University of Antwerp
Faculty of Sciences
Department of Computer Science

Towards an Energy-efficient, Responsive and Reliable Industrial Internet of Things

Glenn Daneels

IDLab
INTERNET & DATA LAB

Submitted in fulfillment of the
requirements for the degree of
Doctor in Science: Computer Science
Academic year 2020-2021

University of Antwerp
Faculty of Sciences
Department of Computer Science

Doctoral committee

Chairman
Prof. Dr. Chris Blondia (UAntwerpen, Belgium)
Supervisors
Prof. Dr. Jeroen Famaey (UAntwerpen, Belgium)
Prof. Dr. Steven Latré (UAntwerpen, Belgium)
Other members
Dr. Carmen Delgado (i2CAT Foundation, Spain)
Prof. Dr. Herbert Peremans (UAntwerpen, Belgium)
Prof. Dr. Sofie Pollin (KULeuven, Belgium)
Prof. Dr. Xavier Vilajosana (Universitat Oberta de Catalunya, Spain)

University of Antwerp
Faculty of Sciences

Department of Computer Science
Sint-Pietersvliet 7, B-2000 Antwerp, Belgium

Submitted in fulfillment of the
requirements for the degree of
Doctor in Science: Computer Science
Academic year 2020-2021

Samenvatting

Het internet der dingen (IoT) paradigma is een verschuiving naar een wereld waarin alle dingen geconnecteerd zijn met het Internet. Terwijl het IoT een impact heeft op elk aspect van onze samenleving, wordt het specifiek heel efficiënt toegepast bij de verdere revolutie van de automatisatie en controle van traditionele fabricage en industriële processen in de huidige vierde industriële revolutie. De introductie van het IoT in de industrie leidt dikwijls tot de term *het industriële internet der dingen (IIoT)*. Om te voldoen aan de hoge eisen van IIoT applicaties, werden industriële sensoren en actuatoren initieel allemaal bedraad verbonden met elkaar. Terwijl het duidelijk is dat bedrade communicatie heel betrouwbaar is, is het ook eerder duur en kan het onpraktisch zijn voor moeilijk te bereiken plaatsen en mobiele machines. Daardoor werd de transitie naar draadloze communicatie onvermijdbaar. Opdat deze transitie echter succesvol zou zijn, moest draadloze communicatie quasi dezelfde betrouwbaarheid als bedrade communicatie kunnen aantonen.

Dus om machines en industriële processen te automatiseren, is uiterst betrouwbare draadloze communicatie nodig in uitdagende omgevingen waar het signaal ernstig kan verstoord worden door de externe storingen (van verschillende aanwezige draadloze technologieën) en vernietigende multipad effecten (door de vele metalen objecten). Daarbovenop is het belangrijk dat zo een draadloos toestel lange tijd operatief kan blijven zonder dat de batterij vervangen moet worden omdat deze toestellen zich dikwijls op een moeilijk te bereiken plaats bevinden. Dat betekent dat tegelijkertijd de communicatie heel betrouwbaar moet zijn en er heel weinig energie mag verbruikt worden. Een relatief recente technologie, genaamd IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH), heeft bewezen aan deze eisen te kunnen voldoen. Het combineert frequentie diversiteit met strikte tijdssynchronisatie en slaagt erin om een betrouwbaarheid van meer dan 99.999% te behalen, terwijl het weinig energie verbruikt. Daarom wordt TSCH tegenwoordig gebruikt als de Medium Access Control (MAC) laag van de meer en meer populair wordende 6TiSCH netwerk architectuur, die industriële netwerken integreert in de traditionele IPv6 internetarchitectuur.

In dit doctoraatsboek bestudeer ik deze TSCH MAC laag en hoe deze verder

kan verbeterd worden zodat deze nog meer succesvol kan gebruikt worden in industriële netwerken. Meer specifiek focus ik op 3 onderzoeksvragen met betrekking tot het energieverbruik, de wachttijd en de betrouwbaardheid van TSCH netwerken. Eerst onderzoek ik hoe het energieverbruik van TSCH kan gekarakteriseerd worden. Daarvoor stel ik een accuraat energiemodel op, zowel van toepassing in de sub-GHz en 2.4 GHz frequentie band, gebruikmakend van hedendaagse IIoT hardwaretoestellen. Dit model laat toe toekomstig TSCH onderzoek te toetsen aan de IIoT eisen in termen van laag energievebruik. Hierna verlegt deze thesis de focus naar het minimaliseren van de wachttijd voor het afleveren van periodieke observatiedata, typisch voor IIoT netwerken. Ik stel een nieuwe gedistribueerde TSCH slot allocatiestrategie voor, genaamd *Recurrent Low-Latency Scheduling Function (ReSF)*. Deze strategie alloceert de draadloze middelen op een intelligente wijze zodat de wachttijd geminimaliseerd wordt en tegelijkertijd wordt het periodieke karakter van de data in acht genomen zodat het energieverbruik zo laag mogelijk kan worden gehouden. Ten slotte mik ik in dit boek op het nog verder verbeteren van de betrouwbaarheid van de draadloze communicatie in een industrieel TSCH netwerk, door het simultaan toelaten van verschillende fysieke lagen in éénzelfde netwerk. Terwijl TSCH inherent een hoge betrouwbaarheid biedt door het toepassen van frequentie diversiteit, wordt het momenteel altijd gelimiteerd door de gekozen onderliggende fysieke laag. Om zo'n meerdere fysieke lagen in hetzelfde netwerk te combineren en een toestel toe te laten om de fysieke laag aan te passen aan de propagatiekarakteristieken van zijn draadloze link, stel ik de *slot bonding* techniek voor. De slot bonding aanpak verbetert het aantal datapakketten dat alle toestellen in het netwerk kunnen afleveren doordat het de gealloceerde draadloze middelen op een efficiënte wijze aanpast aan de karakteristieken van de gekozen fysieke laag. Daarbij wordt er ook een heuristiek voorgesteld die TSCH toestellen toelaat om een goede datalink te selecteren in een netwerk dat meerdere fysieke lagen tegelijkertijd ondersteunt.

Samengevat is het doel van deze doctoraatsthesis het opzetten van een energie-efficiënt, responsief en betrouwbaar TSCH netwerk dat geschikt is voor het gebruik in een IIoT omgeving.

Abstract

The Internet of Things (IoT) paradigm is the shift towards a world where all things are connected to the Internet. While nowadays IoT has an impact on every aspect of society, it is being applied particularly efficiently to further revolutionize the automation and control of traditional manufacturing and industrial processes in the current ongoing fourth industrial revolution. This introduction of IoT in industry often leads to the term Industrial Internet of Things (IIoT). To fulfill the high-end requirements of IIoT applications, interconnecting all sensing and actuating devices was initially typically done through wiring. While the reliability advantage of wiring is obvious, it is costly and can be impractical in hard to reach locations or mobile machinery. Therefore, the transition to wireless communication seemed unavoidable and is becoming more and more ubiquitous. However, for this transition to be successful, wireless communication should show wire-like reliability.

Thus to automate machinery and industrial processes, highly reliable wireless communication is required in harsh environments that suffer from external interference (from different wireless technologies) and multi-path fading effects (due to metal infrastructure) that may easily disrupt the wireless signal. Additionally, to avoid having to frequently replace batteries in inconvenient places, the wireless devices should be able to run on limited battery capacity for years. Therefore, while required to be highly reliable, they should also exhibit (ultra) low-power operations. A relatively recent wireless technique that has proven to be successful in fulfilling these requirement, is IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH), that combines frequency diversity with strict time-synchronization, achieving wired-like reliability of more than 99.999% while having ultra-low power consumption. As such, TSCH is used as the Medium Access Control (MAC) layer for the increasingly popular 6TiSCH network stack that enables the integration of industrial networks in the traditional IPv6 Internet architecture.

In this PhD thesis, I study the TSCH MAC layer and how it can be further improved to deploy it successfully in industrial networks. More specifically, I focus on 3 research questions related to the energy consumption, latency and reliabil-

ity of TSCH networks. First, I investigate how the TSCH power consumption can be precisely characterized. Therefore, an accurate TSCH energy consumption model has been defined, for both the sub-GHz and 2.4 GHz frequency bands, using state-of-the-art IIoT hardware. This allows future research efforts to think about new TSCH solutions that satisfy the low-power operation requirement of IIoT. Second, I aim at minimizing the communication delay of recurrent monitoring data that is typical for IIoT applications. I propose a new distributed TSCH scheduling approach, called the *Recurrent Low-Latency Scheduling Function*, that intelligently allocates schedule resources to minimize the latency and takes into account the recurrent pattern of the traffic to maintain the low-power operation. Finally, I aim at further improving the industrial network's overall reliability by introducing multiple physical (PHY) layers simultaneously in a single TSCH network. While TSCH inherently introduces frequency diversity to enhance its reliability, currently it is limited by the characteristics of the chosen PHY layer. To facilitate the use of multiple PHYs within a single TSCH network and allow each device to adapt its PHY layer to the link's propagation characteristics, I present a method called *slot bonding*. The slot bonding approach improves the network's number of delivered packets by adapting the allocated resources to each PHY's requirements in an efficient manner. Additionally, a heuristic is also proposed that helps a device in making an appropriate parent and PHY selection in multi-PHY TSCH networks.

In summary, this PhD thesis targets an energy-efficient, responsive and reliable TSCH network, thereby contributing to a wireless network deployment that is ready for the IIoT.

Acknowledgements

A PhD is not completed individually, it is definitely a team effort.

My special thanks goes out to Jeroen Famaey. I could not have conduct the research and written this thesis without his mentorship. His continuous support, helpful advice and intelligent feedback shaped me as a researcher. It is quite remarkable how Jeroen, as a supervisor, always sees the bigger picture, while at the same time he is always available to help you out, even with the smallest detail. Throughout the years I saw not only myself evolve (under the guidance of Jeroen), but also Jeroen himself (and his haircut): he deserves absolute credit for being a boss without being *a boss* and building a solid, research-oriented team of excellent researchers and friends. Thank you, Jeroen.

I would also like to express my gratitude to Steven Latré. I vividly remember how excited I was when Steven hired me to become his first PhD student in Antwerp, more than six years ago. In that first period of my PhD, Steven introduced me to the world of research and proofed to be a kind, intelligent and motivated supervisor. It is quite incredible what Steven has achieved with this group, and I have always been proud to be a part of that from the start. For those of you who were not there in the beginning, a fitting example of this metamorphosis: while now you are enjoying your delicious free fruit at your own coffee bar Café René while playing some Ms. Pac-man, back then we were sipping coffee in a dark, chilly server room (that fitted max. 6 people) above the library at Middelheim.

I also have to thank Carmen Delgado. Carmen joined our team at a later stage in my PhD, but her continuous help, intelligence, kindness and answers to my many questions, lifted this PhD definitely to a higher level. I am also grateful to Chris Blondia, one of the founding fathers of our group, who introduced me to the wonderful world of telecommunications when I was still a student at the university. Later during my PhD, he was also of invaluable help cracking the more theoretical puzzles that had to be solved.

In addition to Jeroen, Steven, Carmen and Chris, my sincere thanks also

goes to the people that complete my doctoral committee: Herbert Peremans, Sofie Pollin and Xavier Vilajosana. Thank you for willing to read this thesis book thoroughly and providing me with insightful comments that have made it more rigorous and complete.

I also have to especially thank Esteban Municio: we spent many years in the same office, while enjoying the only air conditioner available, contributing to each other's research and working closely together on the same topic and projects. Not only Esteban's Spanish accent (that always reminded me of holidays), but also his intelligence, hard labor, friendliness and relaxed state of being, made this PhD so much easier, more fun and interesting. I am also grateful to Bart Spinnewyn, who contributed to the research in this PhD, and also offered me lots of interesting and entertaining discussions during a lot of coffee breaks. Additionally, I would also like to thank Bruno Van de Velde who - as a student - made a very valuable contribution to this thesis.

I made unforgettable memories while working with an incredible team of talented, friendly and generous colleagues. I enjoyed every moment of taking too many breaks, having endless discussions, drinking too much coffee, cleaning out Middelheim's coffee machine, playing ping pong, learning about other cultures, drinking beer and eating MEGA mix-max plates at BierCentral, etc. I am proud that I can call many of you my friends now. Listing everyone is probably as though as completing this PhD, but I will try, thanks: Le, Miguel, Serena, Pedro, Niels, Ensar, Sean, Filip, Patrick, Jakob, Tom, Ruben, Jeremy, Carlos, Ashish, Adnan, Raja, Nelson, Paola, Johan, Daniel, Kathleen, Bart B., Bart L., Bart S., Farouk, Pierre, Henrique, Yorick, Daniel, Jan, Sean, Przemyslaw, Maarten, Dragan, Glenn, Christophe, Lynn, Hanne, Anne and Céline. I also have to thank many IDLab people of Ghent University with who I had many fruitful collaborations, thanks Steven, Mathias, Pieter, Gilles, Robbe, Dries, Jan, Jeroen and Eli.

Of course, I also have to thank my friends and family. Indirectly, they contributed a huge part (that can *not* be underestimated) to this thesis. Thank you for your love, patience, encouragement and the absolutely necessary distraction.

Hanne, als één persoon mede-auteur moet zijn van dit boek, dan ben jij dat. Bedankt voor jouw onvoorwaardelijke liefde en opofferingen (ik besef, het waren er veel), die mij toegelaten hebben dit boek te vormen tot wat het is. Daarnaast tonen jij en Renée mij dagelijks wat écht belangrijk is in het leven, ik hou van jullie.

Aan mijn lieve ouders, Bea en Raoul. Zonder jullie liefde, motivatie, geduld en alle mogelijke kansen die jullie mij altijd gaven, had dit boek nooit tot stand gekomen. Daarom, dit boek behoort jullie toe. Ik hou van jullie, bedankt voor alles.

April 2021, Antwerp
Glenn Daneels

"... *ba babaa babababaaa* ..."

My 1.5-year-old daughter Renée, not caring at all about this PhD.

And rightfully so.

Table of Contents

List of Figures

List of Tables

List of Acronyms

3GPP	3rd Generation Partnership Project
6LoWPAN	IPv6 Over Low-Power Wireless Personal Area Network
6P	6top Protocol
6TiSCH	IPv6 Over The TSCH Mode Of IEEE 802.15.4e
6top	6TiSCH Operation Sublayer
ACK	Acknowledgement
AGT	Acknowledgment Guard Time
ARQ	Automatic Repeat ReQuest
ASN	Absolute Sequence Number
BER	Bit Error Rate
BLE	Bluetooth Low Energy
CA	Collision Avoidance
CDF	Cumulative Distribution Function
CDU	Channel Distribution And Usage
COAP	Constrained Application Protocol
COJP	Constrained Join Protocol
CONAMO	Continuous Athlete Monitoring
CRC	Cyclic Redundancy Check
CSI	Channel State Information
DAO	Destination Advertisement Object
DeTAS	Decentralized Traffic Aware Scheduling
DIO	Destination Oriented Directed Acyclic Graph Information Object
DODAG	Destination Oriented Directed Acyclic Graph
DSME	Deterministic And Synchronous Multichannel Extension
EB	Enhanced Beacon
ELLSF	Enhanced Low Latency Scheduling Function
ETX	Expected Transmission Count
FEC	Forward Error Correction
FSK	Frequency Shift Keying
GA	Genetic Algorithm
GMPLS	Generalised Multiprotocol Label Switching
GND	Ground
GUI	Graphical User Interface

HART	Highway Addressable Remote Transducer
HTTP	Hypertext Transfer Protocol
ICMPv6	Internet Control Message Protocol For IPv6
IE	Information Element
IETF	Internet Engineering Task Force
IIoT	Industrial Internet Of Things
ILP	Integer Linear Program
IoT	Internet Of Things
IP	Internet Protocol
IPv6	Internet Protocol Version 6
ISM	Industrial, Scientific And Medical
ITU-R	International Telecommunication Union - Radiocommunications Sector
IWSAN	Industrial Wireless Sensor And Actuator Network
JP	Join Proxy
JRC	Join Registrar/Coordinator
LCM	Least Common Multiple
LLN	Low-power Lossy Network
LLSF	Low Latency Scheduling Function
LQE	Link Quality Estimation
MAC	Medium Access Control
MCF	Multi Commodity Flow
MCS	Modulation And Coding Scheme
MILP	Mixed Integer Linear Program
MRHOF	Minimum Rank With Hysteresis Objective Function
MSF	Minimal Scheduling Function
MTU	Maximum Transmission Unit
MuSCLe-IoT	Multimodal Sub-Gigahertz Communication And Localisation For Low-Power IoT Applications
NB-IoT	Narrowband-IoT
O-QPSK	Offset-Quadrature Phase Shift Keying
OF	Objective Function
OF0	Objective Function Zero
OFDM	Orthogonal Frequency Division Multiplexing
OH	Overhearing
OSCORE	Object Security For Constrained RESTful Environments
OTF	On-the-Fly Scheduling Function
PDR	Packet Delivery Ratio
PGT	Packet Guard Time
PHR	PHY Header
PHY	Physical
PRR	Packet Reception Rate
PSDU	Physical Service Data Unit
RAW	Reliable And Available Wireless
RE	Replication And Elimination
ReSF	Recurrent Low-Latency Scheduling Function
RFC	Request-for-Comments
RPL	Routing Protocol For Low-power And Lossy Network

RSME	Root Mean Squared Error
RSSI	Received Signal Strength Indicator
RSVP	Resource Reservation Protocol
RSVP-TE	Resource Reservation Protocol - Traffic Engineering
SCADA	Supervisory Control And Data Acquisition
SF	Scheduling Function
SF0	Scheduling Function Zero
SF1	Scheduling Function One
SFD	Start-of-Frame Delimiter
SFX	6TiSCH Experimental Scheduling Function
SHR	Synchronization Header
SoC	System-on-a-Chip
SPI	Serial Peripheral Interface
SUN	Smart Utility Network
SUN-FSK	Frequency Shift Keying
SUN-OFDM	Orthogonal Frequency Division Multiplexing
SUN-OQPSK	Offset-Quadrature Phase Shift Keying
SWO	Serial Wire Output
TASA	Traffic Aware Scheduling Algorithm
TCP	Transport Control Protocol
TDMA	Time Division Multiple Access
TSCH	Time-Slotted Channel Hopping
TSMP	Time Synchronized Mesh Protocol
UDP	User Diagram Protocol
VCC	Voltage Common Collector
WG	Working Group
WSN	Wireless Sensor Network
YANS	Yet Another Network Simulator

Chapter 1

Introduction

1.1 Context

The Internet of Things (IoT) paradigm is the shift towards a world where all *things* are connected to the Internet. Those connected objects can be, among others, home appliances, vehicles, or industrial machinery, e.g., a fridge, your car or an industrial valve. However, not only things are being connected as also living organisms are becoming part of this interconnected ecosystem, e.g., farm animals, wildlife or humans in a sports or healthcare context. The paradigm allows for a wide variety of new innovations that should connect, monitor, analyse and improve our lives and the world around us. As the possibilities seem endless, the number of devices being connected keeps increasing, with predictions claiming that in 2025 there will be 8 connected devices per person [1]. Concisely, IoT can be summarized as "*people and things to be connected anytime, anyplace, with anything and anyone, ideally using any path/network and any service*" [2].

While nowadays IoT has an impact on every aspect of society, it is being applied particularly efficiently to support the current ongoing fourth industrial revolution, called *Industry 4.0*. While the first revolution introduced mechanization, the second revolution launched the intensive use of electrical energy and the third focused on widespread digitization, this fourth revolution's focal point is the use of *smart* machinery combined with IoT communication technology, allowing to further revolutionize the automation and control of traditional manufacturing and industrial processes [3]. This introduction of IoT in industry often leads to the term *Industrial Internet of Things (IIoT)*. Achieving industrial automation and control typically means the management of thousands of sensors (e.g, temperature, pressure, position) and actuators (e.g., robotic arms, remotely controlled switches) that respectively sense the industrial site and control and execute repetitive and high-precision processes [4]. Examples of industrial automation, control and monitoring applications include, among

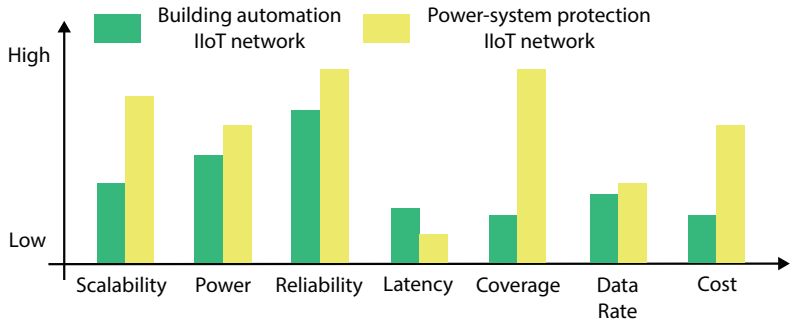


Figure 1.1: A (sub-)set of important metrics for IIoT networks, in this case a building automation and power-system protection network. Exact metric trade-offs depend on the industrial context and the applications running on the network.

others, automated manufacturing robots, oil and gas automation on offshore drilling rigs, the localization of stacked tank containers on freight trains and the monitoring of valve positions.

To fulfill the high-end requirements of industrial automation and process control, interconnecting all sensing and actuating devices was initially typically done through *wiring*. While the reliability advantage of wiring is obvious, it is rather costly (approx. 20 $/m) and can be impractical in hard to reach locations or rotating/mobile machinery [5]. Therefore, the transition to wireless communication seemed unavoidable and is becoming more and more ubiquitous. However, for this transition to be successful, wireless communication should show similar wire-like reliability. Achieving this in harsh industrial environments is a challenge as external interference (from other wireless technologies) and multi-path fading effects (due to metal infrastructure) may easily disrupt the wireless signal. Additionally, to avoid having to frequently replace batteries in inconvenient places, the wireless devices should be able to run on limited battery capacity for years. Therefore, while fulfilling the strict reliability requirement, they should also exhibit (ultra) low-power operations.

As the industrial automation and control processes range from, among others, safety systems and closed loop supervisory systems to open loop systems and alerting systems, there is a wider variety of performance metrics (than only reliability and low energy consumption) in which so-called Industrial Wireless Sensor and Actuator Networks (IWSANs) should score well. Figure 1.1 shows a (sub-)set of important metrics for IIoT networks [5]. These metrics include deployment and maintenance cost (e.g., *what is the cost to replace a broken device?*), coverage (e.g., *can the network cover the whole industrial site?*), scalability (e.g., *how many devices can be deployed without saturating the network?*), data rate (e.g., *will the network be fast enough to transmit all data?*), latency (e.g., *what is the maximum delay when receiving monitored data?*),

power (e.g., *how many times does the device need battery replacement?*) and reliability (e.g., *will all devices have a stable connection to deliver the data?*). Of course, it is dependent on the IIoT application which metrics are most important. Additionally, there is a growing need for industrial networks to seamlessly integrate into the traditional Internet architecture, allowing supervisory control and data acquisition (SCADA) systems to evolve to cloud-based solutions [4]. Currently, there are a lot of wireless contenders that are all characterised by their own set of (dis-)advantages for use in an industrial context, such as Narrowband-IoT (NB-IoT), SigFox, LoRa, WirelessHart, Bluetooth Low Energy (BLE), IEEE 802.11ah (Wi-Fi HaLow) and IEEE802.15.4e Time-Slotted Channel Hopping (TSCH) [6, 7, 8, 9, 10, 11, 12].

Among all of these technologies, IEEE 802.15.4e TSCH has become an increasingly popular wireless solution for IIoT applications. Its combination of a Time Division Multiple Access (TDMA) approach and frequency diversity has been proven to be highly reliable and capable of low-power operation [13, 14, 15]. Additionally, this approach has a successful track record in predecessors such as Time Synchronized Mesh Protocol (TSMP), WirelessHART and ISA100.11A [8, 14, 16]. TSCH can run on a varied set of physical (PHY) layers (i.e., the IEEE 802.15.4g amendment introduced several new PHYs), in both the sub-GHz and 2.4 GHz frequency bands and for different data rates, allowing reported ranges up to 800 meters (i.e., in non-industrial environments) and a robust and full coverage of industrial sites [17, 18]. Furthermore, the Internet Engineering Task Force (IETF) started the IPv6 over the TSCH mode of IEEE 802.15.4e (6TiSCH) Working Group (WG), acknowledging the necessity of an industrial network integration in the traditional Internet architecture. 6TiSCH defines a complete Internet Protocol Version 6 (IPv6) compliant stack, built on top of the IEEE 802.15.4e TSCH Medium Access Control (MAC) layer, thereby offering a complete solution for IIoT applications.

1.2 Problem Statement

While the time-synchronized nature of TSCH provides low-power operation (i.e., by only turning on the radio when nodes have to transmit/receive and sleep otherwise) and the channel hopping feature enhances the wireless reliability (i.e., by pseudo-randomly changing frequency channels for every transmission), a lot of research still has to be carried out to investigate how 6TiSCH networks can be optimized for IIoT applications. A crucial element is the scheduling of TSCH resources between nodes, which is implementation-dependent (thus not standardized). The scheduling approach influences almost all important metrics such as energy consumption, latency, throughput, scalability and reliability. An important IIoT challenge for TSCH scheduling is minimizing the communication delay for (time-critical) industrial processes, while maintaining the device's low-power operation [5]. A scheduling approach can also affect the network reliability by adapting the resources to different link conditions, but it will

always be limited by the characteristics of the underlying PHY layer, i.e., the communication range, bandwidth, data rate, energy consumption and reliability. As it is shown that the support for multiple PHY layers in such harsh industrial environments does further improve the reliability, it is a remaining challenge of introducing these different PHYs in a TSCH network simultaneously [18, 19].

To move the research forward on these topics, the research questions of this thesis are as follows:

1. **How do we precisely characterize the TSCH power consumption?** An accurate power consumption model allows researchers to quickly characterize the energy consumption of new network topology formations, new TSCH scheduling approaches or higher-layer algorithms/applications that satisfy the low-power operation requirements of IIoT.

2. **How do we achieve low-latency communication in a TSCH network?** Minimizing the communication delay while maintaining the TSCH low-power operation is an important requirement for (time-critical) industrial automation and control processes.

3. **How do we use multiple PHYs in TSCH to increase the reliability?** While TSCH inherently introduces frequency diversity to enhance its reliability, it is limited by the characteristics of the chosen PHY layer. The industrial network's overall reliability could clearly benefit from the introduction of multiple PHY layers in a single network to adapt the PHY layer to the link's propagation characteristics. Additionally, when multiple PHYs are available, an important, second question rises: *How do we select the most appropriate link and PHY layer to achieve the best network performance?*

1.3 Research Contributions

The research presented in this thesis focuses on the IEEE 802.15.4e TSCH MAC mode and the improvement of its suitability for IIoT by answering the research questions posed in the previous section. Following is a list of contributions:

1. An accurate and extendable energy consumption model for IEEE 802.15.4e TSCH mode, for both the sub-GHz and 2.4 GHz frequency bands. This model is accompanied with an elaborate set of TSCH state duration and energy consumption measurements for state-of-the-art IoT hardware and firmware (i.e., **Chapter 3**).

2. A new TSCH distributed scheduling approach, called Recurrent Low-Latency Scheduling Function (ReSF). This scheduling approach minimizes the latency of recurrent traffic, such as sensor data with a fixed reporting interval, while keeping the energy consumption to a minimum (i.e., **Chapter 4**).

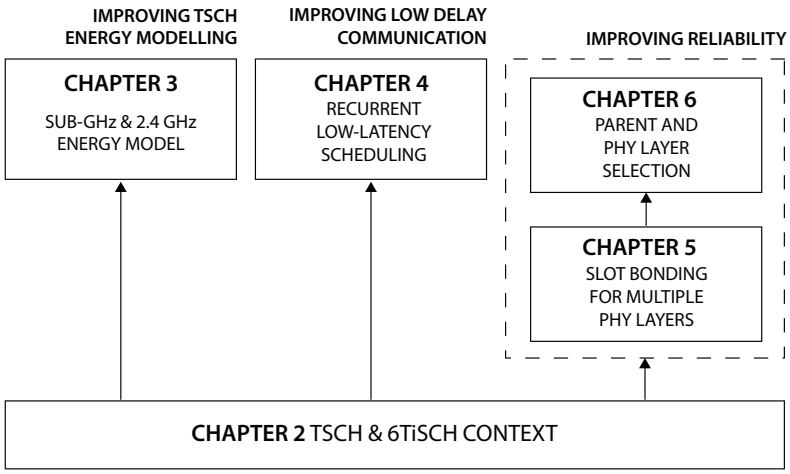


Figure 1.2: Schematic overview of the structure of this dissertation.

3. A method of facilitating the use of multiple PHYs within a single TSCH network, called *slot bonding*. The slot bonding approach allows improving the network's number of delivered packets by efficiently adapting the allocated resources to each PHY's requirements (i.e., **Chapter 5**). Additionally, a distributed heuristic is provided that allows nodes in a slot bonding TSCH network with multiple available PHYs to select an appropriate parent and PHY to optimize the network's number of delivered packets (i.e., **Chapter 6**).

1.4 Outline

In this section, I present an outline of this PhD thesis that addresses the research questions and the different contributions. This outline is illustrated in Figure 1.2. The thesis is structured as follows:

Chapter 2 introduces the IEEE 802.15.4e TSCH MAC mode that provides high reliability and low-power operation in the IIoT. Afterwards, I present the IoT stack and specifications built on top of TSCH, defined by the IETF 6TiSCH WG to bring the Internet Protocol (IP)-ecosystem to constrained devices, filling in the need of industrial networks being able to seamlessly integrate into the traditional Internet architecture.

Chapter 3 focuses on the low-power operation requirements for IIoT. The chapter presents an accurate and extendable energy consumption model for the IEEE 802.15.4e TSCH mode, for both the sub-GHz and 2.4 GHz frequency bands. The model is defined for an elaborate set of time slots and states for variable packet sizes, while using state-of-the-art IoT hardware and firmware. Additionally, it provides the detailed set of state duration values and state energy

consumption measurements. Finally, using simulation the proposed model is used to gain insights in the end-to-end performance and energy consumption of TSCH networks in the sub-GHz and 2.4 GHz frequency bands. The work in this chapter is based on [20].

Chapter 4 takes on the challenge of providing low delay communication in (industrial) IoT. I take into account the recurrent nature of industrial monitoring applications and apply this knowledge in a novel TSCH scheduling mechanism, called ReSF. ReSF is a distributed TSCH scheduling approach that targets low-latency communication while keeping the energy consumption to a minimum by only activating energy-demanding resources when necessary. The work in this chapter is based on [21, 22].

Chapter 5 targets the improvement of TSCH reliability in harsh industrial environments. In traditional TSCH networks all nodes run on one and the same IEEE 802.15.4 PHY. The performance of these nodes could significantly improve from different PHYs being available and adapting the PHY of each link to the local propagation characteristics and the application's requirements. Therefore, I present the TSCH slot and channel bonding techniques that facilitate the use of multiple PHYs with different data rates and bandwidths within a single TSCH network. I formulate the TSCH slot bonding problem and solve it in a near-optimal fashion to show the resource-efficiency of the proposed slot bonding approach and its beneficial effect on the network's overall packet delivery ratio. The work in this chapter is based on [23, 24].

Chapter 6 transfers the slot bonding approach of Chapter 5 to a real-world hardware implementation. First, I define a heuristic that can be integrated in distributed TSCH scheduling approaches and/or a routing objective function and selects the preferred routing parent and PHY to that parent to optimize the packet delivery ratio. The heuristic approximates the near-optimal solutions to the slot bonding problem of Chapter 5. I combine it with the slot bonding hardware implementation and validate it in an office sensor testbed, showing the advantages of the multiple PHY layers in a single TSCH network. The work in this chapter is based on [25].

1.5 Publications

This section lists all publications, published in international journals and proceedings of international conferences, that directly or indirectly contributed to this PhD dissertation.

1.5.1 A1: Publications in international journals indexed by the Web of Science

1. Steven Bohez, **Glenn Daneels**, Lander Van Herzeele, Niels Van Kets, Sam Decrock, Mathias De Geyter, Glenn Van Wallendael, Peter Lambert, Bart Dhoedt, Pieter Simoens, Steven Latré, and Jeroen Famaey (2018). *The*

Crowd as a Cameraman: On-stage Display of Crowdsourced Mobile Video at Large-scale Events. Multimedia Tools and Applications, 77(1), 597-629. [Impact Factor: 2.313]

2. **Glenn Daneels**, Esteban Municio, Bruno Van de Velde, Glenn Ergeerts, Maarten Weyn, Steven Latré, and Jeroen Famaey (2018). *Accurate Energy Consumption Modeling of IEEE 802.15.4e TSCH using Dual-band OpenMote Hardware.* Sensors, 18(2), 437. [Impact Factor: 3.275]

3. **Glenn Daneels**, Bart Spinnewyn, Steven Latré, and Jeroen Famaey (2018). *ReSF: Recurrent Low-latency Scheduling in IEEE 802.15.4e TSCH Networks.* Ad Hoc Networks, 69, 100-114. [Impact Factor: 3.643]

4. Esteban Municio, **Glenn Daneels**, Mališa Vučinić, Steven Latré, Jeroen Famaey, Yasuyuki Tanaka, Keoma Brun, Kazushi Muraoka, Xavier Vilajosana, and Thomas Watteyne (2019). *Simulating 6TiSCH Networks*. Transactions on Emerging Telecommunications Technologies, 30(3), e3494. [Impact Factor: 1.594]

5. Esteban Municio, **Glenn Daneels**, Mathias De Brouwer, Femke Ongenae, Filip De Turck, Bart Braem, Jeroen Famaey, and Steven Latré (2019). *Continuous Athlete Monitoring in Challenging Cycling Environments using IoT Technologies*. IEEE Internet of Things Journal, 6(6), 10875-10887. [Impact Factor: 9.936]

6. Robbe Elsas, Jeroen Hoebeke, Dries Van Leemput, Adnan Shahid, **Glenn Daneels**, Jeroen Famaey, and Eli De Poorter (2020). *Intra-Network Interference Robustness: An Empirical Evaluation of IEEE 802.15.4-2015 SUN-OFDM*. Electronics, 9(10), 1691. [Impact Factor: 2.412]

7. **Glenn Daneels**, Carmen Delgado, Robbe Elsas, Eli De Poorter, Steven Latré, Chris Blondia, and Jeroen Famaey (2021). *Slot Bonding for Adaptive Modulations in IEEE 802.15.4e TSCH Networks*. IEEE Internet of Things Journal. [Impact Factor: 9.936]

1.5.2 Publications (to be) submitted in international journals indexed by the Web of Science

1. **Glenn Daneels**, Dries Van Leemput, Carmen Delgado, Steven Latré, Eli De Poorter and Jeroen Famaey. (2021). *Parent and PHY Selection in TSCH Slot Bonding Networks*. To be submitted.

2. Serena Santi, Tobia De Koninck, **Glenn Daneels**, Filip Lemic and Jeroen Famaey. (2021). *Location-Based Vertical Handovers in Wi-Fi Networks with IEEE 802.11ah* . Submitted to IEEE Access. [Impact Factor: 3.745]

1.5.3 Conference proceedings indexed by the Web of Science

1. **Glenn Daneels**, Jeroen Famaey, Steven Bohez, Pieter Simoens, and Steven Latré (2015, December). *Upstream Content Scheduling in Wi-Fi DenseNets during Large-scale Events*. In 2015 IEEE Globecom Workshops (GC Wkshps) (pp. 1-7). IEEE.

2. **Glenn Daneels**, Esteban Municio, Kathleen Spaey, Gilles Vandewiele, Alexander Dejonghe, Femke Ongenae, Steven Latré, and Jeroen Famaey (2017, October). *Real-time Data Dissemination and Analytics Platform for Challenging IoT Environments*. In 2017 Global Information Infrastructure and Networking Symposium (GIIS) (pp. 23-30). IEEE.

3. **Glenn Daneels**, Steven Latré, and Jeroen Famaey. (2019, June). *Efficient Recurrent Low-Latency Scheduling in IEEE 802.15.4e TSCH Networks*. In 2019 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom) (pp. 1-6). IEEE.

1.5.4 Other international conference proceedings

1. Mathias De Brouwer, Femke Ongenae, **Glenn Daneels**, Esteban Municio, Jeroen Famaey, Steven Latré, and Filip De Turck (2018, April). *Personalized Real-Time Monitoring of Amateur Cyclists on Low-End Devices: Proof-of-Concept & Performance Evaluation*. In Companion Proceedings of the The Web Conference 2018 (pp. 1833-1840).

2. **Glenn Daneels**, Carmen Delgado, Steven Latré, and Jeroen Famaey (2020, June). *Towards Slot Bonding for Adaptive MCS in IEEE 802.15.4e TSCH Networks*. In IEEE International Conference on Communications (ICC). IEEE.

Chapter 2

6TiSCH: Wireless Industrial Networks

2.1 Context

Introducing wireless connectivity in industry is a challenge. To automate machinery and industrial processes, highly reliable communication is required. Therefore, for a long time wired automation solutions were the obvious choice while tolerating the disadvantages such as high installation costs for wiring and being unpractical with rotational devices and mobility [4, 5]. Additionally, the harsh industrial environments, filled with metal infrastructure and the presence of different wireless technologies, make achieving wireless reliability extra difficult. Moreover, if one would succeed in setting up a reliable wireless connection and wiring would not be necessary anymore, frequently replacing batteries of energy-demanding wireless sensors nodes at inconvenient locations would still be a limiting factor.

The unreliable nature of a wireless signal is often caused by external interference and/or multi-path fading. External interference is caused by wireless signals from other present wireless technologies using the same frequency bands that may interfere with and disrupt the transmitted signal. Multi-path fading is caused by the reflection, refraction or diffraction of a transmitted signal, resulting in multiple transmission paths reaching and deteriorating the signal at the receiver, as illustrated in Figure 2.1. As the impact of external interference and multi-path fading was found to vary over frequency and time, frequency diversity techniques such as channel hopping were observed to be successful in increasing the wireless reliability [26]. Furthermore, efforts in combining channel hopping with strict time-synchronization (that results in nodes knowing exactly when to transmit/receive and thereby avoid wasting energy) resulted in achieving wire-like reliability of more than 99.999% while having ultra-low

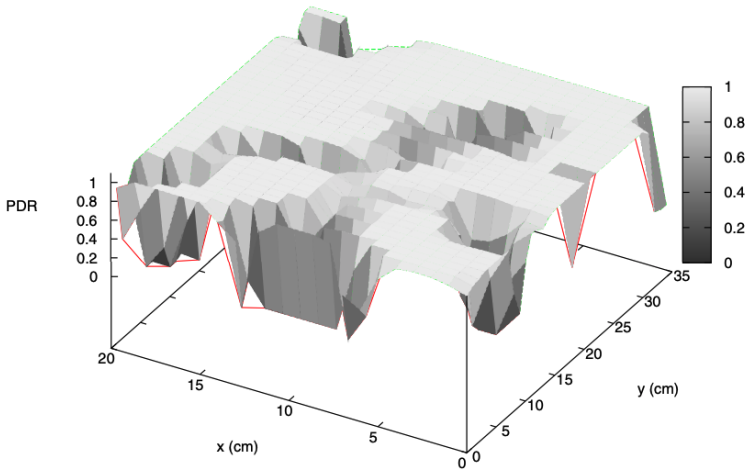


Figure 2.1: Watteyne *et al.* showed the effects of multi-path fading on the packet delivery ratio (PDR) [26]. The figure illustrates that the PDR can heavily fluctuate (from PDR 0 to 1) when only the position of the transmitter changed. Additionally, the authors showed that moving away from PDRs < 0.05 can be done by physically moving the transmitter or changing the operating frequency.

power consumption [13, 14, 15]. As such, in 2007 the widely popular wired highway addressable remote transducer (HART) industrial automation protocol got extended with the backwards compatible WirelessHART, incorporating the principles of time synchronization and frequency diversity [8]. Nowadays, WirelessHART is used worldwide in industrial process automation. In 2012, IEEE published the IEEE 802.15.4e-2012 amendment to the IEEE 802.15.4 standard, that defines PHY and MAC layers for low-power wireless networks and introduced the TSCH MAC mode [12]. Reusing the same core ideas of the earlier introduced TSMP and WirelessHART, TSCH merges a tightly-synchronized time-slotted schedule with channel hopping and was finally added to the standard in the 2015 release [8, 14, 27]. Exactly like its predecessors, the primary goal of TSCH is to provide wire-like reliability while maintaining low-power operation.

2.2 TSCH

In a TSCH network, each node maintains a matrix-formatted schedule, i.e., the so-called Channel Distribution and Usage (CDU) matrix, in which the number of rows equals the number of available physical channels and the number of columns determines the period of scheduling operation (i.e., this equals the so-called *slotframe* size). One element in the schedule is called a *cell* and denoted by $(timeOffset, channelOffset)$, with respectively the time offset and the

channel offset in the schedule. As such, each node is time-synchronized, with time being split up into fixed-duration time slots which are indexed by the time offset and at every slot a node can use 1 of the available channels which is calculated based on the channel offset. A default time slot is long enough to send a packet of 127 bytes and receive an acknowledgement (ACK) (a time slot is typically 10 ms or 15 ms long). Initially, the Maximum Transmission Unit (MTU) was set to 127 bytes, but with the introduction of the Smart Utility Network (SUN) PHYs in the IEEE 802.15.4-2015 standard, the MTU was extended to 2047 bytes [27]. Time slots are grouped in slotframes which are repeated over time, and whose size is application-dependent [28].

To achieve frequency diversity by channel hopping, the channel offset is mapped to the physical channel in a pseudo-random fashion by employing the following calculation:

$$ch = hopSeq[(ASN + channelOffset) \mod hopSeqLen] \tag{2.1}$$

where ch represents the physical channel in which the node will transmit or receive data, $hopSeq$ is a lookup table with all available physical channels, $channelOffset$ is the channel offset and $hopSeqLen$ is the size of the look up table (e.g., 16 channels when using a radio compliant with IEEE 802.15.4 at 2.4 GHz). The Absolute Sequence Number (ASN) defines the absolute time slot in TSCH, shared by all nodes in the network. It starts at 0 being the first ever time slot and increments with 1 in every time slot that follows. It is represented by a 5 byte counter that can support networks for hundreds of years before wrapping (the exact number of years is dependent on the time slot duration). Two (synchronized) communicating nodes that respectively transmit and receive in a negotiated cell, will calculate the same physical channel as they will use the same ASN and $channelOffset$. However, the calculation will result in a different channel in every slotframe, looping through all available channels (i.e., to loop over all available channels, the slotframe length should be a prime number).

The TSCH schedule instructs a node what to do in each slot of a slotframe: sleep, transmit or listen. When a cell is not scheduled, the node will sleep, keeping its energy consumption to the absolute minimum. When a cell is scheduled, it will either transmit or receive. In a *transmission cell*, a node transmits the first packet waiting in the queue to the destination and expects an ACK in the same time slot. If there is no packet in the queue, the node will go to sleep. In a *reception cell* a node is listening for an incoming data packet and answers with an ACK to the transmitter. If no packet was received during a certain guard time, the node goes to sleep. A cell can also be a transmission and reception cell at the same time: when there is no data to send, the node will listen for incoming data. For scheduling a transmission or reception cell, a given neighbor address denotes the neighbor with which the node communicates. However, a cell can also be a *broadcast cell* when the neighbor address equals the broadcast address. By default, a cell is scheduled as *dedicated*: it can only

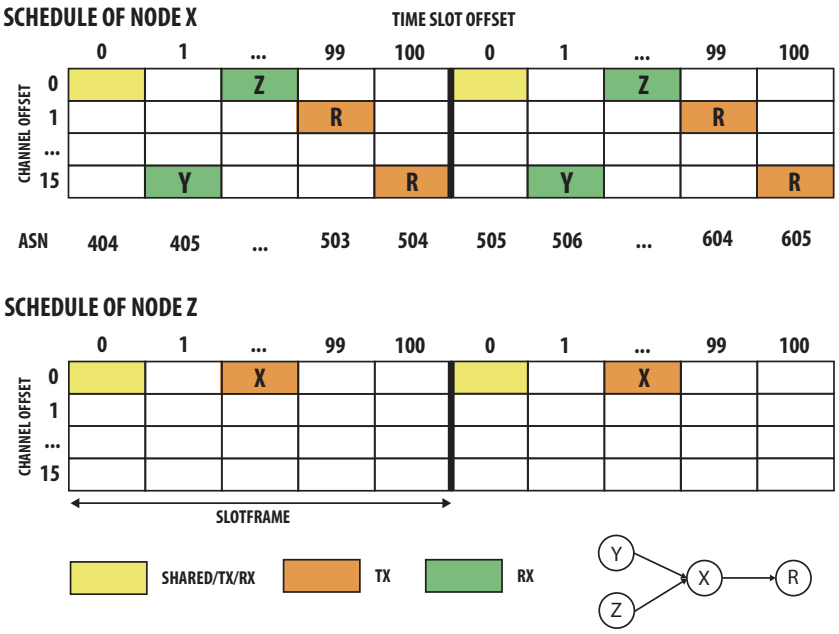


Figure 2.2: Example of a TSCH network, showing the schedules of node X (top) and node Z (bottom).

be used by one node for the transmission of packets to another node. However, the standard also allows a cell to be *shared*: the cell can be used by more than one node for transmission. To decrease the probability of collisions, a retransmission back-off algorithm is applied in shared cells. When a cell is scheduled as a transmission, reception and shared cell at the same time, the cell will have slotted-ALOHA behavior. Figure 2.2 shows a simple topology with 4 nodes and the schedules of both nodes X (i.e., top schedule) and Z (i.e., bottom schedule). All schedules have a shared cell at $(0, 0)$ with slotted-ALOHA behavior to bootstrap the network, and to broadcast network and routing information. Node X has 2 dedicated reception cells for nodes Z and Y and 2 transmission cells towards node R. Node Z has one transmission cell to node X at the same schedule location where X scheduled the reception cell for node Z.

To maintain the necessary synchronization among the nodes in a TSCH network, a node has a time source neighbor to which it regularly synchronizes. There are 2 methods in which a node can synchronize: packet-based and/or acknowledgement-based synchronization. With the packet-based approach, a node adjusts the duration of the current time slot whenever it receives a packet from its time source neighbor by calculating the synchronization error using the expected reception timestamp and the actual reception timestamp. The acknowledgement-based approach is similar, but now the node sends a frame to

its time source neighbor which timestamps the reception of that frame. The time source neighbor then calculates the synchronization error using this timestamp and reports the error back in the Time Correction Information Element (IE) of the acknowledgement. Subsequently, the node adjust its current time slot length based on the reported error. There have been several works that investigated how to further minimize the clock drift between network nodes [29, 30, 31].

2.2.1 PHY Layers

While most TSCH deployments run on top of the O-QPSK PHY layer in the 2.4 GHz version of the standard, the IEEE 802.15.4 standard actually has support for many different PHY layers and operation in both the sub-GHz and 2.4 GHz worldwide Industrial, Scientific and Medical (ISM) frequency bands, allowing reported ranges up to 800 meters (i.e., in non-industrial environments) [17]. For example, the IEEE 802.15.4g-2012 amendment (that initially solely targeted SUN applications) added the Frequency Shift Keying (SUN-FSK), Offset-Quadrature Phase Shift Keying (SUN-OQPSK) and Orthogonal Frequency Division Multiplexing (SUN-OFDM) modulation families [27, 32]. Consequently, the current standard addresses a wide variety of applications, ranging from smart home applications to automation in harsh industrial environments. With proper adjustments of MAC layer transmission and processing timings, TSCH networks can run on many of these PHY layers and even on multiple PHY layers simultaneously, as is researched in Chapters 5 and 6 [18, 19, 33, 34].

2.3 6TiSCH

In the last decade there has been a major effort by the IETF to bring the IP-ecosystem to constrained devices, filling in the need of industrial networks being able to seamlessly integrate into the traditional Internet architecture. Various protocols such as routing, IPv6 adaptation and web transfer protocols were standardized, dealing with limiting factors of constrained networks such as payload size, computing capacity and non-trivial topologies while guaranteeing to be IP-compliant. To integrate these efforts with the industrial performance of the IEEE 802.15.4 TSCH MAC mode, the IETF also introduced the 6TiSCH WG. 6TiSCH provides a management plane for the underlying IEEE 802.15.4 TSCH network, by providing solutions for bootstrapping, security and TSCH schedule management [35]. Additionally, it also proposes a full stack that combines the IEEE 802.15.4 TSCH link layer, the 6TiSCH adaptations, and other higher layer IETF solutions for a fully operating IPv6-compliant industrial IoT network.

2.3.1 Overview

The 6TiSCH WG proposes a full stack from the link layer up to the application layer [35]. In addition, it also provides specifications that specify bootstrapping

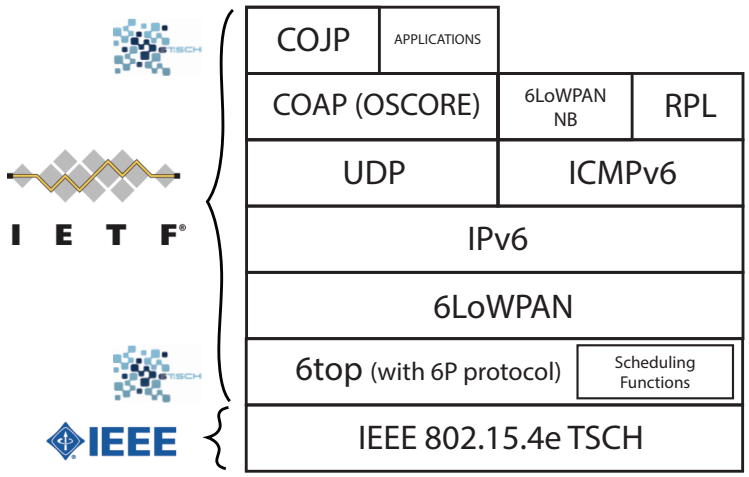


Figure 2.3: The stack proposed by the 6TiSCH WG.

and securely joining a 6TiSCH network and minimal resource scheduling [36, 37, 38]. All these specifications are documented in active IETF drafts or Request-for-Comments (RFC) and many of the proposed concepts are thus still being improved at the time of writing.

A complete overview of the stack proposed by 6TiSCH, starting at the link-layer (more information on possible PHY layers is given in Section 2.2.1), is shown in Figure 2.3. At the link-layer, it is obvious that the IEEE 802.15.4 TSCH MAC mode is used to provide the low-operation and wire-like reliability as discussed in Section 2.2 [27]. The 6TiSCH Operation Sublayer (6top), proposed by the 6TiSCH WG, sits just above the IEEE 802.15.4 TSCH MAC layer and contains the 6top Protocol (6P) protocol and one or more Scheduling Functions (SFs) [39]. The 6P protocol facilitates distributed scheduling in 6TiSCH networks by enabling neighbor negotiation for reserving TSCH cells, as explained in further detail in Section 2.3.2. A SF decides when and how to add TSCH cells to the schedule to answer the application's requirements. Section 2.3.3 gives an overview of different types and examples of 6TiSCH scheduling approaches. The remainder of the 6TiSCH stack proposed is standardized by other IETF WGs (except for Constrained Join Protocol (COJP), which is also developed by the 6TiSCH WG), but collaboration with the 6TiSCH WG and the development of the 6TiSCH specifications has also driven the further improvement of these standards. On top of the 6top sublayer, the IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) layer refers to a set of standards that enable transporting IPv6 datagrams on top of the constrained MAC layer. The entire 6LoWPAN framework contains specifications for IPv6 and User Diagram Protocol (UDP) header compression, neighbor discovery and fragmentation enabling IPv6 in the stack [40, 41, 42, 43, 44]. The Routing Protocol for Low-power and Lossy

network (RPL) is used as it is a specialized routing protocol for wireless networks limited by low power consumption and susceptible to packet loss [45, 46]. RPL uses Internet Control Message Protocol for IPv6 (ICMPv6) to encapsulate its routing control messages [47]. There is no mandatory RPL Objective Function (OF) specific for 6TiSCH networks, as this is left implementation-specific to meet the application's requirements. A more in-depth RPL overview is given in Section 2.3.4. The transport layer uses the connection-less UDP as opposed to a connection-oriented approach such as Transport Control Protocol (TCP) [48]. At the application layer, Constrained Application Protocol (COAP) is chosen as the web transfer protocol. COAP can easily interface with Hypertext Transfer Protocol (HTTP) to integrate with the Web while it was designed for constrained environments, maintaining simplicity and very low overhead [49]. COJP is used to let a new node securely request admission to a 6TiSCH network [37]. Requiring only minimal overhead for joining, COJP is a lightweight protocol over COAP and Object Security for Constrained RESTful Environments (OSCORE), which is a method for application-layer protection of COAP [37, 50].

In addition to the stack, the 6TiSCH WG also proposes 3 additional specifications: a *minimal 6TiSCH profile*, a *minimal security framework* and the *Minimal Scheduling Function (MSF)* [36, 37, 38]:

- The **minimal 6TiSCH profile** describes a minimal mode of operation, providing a baseline set of parameters to configure different key protocols of a 6TiSCH network (such as different TSCH and RPL parameters) [36]. It allocates a so-called *minimal cell* in the TSCH schedule by reserving a shared cell operating in slotted-ALOHA fashion (as explained in Section 2.2) that provides bandwidth for network advertisements (i.e., IEEE 802.15.4 Enhanced Beacons (EBs)) and join traffic, enabling network and security bootstrapping, as shown in Figure 2.2.

- The **minimal security framework** describes exactly how a node securely joins a 6TiSCH network, by providing the COJP protocol [37]. It defines how a new node, called a *pledge*, joins a network by exchanging a join request and join response with a central entity called the Join Registrar/-Coordinator (JRC). To reach the JRC, the pledge communicates over an end-to-end secure channel provided by OSCORE with one of its radio neighbors that already joined the network, called the Join Proxy (JP), and that serves an application-level relay to the JRC. After successful authentication and authorization, the JRC provides the pledge with link-layer cryptographic keys and other parameters (e.g., a short address).

- The **MSF** specification provides a complete joining process (with among others, scanning for EBs with network information, the installment of resource cells to the JP, securely joining the network, the selection of the routing parent and the negotiation of the first transmission cell to/from that routing parent). In addition, it defines a *minimal* scheduling approach, which is considered the default SF of 6TiSCH [38]. To provide all of

this, the MSF specification is built upon the previously described 6TiSCH minimal profile, the 6P protocol and the minimal security framework, and it is also closely related to the RPL standard [36, 37, 39, 45]. In Section 2.3.3.4, we explain the MSF cell management for a joining node and the scheduling approach in more detail.

2.3.2 6top Protocol

6P allows neighboring nodes in a 6TiSCH network to negotiate and subsequently allocate cells in their TSCH schedule [39]. Together with one or more SFs, 6P is part of the 6top layer, i.e., a layer just above the IEEE 802.15.4 TSCH MAC layer. A SF has the task of employing 6P to facilitate the application's requirements.

6P provides commands to add, delete, relocate, count, list and clear cells in the schedule of a neighbor, and additionally it allows to signal generic (SF) commands to other nodes. All such 6P messages are encapsulated within the IEEE 802.15.4 IE structure, which was defined in the standard for the distribution of additional MAC information. A complete negotiation between 2 nodes, is called a *6P transaction*. 6P facilitates both 2-step and 3-step transactions. A 2-step transaction to negotiate new cells, as illustrated in Figure 2.4, between node X and node Y means that node X sends a 6P ADD Request to node Y, specifying the number of cells it wants and the list with cells out of which node Y can pick, with a cell being $(timeOffset, channelOffset)$. From the moment the 6P Request is transmitted to node Y, the suggested cells are locked at node X, meaning that no other transaction going on at node X can use these cells. When the ACK of the 6P Request is received, a timer is set to abort the transaction when the receiver did not respond when the timer expires (i.e., the locked cells will be freed). Node Y answers with a 6P Response containing the cells that suit node Y according to its SF. The cells that node Y responded to node X in the 6P Response are locked until the ACK is received at node Y. In contrast, in a 3-step transaction, not node X but node Y would propose cells in its Response message towards node X. Subsequently, node X would send a confirmation with acceptable cells.

2.3.3 Scheduling Functions

To fulfill the needs of a network application, a 6TiSCH node needs both the 6P protocol and a SF. While the former facilitates the cell negotiation between nodes, the latter decides when to add or delete cells in the TSCH schedule and can try to optimize one or more metrics such as latency and/or energy efficiency. A node may support different SFs at the same time and each SF is identified by a `SFID`. Because of its importance in 6TiSCH networks, there has been a large research effort to scheduling approaches and over the years, also the 6TiSCH WG is doing efforts to standardize a default approach, i.e., MSF [38]. The different SFs can be subdivided in 3 types: centralized, distributed and

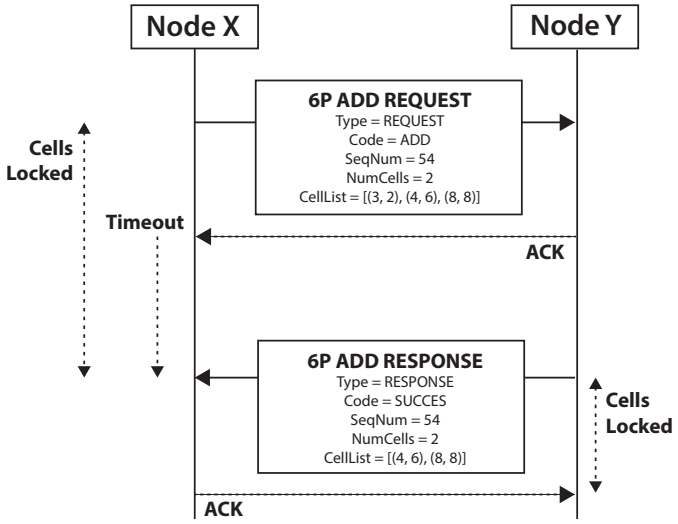


Figure 2.4: A 2-step 6P transaction.

autonomous scheduling. However, a SF can also be a combination of these types, e.g., MSF combines distributed and autonomous scheduling.

2.3.3.1 Centralized Scheduling

A centralized scheduling approach requires a central entity that builds the schedule for all nodes, based on reported or monitored network information. They typically have the advantage of being able to build near-optimal, collision-free schedules that approach the theoretical optimum. However, scalability limitations and the extra signaling overhead make centralized schedulers less interesting for dynamic network topologies. A well-known example of a centralized scheduler is Traffic Aware Scheduling Algorithm (TASA), proposed by Palattella *et al.* [51]. It uses matching and coloring procedures to build a TSCH schedule and assumes full topology and traffic awareness.

2.3.3.2 Distributed Scheduling

In networks with a distributed scheduling approach, nodes maintain their own schedule. They negotiate new cell allocations with neighbor nodes using the 6P protocol. These networks typically scale better compared to centralised approaches because there is less signaling overhead. However, less network information also makes it more complex to build efficient schedules that are responsive to the traffic's requirements, while minimizing energy consumption.

One of the earliest distributed scheduling functions was the 6TiSCH Experimental Scheduling Function (SFX) which was standardized by 6TiSCH WG as

Scheduling Function Zero (SF0) before, but originally introduced as On-the-Fly Scheduling Function (OTF) by Palattella *et al.* [52,53,54]. SFX makes an explicit difference between *allocated* and *used* cells: used cells are the allocated cells which are actually used to send traffic to that neighbor (which can be a subset of the allocated cells to that neighbor). Subsequently, SFX dynamically adapts the amount of allocated cells between two neighbors whenever the number of *used* cells changes. The exact event that triggers when to check if the number of used cells changed, is implementation-specific, but can for example be checked periodically (i.e., a so-called housekeeping period). Whenever SFX is triggered, it runs its cell estimation algorithm that calculates the new number of required cells it would need: it adds the current number of *used* cells and an extra number of cells (i.e., the `overprovision` parameter which is calculated as a percentage of the current number of cells and serves to account for unexpected traffic). This total is passed to the allocation algorithm which checks if the difference between the current number of *allocated* cells and new total required cells exceeds the SFX (de-)allocation threshold. If so, SFX randomly deletes or allocates one or more cells to the neighbor. Based on SF0/SFX, Chang et al. developed the Low Latency Scheduling Function (LLSF) that uses the same traffic adaptation algorithm, but daisy-chains cells over the different links up to the root, rather than picking them randomly [55]. This leads to lower latency for packets traversing from the source node to the root. The idea of daisy-chaining cells was integrated in ReSF, the SF proposed in Chapter 4.

Over the years, research on distributed scheduling approaches has spiked [55, 56, 57, 58, 59, 60, 61, 62, 63]. For example, DeTAS is the distributed mode of the previously mentioned centralised TASA approach [56]. It minimizes buffer overflows and allows for efficient queue management using a so-called *macro*-schedule that contains *micro*-schedules (with alternating TX and RX cells). Municio *et al.* focus on improving scalability with the DeBras scheduler by utilizing selective broadcasting to inform other nodes on each other's schedules [57]. The Wave SF builds a schedule with so-called *waves* with non-conflicting transmissions to target minimised delays of data collected at the sink [58]. In Chapter 4, we discuss more distributed approaches related to low-latency scheduling.

2.3.3.3 Autonomous Scheduling

A recent advancement in TSCH scheduling, is the shift towards autonomous scheduling. Nodes independently add/delete so-called *autonomous cells* to/from their schedule, responding to available information from the routing layer. For example, when a node chooses a new preferred routing parent, the cell to the old parent is removed and a cell to the new parent is automatically added. There is no need for node negotiation (i.e., 6P transactions), thereby avoiding the disadvantages of extra signaling overhead. The calculated time and channel offset of an autonomous transmission/receiving cell are based on a hash of an identifier of the transmitting or receiving node (e.g., MAC address or unique network node ID). S. Duquennoy *et al.* were the first to propose such an approach,

called *Orchestra*. It runs scheduling rules that describe how to maintain TSCH slotframes (for different traffic types) and (autonomous) slots as a function of the routing topology. S. Kim *et al.* proposed the *ALICE* scheduling function that implements directional link-based autonomous scheduling to minimize contention/collision issues that were observed with Orchestra scheduling. *Escalator* is another interesting approach that focuses on minimizing the end-to-end delay by daisy-chaining time slots up to the sink in an autonomous fashion [64], in contrast to the distributed LLSF approach. Finally, *TESLA* is a traffic-aware elastic slotframe adjustment scheme that aims to minimize energy consumption while maintaining the network's high reliability. In contrast to Orchestra and ALICE, it combines autonomous and distributed approaches to offer a robust, but still flexible scheduling solution.

2.3.3.4 Minimal Scheduling Function

MSF is considered as the default scheduling approach for 6TiSCH networks and the successor of the SF0 and SFX SFs [38]. Next to a scheduling approach, its specification also contains the exact steps to manage resource cells while bootstrapping the network. Therefore, the specification is heavily dependent on the *minimal security framework* specification (as explained in Section 2.3.1) and RPL. MSF combines an approach of autonomous cells and distributed scheduling.

MSF uses autonomous cells for initial bootstrapping and the transport of join traffic. A node always keeps an autonomous RX cell installed in the schedule (after synchronization), while it only installs an autonomous TX cell in the schedule when it has to send a frame (and removes it afterwards). During the join process, after receiving EBs advertising the TSCH network, the pledge (i.e., the node that wants to join the network) installs an autonomous transmission (TX) cell to a chosen JP (one of the nodes from which it received EBs). This JP receives the Join Request in its autonomous reception (RX) cell. The values in the (time slot offset, channel offset) tuples that represent these autonomous cells are computed as a hash of the layer 2 address of the receiving node (using the SAX hash function) [65]. When forwarding the Join Response (with cryptography keys and other parameters) from the JRC back to the pledge, the JP also uses an autonomous TX cell to the pledge that receives the Join Response in its autonomous RX cell. In both cases, the autonomous TX cells are removed directly after the transmission. Using those cryptography keys, a node can decrypt incoming routing information messages (i.e., Destination Oriented Directed Acyclic Graph Information Objects (DIOs), see Section 2.3.4) and it can choose a preferred routing parent. When the node has chosen a preferred parent, it installs an autonomous TX cell to this parent to send out a 6P ADD Request to request 1 *negotiated* cell to its parent. The autonomous TX cell is again removed after the transmission of the request. The 6P ADD Request is received in the autonomous RX cell of the parent. Subsequently, the communication of the 6P ADD Response also happens in autonomous TX and RX cells, in opposite direction. At the end of the joining process, the node/pledge

sends out its own EBs and routing messages on the minimal cell, has 1 installed autonomous RX cell and 1 negotiated cell to its preferred parent.

Once a node has joined the network it can add/delete/relocate negotiated cells (i.e., cells scheduled with 6P) with the selected parent in a distributed fashion to dynamically support the application traffic. To do so, it monitors the current usage of the cells it has to its preferred parent by incrementing *numCellElapsed* at each elapsed cell and the *numCellUsed* each time the elapsed cell was effectively used to send or receive a frame to that parent. When numCellElapsed equals *MAX_NUM_CELLS*, MSF checks to see if the number of used cells numCellUsed is within bounds. MSF issues a 6P ADD or DELETE Request depending on if more or less resources are needed to adapt to the traffic. The value of MAX_NUM_CELLS can be changed to adapt to the traffic type of the network (i.e., smaller values reduce latency for bursty traffic, while larger values decrease the 6P overhead in case of periodic traffic), but finding a good value, especially with varying traffic loads, might not always be straightforward [66,67]. MSF also has a relocation procedure for cells of which the PDR is suspiciously low compared with other cells to that same parent. This can happen when two disjoint pairs of nodes allocate the same cell in the schedule (i.e., called a *schedule collision* in the MSF context) and thereby disrupting each other's signal. If this is the case, the node issues a 6P RELOCATION request to its parent to move the cell to another location in the schedule. This method of relocating cells has proven effective by Muraoka *et al.* [68].

2.3.4 RPL

The RPL routing protocol is a distance vector routing protocol that can build bidirectional routes between a border router and up to possibly thousands of resource-constrained nodes [45]. To do so, RPL builds a so-called Destination Oriented Directed Acyclic Graph (DODAG), which is rooted at the border router. This means that the routing topology is organised in a tree topology in which each node can have zero or more child(ren), and one or more parent(s). A node that has no children, is a leaf node, and the node in the tree that does not have any parents, is the root node of the tree. Each network may have one or more RPL Instances, with one Instance containing one (or even more) DODAG(s) with its own root(s). These RPL Instances can be optimized for different application objectives.

The location of a node in the DODAG is determined by the so-called *Rank* value. The node calculates this value itself and disseminates this information, in DIO messages, which are - like all RPL control messages - encapsulated in ICMPv6 messages. These DIO messages contain information that allows a node to discover a RPL Instance, learn its configuration parameters, select a parent set, and maintain the DODAG. Broadcasting of those DIO messages is managed by the Trickle algorithm [69]. To calculate its Rank value, the RPL node applies an Objective Function (OF) that uses one or more routing metrics to approximate the distance of the node to the DODAG root. The OF also defines

how to select the preferred routing parent(s) of the node, which are used to propagate data upwards to the root. OFs are implementation-specific, but the most commonly used ones are Objective Function Zero (OF0) and Minimum Rank with Hysteresis Objective Function (MRHOF) [46, 70]. The former is considered the basic OF for an RPL network, while the latter applies hysteresis to improve Rank stability. While both allow various routing metrics and constraints (which can be disseminated in DIOs), by default both OFs (suggest to) use Expected Transmission Count (ETX) metric to compute the Rank [71]. When a node has selected its routing parent(s), it transmits a Destination Advertisement Object (DAO) message containing destination information upwards along the DODAG in order to also support downwards traffic. RPL supports a *storing* and a *non-storing* mode. In the former, every node stores a routing table with information learnt from received DAOs while in the latter only the root keeps such a table.

The minimal 6TiSCH profile mandates to use RPL in non-storing mode (while nodes which are capable should use the storing mode), and as the OF it must use OF0 combined with the ETX metric [36].

2.3.5 6TiSCH Implementations & Hardware

As the need for IP-compliant IIoT solutions grows, 6TiSCH implementations can be found in more and more reference platforms for constrained (IoT) devices. As such, 6TiSCH can be found in at least the following well-known and established open-source projects aiming at IoT: OpenWSN, Contiki-NG, RIOT and TinyOS [72, 73, 74, 75]. As both OpenWSN and Contiki-NG are used during the research in this thesis, we will shortly highlight them.

The OpenWSN project is a fully standards-based protocol stack based on the IEEE 802.15.4e TSCH link layer, also featuring the IPv6-enabled IETF upper stack (6LoWPAN, RPL, COAP), written in C. OpenWSN is considered the reference 6TiSCH implementation, closely following the standardization efforts of the WG [72, 76]. The OpenWSN firmware is often related with the OpenMote hardware (as the OpenMote company was a spin-off of the OpenWSN project), of which the newest OpenMote B board (see Figure 2.5a[1]) supports the IEEE 802.15.4g-compliant Atmel AT86RF215 radio chip. The earlier OpenMote-CC2538 also supports 2.4 GHz and sub-GHz communication by combining the System-on-a-Chip (SoC) CC2538 (that features the ARM Cortex-M3) and the CC1200 radio chip, both products of Texas Instruments [76, 77]. The OpenMote-CC2538 device is used for the TSCH energy model in Chapter 3, while the OpenMote B is used in Chapter 4. In addition, the OpenWSN platform is also ported to many well-known IoT hardware platforms with among others, the TelosB, Nordic NRF51822 and the IoT Lab M3 devices [78, 79, 80].

Contiki is also an open-source, cross-platform operating system, also written

[1] `https://www.industrialshields.com/web/image/product.template/721/image?unique=3a7b8c9`

(a) The OpenMote B.

(b) The Zolertia RE-Mote.

Figure 2.5: Two 6TiSCH-compliant IIoT devices, supporting the sub-GHz and 2.4 GHz frequency bands.

in C, aiming at constrained devices, while featuring a full IPv6 stack [73]. While already started in 2003, nowadays the Contiki operating system still has a large community in both academia and industry. The Contiki fork *Contiki-NG* was later released in 2017, and focuses on dependable (secure and reliable) low-power communication and standard protocols (for *next-generation* IoT devices), such as IPv6/6LoWPAN, 6TiSCH, RPL, and COAP. Thereby it offers a general clean-up of the original project, together with an updated 6TiSCH with 6P and 6top support (and other features for constrained IoT devices) [81]. While Contiki-NG is also able to run on a variety of state-of-the-art constrained IoT hardware, it is often used on the Zolertia Zoul platform, for example the Zolertia RE-Mote (see Figure 2.5b)[2] [82]. Similar to the OpenMote-CC2538, this board is based on the Texas Instruments' CC2538 SoC, combined with the CC1200 radio chip to support communication in both frequency bands. The Contiki-NG platform together with the Zolertia Re-Mote devices are used for the TSCH slot bonding implementation in Chapter 6.

2.3.6 6TiSCH Simulation

Network simulation allows for evaluating new research contributions in a standard-compliant manner without the practical burden of real-world deployments, while still being more realistic and accurate than mathematical modelling. For those reasons, the *6TiSCH simulator* was developed [83].

The 6TiSCH simulator is a Python-based discrete-event simulator that allows for rapid prototyping in large-scale networks up to several hundreds of nodes. The simulator includes a 6TiSCH standard-compliant implementation (in a behavioral sense) of TSCH, 6P, MSF, COJP, RPL (non-storing mode, OF0), 6LoWPAN Fragmentation and the Minimal 6TiSCH Configuration [36, 37, 38, 39, 45, 84]. As such, one can easily test new research contributions including among

[2] `https://zolertia.io/wp-content/uploads/2017/05/re-mote-1-zolertia.jpg`

others network formation, scheduling approaches and routing optimizations in a full 6TiSCH solution with flexible parameter configuration. Additionally, the simulator integrates the energy model introduced by Vilajosana et al. which allows to determine the energy footprint of the evaluated solutions [77]. Other metrics such as packet delivery ratio and latency are easily retrieved and one can use the web-based Graphical User Interface (GUI) to monitor the network topology, the TSCH schedule and a pre-determined set of key metrics in real-time. Because of its low-complexity and the continuous support of the 6TiSCH WG, the 6TiSCH simulator was used throughout this thesis as the primary simulation tool. As the development of the 6TiSCH simulator is an ongoing process, the simulator implementation state also changed during the different chapters of this book. Therefore, each chapter refers to the specific simulator version that was used to conduct the research.

However, there are also other options such as *Cooja* and *OpenSim* which are both emulators, not simulators, of the Contiki(-NG) and OpenWSN platforms respectively [72, 73, 85, 86]. Being emulators, they run the binary that can also be flashed on the actual hardware allowing for bit-level accuracy of real-world implementations (except for the physical conditions). However simultaneously, it also limits their scalability up to only tens of nodes and the flexibility to implement new solutions rapidly. Recently, *Elsts* proposed a new 6TiSCH discrete event simulator *TSCH-Sim* that shows to be an order of magnitude faster in simulating very large networks (e.g., 10 000 nodes) in comparison with the 6TiSCH simulator and Cooja [87]. Other well-known network simulators such as *NS-3* or *Omnet++* do not support 6TiSCH [88, 89].

Chapter 3

TSCH Energy Modeling

The content of this chapter is partially based on:

- Glenn Daneels, Esteban Municio, Bruno Van de Velde, Glenn Ergeerts, Maarten Weyn, Steven Latré, and Jeroen Famaey (2018). *Accurate Energy Consumption Modeling of IEEE 802.15.4e TSCH using Dual-band OpenMote Hardware.* Sensors, 18(2), 437. [Impact Factor: 3.275]

3.1 Introduction

Low energy consumption is generally expected of connected devices, while at the same time being confronted with challenges such as a low expected manufacturing cost, mobility while being connected and deployment in often difficult-to-reach (industrial) places. This makes minimizing the energy consumption, while still fulfilling strict reliability demands, one of the major challenges of IIoT communications.

To achieve minimal energy consumption while maintaining high reliability, many research works have been conducted on MAC protocols featuring these requirements [90]. An important development was the IEEE 802.15.4 MAC layer and more specifically the IEEE 802.15.4e MAC amendment that proposed the TSCH mode [12,27]. As already discussed in Section 2.2, TSCH uses channel hopping to improve reliability while at the same time being energy-efficient, by using a time-synchronized schedule that tells a node exactly when to send and receive data and thus avoids wasting energy during contention periods and idle listening. The deterministic nature of TSCH scheduling allows for precise modeling of the energy consumption as has been done in the work by Vilajosana et al. [91]. Such a model allows for a detailed energy analysis of new TSCH scheduling functions or new protocols on top of the TSCH MAC layer (e.g., routing protocols), during simulated or real-world experiments.

In this chapter, we propose a novel, more accurate and up-to-date energy consumption model for the IEEE 802.15.4e TSCH mode. It consists of two main contributions. First is a new energy consumption model, based on the work by Vilajosana et al. [91], built from the ground up. It includes a more up-to-date and elaborate set of time slots and states, while using state-of-the-art IoT hardware and firmware. Additionally, the model is extended to support variable packet sizes, a feature absent in the previous work, that allows for a more accurate energy consumption analysis for all packet sizes. As a second contribution, new state durations and state energy consumption measurements are presented for both the sub-GHz (i.e., 868 MHz in Europe) and 2.4 GHz frequency bands, using state-of-the-art OpenMote hardware [77, 92]. Moreover, we experimentally verify the accuracy of our model by comparing the calculated values for both the sub-GHz and 2.4 GHz band to the measured values and compare the model to that of Vilajosana. Finally, the new measurements are used to analyze the end-to-end performance of a TSCH network using the official 6TiSCH simulator [83].

The remainder of this chapter is structured as follows. In Section 3.2, we give information about the used OpenMote hardware and OpenWSN firmware and the related work on energy modeling. Subsequently, Section 3.3 introduces the model itself. In Section 3.4, the measurement methodology is discussed and the measurement values are presented. Afterwards, the proposed model is evaluated in Section 3.5 by analyzing calculated values and measured consumption values and comparing the model to a state-of-the-art model. That section also shows the results of the TSCH network simulations to show the energy consumption effects of 868 MHz and 2.4 GHz communication. Finally, Section 3.6 presents the conclusions of our work.

3.2 Background and Related Work

While we already introduced the OpenMote hardware and OpenWSN firmware in Section 2.3.5, in this section we specify the exact version we used for this research. Afterwards, we compare the work in this chapter to existing energy consumption models.

3.2.1 OpenMote Hardware

The measurements presented in this chapter are performed using OpenMote, a modular open-hardware ecosystem designed for the industrial IoT [77]. The platform was developed at UC Berkeley and is designed to efficiently implement IoT standards such as 6TiSCH.

The OpenMote-CC2538 is the core of the OpenMote hardware ecosystem. It is the most important component, and other components (e.g., the OpenBattery) are considered to be extensions of it. It features a Texas Instruments CC2538

SoC that consists of a 32-MHz micro-controller with 32 kB of RAM and an IEEE 802.15.4-compliant 2.4 GHz radio.

The OpenUSB version used here has a CC1200 radio chip. Unlike the CC2538, which has a 2.4 GHz radio, the CC1200 is a radio transceiver that operates in the 900-MHz range, e.g., the 868 MHz band in Europe. This allows for longer-range communication between the motes. As OpenUSB only holds the CC1200 radio transceiver, it needs to be connected to the OpenMote-CC2538, which holds the microprocessor to control it.

3.2.2 OpenWSN

OpenWSN is an open-source project that implements the 6TiSCH architecture [72], as discussed in Chapter 2. The newest update of the OpenWSN firmware at the time of writing was used when rebuilding and extending the energy model and is made separately available [93]. The hierarchical design of the project makes it relatively easy to port the project to new hardware platforms. Hardware drivers for most common IoT hardware are already available as part of the OpenWSN project itself.

Next to the firmware, useful software such as the OpenVisualizer is also provided (and used in this research). Although the main use of the OpenVisualizer project is to connect the OpenWSN network to the Internet, it also provides the ability to monitor the network. The tool shows the internal state of all the motes that are physically connected to the computer running the OpenVisualizer, e.g., the neighbor table, scheduling table and packet queue. It also has the ability to run simulated motes and to debug the communication with Wireshark [94].

3.2.3 TSCH Energy Modeling

As minimizing the energy consumption is one of the major challenges of IoT networks, a lot of research has already been conducted on this topic. Some of the research already focused on TSCH energy modeling.

Some works target specific features in TSCH. De Guglielmo et al. proposed an analytical model of the IEEE 802.15.4e TSCH CSMA-CA algorithm that is used in shared time slots [95]. The authors also observed that the capture effect has a significant impact on the performance of the CSMA-CA algorithm. Papadopoulos et al. investigated the impact of the guard time in TSCH [96]. The authors decreased the guard time duration when motes were closer to their sink and concluded that this results in significant savings in energy consumption without compromising network reliability. While these works only aim at specific TSCH elements, the proposed work in this chapter provides an energy consumption model for the whole of the IEEE 802.15.4e TSCH mode. Other works such as Juc *et al.* compared the performance of the TSCH and Deterministic and Synchronous Multichannel Extension (DSME) modes of IEEE 802.15.4e [97]. The authors do not propose a model themselves. They observed that TSCH mode tends to consume more energy than DSME mode. This is due to the

large fixed guard time in TSCH and because DSME can aggregate multiple acknowledgments and transmit a single group of acknowledgments.

Finally, Vilajosana *et al.* presented an energy model for TSCH networks, using the OpenMote and OpenWSN for their experimental validation [91]. The values from the model were compared to measurements on the GINA and OpenMote-STM32 platforms. This chapter continues the work of Vilajosana *et al.*, but explores several differences and improvements. As such, we propose a model with an extra time slot type (i.e., `TxDataRxNoAck`), provide an extended and a more up-to-date set of states per time slot and extend the model to support variable packet sizes. Furthermore, the OpenWSN firmware has been continuously updated, and the current software version has changed substantially since the version used by Vilajosana *et al.* in 2013. Finally, by using the OpenMote-CC2538 and OpenUSB board, this chapter focuses on state-of-the-art hardware. This allows us to consider the TSCH energy consumption in both the sub-GHz and 2.4 GHz band. To the best of our knowledge, we are the first to do this. We also explicitly look at the difference in power consumption between using a SoC and the case with a separate micro-controller and radio chip. All the steps in developing the model are explained in detail, allowing it to be used for different types of hardware by simply changing the measured consumption values.

3.3 TSCH Energy Model

In this section, the proposed TSCH energy model is introduced. First, all types of time slots are discussed, followed by a more detailed examination of the states in these time slots. Afterwards, the time slot energy model is presented. Finally, we explain how the slot model could be adapted for use with different hardware. The implementation of the proposed model is publicly available[1].

3.3.1 TSCH Time Slots

A TSCH schedule can contain different types of time slots, i.e., cell types, to indicate that a node should transmit, listen or put its radio to sleep. In IEEE 802.15.4e, seven different types of time slots can be identified:

- `TxDataRxAck`: The mote sends a frame during this time slot and receives an `ACK` when the data have been received successfully.
- `TxData`: The mote sends a frame during this time slot, but does not expect an `ACK` (e.g., broadcast or multicast frames such as RPL DIO messages).
- `RxDataTxAck`: The mote listens and receives a frame in this time slot and replies with an `ACK` to indicate that it successfully received the frame.

[1] `https://github.com/imec-idlab/TSCH-energy-model`

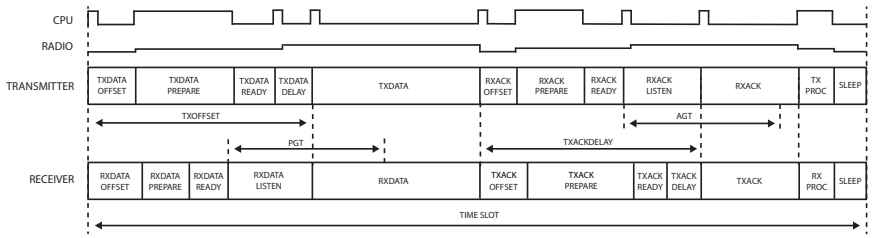


Figure 3.1: States in `TxDataRxAck` (transmitter) and `RxDataTxAck` (receiver) time slots, together with the CPU and radio activity in the `TxDataRxAck` time slot.

- `RxData`: The mote listens and receives a frame in this time slot, but no `ACK` is sent (e.g., broadcast or multi-cast frames).
- `RxIdle`: The mote listens, but does not receive a frame in this time slot.
- `Sleep`: The mote does not transmit or receive during this time slot.
- `TxDataRxNoAck`: The mote sends a frame and expects an `ACK`, but no `ACK` is received. This could be caused by propagation loss or a collision of the data frame.

The proposed model divides each time slot into different states. Figure 3.1 illustrates this and presents a general overview of the activity of a transmitter and receiver during a `TxDataRxAck` time slot and `RxDataTxAck` time slot, respectively. Some of the states seen in Figure 3.1 consist of two parts: one part where the CPU is active and one part where the CPU is sleeping. These two parts are considered as separate states in our model. The state of the radio in our model only changes at moments when the CPU state changes. This is a simplification as in the real world, the radio state changes slightly before or after this moment, typically while the CPU is active.

The remainder of this section explains the `TxDataRxAck` time slot in full detail. The other slots are modeled similarly, and we limit the discussion to highlighting the differences with the `TxDataRxAck` slot.

3.3.1.1 Time Slot `TxDataRxAck`

The different states of the `TxDataRxAck` time slot are shown in Figure 3.1, and Table 3.1 lists the exact CPU and radio states at each moment. As can be seen in the table, the CPU has two states, i.e., Sleep and Active, while the radio has five states, i.e., Sleep, Idle, Listen, Transmit (TX) and Receive (RX).

At the beginning of each time slot, the CPU wakes up and performs the tasks required for any slot. This includes incrementing the ASN and scheduling the next state depending on the type of the slot. The CPU then sleeps again during `TxDataOffset` until the moment the radio is needed.

Table 3.1: States in a TxDataRxAck slot.

State	CPU State	Radio State
TxDataOffsetStart	Active	Sleep
TxDataOffset	Sleep	Sleep
TxDataPrepare	Active	Idle
TxDataReady	Sleep	Idle
TxDataDelayStart	Active	Idle
TxDataDelay	Sleep	TX
TxDataStart	Active	TX
TxData	Sleep	TX
RxAckOffsetStart	Active	Sleep
RxAckOffset	Sleep	Sleep
RxAckPrepare	Active	Idle
RxAckReady	Sleep	Idle
RxAckListenStart	Active	Idle
RxAckListen	Sleep	Listen
RxAckStart	Active	RX
RxAck	Sleep	RX
TxProc	Active	Idle
Sleep	Sleep	Sleep

During TxDataPrepare, the radio wakes up, the channel is set and the bytes to transmit are loaded into the radio. The duration of this state is variable, mainly because the time necessary to load the bytes depends on the frame size. Since this state always starts at the same offset and has a variable duration, there is some time left between the TxDataPrepare and the actual transmission. During this TxDataReady state, the radio is in Idle mode, while waiting until it is time to transmit. To minimize the energy consumption of the mote, the duration of the TxDataReady state should thus be as short as possible.

The first byte behind the Start-of-Frame Delimiter (SFD) has to be transmitted exactly TxOffset ms after the start of the time slot. In order to do so, the time required to switch the radio from Idle to TX mode has to be taken into account. The duration of the TxDataDelay equals the time between the TX command being sent to the radio and the moment the SFD has been transmitted.

After the RxAckOffset that follows where the mote sleeps, the state RxAckPrepare then prepares the radio again by waking it up and setting the correct channel. Any time less than the maximum duration of RxAckPrepare is then spent in the RxAckReady state.

The ACK is transmitted TxAckDelay ms after the end of the TxData state. Because the clocks of the transmitting and receiving node may not be perfectly synchronized, the ACK might arrive slightly earlier or later than expected. The radio is thus turned on at the start of the RxAckListen instead of just in time

for the data. If no ACK is received during the Acknowledgment Guard Time (AGT) period, the mote turns off the radio and considers the transmission failed. The duration of the AGT is defined as 1000 µs in OpenWSN (i.e., the total length of the time slot is 15 ms). When the clocks between the motes are perfectly synchronized, the RxAckListen state has a duration of AGT/2 plus the time to change the radio from Idle mode to RX mode (which is considered to be instantaneous in OpenWSN).

During the TxProc state, the ACK is read from the radio and the transmission is considered successful when the ACK is valid. The mote also synchronizes its clock based on the offset between TxAckDelay and the actual data reception time, if the ACK came from its parent in the network routing graph. For the remaining part of the time slot, both the CPU and radio are in Sleep mode.

3.3.1.2 Time Slot RxDataTxAck

This time slot can be considered the opposite of the TxDataRxAck. The states to handle the data in TxDataRxAck are found in handling the ACK in RxDataTxAck and vice versa. All states of the RxDataTxAck time slot can be found in Table A.1 in Appendix A.

The guard time for the data is however larger than the AGT that is used for ACKs. The Packet Guard Time (PGT) determines how long the radio listens for the data before the radio is turned off. When no data are received during the PGT period, we classify the time slot as RxIdle instead of RxDataTxAck. In OpenWSN, the PGT is defined as 2600 µs.

3.3.1.3 Time Slot TxData and RxData

When no ACKs are required (e.g., for broadcasts), only the first half of the time slot is used. During the TxData and RxData slots, the mote sleeps once the data have been transmitted or received. The states for both TxData and RxData are shown in Tables A.2 and A.3 in Appendix A, respectively.

3.3.1.4 Time Slot RxIdle

When the transmitter has no data to send, the slot that could have been a TxDataRxAck becomes a Sleep slot. However, on the receiver side, a different type of slot is needed to represent the behavior of the node: the RxIdle slot occurs when the receiver expects data, but does not receive anything. The states of RxIdle are shown in Table A.4 in Appendix A. The behavior of RxIdle is not an error; it simply means that a slot was reserved, but the transmitter did not have any data to send at that moment.

3.3.1.5 Time Slot `Sleep`

In time slots where no data have to be transmitted or received, the node sleeps during the whole duration of the slot. The CPU of the node only briefly wakes up at the start of the slot, e.g., to increment the ASN. The states of the `Sleep` time slot are shown in Table A.5 in Appendix A.

3.3.1.6 Time Slot `TxDataRxNoAck`

There are many error states in OpenWSN. The code would go into an error state when, for example, the radio remains active too long or when the prepare state lasts longer than the maximum allowed duration. It is unlikely that the code would end up in most of these error states unless there is a configuration issue. However, there is one error state that is likely to occur eventually: a missing `ACK`. In the `TxDataRxAck` slot, data are transmitted and an `ACK` is received, but in the slot that we refer to as `TxDataRxNoAck`, the `ACK` is expected, but not received. In this case, the node stays in the `RxAckListen` state during the AGT period and does not enter the `RxAck` state. After the AGT period, the radio goes to sleep during the `TxProc` and `Sleep` state, as can be seen in Table A.6 in Appendix A.

3.3.2 TSCH Energy Consumption Model

Having identified all states per time slot, the model for the charge drawn during a time slot can be constructed. The resulting charge (in coulombs) drawn from the battery during a slot, Q_{Slot}, is represented by:

$$Q_{Slot} = \sum_{State \in Slot} \Delta t_{State} \times I_{State} \tag{3.1}$$

with Δt_{State} and I_{State} being the state duration and current drawn in each state, respectively. The unit of the duration is milliseconds (ms), while the unit of the current is milliamperes (mA), meaning that the unit of the resulting charge is microcoulombs (μC). This can be used to calculate the total charge drawn for each of the slot types discussed in Section 3.3.1.

Subsequently, the model previously proposed by Vilajosana et al. can be employed to calculate the total charge drawn across a slotframe [91]. This in turn can be used to compute the lifetime of a mote. That model, however, has one major shortcoming. It does not consider the actual packet size when calculating the charge drawn by a slot. Instead, it takes the consumed charge values for the maximum packet size and scales those linearly based on the actual packet size:

$$Q_{slot}^{N_{sent}} = \frac{N_{sent}}{max_{PktSize}} \times Q_{slot}^{max_{PktSize}} \tag{3.2}$$

With N_{sent} the number of bytes being sent in the packet and $max_{PktSize}$ the maximum packet size for which measurements were performed. However,

this leads to highly inaccurate estimates, especially for small packet sizes, as the duration of most states with the slot is independent of the packet size. In contrast, we propose a more accurate estimation of the charge drawn in a slot $Q_{slot}^{N_{sent}}$, based on actual measurements with different packet sizes. This is achieved by expressing the duration of each state that depends on the packet size, as a linear function of the packet size, rather than a fixed value for the maximum packet size. This is further explained in Section 3.4.

3.3.3 Different Hardware Support

Since the model has an elaborate set of parameters, adapting the model to different hardware while maintaining an equal level of accuracy is a burdensome task. However, at the cost of a slight decrease in accuracy, the model can easily be simplified in order to apply it to different hardware. For example, one can set the duration of short states to zero (e.g., `TxDataDelayStart` and `RxAckOffsetStart`) and only update the states that have the most impact on consumption. Alternatively, the duration can be estimated instead of measured as most durations will be very similar to the ones presented in this chapter. Furthermore, the consumption of the CPU and radio does not have to be measured: these values can be found in the data sheet of the manufacturer. The resulting model will be slightly less accurate, but no or only a few additional measurements have to be made to use this model to estimate the charge drawn by other TSCH hardware.

3.4 Measurements

This section first presents the setup used to measure the duration and energy consumption of each state of each slot type. Afterwards, the measurements of the time slot state durations are discussed together with how the duration values are affected by the packet size. Finally, the consumption of each device state is presented with a detailed discussion for each of the two evaluated radios.

3.4.1 Methodology

In this section, the necessary adaptations to the OpenWSN firmware, that allowed performing the measurements, are briefly explained. Additionally, the two measurement setups for both the state duration and energy consumption measurements are discussed.

3.4.1.1 Firmware Changes

To perform valid measurements, the firmware code that toggles debug pins and LEDs on the OpenUSB board was disabled. Furthermore, the serial communication code was also completely disabled because even when the OpenUSB is not

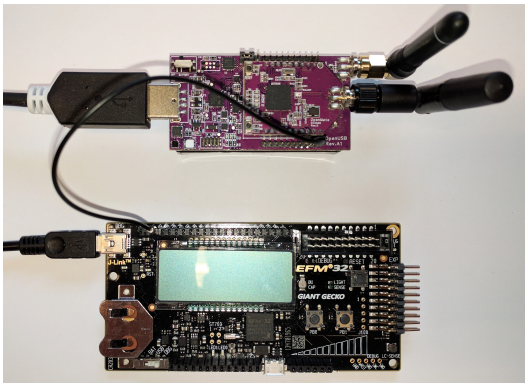


Figure 3.2: Setup used to measure state durations with a connection from the PB9 pin on the Gecko board (bottom) and to the PD2 pin on the OpenUSB (top).

connected to a computer, the code would still try to output data, unnecessarily increasing power consumption.

In order to prepare the 2.4 GHz driver for the measurements, only small adaptations had to be made to the original firmware[2]. The firmware for the 868 MHz driver however required additional implementation effort, as there were no working drivers for the CC1200 radio chip on the OpenUSB at the time of writing. Based on a branch of the official OpenWSN repository[3], we implemented a working CC1200 radio driver, which is publicly available[4]. The data rate of both the 2.4 GHz and the 868 GHz drivers was set to 250 kbps.

3.4.1.2 State Duration Measurements

All state duration measurements were done using the EFM32GG-STK3700 Giant Gecko Starter Kit from Silicon Labs [98]. The setup is shown in Figure 3.2. Using the Gecko board, an OpenUSB pin was connected to pin PB9 of the Gecko board, enabling the Gecko to measure how long the connected OpenUSB pin was made low. The OpenMote firmware would then make the connected pin low at the beginning of the measurement and high at the end of the measurement. The output was sent over Serial Wire Output (SWO) to the console in the proprietary software Simplicity Studio on the connected computer, where post-processing of the duration data was applied [99]. The duration measurements were averaged in case variability between different measurements was noticed.

[2]The 2.4 GHz measurements were performed using the repository at this commit: `https://github.com/openwsn-berkeley/openwsn-fw/commit/5a29808`.

[3]`https://github.com/openwsn-berkeley/openwsn-fw/tree/develop_FW-493`

[4]`https://github.com/imec-idlab/openwsn-fw`

3.4.1.3 Energy Consumption Measurements

In order to perform the different energy consumption measurements, a setup different from the Gecko setup, described in Section 3.4.1.2, was needed to be used. As the consumption of the OpenMote hardware happened to exceed 50 mA (i.e., the maximum of the Gecko measuring range), we switched to using the Keysight N6705B DC Power Analyzer [100]. Using the two-wire mode, the Voltage Common Collector (VCC) and Ground (GND) pins of the OpenMote-CC2538, in the 2.4 GHz measurement setup, and of the OpenUSB with the OpenMote-CC2538 attached, in the 868 MHz measurement setup, were connected to the power supply output of the N6705B, which was configured to provide an input voltage of 3.0 V. This is the nominal voltage of two serially-connected AA batteries, which can be used to power an OpenMote via an OpenUSB or OpenBattery module. The measurement setups for the 2.4 GHz and 868 MHz measurements are shown in Figure 3.3. For the 868 MHz measurements, the OpenUSB has to be attached to the OpenMote-CC2538 because the former only holds a CC1200 radio transceiver and needs the microprocessor on the latter to control it.

For most device states, the consumption was averaged over a period of 500 ms. However, some states (e.g., RX and TX states) only last as long as the radio takes to send all bytes. For these states, the average was taken over a period between 3 ms and 4 ms.

3.4.2 Time Slot State Durations

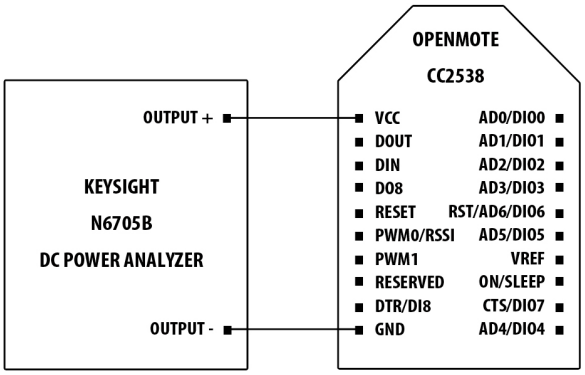
We measured the duration of each state in every time slot where the CPU is active. The durations in which the CPU is sleeping can then be trivially calculated, using the active durations and the timing constants found in OpenWSN firmware. The total length of a time slot was set to 15 ms. The state durations for all time slots are shown in Table 3.2 and Tables A.7–A.12 (in Appendix A).

States do not always have the exact same duration for a variety of reasons. There can be multiple code branches (i.e., different execution paths); the packet size can vary and have an influence; or the duration of an operation can simply be variable (e.g., waking up the CC1200 chip). Therefore, multiple measurements were executed to find a single duration that could be associated with the state.

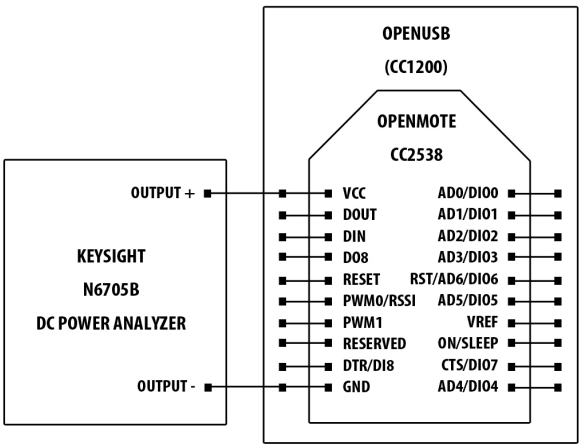
Changing the mode of the CC1200 radio from Sleep to Idle takes between 246 µs and 343 µs, which causes every state where the radio wakes up to have a variable duration. To avoid being susceptible to outliers, the wakeup time was measured over ten thousand times, and an average duration of 273 µs was observed.

For states with multiple code branches, the median value of multiple measurements was chosen. For example, in state `TxProc`, the execution path is different when a data packet has no retries left. However, such small variations of the duration only have a limited impact on the total slot consumption.

The durations of states where packets are loaded to and read from the radio were measured before and after the radio was accessed. Afterwards, the



(a) 2.4 GHz measurement setup.



(b) 868 MHz measurement setup.

Figure 3.3: Energy consumption measurement setups: for the 2.4 GHz measurements, only the OpenMote-CC2538 was used, while for the 868 GHz measurements, the power analyzer was connected to the OpenUSB to which the OpenMote-CC2538 was attached.

communication with the radio for different packet sizes (from 0 bytes–125 bytes with steps of 25 bytes) was measured. Linear interpolation was applied on the measured durations to come up with a formula that fits well to all packet sizes. The difference in durations in states where data are transferred between the radio and CPU (e.g., `TxDataPrepare` and `TxProc`) is caused by the way these bytes are transferred: the CC2538 radio is combined with the CPU in one chip, so data can just be copied in/out of memory while the CC1200 needs to use the slower Serial Peripheral Interface (SPI) to transfer data to and from the CPU in the CC2538 chip, resulting in longer durations. The duration of transmitting and receiving also depends on the packet size. Since the radio

Table 3.2: State durations in the TxDataRxAck time slot with a total length of 15 ms and s being the packet size in bytes.

State	Duration (µs)	
	CC2538	**CC1200**
TxDataOffsetStart	105	105
TxDataOffset	1515	1454
TxDataPrepare	$60+(s\times 0.875)$	$738+(s\times 8.152)$
TxDataReady	$1954-(s\times 0.875)$	$1276-(s\times 8.152)$
TxDataDelayStart	17	58
TxDataDelay	349	369
TxDataStart	16	16
TxData	$(3+s)\times 32-16$	$(3+s)\times 32-16$
RxAckOffsetStart	32	75
RxAckOffset	3769	3116
RxAckPrepare	38	587
RxAckReady	267	328
RxAckListenStart	17	58
RxAckListen	483	442
RxAckStart	16	15
RxAck	880	881
TxProc	225	619
Sleep	$5177-(s\times 32)$	$4783-(s\times 32)$

has a baud rate of 250 kbps, the time it takes to transmit one bit is 4 µs, which makes the time to transmit a byte 32 µs. To calculate the duration, the amount of transmitted bytes has to be multiplied with 32 µs. The PHY header byte and two-byte Cyclic Redundancy Check (CRC) also have to be included as they are sent with the packet. To verify that this calculation is valid, the time between the start-of-frame interrupt and the end-of-frame interrupt was measured: the average error was only 0.13%.

To model the guard time, we assumed that the clocks are synchronized. Our model thus assumes that the packet always arrives exactly in the center of the guard interval.

3.4.3 Device State Current Consumption

The consumption of the OpenMote-CC2538 connected to the OpenUSB was measured during all possible device states. Since the CPU and radio are the two components responsible for the majority of the current consumption, these device states are all combinations between CPU and radio modes. Instead of measuring the consumption of the CPU and radio separately, we measured the consumption of the entire device. The result is that any current consumption

Table 3.3: Current drawn during different device states.

CPU State	Radio State	Consumption (mA)	
		CC2538	**CC1200**
Active	Sleep	13.97	15.06
Active	Idle	13.97	17.49
Active	Listen	31.14	40.13
Active	RX	26.94	50.63
Active	TX	31.47	54.26
Sleep (PM_NOACTION)	Sleep	10.06	11.42
Sleep (PM_NOACTION)	Idle	10.06	13.82
Sleep (PM_NOACTION)	Listen	27.18	36.18
Sleep (PM_NOACTION)	RX	23.16	46.73
Sleep (PM_NOACTION)	TX	27.55	50.24
Sleep (PM2)	Sleep	0.00156	0.27
Sleep (PM2)	Idle	0.00156	2.64

not related to the CPU or radio (e.g., SPI or timers) are measured as part of the CPU usage. This allows for a slightly more accurate prediction of the charge drawn compared to models that ignore these other components.

3.4.3.1 2.4 GHz CC2538 Radio

In Table 3.3, the consumption values of the different device states when using the CC2538 radio, i.e., the 2.4 GHz radio, are shown. The values for the TX state were measured when the transmit power of the radio was set to 0 dBm. When the transmit power was set to 3 dBm, i.e., the current default in OpenWSN, the consumption values of the TX states are 33.04 mA and 29.01 mA, for the CPU in Active and Sleep state, respectively.

The CC2538 radio has an identical consumption of 13.97 mA when the radio is in Sleep or Idle state because the OpenMote-CC2538 consists of both the CPU and radio, and the radio itself does not have a separate Idle or Sleep state. Instead, it has a single Off state for which the consumption was used for both the Sleep and Idle states. Thus, the CC2538 radio has only four states: TX, RX, Listen and Off.

As expected, the difference in the consumption between an active or a sleeping CPU is nearly identical for all radio states: the CPU in active mode consumes on average 3.92 mA more than when being in sleep mode, with a standard deviation of only 0.07 mA.

When switching the CPU of the CC2538 chip to the deeper sleep mode PM2 instead of PM_NOACTION[5], while the radio was in the Sleep and Idle state

[5] http://www.ti.com/product/CC2538/datasheet/

(which are actually both the Off state in the CC2538 radio), the consumption dropped to 1.56 µA.

3.4.3.2 868 MHz **CC1200 Radio**

The device state consumption values when using the CC1200 radio, i.e., 868 MHz, are shown in Table 3.3. The values for the TX state were measured when the transmit power of the radio was set to 0 dBm. When the transmit power is set to 14 dBm, i.e., the current default in OpenWSN, the consumption of the TX states is 91.94 mA for an active CPU and 88.25 mA for a sleeping CPU. The CC1200 radio Sleep state is the Idle state with the crystal oscillator turned off[6]. Consumption is expected to be lower when the Sleep state of the CC1200 chip is used or when the CC1200 is turned completely off.

As expected, the difference in the consumption between an active or a sleeping CPU is nearly identical for all radio states: the CPU in active mode consumes on average 3.81 mA more than when being in sleep mode, with a standard deviation of only 0.15 mA.

When both the CPU and the radio are put in the Sleep state, the consumption is still high. This is caused by the high current consumption of the CPU, which is put in the least possible sleep mode `PM_NOACTION`. When putting the CPU in a deeper sleep, i.e., the `PM2` power mode, while the CC1200 radio is in Sleep and Idle state, the consumption dropped to 0.27 mA and 2.64 mA, respectively.

3.5 Evaluation

In this section, the accuracy of the model is verified. First, the charge drawn per slot type for both radios is calculated and compared to the measured values. Afterwards, the accuracy of the charge drawn during a slotframe is validated using a small-scale test network. The developed packet size-aware model is also compared to the state-of-the-art model of Vilajosana et al. to show the accuracy improvement when including the packet size in the model. Finally, using the measured charge consumption values for both frequency bands' communication, several TSCH network simulations were conducted to observe the energy consumption effects in an end-to-end context.

3.5.1 Slot Charge Consumption

Using the duration and consumption of each state, the charge drawn during each type of slot is calculated using the formula shown in Equation (3.1). To verify the accuracy of our model, the entire consumption of each type of slot was also measured separately. Table 3.4 compares the measured and calculated values for both types of radio. Both radios are configured with a transmit power

[6] `http://www.ti.com/product/CC1200/datasheet/`

Table 3.4: Measured and calculated charge drawn for each slot type.

Slot Type	Measured (μC)		Calculated (μC)	
	CC2538	CC1200	CC2538	CC1200
TxDataRxAck	250.35	420.01	250.94	407.81
RxDataTxAck	253.2	432.09	251.32	417.2
TxData	229.8	360.2	230.13	357.12
RxData	235.1	373.55	228.72	362.12
RxIdle	197.4	245.2	196.35	240.98
Sleep	152.4	168.65	151.12	171.51
TxDataRxNoAck	246.95	395.65	246.79	384.94

of 0 dBm and a packet size of 127 bytes, i.e., the maximum packet size when including the CRC bytes.

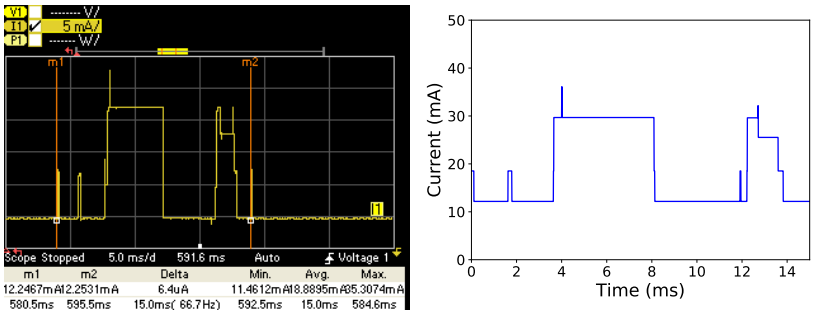
As seen in Table 3.4, the difference between the measured and calculated values is close to negligible. Among the main contributors to these differences are measurement errors and the variations in guard time duration. In the measured data, the guard time can be smaller or larger than in the calculated data, which assumes perfectly synchronized clocks. On average, the difference is limited to 5.08 μC or 1.55% with a standard deviation of 3.3 μC and a maximum difference of 14.89 μC, which proves the accuracy of our model.

More specifically, for the CC2538, the average relative difference is 0.75%, while for the CC1200 chip, the average relative difference is 2.3%. In the case of the CC1200 chip, the larger relative difference is explained by the fact that a specific device state (e.g., CPU is active and radio is sleeping) does not always result in exactly the same current drawn, which we abstracted in Table 3.3.

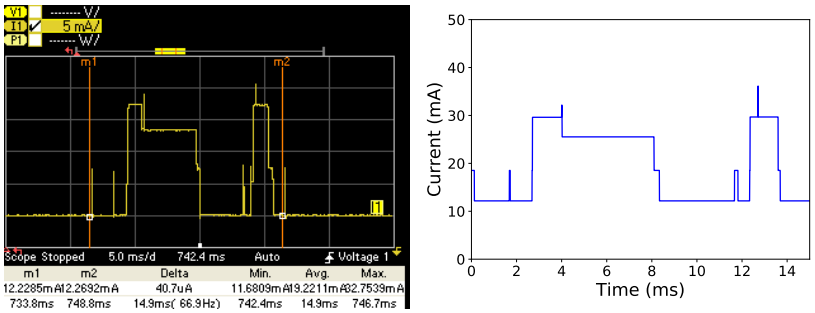
Figures 3.4 and 3.5 show the current drawn over time according to both the model and the measurements, for the `TxDataRxAck` and `RxDataTxAck` time slots, when using the CC2538 and CC1200 radio respectively. Figures A.1 and A.2 (in Appendix A) show the current drawn for the remaining time slots (except for the error time slot `TxDataRxNoAck`) for both radios. For all time slots, the measured graphs and their modeled counterpart look very similar. The peaks on the graphs however do not perfectly match, because the model simplifies certain states. The radio state may be changed while the CPU is active, causing the CPU and radio to be active at the same time, while the model might only consider the radio as active once the CPU goes to sleep. This results in a peak in the measured time slot where there is no peak in the model.

3.5.2 Slotframe Charge Consumption

When considering the charge consumption in the different time slots, the charge consumption of a slotframe can be calculated. To further verify the accuracy of our model, the calculated slotframe charge consumption of a small-scale,



(a) `TxDataRxAck` time slot.



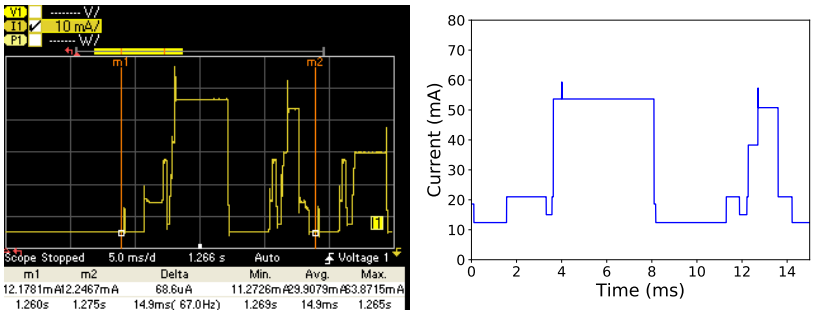
(b) `RxDataTxAck` time slot.

Figure 3.4: Measured (left, between vertical lines m1 and m2) and modeled (right) current comparison for each time slot when using the CC2538 radio.

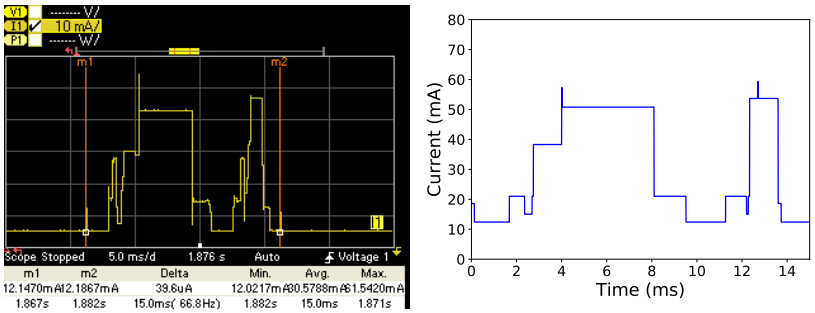
real-world 6TiSCH network is compared with the measured values, for both 868 MHz and 2.4 GHz.

The experiment network topology, depicted in Figure 3.6, used the OpenMote-CC2538 and the OpenMote-CC2538/OpenUSB board combination as hardware nodes for 2.4 GHz and the 868 MHz measurements, respectively. The root node is connected to a computer using OpenVisualizer to monitor the network. The leaf mote was configured to send a packet of 127 bytes (including CRC) every two seconds. The slotframe size was 51 time slots. Since there are 51 slots in a slotframe and every time slot lasts 15 ms, the duration of each slotframe is 765 ms. The first time slot in every slotframe was reserved for management messages, e.g., EBs, RPL DIOs, RPL DAOs and 6top messages, but these were not considered. As such, the first time slot in each slotframe is thus considered to be of type `RxIdle`.

The slotframe of the leaf mote always consists of one `RxIdle` slot and at least 49 `Sleep` slots. The last slot will either be of the type `TxDataRxAck` when there are data to send or another `Sleep` slot when there are no data. As such, two slotframe types were considered for the leaf node: a slotframe where no data were sent and a slotframe where the packet was sent in a `TxDataRxAck`



(a) `TxDataRxAck` time slot.



(b) `RxDataTxAck` time slot.

Figure 3.5: Measured (left, between vertical lines m1 and m2) and modeled (right) current comparison for each time slot when using the CC1200 radio.

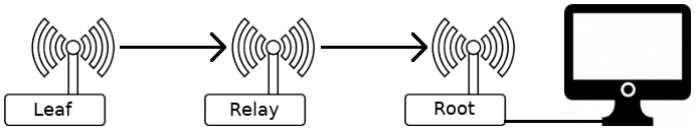


Figure 3.6: Topology used while comparing the charge consumption during a slotframe.

slot. The charges consumed in each of these two slotframe types are represented by:

$$Q_{leaf\_Sleep} = Q_{RxIdle} + 50 \times Q_{Sleep} \tag{3.3}$$

and:

$$Q_{leaf\_TxDataRxAck} = Q_{RxIdle} + Q_{TxDataRxAck} + 49 \times Q_{Sleep} \tag{3.4}$$

For the relay node, a slotframe was considered where the packet coming from the leaf was received in the first slot of the slotframe; subsequently, the relay forwarded the packet to the root, but no acknowledgment was received, followed by a successful retransmission. As such, there are `RxDataTxAck`,

Table 3.5: Measured and calculated charge drawn during a slotframe.

Mote Type	Measured (μC)		Calculated (μC)	
	CC2538	**CC1200**	**CC2538**	**CC1200**
Leaf (`Sleep`)	7833.6	8698.05	7752.35	8816.48
Leaf (`TxDataRxAck`)	7910.1	8942.85	7852.17	9052.78
Relay	8086.05	9348.3	8002.81	9442.96

`TxDataRxNoAck` and `TxDataRxAck` slots when a packet was received and forwarded, while the remaining 48 slots are `Sleep` slots. The charge drawn during the slotframe of the relay node is represented by the following formula:

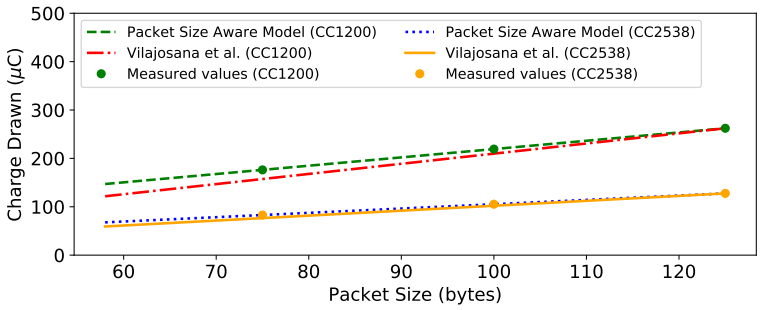
$$Q_{relay} = Q_{RxDataTxAck} + Q_{TxDataRxNoAck} + Q_{TxDataRxAck} + 48 \times Q_{Sleep} \quad (3.5)$$

The charge consumption of the root node is not considered as the root device is typically connected to a computer using OpenVisualizer, serving as a gateway to the Internet. Therefore, the root typically does not run on batteries. Additionally, the serial communication between the root and OpenVisualizer cannot be disabled, making the comparison between the measured consumption and the proposed model invalid.

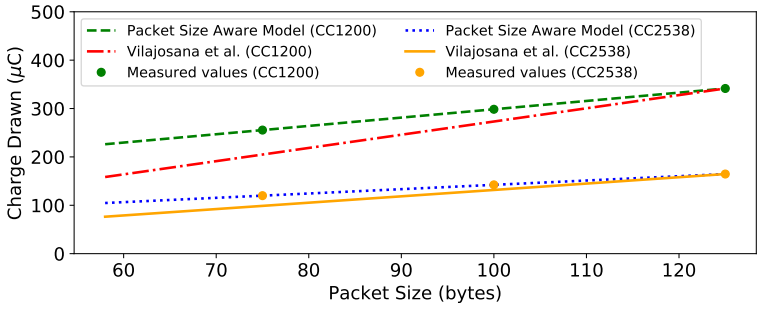
We measured the charge consumed over the length of an entire slotframe for both the leaf and relay node and compared these values to the values calculated using the proposed model and Equations (3.3)–(3.5). Table 3.5 shows the results. On average, the error between the calculated and measured values is lower than 1%. The differences between the CC2538 measured and calculated consumptions, for the leaf and relay nodes, are limited to 1.3% and 1.03%, respectively. For the CC1200 radio, the differences are even slightly smaller: respectively 0.88% and 1.01%. The consumption comparison results again show that our model is accurate, even when measuring across an entire slot frame.

3.5.3 Energy Model Comparison

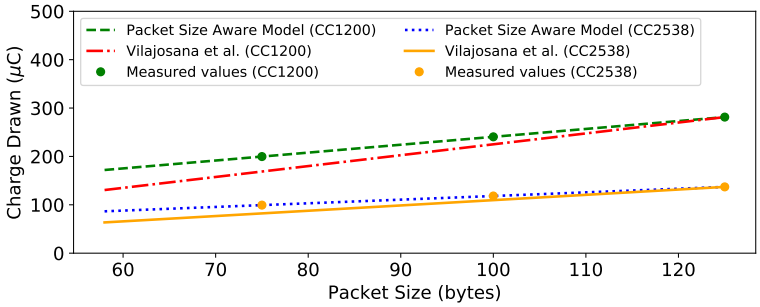
In order to indicate the accuracy gain of using the proposed packet size-aware model over the model introduced by Vilajosana et al., these two models are compared in Figures 3.7 and 3.8. The two models are compared for packet sizes going from 58 bytes, which is the minimum 6TiSCH packet size without additional payload, up to the maximum packet size, i.e., 127 bytes (125 bytes and 2 CRC bytes). The consumption of both models is compared to the packet size-aware model using the exact duration measurements for the packet sizes of 75, 100 and 125 bytes. The device state current values of Table 3.3 are used for this comparison. In the states where the CPU is sleeping and the radio is sleeping or in Idle mode, the `PM2` values were preferred over the `PM_NOACTION` values.



(a) `TxData` time slot.

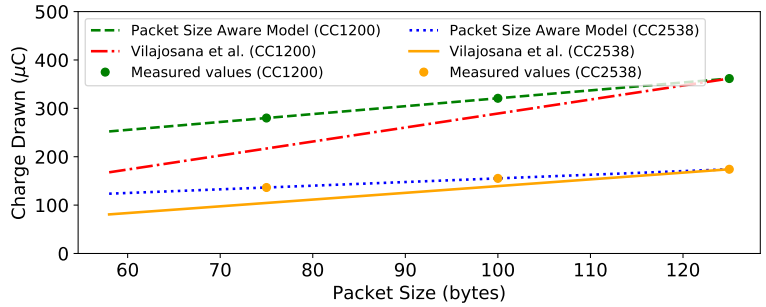


(b) `TxDataRxAck` time slot.

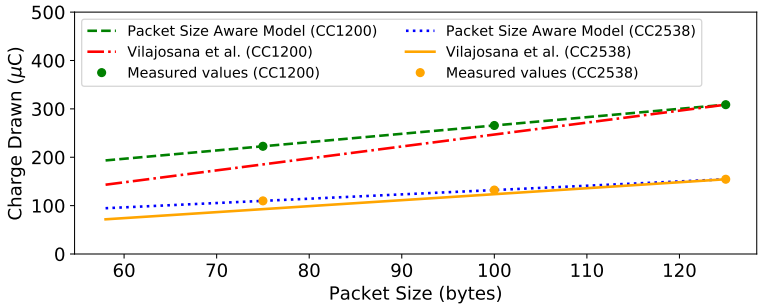


(c) `RxData` time slot.

Figure 3.7: The proposed packet size aware model compared with the Vilajosana et al. model which linearly scales the charge drawn based on the packet size, for the `TxData`, `TxDataRxAck` and `RxData` time slots.



(a) `RxDataTxAck` time slot.



(b) `TxDataRxNoAck` time slot.

Figure 3.8: The proposed packet size aware model compared with the Vilajosana et al. model which linearly scales the charge drawn based on the packet size, for the `RxDataTxAck` and `TxDataRxNoAck` time slots.

As can be seen in the graphs, the proposed model accurately represents the charge consumption for all packet sizes. The model of Vilajosana et al., however, becomes highly inaccurate especially when the packet size decreases. Looking at a packet size of 75 bytes, the average relative errors of the Vilajosana et al. model compared with the packet size-aware model using the exact duration measurements are 17.01 % and a standard deviation of 4 % (i.e., on average 40.18 µC with a standard deviation of 15.34 µC) for the CC1200 radio and 16.31 % and a standard deviation of 5.07 % (i.e., on average 18.74 µC with a standard deviation of 8.23 µC) for the CC2538 radio. Of course, for the maximum packet size, both models estimate the consumption correctly.

The reason for this large inaccuracy introduced by the model of Vilajosana et al. is that their approach linearly scales the entire slot consumption. This is not the correct approach, as only the states in which data are transmitted over the radio or copied between the radio and the CPU can be scaled. Since a time slot consists of many more states than only data processing states, those states should not be scaled. Because the proposed model differentiates between the state durations that depend on the packet size and the durations that are independent

Table 3.6: Parameter configuration in the 6TiSCH simulator.

Parameter	**868 MHz**	**2.4 GHz**
Time slot duration	15 ms	
Slotframe size	101 time slots	
No. of SHARED cells	1	
Inter-node distance	70 m	
No. of stable neighbors	1	
RPL parent set size	1	
Traffic period	5 s	
Traffic period variability	0.05	
EB period	10 s	
EB probability	0.15 s	
DIO period	30 s	
DIO probability	0.15	
OTF threshold	2	
OTF housekeeping period	10 s	
6top housekeeping	False	
TX power	0 dBm	
No. of channels	1	16
Stable RSSI	−83 dBm	−78 dBm

of the size, as can be seen in Table 3.2 and Tables A.7–A.12, it accurately models the slot consumption for different packet sizes.

3.5.4 Frequency Band Consumption Comparison

Using the measured energy consumption values for both 868 MHz and 2.4 GHz, we conducted several TSCH network simulations to analyze the end-to-end network performance and energy consumption at these frequency bands.

3.5.4.1 Simulation Setup

To perform the experiments, the 6TiSCH simulator, which was discussed in Chapter 2, was used. The configuration parameters for the simulation experiments discussed in this chapter, are listed in Table 3.6. To be able to compare the energy consumption for both 868 MHz and 2.4 GHz, we changed the default propagation model of the simulator (i.e., the so-called Pister-hack) to the International Telecommunication Union - Radiocommunications sector (ITU-R) Rural Macro model, which is applicable to both frequency bands [101]. To have a realistic low-power energy consumption comparison between 868 MHz and 2.4 GHz, we re-calculated the charge consumption values of Table 3.4, using the device state consumption values of Table 3.3 and adjusted them to make

Table 3.7: Calculated charge drawn for each slot type, used in the simulator experiments.

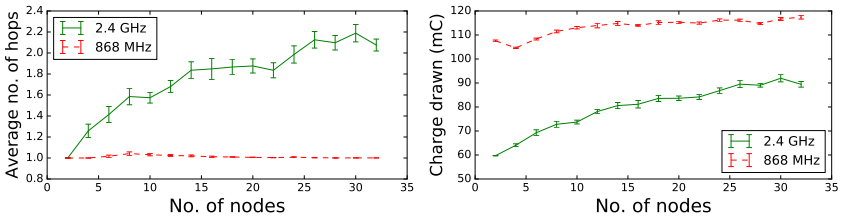
Slot Type	Calculated (μC)	
	CC2538	CC1200
TxDataRxAck	106.45	275.61
RxDataTxAck	107.66	286.76
TxData	83.07	210.32
RxData	82.97	219.8
RxIdle	47.54	81.57
Sleep	0.82	0.89
TxDataRxNoAck	100.32	246.98

sure the measured CC2538 PM2 power mode (instead of the PM_NOACTION mode) and the CC1200 power down Sleep state (instead of the Idle state with the crystal oscillator turned off) were used. Both states are expected to be used when running OpenWSN in an energy-efficient manner. However a software timer issue prevented OpenWSN to use the PM2 power mode[7]. Additionally, setting the CC1200 state to the power down Sleep state in our CC1200 driver implementation actually spiked the consumption instead of decreasing it and could therefore not be used in the measurements of Table 3.3. Therefore, the measured CC2538 PM2 power mode consumption value of 1.56 μA was used to replace the PM_NOACTION power mode values in all states where the CPU was sleeping. This was done by looking at the difference between the current consumption of the PM_NOACTION and PM2 mode when the CPU and the CC2538 radio were sleeping, and applying this difference in all other states when the CPU was sleeping. When the CPU and CC1200 radio were sleeping, the Idle state (with the crystal oscillator turned off) consumption value of the CC1200 chip, i.e., the radio Sleep state, was also replaced by the power down Sleep state consumption value of the CC1200 datasheet, which is 0.5 μA[8]. The resulting slot consumption values are listed in Table 3.7. The 6TiSCH simulator implementation used for these simulation experiments is publicly available[9].

3.5.4.2 Simulation Results

In the first TSCH network experiment, the number of nodes in a random topology varies from 2 to 32 nodes. Figure 3.9 shows the average hop count and the total charge drawn per node over a period of 300 s. As seen in Figure 3.9a, for 2.4 GHz, the average hop count increases as the number of nodes in the network increases. For 868 MHz, the hop count stabilizes to one. Communication at 868 MHz is stable over longer ranges than communication at 2.4 GHz, resulting

[7] https://openwsn.atlassian.net/browse/FW-361
[8] http://www.ti.com/lit/ds/symlink/cc1200.pdf
[9] https://github.com/imec-idlab/6tisch-sim-src/tree/energy



(a) Average hop count.

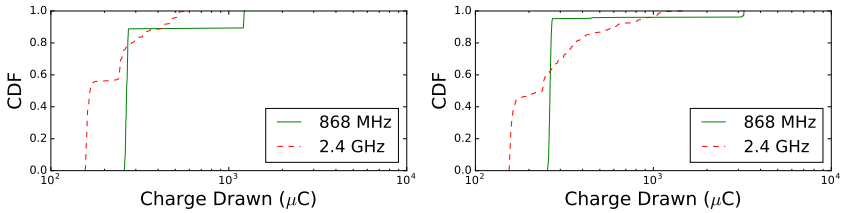
(b) Total charge drawn per node.

Figure 3.9: Total charge drawn per node and average hop count for 868 MHz and 2.4 GHz frequency communication in a random topology as a function of the number of nodes.

in a lower hop count to reach the root node. For the shorter range 2.4 GHz communication, the increase in consumption is significant when the average hop count increases. Since there are more nodes that have to relay additional packets towards the root, the total consumption per node increases. Figure 3.9b also clearly shows that the total consumption per node is higher for 868 MHz than for 2.4 GHz as is explained by the absolute slot consumption values in Table 3.7.

In the second TSCH network experiment, the average charge drawn per node per slotframe was observed over a period of 300 s. The results are shown in Figure 3.10 which shows the Cumulative Distribution Function (CDF) for 2 grid topologies with 9 and 25 nodes, respectively. In the network of 9 nodes, all 8 nodes directly connect to the root node for both 868 MHz and 2.4 GHz. The CDF in Figure 3.10a shows that almost all of the nodes consume less charge when using 2.4 GHz compared to when using 868 MHz. The difference in consumption is explained by the measured slot consumption values shown in Table 3.7, which indicates that 2.4 GHz consumes less energy than 868 MHz. However, when looking at nodes between 0.8 and 0.9, we observe that 868 MHz consumes less. The same effect is observed in the results for 25 nodes: 60% of the nodes that use 2.4 GHz consume less than the nodes using 868 MHz. These nodes represent leaf nodes and nodes that do not have to forward many data packets originating from children in the routing graph. Apart from these nodes, there are also other intermediate nodes, as indicated by the 2.4 GHz hop count of 2.49 ($\sigma = 1.21$) for the grid scenario with 25 nodes, which have to relay many more packets towards the root and consume more energy. When using 868 MHz, the average hop count was 1.01 ($\sigma = 0.1$), which means that all nodes are directly connected to the root and thus do not relay other packets. For both the grid networks of 9 and 25 nodes, the root nodes for 2.4 GHz consume less than those of 868 GHz, which again can be expected by looking at the consumption values in Table 3.7.

Looking at the absolute energy consumption values for both 868 MHz and 2.4 GHz, an increased energy consumption for all 868 MHz communication is



(a) Grid topology of 9 nodes.

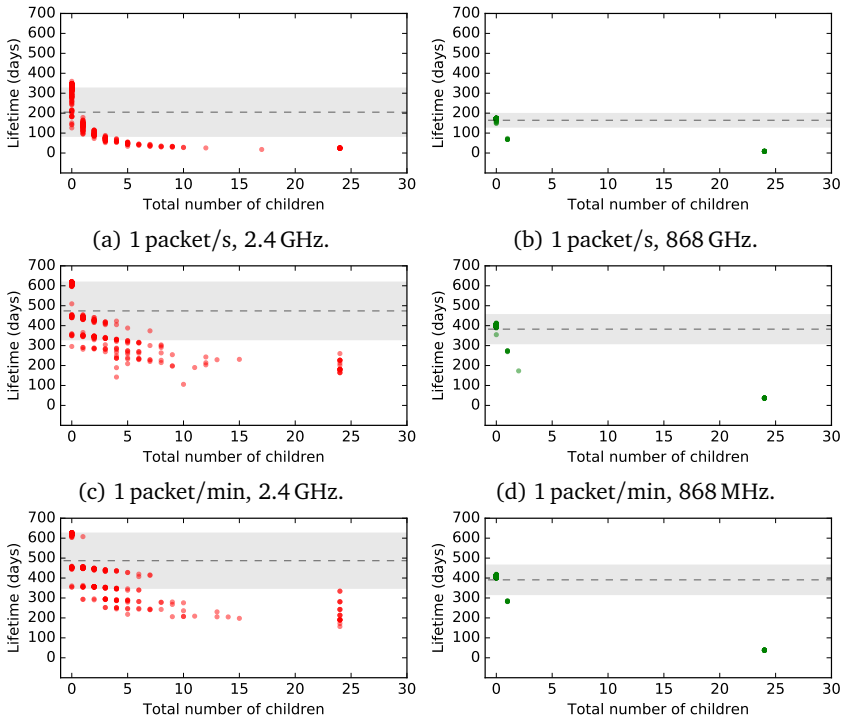
(b) Grid topology of 25 nodes.

Figure 3.10: Comparison of charge drawn per cycle per node for 868 MHz and 2.4 GHz frequency communication in a grid topology of 9 nodes and 25 nodes.

expected. However, these simulation results show that due to the longer-range capabilities of 868 MHz communication, there can be nodes that consume less energy compared to when using 2.4 GHz communication.

In the third TSCH simulator experiment, we observe the lifetime of all TSCH nodes in a grid of 25 nodes for different packet periods. Each node is assumed to be running on two AA batteries, i.e., a battery capacity of 2000 mA h. Figure 3.11 shows the results. The total number of children are all children of a node, e.g., the root node will have 24 children. It is clear that in the case of 2.4 GHz communication, there is much more variability in the number of children a node has, compared to when using 868 MHz communication. This is due to the longer range communication of 868 MHz that allows nodes to directly connect to the root over longer distances. In this 25-node grid topology, however, it is still possible that a 868 MHz leaf node needs multiple hops to reach the root: as observed in Figure 3.11, there are some nodes that have one, two or three children, which indicates that the signal of those children to their parent was better compared to the signal of their link to the root. With 2.4 GHz communication that lacks such longer range capability, a packet typically has to traverse more hops to reach the root. For 2.4 GHz, there is also more variability in the lifetime of nodes with the same amount of children. For 868 MHz, we do not observe this effect. This is because the quality of the different links between the 2.4 GHz nodes differs in every experiment, resulting in a variable number of transmission cells and retransmissions that are necessary to deliver packets, which in turn also influences the energy consumption. Most 868 MHz nodes however are directly connected to the root with good link quality, resulting in almost no variability.

The results show that for a higher packet frequency, the average number of days a node lasts decreases, e.g., the average lifetime for 1 packet/s is 204 days compared to 487 days when having a frequency of 1 packet/h, for 2.4 GHz. The graph also shows that on average, the lifetime in a 868 MHz network is lower, because of the higher consumption values shown in Table 3.7. However, the results in Figure 3.10 showed that this does not necessarily hold for all nodes in a TSCH network.



(a) 1 packet/s, 2.4 GHz.

(b) 1 packet/s, 868 GHz.

(c) 1 packet/min, 2.4 GHz.

(d) 1 packet/min, 868 MHz.

(e) 1 packet/h, 2.4 GHz.

(f) 1 packet/h, 868 MHz.

Figure 3.11: Comparison of the lifetime of a TSCH node, running on two AA batteries, between 2.4 GHz and 868 MHz communication for different packet periods in a grid topology of 25 nodes.

3.6 Conclusion

In this chapter, we propose a more accurate energy model for IEEE 802.15.4e TSCH using dual-band OpenMote hardware. The model differs from previous work in several ways. First, it includes an elaborate and up-to-date set of time slots and states and accurately models variable packet sizes. Second, we present state durations and energy consumption measurements for both the 868 MHz and 2.4 GHz frequency bands, using the CC1200 and CC2538 radio, respectively. We have experimentally verified the accuracy of the proposed model by comparing measured values of all time slots to their modeled counterpart. Furthermore, the energy consumption of a small-scale TSCH network was compared with its modeled consumption. For both the time slot comparison and the small-scale network experiment, the average error was less than 3%, including measurement inaccuracies and variations of the guard time. Using the measured energy slot consumption for both 868 MHz and 2.4 GHz communication, we also conducted

several TSCH network simulations to observe the energy consumption effects for both frequency bands in an end-to-end context. We have also shown that the proposed model can accurately model all packet sizes, a feature absent in current TSCH energy consumption models, which only consider the maximum packet size. These results prove that our model is suitable to accurately predict the energy consumption of TSCH networks.

Chapter 4

Recurrent Low-Latency TSCH Scheduling

The content of this chapter is partially based on:

- Glenn Daneels, Bart Spinnewyn, Steven Latré, and Jeroen Famaey (2018). *ReSF: Recurrent Low-latency Scheduling in IEEE 802.15.4e TSCH Networks.* Ad Hoc Networks, 69, 100-114. [Impact Factor: 3.643]
- Glenn Daneels, Steven Latré, and Jeroen Famaey. (2019, June). *Efficient Recurrent Low-Latency Scheduling in IEEE 802.15.4e TSCH Networks*. In 2019 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom) (pp. 1-6). IEEE.

4.1 Introduction

The attractive characteristics in terms of high-reliability and low-power operation have made TSCH very popular in IoT and industrial environments. However, many applications in such environments also require low-latency connectivity. For example, the remote control of important industrial actuators or continuous monitoring are time-critical processes that require minimal delay. To achieve this in TSCH networks while maintaining a reliable and low-power network operation, intelligent scheduling of the time-synchronized resources is crucial.

In this chapter, we focus on power-constrained sensor networks where each node periodically sends measurement updates to a sink (i.e., recurrent traffic) and demands low-latency dissemination of these data. Current state-of-the-art scheduling functions, such as the LLSF, also focus on low-latency forwarding of packets but do not anticipate the recurrent behavior. This results in higher

latency values, because starting the time-expensive resource allocation process when the recurrent traffic already arrived is too late to maintain a low packet delay. We propose the distributed Recurrent Low-Latency Scheduling Function (ReSF). It explicitly supports recurrent traffic and as far as we know, is the first TSCH scheduling function that takes this recurrent traffic behavior into account when scheduling resources. ReSF reserves a minimal-latency path from source to sink and only activates this reserved path when traffic is expected. This allows resources to be reused more intelligently, thus improving throughput and latency. As such, it has only minimal impact on battery life while significantly decreasing the packet delay.

This chapter presents three contributions. First, we define the problem of minimal-latency scheduling of recurrent data transmissions in sensor networks formally, using an Integer Linear Program (ILP). Second, we present ReSF as well as its collision prevention algorithm and integration with 6TiSCH. ReSF is a distributed scheduling function that solves the presented recurrent data transmission scheduling problem in real-time. Lastly, an updated version of ReSF is provided that improves the original ReSF with collision solving heuristics, improved collision avoidance and better support for sporadic traffic. Additionally, we provide extensive simulation results based on the official 6TiSCH simulator, which we extended with a fully functional 6P implementation. A comparison is provided with an extended version of LLSF [55], the state-of-the-art minimal-latency scheduling function for 6TiSCH. We also tested the computational performance of the different collision solving approaches on the OpenMote hardware.

The remainder of this chapter is structured as follows. First, we introduce related scheduling functions in Section 4.2. Subsequently, Section 4.3 introduces the recurrent minimal-latency scheduling problem and Section 4.4 proposes our ReSF algorithm to solve the problem in a distributed manner in real-time. In Section 4.5, different improvements are proposed in the form of an improved ReSF. The proposed scheduling function and its improved version are evaluated and compared to state-of-the-art scheduling functions (and in case of the former, also to the theoretical optimum) in Section 4.6. Finally, Section 4.7 presents the conclusions of this chapter.

4.2 Background and Related Work

In this section we briefly repeat the necessary information on the 6P protocol that manages the resources in 6TiSCH and give an overview of related TSCH scheduling functions.

4.2.1 6P

6P allows neighboring nodes in a 6TiSCH network to add/delete/relocate cells and is part of the 6top IEEE 802.15.4e sublayer which provides the mechanisms

to do distributed scheduling in 6TiSCH [39]. It is the scheduling function that decides when to add or delete cells, and it uses 6P to effectively execute the resource allocation. A more detailed explanation can be found in Section 2.3.2.

At the time of conducting the research for this chapter, the 6TiSCH simulator did not support the 6P protocol and downwards management traffic. As such, we extended the 6TiSCH simulator and added these features to allow for the simulation of more realistic cell reservations and removals to both parents and children.

4.2.2 Related Scheduling Approaches

SF0 was one of the earliest distributed scheduling functions, dynamically adapting a node's resources to the traffic demand [53]. It was already introduced in Section 2.3.3.2 in Chapter 2, explaining the cell allocation and adaption algorithm. Two major drawbacks of SF0 are (i) that it does not take into account the recurrent behavior of traffic, meaning that each reserved cell repeats itself every slotframe and thus wastes resources if packet generation is not equally frequent and (ii) cells are randomly allocated in a slotframe, risking that packets can not be forwarded immediately and have to wait an additional slotframe. These major issues are addressed by ReSF. SF0 is used as a baseline in the experimental evaluation. Based on SF0, Chang et al. developed an improved version called LLSF that daisy-chains cells over the different links up to the root, rather than picking them randomly [55]. While LLSF does not introduce extra overhead, it also does not anticipate recurrent behavior and thus still leaves room for improvement. An enhanced version of LLSF, which is discussed in Section 4.4.6, is also used for comparison in the experimental evaluation.

Another scheduling function Internet-Draft investigated by the 6TiSCH community, is Scheduling Function One (SF1) [61]. SF1 is and end-to-end distributed resource scheduler with hop-by-hop reservation, using a distributed Resource Reservation Protocol (RSVP). It allocates a dedicated path from source to destination which is called a *track*. In contrast to ReSF, so-called TrackIDs are used to filter data packets for certain tracks. In ReSF recurrent cells can be used by any data packet in the queue. The draft was still in an early stage when eventually becoming inactive and specific features taking into account recurrent behavior or algorithms to efficiently allocate cells were not present. As such, we consider our work complementary to SF1, and it could serve to inspire standardization. Morell et al. [62] worked out a solution that combines Resource Reservation Protocol - Traffic Engineering (RSVP-TE) and Generalised Multiprotocol Label Switching (GMPLS) to manage the schedule and connect the network nodes using labelled switched paths. However, the authors do not focus on the recurrent traffic and their numerical evaluation does not take into account the 6P signaling process. Theoleyre et al. [63] also focus on traffic isolation by introducing different tracks for different applications and consider contiguous reserved cells as well as random reserved cells. Their distributed algorithm calculates the required number of cells based on the amount of for-

warded traffic, but it also does not consider the recurrent behavior of sensor data generation, where the reserved cell may only be needed every so many slotframes. It thus risks wasting resources because of a repeated reservation process. A different, but very interesting approach is Escalator, that focuses on minimizing the end-to-end delay by daisy-chaining time slots up to the sink in an autonomous fashion [64]. While such an approach results in a robust, low-overhead schedule, it is less flexible than a distributed approach like ReSF.

In contrast to other scheduling functions (see Chapter 2) and the related works mentioned here, ReSF specifically focuses on (industrial) IoT applications with recurrent traffic patterns. The state-of-the-art approaches either reserve isolated tracks towards the root and/or reserve resources that are continuously activated. Meanwhile, ReSF tries to guarantee fairness, low-latency and energy-efficiency by not isolating the low-latency paths and only activates resources when traffic is expected, maximizing cell reuse and performance.

4.3 Recurrent Low-Latency Scheduling

In this section, we introduce the recurrent low-latency scheduling problem, which minimizes end-to-end data dissemination latency for devices with recurrent data transmissions (e.g., transmission of sensor measurement values after fixed periods). First, we discuss our motivation to tackle the recurrent low-latency problem. Second, we formally formulate the problem and the associated optimal solution as an Integer Linear Program (ILP).

4.3.1 Motivation

Periodical traffic patterns are typical for Wireless Sensor Networks (WSNs) as sensors usually perform measurements at fixed intervals and/or measured data is accumulated over fixed periods and only then sent to the root. ReSF was developed to exploit this recurrent behavior by allocating resources only when traffic is expected. Existing scheduling functions such as SF0 and LLSF allocate new resources based on historical data: during the so-called periodic housekeeping periods the scheduling function monitors how many packets arrive and are generated by the node itself. Afterwards, it calculates the number of cells it needs to reserve during the next housekeeping period. When reserving these cells, the exact arrival times of the incoming/generated packets are not taken into account, which leads to inefficient resource allocation in terms of latency and power consumption.

A simplified example of such inefficient resource allocation is given in Figure 4.1. The given schedule (for the simplicity of this example with only one channel) is the one of the forwarding node B. Node A sends traffic every 8 slotframes to node B, which should forward it as quickly as possible to the root. During housekeeping at node B, the node decides to reserve a cell to its parent so the incoming packet can be forwarded. To do so, first a 6P `ADD` transaction

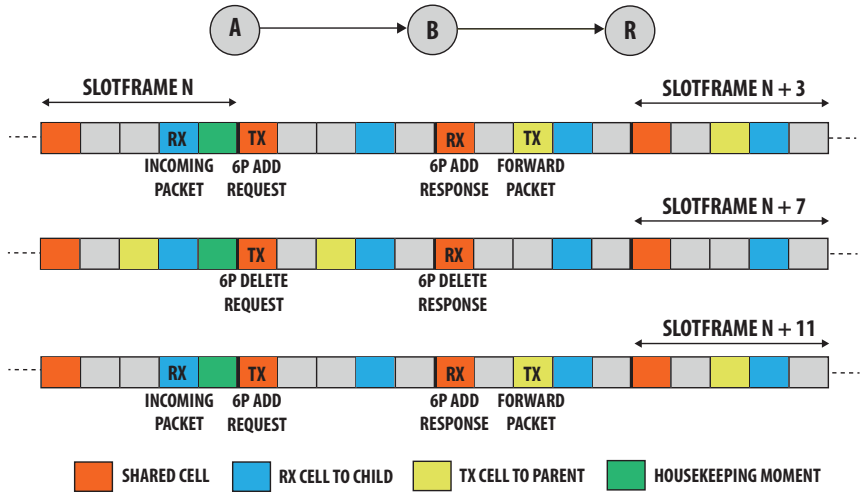


Figure 4.1: Illustration of inefficient resource allocation when not taking into account the recurrent behavior of sensor traffic. The schedule shown belongs to node B.

between node B and its parent needs to take place. Meanwhile the latency of the packet is increasing as it is only in the third slotframe (after 9 time slots) that a TX cell is available to forward the packet to the root. Node A only generates traffic periodically every 8 slotframes, so the next 2 TX cells to the root are not used (and thus radio resources at the receiver are wasted) and even after the housekeeping a third cell is still reserved because the deletion of the cell is not yet complete. One should also take into account the power consumption for sending/receiving these 6P DELETE and 6P RESPONSE messages. When the next periodical packet of A arrives, the process at node B repeats itself. Taking into account the recurrent behavior of the traffic of node A, the increased latency and power consumption introduced by the reservation process can be avoided. This results in a more efficient scheduling function process, which is exactly the goal of ReSF.

4.3.2 Problem formulation

In this section we formally approach low-latency end-to-end packet scheduling in a WSN with recurrent transmissions, as a resource allocation problem. We formulate the underlying optimization problem as an ILP, consisting of input variables, decision variables, constraints and an objective function. Algorithms such as branch and bound can be used to find the optimal solution, by determining the values of the decision variables that maximize (or minimize) the objective function, while satisfying the constraints, given the inputs. As such, the formulation should be considered a resource allocation optimization problem.

The ILP formulation assumes any generated packet to be directed towards the root node. The ILP solution is only used for theoretical comparison purposes in Section 4.6. Solving an ILP on real hardware nodes would be infeasible due to its computationally complexity and the need to be solved centrally. The remainder of this section describes the different aspects of the ILP formulation.

4.3.2.1 Input variables

The WSN comprises a set of nodes N. Packets originate from a set of sources $V \subset N$, with the root node $n_R \in N$ as destination. These sources are recurrent, i.e. they generate a packet at fixed time intervals $S_v + k \cdot T_v, k \in \mathbb{N}_0$. For each source $v \in V : T_v$ and S_v are its period and offset respectively. Note that the period T_v may change over time.

The system is analysed when it has reached steady state conditions, i.e., the number of packets generated per period is constant. The nodes each have only 1 radio, which can either send or receive packets. The combined period of the sources, referred to as the system period, is then given by

$$T_{sys} = \text{LCM}(T_0, T_1, \ldots, T_{|V|-1}), \tag{4.1}$$

where LCM is a function that calculates the Least Common Multiple (LCM) of its integer arguments. When the scheduling is also periodic with period T_{sys}, than it suffices to analyse the system during one system period, beginning when all sources have generated at least one packet. Hence we analyse the system only for time slots

$$i \in I, I = \{S_{max} + 1, \ldots, S_{max} + T_{sys} - 1\}, \tag{4.2}$$

where $S_{max} = \max(S_0, \ldots, S_{|V-1|})$. The set of packets generated by source v in the T_{sys} time slots in I is given by

$$J_v = \{1, 2, \ldots, \frac{T_{sys}}{T_v}\}. \tag{4.3}$$

A complete list and description of the input and auxiliary variables introduced above can be found in Tables 4.1 and 4.2 respectively. Moreover, Figure 4.2 illustrates the notations graphically. In this example, both nodes A and B generate traffic: with $T_A = 6$ and $T_B = 3$, therefore they generate a packet every 6 slots and 3 slots respectively. This results in $T_{sys} = 6$ as this is the least common multiple of 6 and 3.

4.3.2.2 Decision variables

The decision variables represent the solution of the ILP model. $x_{v,p,j}(i)$ is a binary variable, and equals 1 if the packet (v, j) is transmitted over edge $P_{v,p}$ during time slot i, else it is 0. $T_n(i)$ and $R_n(i)$ are 1 when node n is respectively

Table 4.1: Input variables.

Symbol	Description
N	Set of nodes
n_R	Root node
V	Set of source nodes
S_v	Offset of source node v
T_v	Period of source v
P_v	Edges on the path from v to n_R
$P_{v,p}$	p^{th} edge on the path from v to n_R

Table 4.2: Auxiliary symbols.

Symbol	Description
T_{sys}	System period, i.e. $LCM(T_0, T_1, \ldots, T_{V-1})$
I	Time slots in one system period, i.e., $S_{max}+1, \ldots, S_{max}+T_{sys}-1$
J_v	Set of packets generated by source v during one system period

transmitting or receiving during time slot i. $D_{v,p,j}$ is an integer variable holding the delay contribution of each packet (v, j). $O_{v,p,j}$ is a binary variable and a value of 1 indicates that the delay contribution of the p^{th} path element for packet (v, j) crosses the edge of the analyzed interval I (Equation 4.2). In this case the packet arrives only in the next system period, hence T_{sys} is added to the arrival time.

4.3.2.3 Constraints

This section outlines the constraints, which determine the allowed values of the decision variables, as a function of the inputs.

Multi Commodity Flow (MCF) The net packet flow leaving node n_1, corresponding to each packet for source v, depends on whether this node is the source, destination (root node) or neither to this specific packet. For each node n_1, and for each packet (v, j), the difference between the total number of packets flowing out and into n_1 corresponding to (v, j), during one system period, equals either 1, -1, or 0, based on the relation of n_1 to this packet (source, root, or

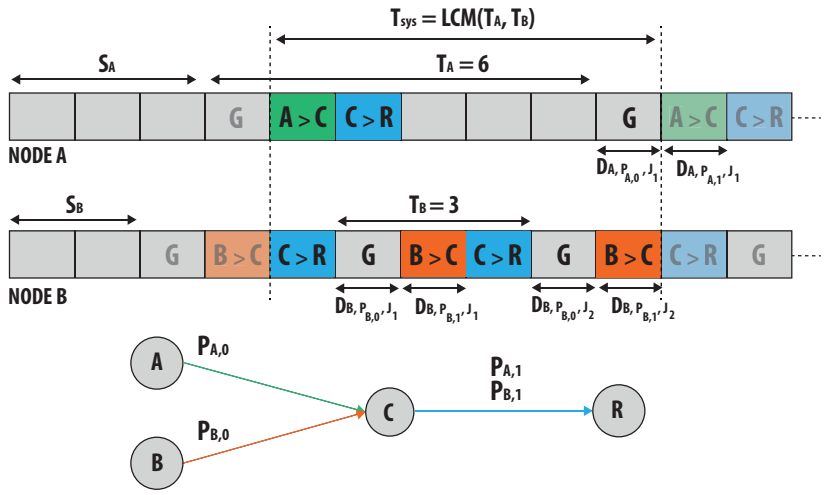


Figure 4.2: Illustration of the notations used in the ILP-formulation. Both nodes A and B generate traffic at time slots denoted by G.

neither).

$$\forall n_1 \in N, v \in V, j \in J_v :$$
$$\sum_{i \in I} \left[\sum_{(n_1,n_2) \in P_v} x_{v,(n_1,n_2),j}(i) - \sum_{(n_3,n_1) \in P_v} x_{v,(n_3,n_1),j}(i) \right] = \begin{cases} 1, & \text{if } n_1 = v \\ -1, & \text{else if } n_1 = n_R \\ 0, & \text{else} \end{cases} \tag{4.4}$$

Transmission and reception A packet is transmitted/received during slot i if a packet corresponding to any of the sources is transmitted or received during this time slot:

$$\forall n_1 \in N, i \in I : T_{n_1}(i) = \sum_{v \in V} \sum_{j \in J_v} \sum_{(n_1,n_2) \in P_v} x_{v,(n_1,n_2),j}(i), \tag{4.5}$$

and

$$\forall n_2 \in N, i \in I : R_{n_2}(i) = \sum_{v \in V} \sum_{j \in J_v} \sum_{n_1 \in N} x_{v,(n_1,n_2),j}(i) \tag{4.6}$$

Single radio A node has only one radio and therefore cannot receive and transmit at the same time.

$$\forall n \in N, i \in I : T_n(i) + R_n(i) \leq 1 \tag{4.7}$$

Delay The delay for each packet is the sum of the individual delay contributions of the edges along the path from the source of the packet towards the root. We distinguish two types of delay contribution. The delay contribution of the first edge on the path is the difference between the time slot at which the packet is first transmitted, and the time slot at which it was generated:

$$\forall v \in V, j \in J_v, p = P_{v,0} : D_{v,p,j} = \\ (\sum_{i \in I} i \cdot x_{v,p,j}(i) + T_{sys} \cdot O_{v,p,j}) - (S_v + j \cdot T_v) \quad (4.8)$$

The delay contribution for the other path elements is the difference between the time slot at which the packet reaches the target of the element, minus the time slot at which it had reached its source:

$$\forall v \in V, j \in J_v, p \in P_v \setminus \{P_{v,0}\} : D_{v,p,j} = \\ (\sum_{i \in I} i \cdot x_{v,p,j}(i) + T_{sys} \cdot O_{v,p,j}) - \sum_{i \in I} i \cdot x_{v,p-1,j}(i), \quad (4.9)$$

where the $T_{sys} \cdot O_{v,p,j}$ term accounts for packets that do not arrive at the source and destination of $P_{v,p}$ during the same system period.

A packet cannot be transmitted by a node during the time slot of its generation, and it cannot traverse multiple edges during one single time slot. Hence, the delay contribution of each path element is limited to

$$1 \leq D_{v,p,j} \leq T_{sys} - 1. \quad (4.10)$$

4.3.2.4 Objective function

The objective is to minimize the average delay of the $\sum_{v \in V} {}^{T_{sys}}/_{T_v}$ packets generated in a system period:

$$\min \frac{1}{|V|} \cdot \sum_{v \in V} \sum_{j \in J_v} \sum_{p \in P_v} \frac{T_v}{T_{sys}} D_{v,p,j} \quad (4.11)$$

4.4 Recurrent Low-Latency Scheduling Function

In the previous section, the underlying resource problem of recurrent scheduling was stated formally and an optimal mathematical solution was provided. However, such an optimal solution is computationally complex and needs to be solved centrally as it assumes perfect network knowledge. Therefore, in this section we propose the new ReSF scheduling function that solves the recurrent low-latency scheduling problem in real-time and a distributed manner. First, the goal and features of ReSF are discussed in a general overview of the scheduling function, which is clarified with an example. Afterwards, detailed information

on the scheduling algorithm, packet loss, the schedule collision prevention and queue housekeeping using Enhanced Low Latency Scheduling Function (eLLSF) are presented. Finally, the integration of ReSF with the 6P protocol is discussed in detail.

4.4.1 General Overview

ReSF was designed to minimize the latency of periodic data transmissions while keeping the reservation overhead, i.e., the number of control messages and the number of reserved slots, to a minimum. It targets IoT systems where traffic is sent periodically, following fixed or slowly changing patterns. The 4 main steps are outlined below:

Scheduling an ReSF reservation ReSF assumes a source node knows its periodic traffic pattern. Using this information, ReSF constructs a so-called *recurrent* path that consists of recurrent cells which are cells that are only activated in slotframes when traffic is expected and are deactivated afterwards. An ReSF reservation is based on the following tuple: $(start, stop, period)$. $Start$ is the ASN at which the first data packet is generated on the source node, $stop$ is the ASN at which point no data will be generated anymore and $period$ represents the periodicity of the data transmission. The tuple is sent from the source node to the next hop and forwarded to the ReSF destination: at each hop the $start$ ASN is incremented to an ASN as closely as possible following the received $start$ ASN, resulting in a daisy-chained ReSF path from source to destination. A path is however not explicitly reserved for one particular packet stream: if a node makes an ReSF reservation that is forwarded all the way up to the destination node, the allocated recurrent cells at an intermediate node along that path can be used by any packet (originating from any node) that is first in the transmission queue of that intermediate node, guaranteeing fairness and lowest average latency.

Anticipating packet loss ReSF anticipates packet loss (cf. Section 4.4.4) by reserving back-up tuples. The number of extra reservations depends on the measured link quality and allows a node to retransmit multiple times consecutively.

Preventing schedule collisions A *schedule collision* is caused when multiple ReSF reservations, located in the same node, want to occupy exactly the same cell at a particular ASN. While theoretically it is possible to search for reservation tuples between sender and receiver that do not share overlapping cells, such a reservation process will take too long and is therefore practically infeasible. Instead, a node that wants to schedule an ReSF reservation message, searches in a pool of reservation tuples, for candidate tuples with the lowest number of unique schedule collisions. The receiver picks the required number of reservations out of the sent candidate tuples by again relying on the lowest number of schedule

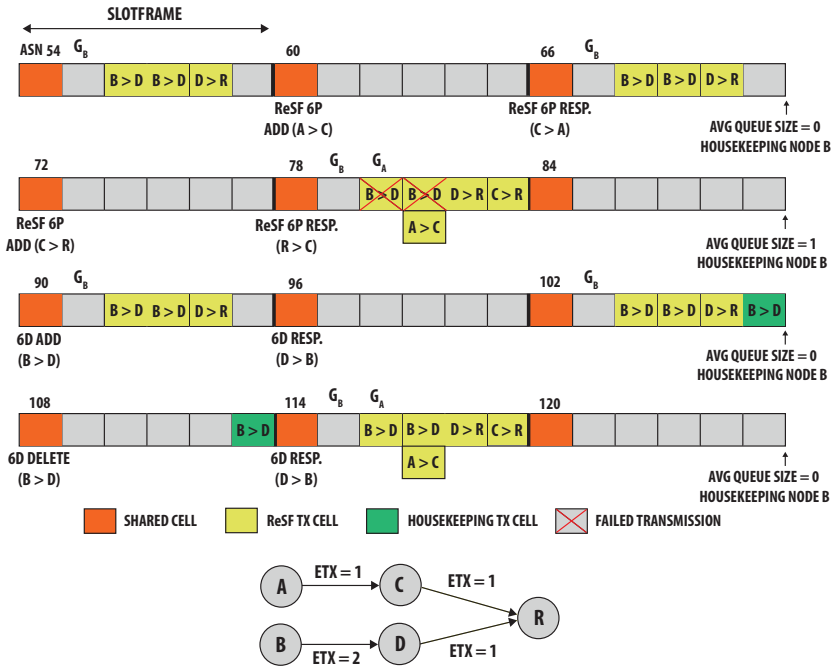


Figure 4.3: ReSF scheduling example with two nodes A and B generating traffic. The schedule represents an aggregation of all individual schedules. The queue size and how this affects the housekeeping of node B is also shown.

collisions. To efficiently calculate this number of unique schedule collisions, we propose the algorithm explained in Section 4.4.5.

Queue housekeeping using eLLSF ReSF prevents failing packets from congesting the queue by doing additional periodical housekeeping. This is done using eLLSF, an enhanced version of the LLSF scheduling function. During this periodical eLLSF housekeeping, a node reserves (or deletes) extra cells that repeat until the next housekeeping moment in order to keep the queue as empty as possible.

4.4.2 Example

As an example, consider Figure 4.3. There are 4 nodes and 1 root R. It is assumed that nodes A and C are far enough away from nodes B and D, so there is no internal interference between these two links. Both node A and node B have a sensor application that periodically generates packets. The traffic destination is root node R. The shown schedule is an aggregated schedule of all individual nodes. The housekeeping moments and decisions of node B are also shown.

The traffic generation pattern of node B looks like ($start = 31, stop = 600, period = 12$), meaning that the first packet starts at ASN 31 and gets repeated every 12 cells. The ReSF reservations from node B to node D and from node D to node R are already installed. Because the ETX of the link between node B and node D is 2, there are two recurrent cells from node B to node D.

The traffic pattern tuple of node A is described by ($start = 80, stop = 790, period = 36$). To reserve the ReSF recurrent cells, node A sends its first ReSF 6P `ADD` message in the `SHARED` cell at ASN 60. Node C responds at its first opportunity in the next `SHARED` cell at ASN 66. To construct a minimal-latency path to the root, node C forwards the reservation to node R, to which the root responds at ASN 78. The ReSF 6P `ADD` reservation from node A to node C contains 6 possible reservation tuples, which is the maximum number of tuples a 6P `ADD` can contain, and that are picked based on the lowest calculated number of schedule collisions. Out of these 6 tuples, node C picks the one with the lowest number of collisions and the closest to the original start ASN, i.e., ($start = 81, stop = 790, period = 36$). From node C to node R the ideal reservation tuple would be ($start = 82, stop = 790, period = 36$) as we assume that the processing of a packet takes less than one cell duration. However, because of the schedule collision that would happen at ASN 82 (with the transmission from node D to R), reservation tuple ($start = 83, stop = 790, period = 36$) has a lower number of schedule collisions and is agreed upon.

Now assume that due to external interference, both transmissions from node B to D at ASN 80 and 81 fail. Because of those failures, the generated data packet of node B ends up in the queue two slotframes in a row. The next housekeeping of node B notices that two out of three times there was an extra packet in the queue at the end of the slotframe. As such, it sends out a reservation at ASN 90 for a housekeeping cell which gets confirmed at ASN 96. Because of this extra cell the queued packet gets forwarded to node D and after that, during the third housekeeping round, the node decides that the housekeeping cell can be deleted.

4.4.3 Scheduling Function Description

The ReSF scheduling function allows nodes to add new recurrent reservations for new data-generating sensor applications, remove deprecated reservations when applications stop generating data and update reservations when an application for example changes the period after which it generates data. To do so, ReSF assumes a source node knows its periodic traffic pattern, e.g., reported by a sensing application, and gets cross-layer updates when this pattern changes. In this section, the different functionalities are described in detail.

4.4.3.1 Scheduling Reservations

We assume that each node periodically generates data for transmission, defined by a ($start$, $stop$, $period$) tuple. ReSF uses the procedures in Algorithms 1

Algorithm 1 Scheduling an ReSF reservation request.

```
1: procedure SCHEDULEREQUEST(start, stop, period, id, dest)
2:     max_tuples ← 6
3:     num_tuples ← getReqTuples(getNextHop(dest))
4:     tuple_pool ← getTuplePool(start + 1, stop, period, resv_buffer)
5:     tuple_list ← list()
6:     while size(tuple_list) ≠ max_tuples do
7:         append(popBestTuple(tuple_pool), tuple_list)
8:     sendRequest(tuple_list, num_tuples, id, dest)
```

Algorithm 2 Receiving an ReSF reservation request.

```
1: procedure RECEIVEREQUEST(tuple_list, num_tuples, id, dest)
2:     ack_tuple_list ← list()
3:     while size(ack_tuple_list) ≠ num_tuples do
4:         append(popBestTuple(tuple_list), ack_tuple_list)
5:     respond(ack_tuple_list, id)
6:     if shouldForward(dest) then
7:         new_start ← getLatestTuple(ack_tuple_list)
8:         scheduleRequest(new_start, stop, period, id, dest)
```

and 2 to, respectively, send a new reservation and/or respond to a received reservation request.

The `scheduleRequest` procedure schedules a new reservation based on the traffic tuple received from the application layer or the tuple derived from an incoming ReSF reservation. It looks for `max_tuples` tuples that equals 6, which is the maximum number of reservations that fit in a 6P `ADD` message, as explained in Section 4.4.7. To determine the number of tuples that should be reserved to anticipate packet loss, `getReqTuples` takes into account the link quality as explained in Section 4.4.4. Using the `getTuplePool` procedure, all tuples in the interval $[(start + 1,\ stop,\ period),\ (start + 1 + reservation\_buffer,\ stop,\ period)]$ are returned. By starting from $start + 1$, ReSF makes sure that a new reservation is reserved after the packet is generated or received from the previous hop: this way cells are *daisy-chained* over multiple hops towards the destination. The `popBestTuple` procedure returns the reservation tuple with the lowest calculated *number of unique schedule collisions* (cf. Section 4.4.5), after checking for collisions with all already activated reservations on that node. Thus, the *reservation_buffer* value represents the trade-off between looking for a minimal number of unique schedule collisions and the risk of introducing more delay by picking a reservation tuple with a large gap between the time slot at which the packet is generated and the actual time slot at which the recurrent packet will be sent. In Section 4.6.1.2, we experimentally determine the best *reservation_buffer* value. Finally, the ReSF request with the

chosen candidate tuples is sent to the next hop.

The `receiveRequest` procedure is straightforward: when it receives an ReSF reservation request, it chooses `num_tuples` tuples from the received `tuple_list`, by using the `popBestTuple` procedure. The receiver acknowledges the sender with this list of `num_tuples` tuples. If the node is not the final destination of the ReSF reservation, it prepares to forward a new reservation, using `scheduleRequest`.

4.4.3.2 Unscheduling and Updating Reservations

When a sensor application stops generating data, all the reserved ReSF slots towards the reservation destination node should also be removed to avoid energy waste. The procedure for removing cells is fairly simple: a node transmits a delete message to the next hop (towards the destination node), complemented with the ReSF reservation `ID`. When the next hop receives this reservation removal message it will forward a delete message with the same `ID` to its next hop which is repeated until the delete message reaches the destination node and all ReSF reservations with that `ID` are removed.

Every reservation has a $stop$ value at which the node stops generating data. When this $stop$ ASN is reached, the reservation gets removed and should be renewed when a node wants to send additional data. Reservations also get removed when they are unused for a fixed amount of time.

It may happen that an ReSF reservation needs to be updated, for example when the sensor data generation periodicity changes. In that case, the node sends a new reservation message to its next hop with the same ID (of the old reservation that needs to be updated) but it will propose a new reservation tuple. This avoids the delay and energy waste of first removing the slots via delete messages and only afterwards reserving new cells. When sending the updated reservation message the same reservation process applies as described earlier in this section. However, there is one additional step: when a node and its next hop agree on a new set of reservation tuples for that particular `ID`, the old reservation is deleted automatically.

4.4.4 Anticipating Packet Loss

To deal with possible packet loss (due to interference) in advance, instead of reserving 1 recurrent cell per packet, ReSF by default allocates a number of recurrent cells equal to the ceiled ETX value of the link to its next hop, $num\_tuples = \lceil ETX(link_{nextHop}) \rceil$, which is calculated in the `getReqTuples` in Algorithm 1. ETX is the expected number of transmissions a packet needs to reach its destination. When forwarding the reservation message, its next hop will apply the same formula for allocating a number of cells to its next hop and so on. Doing this over-provisioning, ReSF copes with possible packet loss due to bad link quality.

4.4.5 Preventing Schedule Collisions

When multiple transmitting and/or receiving ReSF reservation tuples, located in the same node, each expect the exact same cell to be activated at a specific ASN, this is called an ReSF *schedule collision*. Allowing multiple tuples to use the same cell will result in packet transmission/reception failures whereby those packets are queued to be retransmitted and the delay inevitably increases. In order to minimize this delay, such schedule collisions should be kept to a minimum. However, detecting collisions between recurrent reservations that only reoccur every so many slotframes and at different time slots in those slotframes, is much harder than detecting collisions between traditional reservations. The latter ones persist every slotframe at the same time slot, which makes identifying collisions trivial. In this section, we present an algorithm to search for reservation tuples with the least amount of unique schedule collisions.

To efficiently calculate the number of collisions and the exact collision ASNs of two reservations $(start_1,\ stop_1,\ period_1)$ and $(start_2,\ stop_2,\ period_2)$, we search for the solution to:

$$\forall x, y \in \mathbb{Z} : start_1 + period_1 \cdot x = start_2 + period_2 \cdot y \tag{4.12}$$

in the interval $[start_{max},\ stop_{min}]$ with $start_{max} = max(start_1, start_2)$ and $stop_{min} = min(stop_1, stop_2)$. More specifically, we start at $start_{max}$ so we are sure every reservation is already sending. Equation 4.12 can be rewritten into the standard form of a linear Diophantine equation $ax + by = c$ where $a = period_1, b = -period_2, c = start_2 - start_1$ are integers:

$$period_1 \cdot x - period_2 \cdot y = start_2 - start_1 \tag{4.13}$$

A linear Diophantine equation has solutions if and only if the $\gcd(a,b)|c$, meaning that c should be a multiple of the greatest common divisor of a and b. If not, this means there are no collisions between the two reservations. If $\gcd(a,b)|c$, the equation has infinitely many solutions. The solution (x_0, y_0) can be calculated using the extended Euclidean algorithm[1]. Using (x_0, y_0) and the Diophantine solution standard form, one can calculate all solutions:

$$\forall n \in \mathbb{Z} : x = x_0 + n \cdot \frac{b}{gcd(a,b)} \tag{4.14}$$

$$\forall n \in \mathbb{Z} : y = y_0 - n \cdot \frac{a}{gcd(a,b)} \tag{4.15}$$

All (x, y) tuples, for $x, y \in \mathbb{Z}$, are solutions to Equation 4.12. However, we are only interested in the values of n where the two reservation sequences can collide, i.e., in the interval $[start_{max},\ stop_{min}]$. Replacing x, a and b:

[1] http://mathworld.wolfram.com/DiophantineEquation.html

$$start_{max} \leq start_1 + period_1 \cdot (x_0 + n \cdot \frac{-period_2}{gcd(period_1, -period_2)}) \leq stop_{min} \tag{4.16}$$

Based on Equation 4.16, we calculate the lower- and upperbound of the n values that we should consider:

$$n_{start} = (\frac{start_{max} - start_1}{period_1} - x_0) \cdot \frac{\text{gcd}(period_1, -period_2)}{-period_2} \tag{4.17}$$

$$n_{stop} = (\frac{stop_{min} - start_1}{period_1} - x_0) \cdot \frac{\text{gcd}(period_1, -period_2)}{-period_2} \tag{4.18}$$

The number of schedule collisions $numCollisions$ between the two reservations $(start_1, stop_1, period_1)$ and $(start_2, stop_2, period_2)$ equals the numbers of integers in the interval $[n_{stop}, n_{start}]$ (as n_{stop} will always be smaller than n_{start}):

$$numCollisions = \lfloor n_{start} \rfloor - \lceil n_{stop} \rceil + 1 \tag{4.19}$$

A node can immediately and exactly calculate all collision ASNs, by iterating over all the integers in the interval $[n_{stop}, n_{start}]$ using:

$$start_1 + period_1 \cdot (x_0 + n \cdot \frac{-period_2}{gcd(period_1, -period_2)}) \tag{4.20}$$

Subsequently, to calculate the exact number of unique collisions between a new candidate reservation tuple $(start_k, stop_k, period_k)$ and all other already installed $k-1$ tuples on that node, we have to pair-wise calculate the collisions of the tuple $(start_k, stop_k, period_k)$ with every other reservation tuple $(start_i, stop_i, period_i)$, as shown in the `calcNrUniqueCollisions` procedure in Algorithm 3. The candidate reservation tuple $(start_k, stop_k, period_k)$ and the $k-1$ tuples are given as parameters to the procedure.

The procedure iterates over every tuple $(start_i, stop_i, period_i)$ and calculates the $start_{max}, stop_{min}, n_{start}$ and n_{stop} for each tuple t, as defined in Equations 4.17 and 4.18. Then at lines 5 and 6, for every $(start_i, stop_i, period_i)$, we iterate over all n (integer) values in the interval $[n_{stop}, n_{start}]$ and use them to calculate every collision ASN relating to a n value. As show at lines 7 and 8, if the result does not exist yet in the list of unique collisions ASNs, it is added. The returned length of the `ASNList`, at line 9, is the total number of unique collisions of the candidate tuple $(start_k, stop_k, period_k)$. The node will use this number to compare this candidate with the other candidate tuples and select the best candidate, i.e., the candidate with the lowest number of schedule collisions.

Algorithm 3 Exact Collision Solving with Multiple Tuples

```
1: procedure CALCNRUNIQUECOLLISIONS(start, stop, period, tupleSet)
2:     ASNList ← list
3:     for each t ∈ tupleSet do
4:         start_max ← max(start, t.start)
5:         stop_min ← min(stop, t.stop)
6:         nStart, nStop ← ...          ▷ See definition Equations 4.17 and 4.18
7:         for each n ∈ [nStop, nStart] do
8:             ASN ← start + period · (x0 + n · −t.period / gcd(period,−t.period))
9:             if ASN ∉ ASNList then
10:                append(ASN, ASNList)
11:    return length(ASNList)
```

4.4.6 Queue Housekeeping using eLLSF

While extra recurrent cells are reserved to anticipate inferior link quality and action is taken to limit the number of schedule collisions, packets can still end up in the queue at the end of a slotframe when the number of reserved recurrent slots, including the over-provisioned slots, does not suffice due to more unanticipated packet loss or non-recurrent traffic.

To empty the queue and preventing the queued packets from taking up cells that were meant for other ReSF reservations, we use the distributed scheduling function eLLSF. It uses a periodical *housekeeping* moment at which it reserves a required number of cells for the slotframes to come, i.e., cells that repeat *every* slotframe. To keep the delay of those queued packets as low as possible, ReSF calculates the required number of cells for the next eLLSF houskeeping period by averaging the number of queued packets of every slotframe since the last housekeeping, as to have extra resources to clear the queue and further minimize latency. The housekeeping period is a configurable parameter.

eLLSF is actually an extended version of LLSF [55]. The idea behind LLSF is to daisy-chain receiving and transmitting cells used in a multi-hop path in order to decrease the latency. The authors however only described how LLSF behaves in a multi-hop path where each node has one child and one parent. Based on their description, we extended LLSF and implemented eLLSF so it can deal with multiple children and use it for both up- and downstream reservations.

The process of adding and removing cells in eLLSF is similar to LLSF. However, in contrast to LLSF, eLLSF makes a difference between up- and downstream packets when performing slot reservations and removals. This differentiation is crucial as eLLSF considers the (possible) multiple children of a node.

Scheduling eLLSF cells If a node wants to reserve *n* transmit cells to its parent, it uses the following four-step process:

1. For each reception cell from a child, count the number of cells between

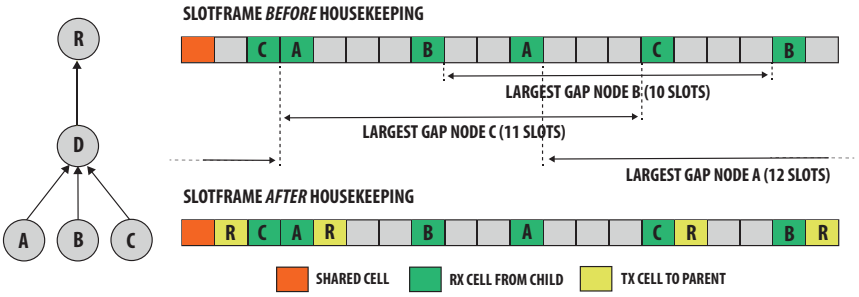


Figure 4.4: eLLSF housekeeping on node D that wants to reserve 4 transmission cells to its parent.

that reception cell and the previous reception cell of that child.

2. For each child, pick the cell with the *largest gap* to its left: this is the largest amount of cells between two reception cells of that child.

3. Distribute the cells that are to be reserved evenly and randomly among all children. For example, if a node wants to reserve five cells to its parent and has three children, first assign one transmit cell to each of the children of the node. After that, assign the remaining two cells to two random children.

4. For each child that is assigned one or more transmission cells, place the transmit cell(s) as closely as possible to the right of the reception cell with the largest gap of that child.

When making a transmit cell reservation in the other direction i.e., to one of its children, the 3-step LLSF reservation process is used: only the reception cells of the child – to which the reservation is made – are considered when looking for the largest gap. This way, minimum-delay communication between that child and its parent is encouraged.

Figure 4.4 shows an example of a eLLSF housekeeping moment. The schedule of node D before and after the housekeeping moment is shown (for simplicity reasons only one channel is used). Node D wants to reserve 4 cells to its parent. Therefore, it first determines the largest gap RX cells of each child. Afterwards, it places a transmission cell as closely as possible to the largest gap reception cell of each child, with one transmission cell to spare (because there are only 3 children). This last transmission cell is randomly assigned to a node: in this case it is assigned to node B and placed in the beginning of the slotframe because there are no empty cells left at the end of the slotframe.

Unscheduling eLLSF cells When a node has too many cells to a neighbor (i.e., parent or child), it will remove 1 or more cells to that neighbor. Again, eLLSF

makes a distinction between removing a cell to a parent or to a child. When removing a transmission slot to a parent, the algorithm looks for the largest gap between a transmission cell to that parent and the reception cells of *all* children. Then it removes the transmission cell with the largest gap to its left. In the case a transmission cell to a child is being removed, we use the unscheduling process of LLSF. It first looks for the largest gap between a transmission cell to that child and the previous reception cell of that same child. Then it removes the transmission cell to that child with the largest gap to its left.

Preventing collisions between eLLSF and ReSF In order to prevent the eLLSF housekeeping from reserving recurrent cells that were meant for ReSF reservations, a 2-step process is used: (1) calculate all the cells occupied by active ReSF reservations during an interval $[now, now + buffer]$ (with $buffer$ being a preset parameter currently fixed at the length of 10 slotframes), (2) schedule the housekeeping cells using eLLSF while not considering the slots calculated in step 1 as available slots.

4.4.7 6P Integration

This section clarifies how to integrate ReSF in the 6P protocol. When looking at the format of a normal 6P `ADD` transaction, an ReSF 6P transaction is very similar: `ADD` and `DELETE` requests contain an additional ReSF `ID` that identifies the reservation, the `Destination` of the reservation and a list of ReSF reservations that have to be added/deleted. The receiver answers with the number of requested ReSF reservation tuples that fit the node best. If the receiver does not agree with any of the proposed tuples, it answers with an empty 6P `RESPONSE` which indicates that the sender should propose other ReSF reservation tuples.

In Figure 4.5, both a 6P `ADD` message and ReSF reservation format are shown. The maximum length of a 6TiSCH packet is 127 bytes. The IEEE 802.15.4 header has a length of 23 bytes (including the `FCS` field) while the 6top header only has a length of 8 bytes when leaving out the list of cells. This leaves 96 bytes to specify the ReSF reservation. The ReSF `ID` can be represented by a 2-byte integer, the `Destination` by 8 bytes. An ReSF reservation has a length of 14 bytes containing two 5-byte ASN values for the $start$ and $stop$ value, a 2-byte channel offset and a 2-byte $period$ value. This means that a 6P `ADD` reservation can include 6 reservations. There are 2 bytes left for future use.

4.5 Improved ReSF

In this section we present an improved version of the original ReSF scheduling function as presented in Section 4.4. This updated ReSF improves over the original ReSF with collision solver heuristics, a new approach for collision avoidance and support for sporadic traffic. In contrast to the original ReSF, the improved ReSF does not include a $stop$ ASN in the reservation tuple, i.e.,

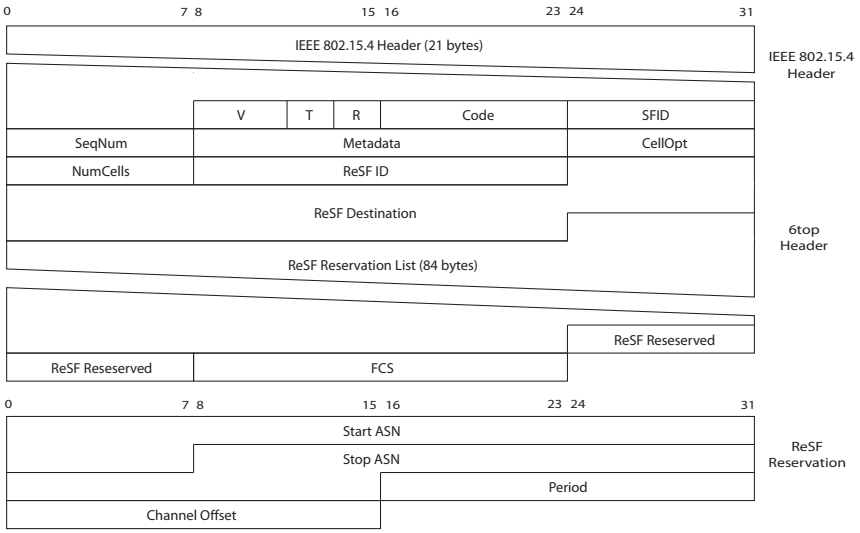


Figure 4.5: Example of a 6P ADD request format for ReSF reservations (maximum length of 127 bytes) and an ReSF reservation that includes the channel offset and the reservation tuple (14 bytes).

$(start, period)$. Therefore, reservations are assumed to continue until they are explicitly removed by a 6P DELETE message.

4.5.1 Fast Collision Solving

As explained in Section 4.4.5, to avoid the negative effects of schedule collisions, an ReSF node aims to keep these schedule collisions to a minimum by applying a collision solver algorithm to a list of possible candidate tuples. Whenever a node proposes a new reservation tuple to another node or when a node receives an ReSF request, the node selects the reservation tuple candidate with the lowest number of schedule collisions as calculated by the procedure in Algorithm 3.

Below we first explain the slight change in the calculation of the number of unique collisions. Afterwards, we introduce less computationally-intensive heuristics.

4.5.1.1 Exact Collision Solving with Multiple Tuples

In this section we explain the slightly changed calculation of the exact number of collisions between one tuple and multiple other tuples (compared with the calculation in Section 4.4.5), due to the missing $stop$ ASN in the reservation tuple of the improved ReSF.

When calculating the exact number of unique collisions between a new candidate reservation tuple $(start_k, period_k)$ and the already $k-1$ installed

Algorithm 4 Exact Collision Solving with Multiple Tuples

```
1: procedure CALCNRUNIQUECOLLISIONS(start, period, tupleSet, startMax, LCM)
2:     ASNList ← list
3:     for each t ∈ tupleSet do
4:         nStart, nStop ← ...                ▷ Now using startMax and LCM
5:         for each n ∈ [nStop, nStart] do
6:             ASN ← start + period · (x0 + n · −t.period / gcd(period,−t.period))
7:             if ASN ∉ ASNList then
8:                 append(ASN, ASNList)
9:     return length(ASNList)
```

reservation tuples on that node, we now pair-wise calculate the collisions with every tuple $(start_i, period_i)$ in the interval:

$$[start_{max}, start_{max} + LCM(period_1, period_2, \ldots, period_k) - 1] \quad (4.21)$$

with $start_{max} = max(start_1, start_2, \ldots, start_k)$ and LCM being the *least common multiple*. More specifically, we start at $start_{max}$ so every reservation is sending and after a length of $LCM(period_1, period_2, \ldots, period_k)$, the same sequence repeats itself. The candidate reservation tuple $(start_k, period_k)$, the $k-1$ tuples, $start_{max}$ and $LCM(period_1, period_2, \ldots, period_k)$ are given as parameters to the new `calcNrUniqueCollisions` procedure in Algorithm 4. The main difference with the procedure in Algorithm 3 is that n_{start} and n_{stop} are now calculated with the given $start_{max}$ and LCM parameters as defined in this section. More specifically, the $start_{max}$ and $stop_{min}$ variables in both Equations 4.17 and 4.18 are replaced with $start_{max} = max(start_1, start_2, \ldots, start_k)$ and $stop_{min} = start_{max} + LCM(period_1, period_2, \ldots, period_k) - 1$ respectively.

Algorithm 4 shows that the computational performance of the exact collision solving approach is dependent on the size of $tupleSet$, but also of the length of the interval $[n_{stop}, n_{start}]$, i.e., the number of schedule collisions, for each tuple in $tupleSet$.

4.5.1.2 Collision Solving Heuristic Using Summation

We introduce a heuristic to approximate the exact algorithm in terms of collision solving performance, while avoiding the computationally-intensive procedure of calculating all the unique collision ASNs between one and multiple ReSF tuples.

The heuristic is demonstrated in Algorithm 5. In the `calcSumCollisions` procedure, we iterate over all the existing tuples in `tupleSet` and calculate the number of collision ASNs, which is obtained by filling in $\lfloor n_{start} \rfloor - \lceil n_{stop} \rceil + 1$, that the candidate tuple $(start_k, period_k)$ has with each of the $k-1$ tuples. All these totals are aggregated in the `sum` variable which results in an estimation of

Algorithm 5 Collision Solving Heuristic Using Summation

```
1: procedure CALCSUMCOLLISIONS(start, period, tupleSet, startMax, LCM)
2:     sum ← 0
3:     for each t ∈ tupleSet do
4:         nStart, nStop ← ...
5:         sum ← sum + ⌊nStart⌋ − ⌈nStop⌉ + 1
6:     return sum
```

the unique number of collisions. Likely, this total is an overestimation as it can include collisions ASNs that were counted multiple times with different tuples.

In contrast to the exact approach, the heuristic is only dependent on the size of $tupleSet$, and not on the number of schedule collisions for each tuple in $tupleSet$, making it computationally more attractive than the exact solution.

4.5.1.3 Collision Solving Heuristic Using Minimal Delay

We introduce another heuristic that always picks the earliest tuple available in the list of candidate tuples (i.e., with the smallest $start$ value), which is the tuple with the time slot directly after when the packet is generated or received. Theoretically, this results in minimal delay and a perfectly daisy-chained path from source to root. However, a possibly high number of schedule collisions with this tuple can lead to packet loss and increased latency. Computationally, always choosing the earliest and thus first tuple makes this heuristic the most attractive.

4.5.2 Improved Collision Avoidance

Next to the collision solving algorithm, we introduce an additional feature that aims to avoid schedule collisions. In the original ReSF the housekeeping scheduling function, i.e., eLLSF, could not reserve cells that were occupied by ReSF for a future period in time (i.e., parameter $buffer$ in Section 4.4.6). However, determining the optimal length of this period is not trivial. In the updated ReSF version we propose an alternative and limit the choice of the eLLSF slots to a pre-determined set. When allocating slots for housekeeping, eLLSF is only allowed to choose slots out of this set and ReSF can never activate a cell at any of these time slots for a recurrent reservation. Additionally, when a node applies the collision solver algorithm to select the best candidate tuple, every slot in that set is included as a separate reservation to compare to. Those slots are characterized by the tuple $(start = ASNSlot, period = slotframeLength)$, where $ASNSlot$ is the ASN of the time slot in the first slotframe. By explicitly incorporating these slots in the collision solver process, the selected candidate tuple will cause less schedule collisions between ReSF and eLLSF.

4.5.3 Supporting Sporadic Traffic

In the previous version of ReSF, allocating extra cells to empty the queue from failed packets was based on a periodic housekeeping moment. During that moment the average number of packets in the queue since the last housekeeping was used as an estimation for the (de-)allocation of extra cells until the next housekeeping. In this updated version of ReSF, the extra cell adaptation algorithm is changed to the traffic adaptation algorithm of MSF [38], which was introduced in Chapter 2. Using this algorithm, ReSF does not look at queue contents, but at the used extra cells during the last *NumCellsPassed* elapsed extra cells. *NumCellsPassed* is a configurable parameter. Additionally, ReSF limits the (de-)allocation of extra cells to only one at a time. In the scenarios with less frequent traffic and more sporadic traffic, by applying these changes the traffic adaptation algorithm should fluctuate less and management signaling in the network decreases.

4.6 Evaluation

This section evaluates the proposed ReSF scheduling algorithm. In the first part of the evaluation, the original ReSF proposed in Section 4.4 is tested extensively. In the second part, the performance of the updated ReSF, combining the different improvements introduced in Section 4.5, is assessed.

4.6.1 Original ReSF Evaluation

This section evaluates the performance of ReSF as introduced in Section 4.4 and compares it to the state-of-the-art TSCH scheduling functions SF0 and eLLSF. A variety of experiments was conducted while observing several metrics including packet latency and charge drawn. First, we present the different experiment parameters. Afterwards, we determine the optimal value of the $reservation\_buffer$. Finally, we show the performance of ReSF with both static and dynamic traffic patterns.

4.6.1.1 Simulation Setup

In order to evaluate the performance of ReSF, we used the 6TiSCH simulator as introduced in Chapter 2 [83][2]. As at the time of conducting the research, the simulator's implemented 6top sublayer did not support a real message-passing 6P protocol, we extended the simulator with this feature. The presented ILP formulation is solved using Gurobi, which uses a hybrid solution procedure that combines three different approaches to find an exact solution: (1) cutting planes, (2) branch and bound, (3) relaxation and decomposition [102].

[2]The specific simulator implementation used for this evaluation is publicly available at `https://github.com/imec-idlab/6tisch-ReSF`

Table 4.3: The default experiment parameters.

Parameter	**Value**
Grid size	4.5 km x 4.5 km
Inter-node default distance	0.230 km
Nr. of runs per experiment	20
Simulated time	30 min
Frequency	2.4 GHz
Nr. of channels	16
Stable RSSI	−93.6 dBm (PDR $\sim$ 0.5)
Nr. of stable neighbors	1
Avg. nr. of hops (25/100 nodes)	3.4 ($\sigma = 0.4$) / 5.8 ($\sigma = 0.4$)
Avg. nr. of children (25/100 nodes)	2.2 ($\sigma = 0.4$) / 1.7 ($\sigma = 0.1$)
Slotframe size	101
Nr. of SHARED cells	3
6top housekeeping	False
SF0 threshold	0
RPL parent set size	1
RPL DIO Period	5 s
ReSF reservation_buffer	64
Housekeeping period	10 s
Time slot duration	10 ms
Packet size	127 bytes

During all experiments, the nodes are placed on a grid, with the root node positioned in the center. The (x, y) grid position of each node (except for the root node) is determined at random – with a different seed for each experiment iteration – following the normal distributions $\mathcal{N}(x_0, (\frac{d}{8})^2)$ and $\mathcal{N}(y_0, (\frac{d}{8})^2)$ respectively, with (x_0, y_0) being the initial grid position of each node. The distance is d, set to 230 meters, to ensure the topologies did not end up being star topologies, while still maintaining end-to-end multi-hop connectivity between the root and every other node. Generating heterogeneous traffic is done sampling a normal distribution $\mathcal{N}(t, (\frac{t}{4})^2)$, with t the mean traffic rate, for which 3 different values are used: 1 packet/min, 6 packets/min and 12 packets/min per node. All network traffic is sent to the same sink node: the network root. The default parameter values are summarized in Table 4.3. The optimal housekeeping period parameter is experimentally determined and is set to 10 s for all three scheduling functions (i.e., SF0, eLLSF and ReSF) as they all employ periodical housekeeping. 6top housekeeping (i.e., relocation of cells) is not enabled as this was not added to the message passing 6P implementation. To bootstrap the network, 3 SHARED cells are installed at the first 3 cells of every slotframe.

All results in this section exclude the initial warm-up period in which the network topology converges, meaning that each node has already selected a

preferred parent and has negotiated 1 dedicated cell to that preferred parent. In case of ReSF, it also means that each ReSF reservation of a node has already reached the root. It is assumed that a link-layer `ACK` message does not fail and that the intermediary processing of a data packet takes less than one cell length (and thus can be forwarded in a cell immediately following the cell in which it was received).

The *latency* metric encompasses the total time it takes a data packet to reach the root, from the moment it is generated. It is important to mention that a packet is only dropped after it has been retransmitted at least 5 times and the queue of the node is full. The *charge drawn* metric is the aggregated charge drawn by the radio of all the nodes, taking into account idle listening, transmission and reception of each data and acknowledgement packet. The charge values used in the simulator that correspond to these different actions are based on the work by Vilajosana et al. who provided a realistic TSCH energy consumption model [91]. All experiment results show the mean of 20 (random) iterations and the associated standard error.

4.6.1.2 Reservation Buffer

The *reservation_buffer* parameter represents the number of tuples a node will compare to all already activated reservations, when looking for the tuple with the smallest collision rate, as explained in Section 4.4.3.1. As can be observed in Figure 4.6, a higher *reservation_buffer* has a significant impact on the network latency when there is a network load of 12 packets/min: taking the optimal value of 64 improves 56 % in delay over not considering any extra tuples (i.e., a value of 0). When looking at lower traffic loads, increasing the buffer leads to higher delays. At lower traffic loads the schedule is less congested and low collision rate tuples are commonly available. Looking at tuples with later $start$ ASNs to find an even better collision rate results in an increased latency because of the extra difference in time between the packet arrival and the moment it can be sent to the next hop. The *reservation_buffer* parameter is set to a value of 64 for all experiments. For the simplicity of parameter configuration, we show the results for one *reservation_buffer* value for the different traffic loads. This, of course, has an impact on the lower traffic loads: for 1 packet/min the latency result is 16 % worse than the optimum (i.e., a *reservation_buffer* value of 0) and for 6 packets/min are 6 % worse than the optimal results at a value of 4.

4.6.1.3 Static Traffic

Figure 4.7 shows the average latency for both 25 and 100 nodes in topologies with static traffic patterns, meaning that a node will not change the period with which it generates traffic throughout the experiment.

The results show that taking into account recurrent traffic behavior combined with daisy-chaining the cells up to the root, results in superior performance in terms of latency. Looking at results of 100 nodes with the traffic load means of

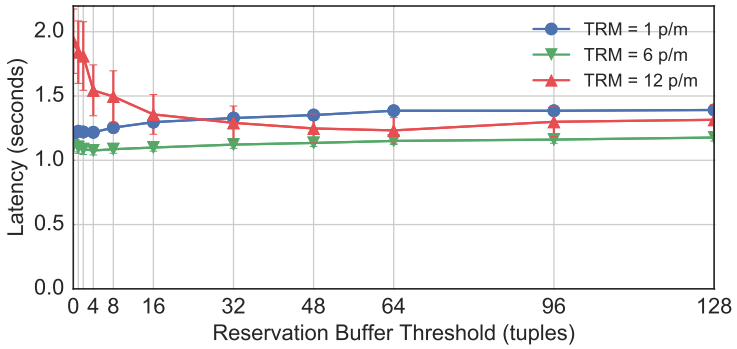
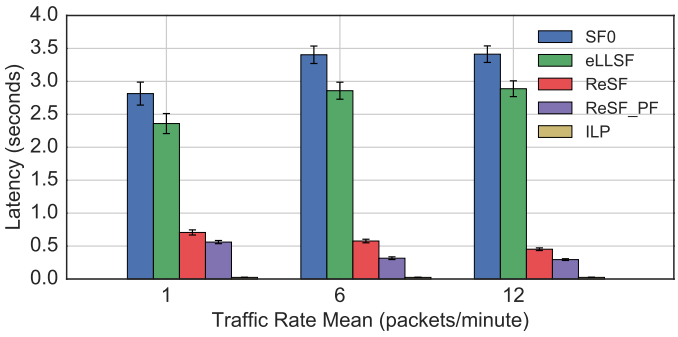


Figure 4.6: Reservation buffer parameter experiment with 100 nodes.

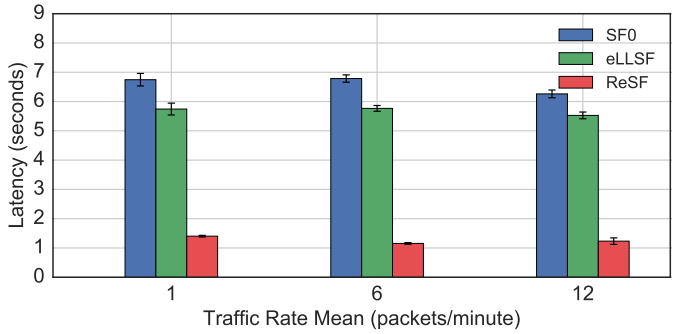
1 packet/min, 6 packets/min and 12 packets/min per node, the relative latency improvements of ReSF over eLLSF are respectively 76 % (from 5.74 s to 1.40 s), 80 % (from 5.77 s to 1.16 s) and 78 % (from 5.53 s to 1.23 s). Looking at the ReSF results for 100 nodes when traffic increases, the latency is almost constant. This means that because of the traffic behavior-aware reservations ReSF is well equipped to deal with different traffic rates. Looking at the results in Figure 4.7a, the latency of ReSF is even slightly decreasing at higher mean traffic rates. This is because ReSF defines the number of cells needed for a packet transmission as the *ceiled* ETX value per link which is dynamically determined based on the number of retransmissions. For example, if a link has an ETX value of 1.1, ReSF will reserve 2 cells per link per packet sent, i.e., an over-provisioning of 1 cell. However, with an ETX value of 1.1 the probability that the packet will actually need two cells is rather small. This means that the over-provisioned cell(s) can be used by other packets. This effect is magnified when dealing with higher traffic rates as there will be more over-provisioned cells.

As expected, due to the daisy-chained paths scheduled by eLLSF, it also improves over the random reserved cells of SF0 with respectively 15 %, 15 %, and 12 %. Packet loss is minimal for all 3 scheduling functions: when considering the high traffic load scenario of 12 packets/ min with 100 nodes, ReSF has the least amount of packet loss with 0.8 % of the packets lost, followed by eLLSF with 1.7 % and SF0 with 1.8 % packet loss. Considering the other traffic loads, the packet loss of ReSF is negligible at a maximum of only 0.04 %. For eLLSF and SF0, the maximal packet loss values are 0.6 % and 0.7 %.

Table 4.4 shows that the same trend holds for larger network sizes up to 200 nodes and a traffic rate mean of 12 packets/ min. For 150 and 200 nodes, ReSF has an average latency of 1.94 s and 2.83 s respectively, while eLLSF has an average latency of 6.93 s and 8.44 s respectively. However, the packet losses introduced by ReSF increase to 4.24 % for 150 nodes and to 9.87 % for 200 nodes, while for eLLSF the packet loss increases to 4.08 % and 9.02 %



(a) Latency for 25 nodes.



(b) Latency for 100 nodes.

Figure 4.7: Latency results for static traffic with 25 and 100 nodes.

respectively. These results show that while ReSF has slightly more packet losses compared to eLLSF when the network scale increases, ReSF scales significantly better than eLLSF in terms of latency.

Figure 4.7a also shows the optimal solution. It is calculated by solving the ILP formulation of the recurrent traffic problem, as defined in Section 4.3.2. Optimal results for 100 nodes are omitted, due to the exponential execution time increase in terms of network size. As the ILP formulation assumes perfect link conditions without interference (i.e., $ETX = 1$), the graph also shows ReSF results under perfect network conditions (i.e., RESF_PF) for a more fair comparison. The ILP results average around 25 ms which is significantly better than the normal ReSF results or the ReSF experiment with perfect links of which the latency for all traffic rates averages around 250 ms. The reason that the ILP solution has a latency about 10 times lower than ReSF_PF is because its result is the theoretical optimal solution that avoids schedule collisions using perfect global information, and it does not take into account interference or signaling overhead.

Table 4.4: Comparison of eLLSF and ReSF latency and packet loss values for a traffic rate mean of 12 packets/ min for different network sizes.

Size	eLLSF		ReSF	
	Latency (s)	Loss (%)	Latency (s)	Loss (%)
25	2.89	0.14	0.45	0.0
100	5.53	1.7	1.23	0.8
150	6.93	4.08	1.94	4.24
200	8.44	9.02	2.83	9.78

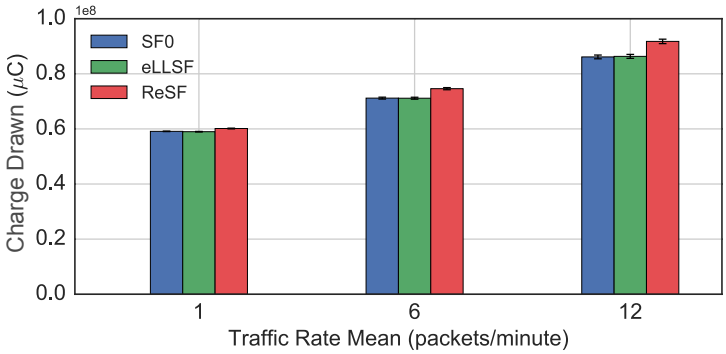


Figure 4.8: Charge drawn for 100 nodes.

The charge drawn results show that there is only a minimal impact of ReSF on battery life: a maximum increase of 6.3 % increase over eLLSF (and 6.54 % over SF0) for a traffic load of 12 packets/min per node and a minimum of 1.96 % (and 1.74 % over SF0) for a traffic load of 1 packet/min. Figure 4.8 shows the charge drawn for 100 nodes (due to the similarity, the results for 25 nodes are not shown). The fact that there is a slight increase in charge drawn is due to the extra cells that ReSF reserves in the slot frames when traffic is expected. For each scheduling function, all nodes in the network have – by default – a dedicated transmission cell to their parents to allow fast management communication (i.e., 6P transactions) and also data transmissions. Next to those default reserved dedicated cells, ReSF also reserves recurrent cells that are only activated when traffic is expected and housekeeping cells to empty the queue of any remaining traffic. SF0 and eLLSF also have to reserve extra cells when dealing with unanticipated high traffic loads and because they do not anticipate the recurrent traffic with recurrent cells they have to send significantly more 6P overhead as shown in Figure 4.9. The figure shows the number of sent 6P packets before and after network convergence. For the lowest traffic load, ReSF has higher total overhead, but this is only because of the sent ReSF reservations during network convergence. When the network has converged, the overhead

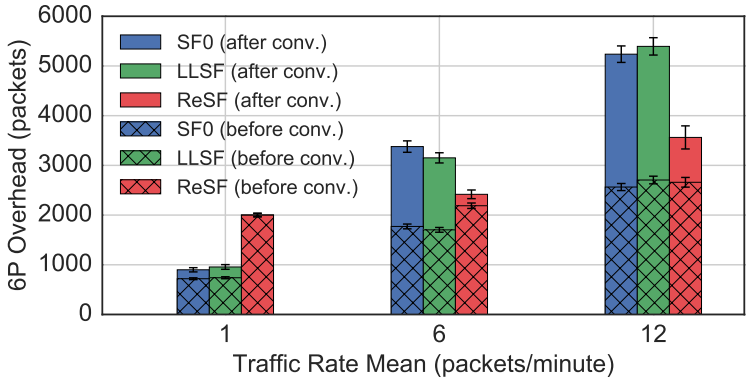


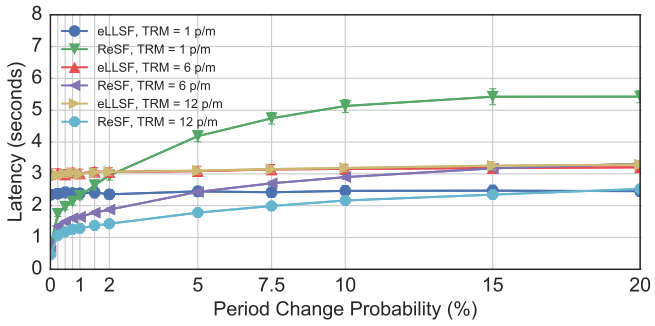
Figure 4.9: Number of sent 6P messages, before (i.e., hatched bars) and after the network convergence.

of SF0 and eLLSF is significantly higher.

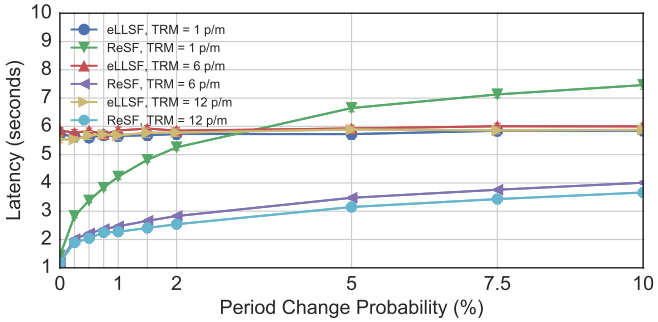
4.6.1.4 Dynamic Traffic

In these experiments, ReSF and eLLSF are tested on how they deal with dynamic traffic. Every second every node has a probability that its traffic generation period changes. When the traffic generation period changes, the new period is drawn from the sample normal distribution out of which the original traffic period was drawn.

The results in Figure 4.10 show that eLLSF is not affected by the traffic period changes. Because the changes in traffic period are not that significant, the housekeeping of eLLSF does not continuously need to send 6P `DELETE` and `ADD` to adjust the number of resources. The number of 6P messages thereby equals the overhead as if there were no traffic period changes, leading to similar latency results. This is in contrast to ReSF, where for each traffic period change at a node, a new recurrent reservation needs to be forwarded to the root. This additional ReSF 6P overhead will hold up more data packets in the queue, which results in additional 6P ReSF housekeeping overhead. All this extra overhead decreases the performance of ReSF. However, the results show that for 25 nodes, 12 packets/min and a probability up to 20 % (meaning that, on average, every second 5 nodes change their traffic period), ReSF can deal with these dynamic traffic periods and it improves on latency while maintaining a throughput equal to eLLSF. The graph shows that ReSF actually deals better with the changing traffic periods when the traffic rate mean is higher. This is because when the traffic rate is higher, recurrent cells are more frequently available to forward the updated 6P ReSF reservations and thus less data packets are obstructed from being forwarded. When observing 100 nodes at 12 packets/min, ReSF performs better than eLLSF up to somewhere between 1.5 % and 2 %. While the latency graph shows a significant improvement by ReSF, the throughput results showed



(a) Latency for 25 nodes.



(b) Latency for 100 nodes.

Figure 4.10: Results for dynamic traffic with 25 and 100 nodes, comparing ReSF to eLLSF, as a function of the probability that the traffic generation period of each node changes every second.

that at 1.5 % the throughput of ReSF is better than eLLSF while at 2 % it was slightly less (i.e., ReSF had 0.28 % less throughput).

It is important to note that such frequent traffic rate changes are unlikely for most real-world sensing applications, such as temperate or heart rate measurements. For example, in a topology with 100 nodes a 2 % change probability means that it is expected that 2 out of 100 nodes change their traffic period every second and every node would change it every 50 s on average. Considering a generation mean of 1 packet/min such a change probability becomes even more unrealistic as a node would change its generation period faster than it generates an actual packet. So, a probability of 2 % is very high which means that ReSF is well-equipped to handle dynamic traffic rates.

Table 4.5: The 6TiSCH simulator parameters.

Parameter	Value
Nr. of runs per experiment	25
Simulated time	1 h
Frequency	2.4 GHz (16 channels)
Stable RSSI	−91 dBm (PDR ∼ 0.75)
Slotframe size	101 (with 15 ms slots)
Nr. of SHARED bootstrap cells	7
6top housekeeping	False
TSCH [min, max] back-off exp.	[0, 1]
Bayesian broadcast probability	0.33
RPL parent set size	1
RPL DAO period	90 s
MSF Join process	False
MSF NumCellsUsed [low, high]	[4, 12]
MSF NumCellsPassed	16
ReSF reservation buffer	20
ReSF reserved back-up slots	20

4.6.2 Improved ReSF Evaluation

In this section, we evaluate a newer version of ReSF that combines the improvements introduced in Section 4.5. First, we present the updated simulation configuration. Afterwards, we evaluate the collision solving heuristics and the collision avoidance feature. Finally, we compare the improved ReSF to the original ReSF and the scheduling function eLLSF.

4.6.2.1 Simulation Setup

To properly evaluate the performance of improved ReSF, we use the official 6TiSCH simulator as introduced in Chapter 2[3]. As the development of the 6TiSCH simulator is an ongoing process, it is important to mention that the simulator used for the research in this section was a far more updated version of the older simulator version used in the evaluation Section 4.6.1. Therefore, comparing results from sections to each other is not recommended.

The simulator is again extended with eLLSF. The resource adapting algorithm of eLLSF is now changed to that of MSF [38]. A summary of the simulator configuration can be found in Table 4.5. The small maximum back-off exponent observed in the configuration was experimentally set as this value resulted in better latency and throughput performance of eLLSF compared to larger values.

[3]The specific simulator implementation used for this evaluation is publicly available at `https://github.com/imec-idlab/6tisch-new-ReSF`

During the different iterations, the nodes are placed on a grid with the root node positioned in the center and each node its final grid position is adjusted slightly following a normal distribution around its initial grid position. The initial inter-distance between the nodes is 100 m and the average hop count is 5.5 ± 0.5. The scheduling functions are compared for three different traffic scenarios, called *fast* (i.e., 3 s, 6 s or 9 s), *frequent* (i.e., 30 s, 45 s or 60 s) and *non-frequent* (i.e., 300 s, 450 s or 600 s). At the start of the experiment, a node picks one of the three transmission intervals of the specific scenario uniformly at random. All network traffic is sent up to the root. The discussed results discard the bootstrap of the network during which the network converges, meaning that each node already has a reserved dedicated `SHARED` cell to its preferred parent (i.e., a SHARED cell only shared between those two nodes). In the case of ReSF, this also means that each node's reservation reached the root.

The *latency* metric discussed in the evaluation is defined as the time it takes for a data packet to reach the root, measuring from the moment it was generated on the source node.

4.6.2.2 Collision Solving Approaches

In this section, the performance of the proposed collision solving heuristics is evaluated. We compare the exact solution to the sum and the minimal delay heuristic and a random approach. The random approach randomly selects a tuple out of the proposed candidates. First, we compare the different approaches by observing the error in estimated schedule collision percentage. Afterwards, we evaluate the computational performance of the heuristics on real hardware. Finally, the different solutions are evaluated in a simulated 6TiSCH network in terms of packet delivery latency.

Collision Percentage Error To evaluate the performance of the different approaches, we introduce the *collision percentage error* metric, defined as the absolute difference in collision percentage (i.e., the number of unique collisions over the total number of transmissions of the candidate tuple), between the tuple chosen by the exact approach and the one chosen by the heuristic.

We performed a standalone experiment (i.e., without the simulator) in which we calculated the best candidate for the ReSF tuple $(start_{new}, period_{new})$ for different numbers of reservations already present on the node. The experiment was repeated 2000 iterations. For each iteration the start time and the period of all tuples were randomly chosen between 101 time slots (i.e., the length of a slotframe) and 6000 time slots (i.e., 90 s) and the number of candidates tested was 64, as determined in our previous work. We limited the least common multiple of the periods to 12 hours (in case the LCM was larger).

Figure 4.11 shows the results. As expected, the sum heuristic outperforms the two other heuristics as its maximal error only goes up to 1.1 %, at 2000 tuples, while the minimal delay and random approach have an error up to 4.4 % and 4.5 %, both at 1000 tuples with outliers that exceed a 10 % error. For

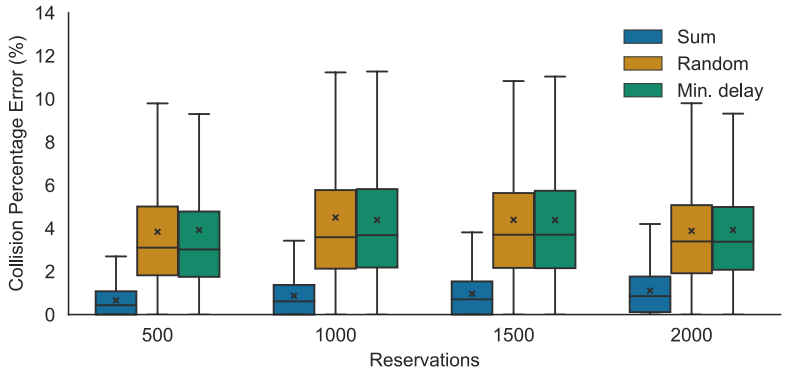


Figure 4.11: Collision percentage error of the different heuristics, compared to the optimal collision approach.

the sum heuristic, this means that on average the tuple that the sum heuristic chooses only has a 1 % difference in terms of number of collisions with the tuple that the optimal approach would calculate to have the best collision percentage.

Hardware Performance on an OpenMote B To show the difference in computational performance of the different collision solving approaches, we tested them on an OpenMote B[4]. The OpenMote B is an open-source hardware board developed to accelerate the development of industrial IoT. It contains an ARM Cortex-M3 micro-controller with 32KB on-chip RAM. We compare the algorithms for different numbers of collisions. The exact algorithm calculates every collision ASN (i.e., here we neglect the additional overhead of adding them to a list and checking if a collision is unique or not) while the sum heuristic only calculates the sum and adds it to a total, as shown in Algorithms 4 and 5. The minimal delay heuristic will just take the proposed tuple, so its duration is less than 1 μs. Table 4.6 shows the results. It is clear that the exact algorithm's execution time increases with the number of collisions, while that of the sum heuristic is constant. As an example, when a node wants to send an ReSF reservation and it considers 20 candidate tuples (i.e., the ReSF reservation buffer is 20) and it has to compare to 50 other reservations with on average 1000 collisions per tuple, this will take a total of 1.1 s (i.e., 1094 us x 20 x 50) and 0.09 s (i.e., 94 us x 20 x 50) for the exact approach and the sum heuristic respectively.

Heuristic Performance in 6TiSCH network To evaluate how the heuristics perform in terms of actual network performance, each heuristic was tested in a 200 node 6TiSCH simulation for all traffic scenarios, with a ReSF reservation buffer of 64. Figure 4.12 shows that the performance of the sum heuristic is nearly identical to the exact one, while the latency values of the minimal and

[4] `http://www.openmote.com/product/openmote-b-bronze-kit/`

Table 4.6: Duration comparison on an OpenMote B board between the exact collision solving algorithm and the heuristic.

Nr. collisions	Duration (μs)		
	Exact Algorithm	Sum	Min. delay
1	94	94	< 1
100	188	94	< 1
500	594	94	< 1
1000	1094	94	< 1

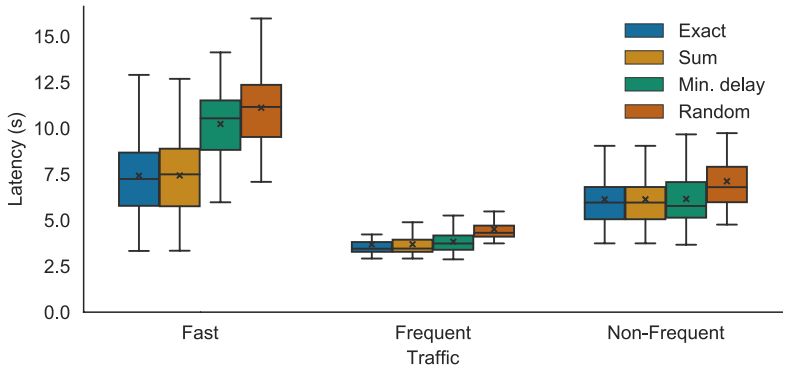


Figure 4.12: Latency comparison between the exact and heuristic ReSF collision solving approaches.

random approach are higher. Especially in the fast traffic scenario, the latency decrease of the sum heuristic is 27.4 % and 33.1 % better than those of the minimal delay and random approach respectively. Because of this significant performance increase, in the remainder of this evaluation we use the sum heuristic. However, when there are a lot of tuples to compare to and the computational performance of the sum heuristic would be a problem, one can always switch to the minimal delay heuristic, which, especially in the frequent and non-frequent scenarios, performs well.

4.6.2.3 Collision Avoidance

To evaluate the collision avoidance (CA) feature, we compare packet loss and latency in a 200 node network. For the new CA feature, we experimentally set the number of slots reserved for the housekeeping cells to 20. Table 4.7 shows the results for ReSF without any CA, with the old CA and the new proposed CA. We observe that for the highly saturated traffic case, i.e., the fast scenario, there are noteworthy improvements of 13 % and 19.3 % in packet loss and the latency decreases with 6.5 s and 13.1 s, compared to ReSF without CA and

Table 4.7: Packet loss and latency for ReSF without, with the old CA and with the new CA feature in a 200 node network.

Traffic	**No CA Loss (%)**	**Old CA Loss (%)**	**New CA Loss (%)**
Fast	40.8	47.1	27.8
Frequent	0.1	0.4	0.2
Non-frequent	0.3	0.3	0.4
	Latency (s)	**Latency (s)**	**Latency (s)**
Fast	15.4	22	8.9
Frequent	2.7	3.1	3.6
Non-frequent	4.7	4.7	5.8

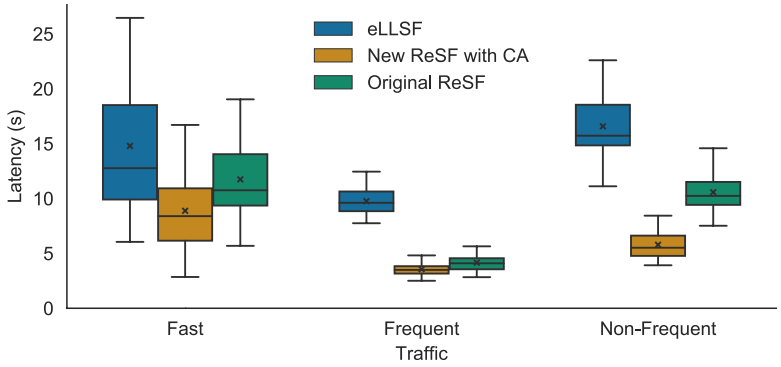


Figure 4.13: Comparison of eLLSF, the original ReSF and new ReSF in a 200 node network with recurrent traffic.

with the old CA respectively. However, for the other scenarios with frequent and non-frequent traffic, we observe that preventing ReSF of using the reserved housekeeping slots actually has a negative effect on the latency. This is due to the reduced amount of contention, and therefore collisions, making the reserved housekeeping slots less useful as they actually prevent ReSF from daisy-chaining all resources from source to root.

4.6.2.4 Recurrent Traffic

In this section we evaluate the improved ReSF, i.e., with the sum heuristic, the new CA feature and the new housekeeping adaptation algorithm, to eLLSF and the original ReSF, i.e., with the periodic housekeeping moment and the old CA, in the different recurrent traffic scenarios. Figure 4.13 shows the results with only recurrent traffic.

In all scenarios, the decrease in latency between eLLSF and the improved ReSF is considerable, with respectively 40 %, 63.5 % and 65.1 %. Moreover, the

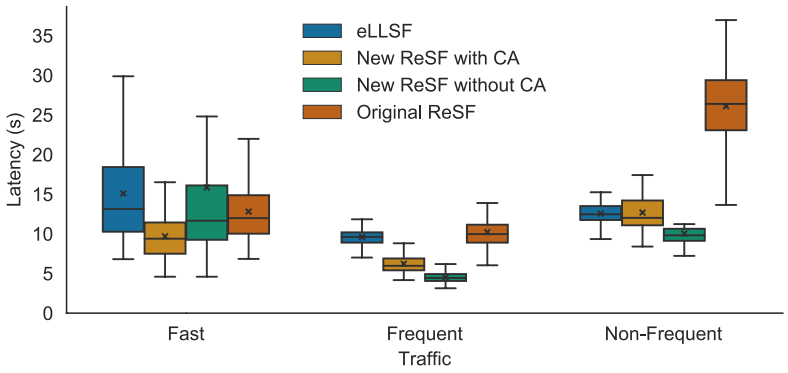


Figure 4.14: Comparison of eLLSF, the original ReSF, the new ReSF with and without CA in a 200 node network with recurrent and sporadic traffic.

PDR of the improved ReSF is 72 % in the highly saturated fast traffic scenario, while that of eLLSF is only 66.7 % (not depicted in graphs). In the frequent and non-frequent scenarios the PDR of ReSF is higher than 99.5 %, while the PDR of eLLSF is 98.6 % and 96.6 % respectively. The improved ReSF also outperforms the original ReSF, especially in the non-frequent scenario with a latency decrease of 45.3 %. The PDR improves from 97 % to 99.5 %. These results show the effectiveness of the improved ReSF.

4.6.2.5 Recurrent and Sporadic Traffic

In this section we compare the new ReSF to eLLSF and the original ReSF, and also show the performance of new ReSF without the CA feature of only allocating eLLSF cells from a pre-defined set. In addition to the recurrent traffic, every node also sends sporadic traffic for which the node randomly waits approximately 30 s, 60 s or 90 s before sending a new sporadic packet. Figure 4.14 shows the results.

In all scenarios, an improved version of ReSF (with or without CA) outperforms the original ReSF in terms of latency up to 61.5 % in the non-frequent scenario. We also observed that in terms of PDR the original ReSF is outperformed by the newer versions, with an increase from 57.1 to 70.8 % in the fast scenario. The traffic adaptation algorithm of the original ReSF seems too responsive to the sporadic traffic and floods the network with resource management signaling (especially in the frequent and non-frequent scenarios), which hinders the data propagation. When comparing the improved ReSF to eLLSF, we observed that the ReSF version with CA performs better in the saturated fast traffic scenario, while ReSF without CA performs better in the other scenarios with a maximum decrease in latency of 53.2 %. While in the saturated scenario the reserved cells for housekeeping help ReSF to avoid schedule collisions, in other scenarios these reserved slots limit ReSF from sending the data with minimal

delay.

4.7 Conclusion

In this chapter, we explicitly focused on minimizing the latency of recurrent traffic in WSNs. First, we stated the problem of minimal-latency scheduling of recurrent transmissions formally, using an ILP. Second, we presented ReSF, a distributed TSCH scheduling function, specifically designed for IoT applications with recurrent traffic, such as sensor measurement applications. ReSF builds a minimal-latency path from source to root and activates the recurrent cells on this path only when traffic is expected, and deactivates them immediately afterwards. Finally, we also presented improvements over the original ReSF in terms of fast collision solving, improved collision avoidance and better support for sporadic traffic. We have conducted numerous experiments using the 6TiSCH simulator, comparing ReSF both to SF0 and eLLSF, and also tested the computational performance of the different collision solving approaches on the OpenMote hardware. The results show significant performance improvements. When considering 100 nodes and each node having a static traffic pattern, ReSF improves up to 80% in terms of latency compared to eLLSF while only having a minimal impact on the battery charge drawn of at most 6.3%. We have also experimentally shown that ReSF can handle a per-second traffic rate change probability up to 20 % when considering 25 nodes and between 1.5 % and 2 % in a topology of 100 nodes. Traffic rate changes in most real-world sensor applications are typically much less dynamic. We conclude that ReSF is well-equipped to maintain a minimal delay in both static and dynamic recurrent traffic rate scenarios.

Chapter 5

Analysing Slot Bonding for Adaptive Physical Layers in TSCH

The content of this chapter is partially based on:

- Glenn Daneels, Carmen Delgado, Steven Latré, and Jeroen Famaey (2020, June). *Towards Slot Bonding for Adaptive MCS in IEEE 802.15.4e TSCH Networks*. In IEEE International Conference on Communications (ICC). IEEE.
- Glenn Daneels, Carmen Delgado, Robbe Elsas, Eli De Poorter, Steven Latré, Chris Blondia, and Jeroen Famaey (2021). *Slot Bonding for Adaptive Modulations in IEEE 802.15.4e TSCH Networks*. IEEE Internet of Things Journal. [Impact Factor: 9.936]

5.1 Introduction

Industry 4.0 and its numerous applications, including time-critical remote control of actuators to continuous monitoring of machinery, pose hard challenges for wireless connectivity [103]. The stringent industrial requirements are often defined by a trade-off between low delay, high reliability and low-power operation. As explained in Chapter 2, the IEEE 802.15.4 standard, first released in 2003, was introduced to tackle the challenges of such applications [27]. It defines the PHY and MAC layers for low-power wireless networks in the sub-GHz and 2.4 GHz frequency band. At the MAC layer, among others, the TSCH mode was proposed to bring reliability and low-power operation in challenging wireless

environments [12]. At the PHY layer, the IEEE 802.15.4g-2012 PHY amendment introduced, among others, three new sets of so-called SUN modulation families, being Frequency Shift Keying (FSK), Offset-Quadrature Phase Shift Keying (O-QPSK) and Orthogonal Frequency Division Multiplexing (OFDM), different data rates and two frequency bands (sub-GHz and 2.4 GHz) [32]. This results in a variety of PHYs that have their own additional characteristics in terms of communication range, bandwidth, energy consumption and reliability. As mentioned in Chapter 1, we define a PHY (layer) as a radio configuration, consisting of a modulation, a data rate and a frequency band.

For many years, TSCH has proven its efficiency in various low-power wireless mesh scenarios. However, traditionally all nodes in the network use the same IEEE 802.15.4 PHY and are thus limited by the characteristics of the chosen PHY. Additionally, the reliability of that one PHY can be heavily affected by the often challenging wireless environments. For example, a network deployed at large industrial sites can experience a lot of external interference from other wireless technologies and its signal is often disrupted by the many blocking metal constructions. Addressing this issue, it is shown that enabling adaptive modulation switching can improve reliability in industrial environments with location-dependent heterogeneous propagation behaviour [18]. Therefore, a TSCH network could significantly improve from being able to use different PHYs for different network links simultaneously and adapt the PHY layer of each link to the local propagation characteristics and the application's requirements.

In this chapter, we explore the exploitation of multiple PHYs in a single TSCH network by introducing the concepts of *slot* and *channel bonding* which allows the creation of different-sized bonded slots with a duration and number of used channels adapted to the data rate and bandwidth of each chosen PHY. Specifically, we focus on the slot bonding technique that thus facilitates the use of multiple PHYs with different data rates within a single TSCH network in a resource-efficient manner. In contrast, traditional TSCH relies on fixed-duration slots, large enough to send a packet of any size given the fixed data rate, which results in wasted airtime if different PHYs with different data rates are used simultaneously.

The contributions of this chapter are three-fold. First, we give a formal description of the TSCH slot bonding problem that allows us to analyse the proposed slot bonding technique, including interference avoidance and parent selection to optimize the overall network PDR while keeping the radio on time minimized. This problem has a high computational complexity when solving it for large network scenarios. Additionally, to allow a node to allocate an arbitrary number of slots to its parent, resulting in flexible and realistic TSCH schedule allocations, a Markov chain model was introduced. To be able to implement and solve the formulated problem with the Markov Chain, we used a Genetic Algorithm (GA) to find the best heuristic solution for the slot bonding problem by selecting the most appropriate parent, PHY, and bonded time slots for each node. Finally, we provide insights into preferred parent selection and PHY configurations by using this heuristic approach during extensive TSCH

simulation experiments, in which the scalability advantage of slot bonding over longer fixed-duration slots in terms of network-wide PDR is also shown.

The remainder of this chapter is structured as follows. First, we introduce the related work on using multiple PHYs simultaneously and other techniques to combat external interference and multi-path fading effects in TSCH in Section 5.2. Second, in Section 5.3, we explain the concept of slot and channel bonding. In Section 5.4, we formally define the slot bonding problem, while in Section 5.5, we introduce our genetic algorithm approach to solve the slot bonding problem heuristically. Afterwards, the slot bonding approach is evaluated in Section 5.6. Finally, Section 5.7 presents the conclusions of this chapter.

5.2 Related Work

Due to the popularity gain of low-power wireless networks and the first capable IEEE 802.15.4g transceivers appearing on the market, there is an increased research effort on the available PHYs and their usage in different real-world scenarios. There have been experimental evaluations in several outdoor scenarios [17, 104], testing their suitability for smart building applications [105] and the interference robustness of the different modulations, i.e., more specifically the impact on the resulting PDR and depending on the length of the transmitted packet [106]. As low-power wireless networks are being used more and more in industrial settings, there is also an increased interest, specifically in the reliability performance of SUN modulations, in such environments. Tuset-Peiró *et al.* presented a large real-world data set consisting of data of 11 nodes being deployed in an industrial warehouse transmitting packets with 3 SUN modulations [18]. The results show large variability and poor PDRs, due to multi-path propagation and external interference effects, making them not suitable for stringent industrial requirements. As such, the authors and other related studies propose packet replication and the use of multiple IEEE 802.15.4 modulations for different packets, i.e., modulation diversity, to increase network reliability [33, 107]. Similarly, J. Muñoz *et al.* hint to the use of a different OFDM Modulation and Coding Scheme (MCS) on a per-link basis [105]. M. Rady *et al.* adjusted the OpenWSN firmware to support O-QPSK (2.4 GHz at 250 kbps), FSK (868 MHz option 1 at 50 kbps) and OFDM (868 MHz option 1 MCS3 at 800 kbps). The different PHYs were tested on a 42 nodes testbed in an office environment. The authors conclude that no PHY outperforms the other PHYs for all other metrics, and therefore the combination of different PHYs should be considered in a 6TiSCH architecture. [19].

Besides progress at the PHY layer in the IEEE 802.15.4 standard, a lot of research effort also went into increasing network reliability by the introduction of the IEEE 802.15.4e TSCH mode in 2015. T. Watteyne *et al.* showed the importance of the channel hopping feature present in TSCH, to mitigate multi-path fading effects and increase reliability of low-power wireless networks [26]. Since its introduction, numerous centralized, distributed and autonomous TSCH

scheduling solutions have been proposed, as discussed in Section 2.3.3. While these scheduling functions aim at finding the optimal trade-off between throughput, latency and energy consumption for the given application, their performance still heavily depends on environmental factors, such as external interference and multi-path fading effects. To cope with those effects, other techniques are proposed to improve reliability, such as channel blacklisting [108] and more recently the PAREO methodology [109] on top of TSCH, which is an application of the IETF Reliable and Available Wireless (RAW) layer-3 approach [110] of Automatic Repeat reQuest (ARQ), Replication and Elimination (RE) and Overhearing (OH).

While all of those techniques offer valid solutions to deal with various link conditions, they are still limited by the employed PHY layer. That is why we introduce the concept of TSCH slot and channel bonding (and provide an in-depth theoretical analysis of slot bonding) to allow PHY diversity in a resource-efficient manner to improve the reliability in one single network. M. Brachmann *et al.* proposed a similar approach of multiple PHYs by presenting, among others, two multi-PHY designs: (i) a design where slower PHYs are scheduled to have logical slots spanning multiple real slots, and (ii) a design where the slot size is based on the slowest PHY [111]. Their multi-template design is similar to our slot bonding approach. Van Leemput *et al.* also use multiple PHYs, but focus on throughput improvement. They define two alternative time slot structures allowing multiple packets transmissions to increase the throughput for higher data rate PHYs while maintaining a fixed slot duration [34]. In this chapter, we analyse the slot bonding technique in detail by presenting a realistic model that includes interference avoidance and also builds an optimal multi-PHY network topology tree.

5.3 TSCH Slot and Channel Bonding

TSCH slot and channel bonding allows a single TSCH network to exploit the characteristics of different available PHYs in the IEEE 802.15.4 standard. By choosing the most appropriate PHY layer, the performance of each link can be tailored to the local environmental conditions and application's requirements to maximize the overall network performance. When using different PHY layers in the same TSCH network, both the time and channel management of a TSCH schedule are affected and therefore we introduce both the concepts of *slot* and *channel bonding*.

As different PHY layers are computationally different and have different data rates, their processing and transmission time directly impacts the optimal TSCH slot length, which should encompass the time it takes to send a data packet of the maximum allowed size (e.g., 127 bytes in TSCH) and receive an ACK. At the same time, different PHY layers can require different bandwidths, which influences both the total number of available frequency channels in a TSCH schedule and the effective number of channels a transmission would

need to bond together to send the data. For both the time slot and channel management when supporting multiple PHY layers, we propose the concepts of slot and channel bonding respectively, which can be applied combined or used separately. The former bonds multiple *regular* time slots into a bonded slot which has the necessary length to transmit or receive data given the data rate and computational time of the selected PHY. The latter bonds multiple available frequency channels to allocate the necessary bandwidth to accommodate the chosen PHY layer. In contrast to a more greedy approach in which the TSCH schedule is configured with cells that are long enough in time and occupy a bandwidth wide enough to be compatible with all supported PHYs at once, the proposed slot and channel bonding technique tailors each bonded slot to the requirements of the specific PHY. It thus has the advantage of limiting the waste of airtime and bandwidth resources.

Figure 5.1 shows an example of a TSCH schedule that applies both slot and channel bonding in a slotframe of 11 slots of 10 ms each. The PHY *PHY1* applied for the link from node X to the root requires a 30 ms bonded time slot, thus bonding 3 regular time slots, and the bonding of two 200 kHz frequency channels to a 400 kHz bandwidth. *PHY2* used for the link from node Y to the root requires the bonding of two 10 ms time slots to a 20 ms time slot in order to be able to transmit a packet and receive an ACK. The minimal shared cell to bootstrap the network, indicated by the orange bonded slot in the figure, is configured with the same PHY1 PHY layer. Usually, the bootstrap cell should use the most robust PHY, to ensure it can be used by all nodes independent of the link quality.

Allocating a bonded slot would be very similar to the traditional 6P transaction to allocate a TSCH cell [39]. In addition to the conventional 2-step or 3-step 6P negotiation between two nodes, the nodes also agree on the employed PHY which can be specified in the reserved bits in the *CellOptions* bitmap when adding the bonded slot. When a node wants to change the PHY layer of a particular bonded slot, it can issue a 6P *RELOCATE* request to its neighbor that does not relocate the cell but mentions another PHY index in the *CellOptions* bitmap.

Also, the coexistence of slot bonding nodes and normal TSCH nodes (i.e., nodes that do not support slot bonding) in one network is possible when 2 requirements are fulfilled: (a) the regular TSCH timings are the same (i.e., the regular time slot length and all its sub-states, such as, the time to wait until transmission of the data and reception of the ACK) (b) the default PHY to join the network should also be used by the slot bonding nodes to send EB messages and let other nodes join.

In this work, we will only focus on slot bonding, by only considering PHYs that require the same bandwidth. In case PHYs with different bandwidth requirements would be considered, this work can be extended to also include channel bonding.

While supporting multiple PHYs, we make use of the flexibility of OFDM as it provides various data rates for a single bandwidth. The OFDM modulation

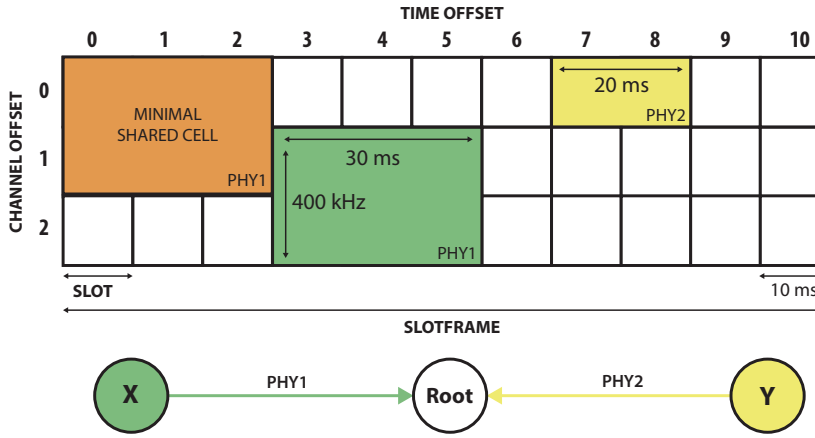


Figure 5.1: A TSCH schedule with regular slots of 10 ms length and 200 kHz bandwidth. For accommodating PHY1 on the link from node X to the root 3 slots and 2 channels are bonded to a 30 ms slot of 400 kHz wide while PHY PHY2 used by node Y only requires 2 10 ms slots bonded together.

that was introduced in the IEEE 802.15.4g-2012 amendment offers 31 PHY variations, which are subdivided in 4 different *options* that determine how many sub-carriers are grouped together in order to form an OFDM channel [106]. Each option has its own specific bandwidth requirement and also a set of different MCS values that determine how each sub-carrier is modulated (and whether frequency repetition and Forward Error Correction (FEC) are applied). As explained earlier, the different characteristics of the MCSs lead to different time slot lengths which translates to bonded slots with different numbers of regular slots for each MCS, as shown in Table 5.1 for regular slots of lengths 10 ms and 40 ms. To calculate the necessary number of regular slots per bonded slot, we take into account the radio on time for a specific MCS and assume 5 ms of CPU processing time and 3 ms for reconfiguration of the radio per bonded slot, in line with the values reported by respectively Daneels *et al.* and Brachmann *et al.* [20, 111]. The radio on time for each MCS was calculated for a 127 bytes data packet and a 27 bytes ACK (i.e., the default size in TSCH to determine the slot duration), including the duration for sending the OFDM Synchronization Header (SHR) and PHY Header (PHR) in the lowest MCS of the chosen OFDM option, i.e., MCS2 of option 4 in this case [32, 112].

Because of the focus on OFDM, throughout the remainder of this chapter, the term PHY may be interchanged with *MCS*, both referring to the used radio configuration.

Table 5.1: MCS2, MCS3 and MCS4 of OFDM option 4 which all require 200 kHz bandwidth.

Mode	Modulation, Code rate, Freq. rep.	Datarate (kbps)	Radio on (ms)	# Regular slots per bonded slot	
				10 ms	40 ms
MCS2	O-QPSK, 1/2, 2x	50	27.84	4	1
MCS3	O-QPSK, 1/2, 0	100	15.48	3	1
MCS4	O-QPSK, 3/4, 0	150	11.28	2	1

5.4 Problem Formulation

This section formally describes the IEEE 802.15.4e TSCH slot bonding problem in which the expected number of delivered packets at the root is maximized, while the total radio on time is minimized. The inputs to this problem are the physical location of the nodes, the possible MCSs every node can employ to communicate with other nodes and the reliability of the link when using a specific MCS. As is often the case within WSN applications (e.g., periodical sensor monitoring at industrial environments), it is assumed that data packets are generated at a fixed rate. We consider interference avoidance, parent selection, MCS selection, and slot assignment.

First, we describe the expected delivered packets calculation and the radio on time calculation. Afterwards, we apply these calculations and propose the new slot bonding problem formulation. The symbols used in this section, are listed in Table 5.2.

5.4.1 Delivered Packets Calculation

In this section we describe how for all resource allocations made in a single TSCH slotframe, the expected total number of delivered packets at the root of the tree topology, originating from all nodes in the network, is calculated. First, we introduce a Markov chain model that helps us to calculate the number of packets a node can successfully transmit to its parent in a slotframe on average. Afterwards, we show how the result of the proposed stochastic process is used to calculate the expected number of delivered packets to the root of the entire network.

Table 5.2: All used symbols and their respective meaning.

Symbol	Meaning
B	Matrix containing b_{v_i,v_j} probabilities at the (i, j)-th entry that represent the chain starting in transient state v_i and being absorbed in state v_j
B^l	B matrix containing $b^l_{v_i,v_j}$ probabilities for a link reliability $l \in [0,1]$
C_n	Set of all children of node n, i.e., $C_n \subset N$
D_n	Set of all descendants of node n, i.e., $D_n \subset N$
F	Set of available channel offsets, i.e., $\{f_0, f_1, ..., f_{max}\}$
I_n	Set of nodes that interfere with receiving node n
J_x	Set of absorbing states in which x packets arrive at the parent, i.e., $J_x \subset V$
M	Set of available MCSs, i.e., $\{m_0, m_1, ..., m_{max}\}$
N	Set of all network nodes, i.e., $\{n_0, n_1, ..., n_{max}\}$
N_0	N without the root, i.e., $\{n_1, ..., n_{max}\}$
P	Markov process transition probability matrix
Q	Max. queue size of the node
R	Probability matrix going from transient to absorbing state
S	Set of possible number of consecutive regular slots bonded together, i.e., $S = \{1, ..., \lvert T \rvert\}$
T	Set of slots in slotframe, i.e., $\{t_0, t_1, ..., t_{max}\}$
V	Set of all Markov chain states
W	Probability matrix going from transient to transient state
a	Number of remaining slots in the slotframe which can all be used by the node for transmission
a_n	Number of TX slots allocated by node n
b_{v_i,v_j}	The probability that the chain will be absorbed in the absorbing state v_j if it starts in the transient state v_i, is the (i, j)-th entry of the matrix B
e_n	Approx. of expected number of packets in queue of node n
g	Number of packets generated per slotframe per node
$h_{C_n,q}$	Probability of q packets arriving at n from its children C_n
l	Link reliability $\in [0,1]$ between node and parent, depends on used MCS
l_n	Link reliability l for a node n to its parent
n_0	Root node of the network, i.e., $n_0 \in N$
o_{state}	Radio on time for a specific TSCH state
p_n	Parent of node n, i.e., $p_n \in N$
q	Number of packets waiting for transmission
q_e	Number of packets left in queue after the previous slotframe
r	Remaining number of retransmissions for current packet
r_{max}	Max. retransmissions before being discarded
s_m	Number of consecutive regular slots necessary for 1 bonded slot for MCS m, i.e., $s_m \in S$
u_n	Avg. number of TX slots that node n is expected to use during one slotframe
v_i	State of the Markov Chain, i.e., $v_i \in V$
x	Number of packets successfully arrived at parent
$y_{x,q,a,l}$	Probability of x packets arriving at the parent at end of slotframe, while at the start of the frame the state was $(q, a, r_{max}, 0)$, over a link with reliability l
z^n_x	Probability that x packets arrive from a node to its parent
$\sigma_{t,f,s,n}$	Binary decision variable that equals 1 when node $n \in N_0$ transmits in a bonded slot, using $s \in S$ consecutive slots allocated at time offsets $\{t, t+1, \ldots, t+(s-1)\} \subset T$ and channel offset $f \in F$ to its parent $p \in N$, else 0
$\gamma_{p,m,n}$	Binary decision variable that equals 1 when node $n \in N_0$ selects $p \in N$ as its parent and $m \in M$ as the MCS to transmit to the parent, else 0

5.4.1.1 Markov Chain Model

In order to calculate the number of packets a node can successfully transmit to its parent, we propose a Markov chain model of a transmitting node in a TSCH network. This model takes as inputs the TSCH schedule slotframe with its set of slots $T = \{t_0, t_1, ..., t_{max}\}$, the number of allocated transmission slots in the slotframe, the maximum queue size Q of the node, the number of packets in the queue at the start of the slotframe, the chosen MCS between the node and its parent that determines the reliability $l \in [0, 1]$ of the link to its parent and the maximum number of retransmissions r_{max} before the node discards the packet. Each node in the network generates a fixed number of g packets per slotframe which are assumed to be generated at the beginning of the slotframe. As for the outputs, the probabilities determined by the model allow us to calculate the probability that the node successfully sent x packets to its parent during a slotframe. More specifically, x denotes all packets that were present in the queue at the beginning of the slotframe and that are sent successfully before the end of the frame (hence leaving an empty queue). Or, x denotes that part of the packets present at the start of the frame in the queue that are sent successfully to the parent when the end of the frame is reached (i.e., leaving a non-empty queue).

Consider the stochastic process defined by the four variables (q, a, r, x), which are considered at the start of each slot t_i of a frame, $t_i \in T$, and at the end of slot t_{max}, denoted by t_{max+1}. q is the number of packets waiting for transmission with $0 \leq q \leq min(Q, q_e + g)$, with q_e being the packets left in the queue at the end of the previous slotframe. a is the number of remaining slots in the slotframe and the node can transmit in all of them, with $a = |T| - i$ at slot t_i, $0 \leq a \leq |T|$. r is the number of remaining transmissions that are allowed for the current packet to be transmitted, $1 \leq r \leq r_{max}$. When a packet is transmitted for the first time, then $r = r_{max}$. If after a transmission that started with $r = 1$ the transmission was unsuccessful, the packet is lost. x is the number of successful packet transmissions in the previous slots of the current frame, $0 \leq x \leq |T| - a$.

$PS(q, a, r, x)$ denotes the stochastic process and V is the set of all Markov Chain states. There are two classes of absorbing states, i.e., $(0, a, r, x)$ and $(q, 0, r, x)$, representing respectively that the process can not transition to any other state when there are no more packets left to transmit or when there are no allocated slots in the slotframe left to transmit in. In what follows we describe the possible transitions in the transition matrix P of the process PS from slot t_i, so starting from state (q, a, r, x) to slot t_{i+1}, for $0 \leq i \leq |T| - 1$, as shown in Figure 5.2:

- **if r > 1, a > 1 and q > 1**: the current packet is transmitted. With probability l the packet reaches the parent and the process goes to state $(q-1, a-1, r_{max}, x+1)$. With probability $1-l$ the packet is lost and the process retries transmitting the packet in state $(q, a-1, r-1, x)$.
- **if r = 1, a > 1 and q > 1**: the current packet is retransmitted one

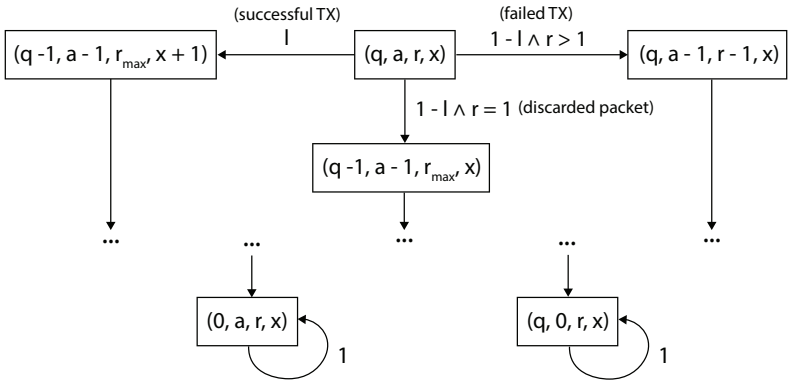


Figure 5.2: Diagram of the Markov chain.

more time. With probability l the packet reaches the parent, and the process goes to state $(q-1, a-1, r_{max}, x+1)$. With probability $1-l$ the transmission fails and the packet is discarded. The process proceeds to the next packet, in state $(q-1, a-1, r_{max}, x)$.

- **if a = 0**: there are no allocated slots left for the node to transmit in, the process stays in the same absorbing state with probability 1.
- **if q = 0**: there are no waiting packets left, the process stays in the same absorbing state with probability 1.

We renumber the states of the transition matrix P in such a way that the transient states come first and the absorbing states last, resulting in following canonical form:

$$P = \begin{pmatrix} W & R \\ 0 & I \end{pmatrix} \tag{5.1}$$

where I is the identity matrix and W and R are non-zero matrices containing the probabilities of going from a transient state to another transient state and the probabilities going from a transient to an absorbing state, respectively.

According to the Theorem 11.6 of [113], the probability b_{v_i,v_j} that the chain will be absorbed in the absorbing state v_j if it starts in the transient state v_i, is the (i, j)-th entry of the matrix B, given $B = (I - W)^{-1} \cdot R$. Note that according to Theorem 11.4 of [113], the inverse matrix $(I - W)^{-1}$ exists.

These probabilities b_{v_i,v_j} allow us to evaluate the probability that, when the system starts in a transient state, the system is absorbed in a state $(0, a, r, x)$, i.e., the queue is emptied before the end of the frame and there have been x packets transmitted, or the system is absorbed in a state $(q, 0, r, x)$, i.e. the end of the frame is reached and there have been x packets transmitted. These probabilities are used in Section 5.4.1.2, to calculate the average number of delivered packets at the root, for all nodes in the network, in one slotframe.

5.4.1.2 Number Of Delivered Packets

We describe how the expected number of packets arriving at the root in a single slotframe is calculated, using the probabilities determined by the Markov chain defined in Section 5.4.1.1.

We define $y_{x,q,a,l}$ as the probability that at the end of the slotframe x packets have arrived at the parent when at the start of the frame the state was $(q, a, r_{max}, 0)$, over a link to the parent with reliability $l \in [0, 1]$:

$$y_{x,q,a,l} = \sum_{v_i \in J_x} b^l_{v_{start}, v_i} \tag{5.2}$$

where $v_{start} = (q, a, r_{max}, 0)$ is the start transient state and J_x the set of absorbing states where x packets successfully arrived at the parent. The probability $b^l_{v_i,v_j} \in [0, 1]$ which is the (i, j)-th entry of the B^l matrix is calculated with the Markov chain model defined in Section 5.4.1.1, for a link reliability $l \in [0, 1]$ from a node to its parent.

Subsequently, the probability z^n_x that x packets successfully arrive from node n to its parent is represented by:

$$z^n_x = \sum_{q=0}^{|D_n| \cdot g} h_{C_n,q} \cdot y_{x,min(Q,q+g),a_n,l_n} \tag{5.3}$$

where D_n is the set of all descendants of node n, a_n the number of slots allocated for transmission by n, l_n the reliability of the link to the parent of n, of which the value is dependent on propagation characteristics of the link between the node to its parent and the MCS used by that node, and $h_{C_n,q}$ is the probability that in total q packets arrived from the children C_n at node n. In the case that node n is a leaf node, $h_{C_n,q}$ is considered 1. Note that $x \leq |D_n| \cdot g + g$, as in the other cases z^n_x equals 0 because a node can not transmit more packets than it received from its children, i.e., $|D_n| \cdot g$, and the packets it generates itself, i.e., g packets. $min(Q, q + g)$ is the maximum number of packets that can arrive at the parent of n from node n, with the maximum queue size being Q and $q + g$ being the number of packets q from the children of n aggregated with the g packets generated at node n. For leaf nodes, $z^n_x = y_{x,g,a_n,l_n}$ with $x \leq g$, as only the generated packets can be transmitted.

$h_{C_n,q}$ is defined as the sum of all permutations of how q packets can arrive at node n from its children. For each permutation, we multiply the probabilities $z^{c_1}_{x_1}, z^{c_2}_{x_2}, \ldots, z^{c_{|C_n|}}_{x_{|C_n|}}$, where $C_n = \{c_1, c_2, \ldots, c_{|C_n|}\}$ and $q = \sum_{i=1}^{|C_n|} x_i$. For example, let us assume that node n has two children $C_n = \{c_1, c_2\}$, then $h_{C_n,q}$ for $q = 2$ is calculated as follows:

$$h_{C_n,2} = z^{c_1}_2 \cdot z^{c_2}_0 + z^{c_1}_0 \cdot z^{c_2}_2 + z^{c_1}_1 \cdot z^{c_2}_1 \tag{5.4}$$

which represents the probability that exactly 2 packets arrive from the children of node n.

The above calculations are performed in an iterative fashion for all nodes in the topology tree, starting from the leaf nodes, continuing with their parents, and so on, until the root is reached. At the root, we can calculate the expected total number of delivered packets as follows:

$$\sum_{q=0}^{|D_{root}|\cdot g} h_{C_{root},q} \cdot q \tag{5.5}$$

5.4.2 Radio On Time Calculation

Here we calculate the expected radio on time of the entire network during a single TSCH slotframe. To do so, we calculate the radio on time of each node separately and aggregate those to one total value. The expected number of packets a node will have in its queue at the beginning of the slotframe and the number of slots that will be used by the node and its parent to effectively transmit and receive these packets, is calculated. Using those values and the different radio on values for the different states a TSCH node can be in, the radio on time for the entire network can be approximated.

To calculate the radio on time for the packets transmitted from node n to its parent, we first approximate the expected number of packets that there will be in the queue of the node, e_n, defined as follows:

$$e_n = round\left(g + \sum_{q=0}^{|D_n|\cdot g} h_{C_n,q} \cdot q\right) \tag{5.6}$$

with g packets being generated at the node, $|D_n| \cdot g$ being the maximal number of packets that can arrive from the children of node n and $h_{C_n,q}$ as defined in Section 5.4.1.2.

In every slotframe, a node n has a_n slots allocated towards its parent. Subsequently, we calculate the average number of slots that node n is expected to use, u_n, out of the a_n allocated slots, to transmit its queued packets e_n to its parent:

$$u_n = a_n - \sum_{v_i \in J_x} \frac{b^l_{v_{start},v_i}}{\sum_{v_i \in J_x} b^l_{v_{start},v_i}} \cdot a' \tag{5.7}$$

where a' is the number of slots not used by node n for every state $v_j = (q', a', r'_{max}, x)$, with q' being the packets left in the queue and r'_{max} being the number of unused retransmissions. $v_i \in J_x$ represents all the absorbing states with x arrived packets at the parent of node n, starting from state $v_{start} = (q, a, r_{max}, 0)$ over a link to its parent with reliability l.

Finally, we can calculate the radio on time for the communication between a node n and its parent as follows:

$$\sum_{x=0}^{min(a_n,e_n)} y_{x,e_n,a_n,l} \cdot (x \cdot (o_{txDataRxAck} + o_{rxDataTxAck}) \quad + \\ (u_n - x) \cdot (1-l) \cdot (o_{txData} + o_{rxIdle}) \quad + \\ (u_n - x) \cdot l \cdot (o_{txDataRxNack} + o_{rxDataTxNack}) + (a_n - u_n) \cdot o_{rxIdle}) \quad (5.8)$$

with $y_{x,e_n,a_n,l}$ being the probability that at the end of the slotframe x packets have arrived at the parent, as described in Equation 5.2. The o_{state} values are the different radio on time values for the different states a TSCH node can be in, using the MCS chosen by node n. These different states are described in Section 3.3.1 of Chapter 3 and the radio on time value of each state is calculated similarly to the values in Table 5.1 for each MCS. The radio on time of every node in the network is calculated and summed, resulting in the radio on time of the entire network.

5.4.3 TSCH Slot Bonding Problem Formulation

Using the expected number of delivered packets and radio on time calculations as described in the previous sections, we propose the slot bonding problem formulation that maximizes the number of delivered packets while keeping the radio on time to a minimum. It is important to note that the problem formulation assigns slots in a single TSCH slotframe and this schedule is assumed to be repeated every slotframe.

5.4.3.1 Input Variables

The network consists of a set of nodes $N = \{n_0, n_1, ..., n_{max}\}$, of which node n_0 is the root, i.e., the sink node. $N_0 \subset N$ represents the set of nodes without the root n_0. $M = \{m_0, m_1, ..., m_{max}\}$ is the set of possible MCSs for each $n \in N_0$. The set of slots in a slotframe are denoted by $T = \{t_0, t_1, ..., t_{max}\}$ and the set of channel offsets at which a node can transmit are represented by $F = \{f_0, f_1, ..., f_{max}\}$. Every node n has a set $I_n \subset N_0$ which contains the set of nodes that interfere with the receiving node n, i.e., all the nodes that have a Received Signal Strength Indicator (RSSI) value at n higher than the channel noise floor, added to the noise figure of the receiver device. Each MCS $m \in M$, requires a bonded slot consisting of $s_m \in S$ regular slots, with $S = \{1, ..., |T|\}$, to transmit a 127 bytes packet and receiving an ACK.

5.4.3.2 Decision Variables

The first binary decision variable $\sigma_{t,f,s,n}$ represents the specific assigned bonded slots a node should select in the TSCH schedule. $\sigma_{t,f,s,n}$ is 1 when node $n \in N_0$ transmits in a bonded slot, using $s \in S$ consecutive regular slots allocated at

time offsets $\{t, t+1, \ldots, t+(s-1)\} \subset T$ and channel offset $f \in F$ in the TSCH schedule matrix to its parent $p \in N$, else it is 0.

The second binary decision variable $\gamma_{p,m,n}$ is 1 when node $n \in N_0$ selects $p \in N$ as its parent and $m \in M$ as the MCS to transmit to the parent, else it is 0.

5.4.3.3 Objective

The objective is to maximize the expected number of delivered packets at the root in one slotframe, while keeping the radio on time to a minimum.

Therefore, the goal is to maximize Equation 5.5 that calculates the delivered packets at the root, given a TSCH network determined by the values assigned to the $\sigma_{t,f,s,n}$ and $\gamma_{p,m,n}$ variables. Simultaneously, for those solutions that provide the same number of delivered packets, the solution with the smallest radio on time of the network, defined as the sum of all radio on time values of all nodes in the network, should be preferred. To calculate the radio on time of the entire network, we need to calculate the radio on time for every node n using the values assigned to the $\sigma_{t,f,s,n}$ and $\gamma_{p,m,n}$ variables and Equation 5.8.

5.4.3.4 Constraints

A node $n \in N_0$ can choose only one parent $p \in N$ and one MCS $m \in M$ to transmit to that parent p:

$$\forall n \in N_0 : \sum_{p \in N} \sum_{m \in M} \gamma_{p,m,n} = 1 \tag{5.9}$$

A node $n \in N_0$ that uses MCS $m \in M$ to its parent $p \in N$, should allocate exactly s_m consecutive slots for a bonded slot transmission:

$$\forall n \in N_0, p \in N, t \in T, f \in F, m \in M, s \in S | s \neq s_m : \gamma_{p,m,n} \cdot \sigma_{t,f,s,n} = 0 \tag{5.10}$$

A bonded slot transmission $\sigma_{t,f,s,n}$ consisting of s consecutive slots should not exceed the slotframe length:

$$\forall n \in N_0, t \in T, f \in F, s \in S : (t+s-1) \cdot \sigma_{t,f,s,n} \leq t_{max} \tag{5.11}$$

A transmission of node $n \in N_0$ to its parent cannot overlap with any other transmission of the same node, nor with any of the transmissions of the nodes that have node n as their parent, as a node can only perform one action (i.e., transmit or receive) during each slot:

$$\forall n \in N_0, t \in T : \left(\sum_{f \in F} \sum_{t' \in [0,t]} \sum_{s \in [t-t'+1, (t_{max}+1)-t']} \sigma_{t',f,s,n} \right) + \left(\sum_{f \in F} \sum_{t' \in [0,t]} \sum_{s \in [t-t'+1, (t_{max}+1)-t']} \sum_{m \in M} \sum_{j \in N_0} \gamma_{n,m,j} \cdot \sigma_{t',f,s,j} \right) \leq 1 \tag{5.12}$$

The transmissions of the children of the root cannot be in the same slot, as the root can only listen to one child at once:

$$\forall t \in T : \sum_{f \in F} \sum_{t' \in [0,t]} \sum_{s \in [t-t'+1,(t_{max}+1)-t']} \sum_{m \in M} \sum_{j \in N_0} \gamma_{n_0,m,j} \cdot \sigma_{t',f,s,j} \leq 1 \quad (5.13)$$

A transmission of node $n \in N_0$ to its parent p cannot overlap with any other transmission of the interferers of the receiving parent node p:

$$\forall n \in N_0, t \in T, f \in F : \left(\sum_{t' \in [0,t]} \sum_{s \in [t-t'+1,(t_{max}+1)-t']} \sigma_{t',f,s,n} \right) \cdot \left(\sum_{t' \in [0,t]} \sum_{s \in [t-t'+1,(t_{max}+1)-t']} \sum_{p \in N} \sum_{m \in M} \sum_{j \in I_p} \gamma_{p,m,n} \cdot \sigma_{t',f,s,j} \right) = 0 \quad (5.14)$$

The output of this slot bonding formulation is a set of MCSs and bonded slot allocations from each node to its chosen parent, represented by the $\sigma_{t,f,s,n}$ and $\gamma_{p,m,n}$ decision variables, in order to maximize the expected number of delivered packets at the root while minimizing the radio on time. However, the proposed slot bonding problem becomes highly computationally complex when solving it for large topologies. Therefore, a GA is proposed in Section 5.5 to solve the problem in a heuristic manner. The slot bonding formulation sub-problem of finding slot allocations for all nodes in a saturated network scenario is equivalent to the scheduling problem formulated by S.C. Ergen *et al.*, which is proven to be NP-complete [114]. Additionally, the sub-problem of building the topology tree by making parent and PHY selections can be seen as equivalent to the construction of the minimum routing cost spanning tree which is NP-hard [115]. As such, solving the slot bonding formulation is not possible in polynomial time.

5.5 A Heuristic Approach

In this section, detailed information is provided on the GA that allows us to implement and solve the slot bonding problem defined in Section 5.4. First, we describe the genetic algorithm and its different operators. Afterwards, the feasibility heuristic that is used to know if a GA solution fits in the TSCH schedule, is explained in detail.

5.5.1 Genetic Algorithm

A GA is a heuristic technique to solve optimization or search problems by simulating natural evolution [116]. It is an iterative process that starts from an initial randomly or heuristically created population. A population consists of so-called individuals that represent candidate solutions of the problem that is being solved.

Each generation (i.e., iteration), the GA applies biologically inspired operators to select appropriate individuals and do cross-over and mutation operations on the individuals in the population, which results in a new offspring population. This offspring combined with the population at the start of the generation will be the population for the next generation in the evolutionary process. To select appropriate individuals for the operators and the next generations, each individual is evaluated and assigned a fitness value during each generation. This value represents its performance measured by the optimization problem's objective function. The goal of a GA is to evolve towards a population that contains the best candidate solutions for the problem at hand.

We applied a GA to find the best candidate solutions for the slot bonding problem defined in Section 5.4. Most importantly, the operators (see Sections 5.5.1.2 and 5.5.1.3) and objective function (see Section 5.5.1.4) of the GA can easily be adjusted to, respectively, respect the validity of the network topology and make the complex delivered packets calculations. Additionally, the choice of a parent, MCS, and the number of allocated TSCH slots can easily be encoded in a GA integer string (see Section 5.5.1.1), similarly to what was argued by M. Ojo *et al.* [117]. Also, as GAs are well-fitted for finding global solutions for problems with large search spaces and the search space of our slot bonding problem is huge, applying a GA for this problem is suitable. For example, for a network with only 5 nodes except the root, so 5 potential parents, 2 potential MCSs and a slotframe length of 9 slots (so 10 possible numbers of allocated slots), the size of the solution space is already $(5 \cdot 2 \cdot 10)^5 = 10^{10}$.

The inputs to the GA are the set of network nodes N, the set of interferers I_n for each node n, the set of different available MCSs M that all result in a specific reliability l for each node-parent pair, the length of the bonded slots for each MCS $s_m \in \mathbb{N}_{>0}$, the radio on time o_{state} for each state a TSCH node can be in and for every MCS m that can be chosen, and the TSCH schedule with its set of slots T and frequency channels F. The output of the GA is a parent selection for each node in the network and MCS and slot allocations for each node to its chosen parent, in order to maximize the expected number of delivered packets at the root while minimizing the radio on time. In this section, we explain the candidate solution representation, the different operators, the fitness function, and provide an exact overview of the GA workflow.

5.5.1.1 Representation

A candidate solution of the slot bonding problem is represented by a GA individual as shown in Figure 5.3. Each node in the individual is represented by 3 so-called *genes*, being an encoder of a characteristic of the node, with each gene value being a natural number. The first gene p_n represents the parent of the node n, m_n represents the MCS that node n applies to communicate with its parent and the last gene a_n is the number of bonded slots it allocates to its parent. The root node does not need representation in the individual as other nodes connect and allocate resources to the root.

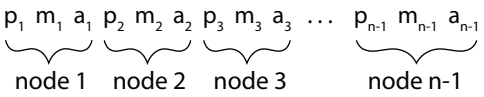


Figure 5.3: GA individual for a topology of n nodes with 3 genes per node. The root is not explicitly represented in the individual.

5.5.1.2 Cross-over Operator

A cross-over operation recombines the genetic information of two individuals into new offspring individuals. A slightly altered version of the traditional two-point cross-over was applied. The two-point cross-over was chosen over the single-point crossover to reduce the positional bias [118]. Traditionally, this technique chooses two random positions in the individuals, after which both parent individuals are cut at those positions and the genetic substring between those positions of the first parent is interchanged with the substring at the same location in the second parent. We applied two changes to that operator implementation. First, the random positions are limited to be multiples of three in order to never separate the three genes of one node. Second, if interchanging the substring between the chosen positions does not result in a new valid topology tree (i.e., there is a path from each node to the root and there is no path loop), we decrement the substring with 3 genes (i.e., 1 node) and retry if this cross-over results in a valid tree. This process is repeated until this results in a valid cross-over or until the length of substring is 0 and thus no cross-over is applied. Figure 5.4 shows a successful crossover operation in which the genetic information of the first 2 nodes is interchanged, resulting into 2 new individuals with valid topologies.

5.5.1.3 Mutation Operator

A mutation operator introduces genetic diversity into the population by randomly altering one or more genes of the individual. We implemented a 3-phase mutation operator. In the first phase, our mutation operator iterates over all nodes in the individual and with a probability p_{gene} alters the parent gene p_n of a node n. This parent is chosen out of the list of valid parents for that node, i.e., based on the topology information and the communication range of the different available MCSs. From this list of possible parents, all nodes that are currently descendants of the node are excluded to prevent the mutation operator from introducing a loop in the network tree. The current parent is also excluded. Figure 5.5 shows an example of such a successful parent mutation. In the second phase, the mutation operator iterates over all MCS genes of all nodes in the individual and, with the same probability p_{gene} or if the parent of this node was changed, randomly alters the gene value m_n. When picking a new MCS, the possible MCSs depend on the current parent of the node to prevent it from choosing an MCS to its parent that results in reliability that is

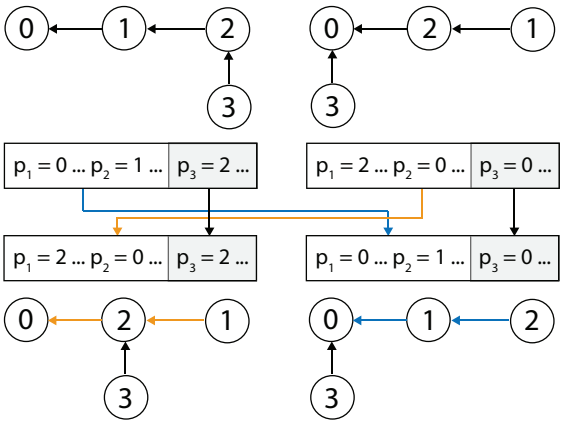


Figure 5.4: Example of successful two-point cross-over operation where the genetic information of the first 2 nodes is interchanged, resulting in 2 new individuals with valid topologies.

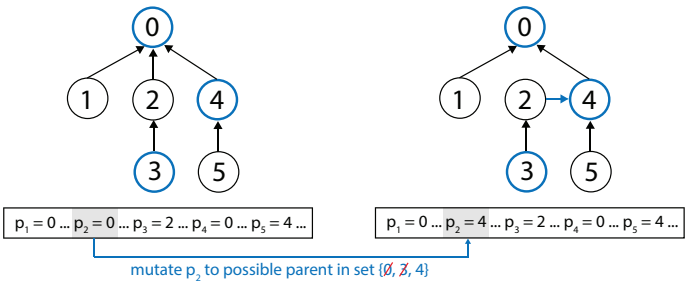


Figure 5.5: Example of the mutation operation in which the parent of node 2 is altered. Assuming that the possible parent set of node 2 is $\{0, 3, 4\}$, the new parent can only be node 4, as the current parent is 0 and node 3 is a descendant of node 2 (which would lead to a loop).

below the predefined threshold to form a link (i.e., see the experiment setup in Section 5.6.1 for the threshold value). In the third and last phase, the operator iterates over the third gene of all nodes, i.e., the number of bonded slots a node allocates a_n. It alters the gene value with the probability p_{gene} or whenever the MCS value of a node was altered (to make sure the new number of slots value does not exceed the slotframe length). The valid range for the number of slots depends on the picked MCS, and is $[0, \lfloor \frac{len(slotframe)}{len(bonded\_slot, MCS)} \rfloor]$ in which $len(bonded\_slot, MCS)$ represents the number of regular slots needed to bond together for the chosen MCS.

Algorithm 6 Genetic algorithm workflow.

```
1: best_ind ← None
2: generation ← 0
3: pop ← initialize_population(pop_size)
4: evaluate_fitness(pop)
5: while generation ≤ max_generations do
6:     parent_pop ← select(pop, pop_size)
7:     offspring ← crossover(parent_pop)
8:     offspring ← mutate(offspring, p_gene)
9:     evaluate_fitness(offspring)
10:    pop ← select(pop + offspring, pop_size)
11:    for ind ∈ pop do
12:        if ind.fitness > best_ind.fitness then
13:            best_ind ← ind
14: return best_ind
```

5.5.1.4 Fitness function

To evaluate the fitness of an individual according to the slot bonding problem formulated in Section 5.4, the fitness function calculates the expected total number of delivered packets at the root and the network radio on time. To calculate the total number of delivered packets, we apply the calculation defined in Section 5.4.1. The radio on time for each node is calculated by applying Equation 5.8. These radio on times are summed, resulting in the network radio on time. Consequently, the individual with the highest expected number of delivered packets and lowest radio on time, in lexicographical order, is considered the best individual. This means that the first goal of the GA is maximizing the expected number of delivered packets and then individuals with identical delivered packets values are ranked according to their radio on time. However, before evaluating the expected number of delivered packets and radio on time, an individual is first checked on being *feasible*, meaning that all allocations should fit the in TSCH schedule. To do so, the individual is passed to the feasibility checking algorithm defined in Section 5.5.2. If the individual is not feasible, the number of delivered packets is set to a negative value (i.e., -100) and the radio on time to an unrealistic high positive value (i.e., 1×10^6 ms). In case the individual is feasible, the fitness function continues to calculate the expected number of delivered packets and the radio on time.

5.5.1.5 Workflow

Algorithm 6 shows the workflow of the genetic algorithm. Before the GA starts for a run of $max\_generations$ generations, the population pop is initialized with pop_size number of individuals. The initial individuals are built by starting from one and the same valid topology (i.e., this initial topology is the same

for all individuals in the initial population), while the m_n and a_n genes are picked uniformly at random out of the set of valid options for that node n to its parent, meaning that only MCSs can be chosen that allow the node to communicate with its parent and the length of all allocated bonded slots does not exceed the slotframe length. Then, the mutation operation of Section 5.5.1.3 is applied 100 times to randomize the individual while guaranteeing that the MCS allocations, the slot allocations and the represented topology stay valid (i.e., each node can reach the root and there are no routing loops). In each generation, on line 6, it first selects pop_size parents for reproduction. Then, the cross-over operator as discussed in Section 5.5.1.2 is applied to all sets of two parents in $parent\_pop$, after which the GA mutates each individual of the $offspring$, with a probability p_{gene} being the probability to alter a gene of the individual, according to the operator described in Section 5.5.1.3. Afterwards, at line 9, the fitness of the $offspring$ is evaluated using the fitness function of Section 5.5.1.4. The population for the next generation is prepared by selecting pop_size individuals out of the population pop that started the generation and the produced $offspring$. At lines 11-14, the fitness of each individual in the new population is checked to be better than the fitness of the current best individual $best\_ind$. If so, that individual becomes the new $best\_ind$. This procedure is repeated until the $max\_generations$ is reached, after which the $best\_ind$ is returned.

The exact probability p_{gene}, and the different selection strategies of line 6 and 10, were selected empirically and are discussed later in Section 5.6.1.

5.5.2 Feasibility Heuristic

An individual returned by the proposed GA should be feasible, meaning that all allocations should fit in one TSCH slotframe while avoiding interference and respecting that a node cannot transmit and receive simultaneously. To check if this holds for the given individual, it is possible to formulate a set of linear constraints that the solution should satisfy, based on the formulation in Section 5.4.3. However, because of the computational complexity of such a feasibility check with linear constraints, instead we employ a greedy bin-packing heuristic that guarantees to identify all infeasible solutions (while it might identify a small percentage of the feasible solutions as infeasible). The heuristic takes as input the network tree topology, the number of bonded slot allocations a_n for the chosen MCS between a node n and its parent, as represented in the tested individual, and the interferers of all nodes in the network. This heuristic is also used to generate a possible TSCH schedule from the best individual found by the proposed genetic algorithm.

The heuristic is shown in Algorithm 7. It iterates over all nodes except the root, N_0, and checks if the number of allocated bonded slots a_n of node n is larger than zero. If not, the heuristic can directly continue to the next node. The for loops of lines 4 and 5 iterate over the entire schedule and the heuristic tries to allocate the bonded slot of length s_{m_n} (with m_n being the MCS picked by node

Algorithm 7 Feasibility heuristic

```
1: for n ∈ N_0 do
2:     if a_n > 0 then
3:         num_slots = a_n
4:         for f ∈ [0, f_max] do
5:             for t ∈ [0, t_max − s_{m_n} + 1] do
6:                 if allocate(n, s_{m_n}, t, f) then
7:                     num_slots = num_slots − 1
8:                     if num_slots > 0 then
9:                         continue with next slot
10:                    else
11:                        continue with next node
12:        if num_slots > 0 then
13:            return False
14: return True
```

n) at each slot (t, f). To allocate a bonded slot starting at a slot (t, f), it checks if the node or the parent of the node is not yet transmitting or receiving in the range $[(t, f), (t + s_{m_n} - 1, f)]$ and if no slot in that range is used by an interferer. If the bonded slot can be allocated successfully, the heuristic continues with the next bonded slot for that node or with the next node, depending on if there are bonded slots left to allocate, i.e., the value of $num\_slots > 0$ at line 8. If it cannot be allocated, the heuristic moves on to the next slot. If the heuristic has iterated over all slots in the schedule and there are still bonded slots to allocate, i.e., $num\_slots > 0$ at line 12, the individual is considered infeasible. If the heuristic can iterate over all nodes and allocate all bonded slots, the individual is considered feasible.

It it important to stress that the bin-packing heuristic will correctly identify *all* infeasible individuals. However, depending on the order of the nodes in N_0, some feasible individuals will be identified as infeasible. Therefore, for each individual, there are different runs of the heuristic in Algorithm 7, each time with N_0 sorted differently, such as sorting the nodes with most bonded slot allocations first or using a breadth-first search of the tree topology. As long as the heuristic labels the individual as infeasible and there are sorting algorithms left to try, the heuristic runs again. From the moment an individual is considered feasible by the heuristic, this process stops and the positive outcome is returned. If none of the sorting algorithms make the heuristic return a feasible result, the individual is returned as an infeasible solution.

5.5.3 Time Complexity Analysis

To analyse the time complexity of the GA, the different steps of a single generation in Algorithm 6 (i.e., the while-loop from line 5 to 13) were analysed.

The time complexity of the cross-over operator (i.e., $O(|N|^3)$), the mutation operator (i.e., $O(|N|^2)$), the selection operator at line 6 (i.e., tournament selection with time complexity $O(k \cdot pop\_size)$ and k being the tournament size), the selection operator at line 10 (i.e., elitist selection with time complexity $O(pop\_size \cdot log(pop\_size))$) are all dominated by the time complexity of the fitness function calculation (specifically, the calculation of the $h_{C_n,q}$ values at every node, as defined in Section 5.4.1.2), resulting in a total time complexity of $O(|N|! \cdot |N|^2)$, while assuming fixed constants for parameters such as $max\_generations, pop\_size, Q, |F|, |T|$ and g. The analysis confirms the high complexity of solving the proposed slot bonding formulation.

5.6 Evaluation

In this section, we evaluate the effectiveness of the decisions taken by the proposed GA, in terms of the parent selections and MCS and slot allocations, to solve the proposed TSCH slot bonding problem by applying them in TSCH network simulations. First, we compare the GA heuristic to the optimal solution obtained from an exhaustive search approach. Afterwards, we test the effectiveness of the feasibility heuristic proposed in Section 5.5.2 and validate the proposed GA by comparing its results to the simulation results. Then, we run the GA for different network sizes while showing the advantages of slot bonding. We also show the advantage of employing multiple PHYs (or MCSs), compared with only using one MCS. Finally, we provide insight in preferred parent selection and MCS configuration when applying slot bonding that can be used as a basis to develop practical scheduling and routing algorithms for multi-MCS TSCH networks.

5.6.1 Experiment Methodology & Setup

The experiments are conducted by following a four-step process: first, the network topologies were generated by the 6TiSCH simulator which was introduced in Chapter 2. Second, those topologies were fed to the GA, implemented in the Python DEAP framework [119], that solves the slot bonding problem defined in Section 5.4 heuristically. Third, the best possible solution found by the GA was used as input for the heuristic defined in Section 5.5.2 to generate a TSCH schedule for all nodes in the network. Fourth, the scheduling solution provided by the heuristic is used as a centralized schedule to make static schedule allocations in the TSCH experiments using the 6TiSCH simulator. This evaluation process that allows us to analyse the slot bonding technique is illustrated in Figure 5.6. The source code changes to the 6TiSCH simulator, the GA implementation and the feasibility heuristic are publicly available[1]. The number of slots per bonded slot are configured as shown in Table 5.1. Every node of the network is configured

[1] https://github.com/imec-idlab/adaptive-mcs-ga-simulator

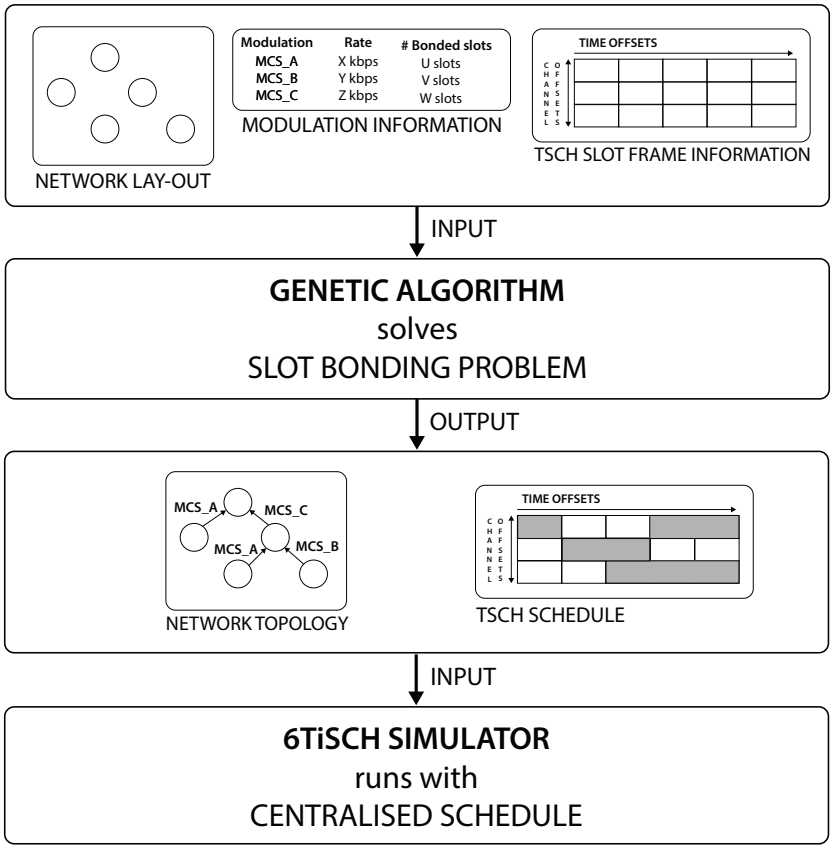


Figure 5.6: Flow diagram showing the evaluation process, showing that the output of the GA is used as a centralised schedule for the 6TiSCH simulator to analyse the proposed slot bonding technique.

to generate 1 packet per slotframe. Each experiment result is averaged over at least 20 iterations. The error bars in the figures represent the standard deviation over these iterations.

5.6.1.1 6TiSCH Simulator

We extended the simulator with a sub-GHz outdoor propagation model based on the 3rd Generation Partnership Project (3GPP) spatial channel model [120] and use 3 channels in the 868 - 868.6 MHz band. The simulator supports the 3 different MCSs in option 4 of the OFDM PHY which all require 156 kHz nominal bandwidth per channel, as listed in Table 5.1. In addition, to determine the link reliabilities for every MCS (as introduced in Section 5.4.1.2), real-world

measurements were performed with the Atmel AT86RF215 transceiver[2] (which is fully compliant with the IEEE 802.15.4g-2012 standard and thus supports OFDM) of two OpenMote-B nodes. After configuring both nodes with the same MCS, attenuating the transmitter by a certain value and transmitting 200 frames with a 127 byte Physical Service Data Unit (PSDU), the packet reception rate (PRR) and average RSSI were calculated from the receiver logs. This process was repeated for each combination of MCS and attenuation in order to map the PRR in function of average RSSI for each of the selected MCSs. The measured data and resulting regression models are publicly available[3].

The topologies were created by randomly placing nodes until a node had at least one reliable link (i.e., a PDR of at least 70% for MCS2) to one of the other nodes. This was done to ensure that the network is fully connected and each node can reach the sink over one or multiple hops. In our experiments, only the PHY and MAC layer were enabled and all other stack layers were disabled. The interference model implemented in the simulator, was changed to the Yet Another Network Simulator (YANS) model [121]. Note that while the interference model was enabled during all simulations, there were no transmission failures due to interference effects because the proposed slot bonding model includes interference avoidance. The receiver noise floor is calculated using the thermal noise calculation $N = 1.381 \cdot 10^{-23} \frac{J}{K} \cdot 290K \cdot 156 \cdot 10^{3}$Hz, added to the noise figure of the Atmel AT86RF215 chip, i.e., 4.5 dB.

5.6.1.2 GA

The GA workflow is configured as described in Section 5.5.1. The population size pop_size is set to 100 individuals, while the maximum number of generations $max\_generations$ is configured to 10000 generations. The parent selection operator, at line 6 in Algorithm 6, is set to tournament selection with a tournament size of 2. The mutation probability of a gene p_{gene} is set to 0.05. Within the mutation operator, the minimum reliability threshold for a node to be able to pick a specific PHY to its parent, is 70 %. To select the population for the next generation, at line 10 in Algorithm 6, elitist selection, retaining the 10 % best individuals of the previous generation, is applied. This GA configuration showed the best overall performance after evaluating a large number of configurations for a wide variety of inputs (results omitted due to space constraints). To calculate the expected number of delivered packets and the network radio on time in the fitness function of the GA as described in Section 5.4.1 and Section 5.4.2, the output of the topology creation is provided as input to the GA, i.e., the different available MCSs and corresponding link reliabilities between all network nodes. The parameter g is set to 1 as each node is configured to generate 1 packet per slotframe. The maximum number of allowed transmission opportunities per

[2] http://ww1.microchip.com/downloads/en/devicedoc/atmel-42415-wireless-at86rf215_datasheet.pdf

[3] https://github.com/imec-idlab/adaptive-mcs-ofdm-measurements

packet, denoted by r_{max} in Section 5.4.1.1, is set to 4 as advised by the 6TiSCH Minimal Configuration [36].

5.6.2 GA Validation

We compare the expected number of delivered packets and radio on time results returned by the GA to the results returned by exhaustive search. Due to the computational complexity, we could only run exhaustive search experiments for topologies up to 7 nodes for 10 ms regular slots (while applying slot bonding) and 8 nodes for 40 ms regular slots. The TSCH schedule was configured to have a slotframe length of 120 ms, 2 channels and 2 MCSs, i.e., MCS2 and MCS4 of Table 5.1. Next to the optimality of delivered packets and the difference in radio on time between the GA and the exhaustive search, Table 5.3 also shows the relative number of candidate solutions (i.e., individuals) checked by the GA and the relative computation time needed by the GA compared with the exhaustive search. The relative number of candidate solutions is the number of unique solutions checked by the GA over the number of candidate solutions checked by the exhaustive search (after filtering out a small set of invalid solutions beforehand). The relative computation time is the time needed by the GA over the time needed by the exhaustive algorithm to find the optimal solution. As expected, for an increasing number of nodes and the shorter the regular slot length (which results in more slots in a slotframe), the relative number of candidate solutions and relative computation time for the GA becomes smaller as the number of candidate solutions increases exponentially. For 5 nodes and a regular slot length of 40 ms, we notice that the GA took slightly more time to complete than the exhaustive search while considering less unique candidates. The reason is that the GA employs a fixed population size (i.e., 100 individuals) and a fixed number of generations (i.e., 10000 generations), resulting in a total of 1000000 solutions that need to be checked by the GA. As for this experiment configuration the number of unique solutions is smaller than 1000000, the total number of individuals checked by the GA includes many duplicate solutions that require extra computation time. The delivered packets and radio on time differences in Table 5.3 show that, for topologies up to 5 nodes and a regular slot length of 10 ms, the GA finds a globally optimal solution, while for 6 and 7 the optimality drops with 0.6 % and 0.4 % respectively. For a regular time slot length of 40 ms a globally optimal solution is found for all topology sizes. These results confirm that the proposed GA is capable of finding (near-)optimal solutions for the slot bonding problem.

To show the accuracy of the feasibility heuristic presented in Section 5.5.2, the output of the heuristic is compared with the output of a computationally-intensive Integer Linear Program (ILP) model that can determine the feasibility of a GA individual with 100 % accuracy. This ILP is extracted from the set of decision variables and constraints defined in the slot bonding formulation in Section 5.4. Table 5.4 shows the results for a slotframe length of 200 ms and 2000 generations. The heuristic proves to be much faster as it needs only 3.9 %

Table 5.3: Averaged GA results compared with exhaustive search results, for different topology sizes and slot lengths.

Nr. nodes, slot length	Delivered packets optimality (%)	Radio on difference (ms)	Relative nr. candidates (%)	Relative comp. time (%)
5 nodes, 10 ms	100	0	0.6	21.5
6 nodes, 10 ms	99.4	1.7	< 0.1	3.1
7 nodes, 10 ms	99.6	5.2	< 0.1	0.5
5 nodes, 40 ms	100	0	3.7	102
6 nodes, 40 ms	100	0	0.3	8.8
7 nodes, 40 ms	100	0	< 0.1	1.1
8 nodes, 40 ms	100	0	< 0.1	0.6

Table 5.4: Feasibility heuristic compared with the exact ILP feasibility model for a slotframe length of 200 ms and 2000 generations.

Topology size	Relative comp. time (%)	True negative rate	False negative rate
8	3.9 ± 0.5	1.0 ± 0	0.17 ± 0.014
14	2.4 ± 0.7	1.0 ± 0	0.058 ± 0.039

and 2.4 % of the ILP computation time. The results also show that the heuristic identifies all infeasible individuals, i.e., the true negative rates are 1. However, the heuristic does label some individuals as infeasible while they are feasible: the false negative rate of 0.017 for 8 nodes increases to 0.058 for 14 nodes. While the increasing number of nodes clearly makes it more difficult for the heuristic to fit all allocations in the TSCH schedule, this has a minimal impact on the expected delivered packets results. When comparing the best solution found when using the heuristic with the best solution found when using the ILP, the average optimality of the number delivered packets is $99.2\,\% \pm 2.6\,\%$ and $98.0\,\% \pm 7.4\,\%$ for 8 and 14 motes respectively.

We also validate the GA by comparing its PDR results (i.e., the expected delivered packets value calculated in Equation 5.5, over the number of generated packets per slotframe) with the PDR results of the TSCH simulation that used the

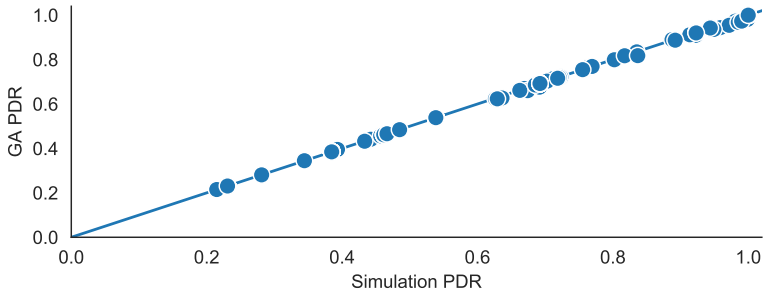


Figure 5.7: GA and simulation PDR for all experiment iterations for the 10 ms bonded slots for 120 ms, 200 ms, 280 ms and 360 ms slotframe lengths in a network with 14 nodes.

GA schedule allocations as input. Figure 5.7 compares the GA and simulation PDR values for all iterations for the 10 ms bonded slots and 40 ms experiments with 120 ms, 200 ms, 280 ms and 360 ms slotframe lengths in a network with 14 nodes. For all iterations, both values are nearly identical with an overall root mean squared error (RSME) of only 0.0044, confirming that the GA model and TSCH simulator behave similar and validating that the expected PDR calculated by our GA model is accurate.

5.6.3 Slot Bonding Scalability

To show the effect of slot bonding on scalability, we compare the resource-efficient slot bonding technique with the static approach without slot bonding where a time slot should have a length long enough to accommodate all possible MCSs (and thus valuable slotframe time is wasted for transmissions using MCSs with fast data rates that do not require the maximum slot length). We compare TSCH schedules that contain 10 ms bonded slots to schedules with 40 ms regular slots. Table 5.1 shows the number of slots needed for each supported MCS. The 40 ms slots are long enough to support all MCSs and thus no slot bonding is required. We compare both techniques for slotframe lengths of 120 ms, 200 ms, 280 ms and 360 ms, meaning that there are 12, 20, 28 and 36 slots for the slotframe with 10 ms bonded slots and 3, 5, 7 and 9 regular slots for the slotframe with 40 ms slots respectively.

Figure 5.8 shows the average PDR GA results for 8 and 14 nodes. It is clear that the slot bonding technique offers a clear PDR advantage over the static approach for different slotframe lengths. For a 120 ms slotframe length and 8 nodes, the PDR increases from 0.426 (± 0.011) to 0.75 (± 0.095), while for 14 nodes the PDR almost doubles from 0.23 (± 0.004) to 0.446 (± 0.042). For the slotframe length of 360 ms, the difference for 8 nodes between both PDRs is negligible as both PDRs are close to 1 because the slotframe length is long

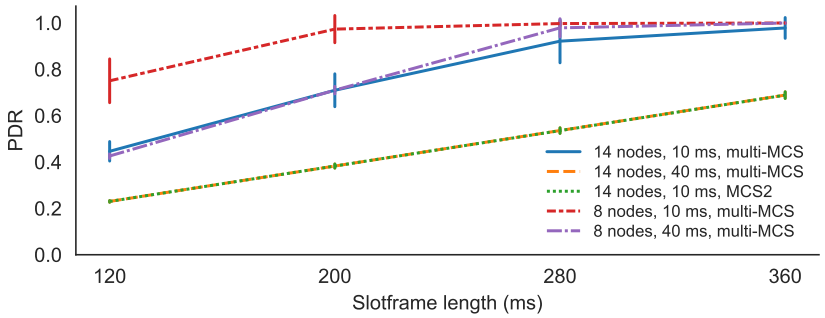


Figure 5.8: PDR comparison for 10 ms bonded slots and 40 ms slots, for different slotframe lengths.

enough to accommodate all the necessary bonded slots. Additionally, applying slot bonding allows to achieve a PDR near 1 at a 200 ms slotframe length, while for 40 ms this requires a 280 ms length. For 14 nodes and a slotframe length of 360 ms, there is still a PDR increase of 42 % with the 40 ms approach having a PDR of 0.689 (± 0.014) while the slot bonding approach has a PDR of 0.978 (± 0.045). As expected, for both techniques the PDR increases as the length of the slotframe increases and the contention decreases.

Table 5.5 shows the average number of drops due to the maximum of retransmissions being reached (i.e., caused by link failures or full queues at the receiver) relative to the number of generated packets, the average number of generated packets that was immediately dropped (because the queue of the transmitter was full), the average link reliability and the average hop count. For the last two metrics, only the nodes for which a packet could reach the root were considered, meaning only nodes for which there was a path to the root with every node along that path having at least one allocated slot to its parent. Because the reliability of the allocated links in both approaches was high, it is clear that the major reason for the difference in PDR is due to saturated queues, either at the transmitter or at the receiver.

These results show that the slot bonding technique is more efficient and can choose to allocate an extra slot with an MCS that requires less time (a bonded slot of 20 ms or 30 ms) which could be just short enough and have enough reliability to serve an extra node or packet in the network. In contrast, for each extra slot the 40 ms static approach wants to allocate, the complete slot length needs to be allocated, i.e., 40 ms. This confirms the scalability advantage of the slot bonding approach over the static approach with fixed 40 ms slot lengths.

Table 5.5: GA averaged results for 14 node topologies, for both 10 ms (with slot bonding) and 40 ms regular slot lengths and different slotframe lengths.

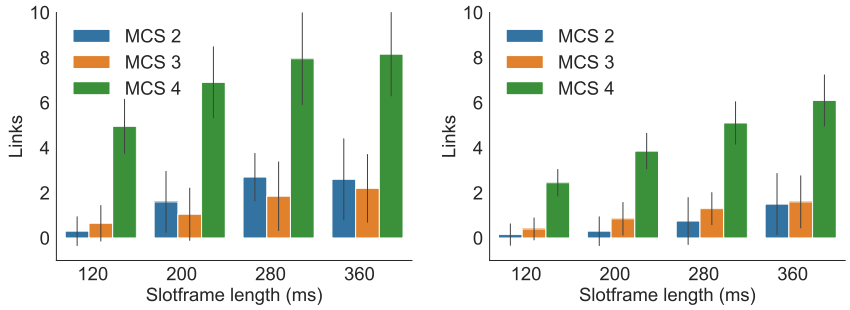
Slot length, slotframe (ms)	Retransmission drops (%)	Full queue drops (%)	Link reliability (%)	Hop count
10, 120	0 ± 0	55.4 ± 4.2	99.1 ± 3.8	1.6 ± 0.6
10, 200	0.1 ± 0.1	29 ± 7.1	98.1 ± 5.5	1.9 ± 0.8
10, 280	0.5 ± 1.7	7.3 ± 9.2	97.6 ± 5.8	2.2 ± 1
10, 360	0.1 ± 0.1	2.1 ± 4.5	98 ± 5.5	2.2 ± 1
40, 120	0 ± 0	77 ± 0.4	99.7 ± 2.3	1.1 ± 0.3
40, 200	0 ± 0	61.7 ± 0.9	99.8 ± 1.9	1.4 ± 0.6
40, 280	0.2 ± 0.6	46.2 ± 0.9	99.5 ± 3.2	1.6 ± 0.7
40, 360	0.6 ± 1.8	30.5 ± 2.1	99.3 ± 3.4	1.8 ± 0.8

5.6.4 Adaptive PHYs

We show the advantage of employing multiple PHYs to adapt to the link conditions compared with only using a single PHY. Figure 5.8 shows the results in a network with 14 nodes when applying the slot bonding approach with 10 ms slots for both multiple MCSs and only employing MCS2. Only employing MCS4 was not possible, as it cannot always create a reliable enough link to create a fully connected network. It is clear that using multiple MCSs combined with slot bonding results in significantly better PDR values. The figure also shows that the PDR performance of only using MCS2 is the same as the performance of a network that only uses 40 ms regular slots (i.e., without slot bonding) for multiple MCSs. The reason for this is that in order to use MCS2, 4 regular slots always have to be bonded together. However, when comparing the radio on time of both experiments, there is a clear difference in the experiment with multiple MCSs as the nodes can choose MCSs that may provide the same reliability as MCS2 but require less transmission time. For the slotframe lengths of 120 ms, 200 ms, 280 ms and 360 ms, there was a respective 56.2 %, 50 %, 46.2 % and 43.7 % decrease in radio on time when using multiple MCSs.

5.6.5 Allocation Analysis

First, to understand how the network tries to optimize the PDR, we observe the different PHYs used by the nodes. Figure 5.9 shows the results for 14 nodes, when applying slot bonding with 10 ms and with 40 ms regular slots. Table 5.5



(a) 10 ms bonded slots.

(b) 40 ms bonded slots.

Figure 5.9: MCSs used in a TSCH network with a topology size of 14 nodes.

shows the average reliabilities for all the links in the network. The figure only shows the MCSs for links of nodes for which a packet could reach the root. It is clear that when applying slot bonding more nodes in the network can make slot allocations, especially for the short slotframe lengths of 120 ms and 200 ms, compared with the 40 ms naive approach. In general, the nodes try to allocate many links using MCS4, which is the fastest MCS and thus requires less transmission time. In the case of slot bonding, this means that only 2 regular slots are bonded together, thus leaving more free space in the schedule for other allocations. Moreover in the slot bonding case, when increasing the slotframe length, also more MCS2 PHYs are allocated in an effort to guarantee more reliability as there is more free space in the schedule to do so.

Additionally, different metrics were tested to thoroughly understand the decisions that were taken when choosing a node's parent and MCS in order to optimize the PDR and radio on time. Table 5.6 shows the different metrics and the average values over the results obtained for the 120 ms, 200 ms, 280 ms and 360 ms slotframe lengths for two GA experiments with respectively 8 and 14 nodes, when applying slot bonding with a 10 ms regular slot length. The results in the table show that when a node chooses its parent and the MCS to that parent, it first focuses on MCS that provide high reliability rather than preferring fast MCSs, as the numbers of picking reliable MCSs are higher than those of picking faster MCSs. These results, combined with the picked MCSs shown in Figure 5.9 and the reliability values listed in Table 5.5 make it clear that many nodes were able to pick the fastest PHY MCS4 while still achieving high reliability. Also, when there are different MCSs available at the parent that provide an equal reliability, the results show that in virtually all cases the node will pick the fastest MCS.

When looking at parent selection choices, parent nodes that are physically closer to the root than the node picking the parent, were clearly preferred with averages of 96.5 % and 94.3 % for 8 and 14 nodes respectively. Due to shorter distance, those neighbors experience less path loss than nodes that are further

Table 5.6: Different metrics tested for two GA runs for respectively 8 and 14 nodes when applying slot bonding with 10 ms, showing the average values over the results obtained for the 120 ms, 200 ms, 280 ms and 360 ms slotframe lengths.

Metric	**8 nodes**	**14 nodes**
Picked most reliable MCS of parent / overall (%)	94.8 ± 7.8 85.3 ± 9.1	95.3 ± 1.6 88.5 ± 4
Picked fastest MCS of parent / overall (%)	70.5 ± 2.7 64.8 ± 2.7	77.2 ± 6.6 72.1 ± 9.1
Picked fastest MCS of equal reliable MCSs of parent / overall (%)	99.9 ± 0.2 94.2 ± 0.9	100 ± 0 95.6 ± 1.9
Picked parent closer to root than itself (%)	96.5 ± 1.2	94.3 ± 2.8
Picked parent with equal or better reliability than itself (%)	77.5 ± 6.3	78.8 ± 7.2
Picked parent with equally fast or faster rate than itself (%)	64.7 ± 14.7	81.8 ± 9.8

away from the root, making them the more reliable choice. More reliability means less transmissions needed to deliver the packets and thus less radio on time and slots that need to be allocated, again leaving more free space in the schedule for other nodes. Additionally, the nodes seemed to have a strong preference for parent nodes that had a reliability towards their respective parent that was equal or higher than the reliability between the node itself and its parent. On average, in 64.7 % and 81.8 % of the cases for the experiments with 8 and 14 nodes respectively, a node also preferred a parent node with equally fast or faster MCSs to their parent compared to the rate of the MCS between the node itself and its parent. Moreover when only looking at nodes that allocated the slowest MCS2 PHY towards their parent, it is significant that on average in 95.3 % and 91.1 % of the nodes (i.e., in the experiments with 8 and 14 nodes respectively) prefer a parent that allocated a MCS with a faster rate than the MCS between the node itself and its parent. When looking at nodes with the fastest MCS MCS4 allocated towards their parent, we observe that especially for the shorter slotframe lengths nodes prefer parents with the same rate (it is not possible to allocate a faster MCS), while for the longer slotframe lengths this effect diminishes, i.e., for a 120 ms (and a network of 14 nodes) 91.3 % of the nodes prefer the same MCS, while for 360 ms length this is only 54.4 %. As there is more space in the schedule with longer slotframe lengths, nodes

probably start to prefer the slower but more reliable MCS2 PHYs.

5.7 Conclusion

This chapter explored the simultaneous use of multiple PHYs in a IEEE 802.15.4e TSCH network by proposing the concepts of slot and channel bonding and performing an in-depth analysis of the slot bonding concept. We formally described the slot bonding problem that maximizes the expected number of delivered packets to the root while minimizing the radio on time. Additionally, we proposed a GA as a static analysis tool, allowing us to implement and solve the computationally complex slot bonding problem up to a network size of 14 nodes. The GA finds an near-optimal solution in terms of parent and MCS selection as well as slot allocations, i.e., the maximum deviation in terms of delivered packets to the root is on average less than 1%.

The results of applying the GA solutions in TSCH network simulations confirmed the scalability advantage of the slot bonding approach in terms of PDR, as it can bond multiple regular slots together in a bonded slot without over-allocating the slot time it needs and thus avoid wasting radio on time and energy. It was also shown that simultaneously using different PHYs in the same network compared to only allowing the most reliable PHY demands less radio on time while maintaining the same PDR. Finally, the simulations showed insight in the preferred PHY and parent selections for the slot bonding approach to optimize the number of delivered packets.

Chapter 6

Parent and PHY Selection in TSCH Slot Bonding Networks

The content of this chapter is partially based on:

- Glenn Daneels, Dries Van Leemput, Carmen Delgado, Steven Latré, Eli De Poorter and Jeroen Famaey. (2021). *Parent and PHY Selection in TSCH Slot Bonding Networks*. To be submitted.

6.1 Introduction

Industrial applications that are deployed on low-power wireless sensor networks often demand a trade-off between data rate, energy consumption, latency and reliability. To fit the exact requirements, 6TiSCH networks allow to tune the network characteristics at different layers in the network stack, starting from the IEEE 802.15.4e TSCH MAC layer up to the application layer [12, 122]. Nevertheless, the IIoT stack is limited by the characteristics of the chosen PHY layer, that is often challenged by the harsh conditions of industrial environments which are filled with the presence of other wireless technologies and metal infrastructure. To overcome this limitation, the TSCH and 6TiSCH community is shifting research efforts to the support of multiple PHY layers, as discussed in Chapter 5 [19, 23, 34, 111].

Employing different PHYs in a single TSCH network also means embracing the different characteristics of each PHY, such as its data rate, bandwidth, computational complexity, energy consumption and wireless reliability. Integrating multi-PHY support in a 6TiSCH network therefore implies on the one hand dealing with the different transmission lengths and bandwidth requirements to fit the PHY allocation in the TSCH schedule, and on the other hand defining

policies on how to select the appropriate PHY for each network link. In Chapter 5, we have answered how different PHYs can be fitted in a TSCH schedule by introducing the concepts of slot and channel bonding. In this chapter, we investigate the parent and PHY selection for each network node, related to making correct routing decisions based on the available PHYs and being able to schedule those PHY resources in the TSCH schedule.

The contributions of this chapter are two-fold. First, we propose a computationally efficient heuristic that is a step towards a distributed PHY and parent selection mechanism for a TSCH network that applies slot bonding. The performance of this heuristic is initially evaluated via simulation results, by comparing it against the near-optimal, but computationally complex multi-PHY scheduler proposed in Chapter 5. Second, we propose an easy-to-configure TSCH slot bonding implementation in the latest version of the Contiki-NG IIoT operating system. Subsequently, this slot bonding implementation is used to extensively evaluate the proposed heuristic on a real hardware sensor testbed.

The remainder of this chapter is structured as follows. First, we introduce the related work on using multi-PHY TSCH in Section 6.2. Second, in Section 6.3, we introduce the parent and PHY selection heuristic. In Section 6.4, we describe the TSCH slot bonding implementation in the Contiki-NG firmware. Afterwards, the heuristic is evaluated in Section 6.5. Finally, Section 6.6 presents the conclusions of this chapter.

6.2 Related Work

The introduction of the IEEE 802.15.4g amendment and the commercial availability of IEEE 802.15.4g-compliant transceivers has spiked research towards the available PHYs and their potential for 6TiSCH networks. Section 5.2 of Chapter 5 already discussed many works carrying out experimental evaluations including range tests, interference robustness and overall suitability of different IEEE 802.15.4g PHYs, tested in outdoor, smart building or industrial scenarios [17, 18, 104, 105, 106, 107, 112].

Currently, research that investigates specifically how different PHYs can be combined in a single TSCH network, is scarce. M. Rady *et al.* adjusted the OpenWSN firmware to support O-QPSK (2.4 GHz at 250 kbps), FSK (868 MHz option 1 at 50 kbps) and OFDM (868 MHz option 1 MCS3 at 800 kbps) [19]. For a fair comparison, the authors used a 40 ms time slot that could facilitate all implemented PHYs. This approach is similar to the baseline approach used in Chapter 5. After evaluating the different PHYs on a 42 nodes office testbed, the authors conclude that no PHY outperforms the other PHYs for all metrics, and therefore the combination of different PHYs should be considered on a frame-by-frame basis (depending on the changing propagation characteristics of the link) in a 6TiSCH architecture. Defining a policy on how to select the appropriate PHY was not in the scope of the article. Gomes *et al.* propose different policies on packet replication and the use of multiple IEEE 802.15.4g

modulations for different packets, i.e., modulation diversity, to increase network reliability [33]. The authors do not discuss how to exactly integrate the different modulations in TSCH, but their defined policies should be considered a PHY selection mechanism, that can be applied on top of our slot bonding approach. Another recent, interesting approach is presented by Van Leemput *et al.* who take into account the different data rates possible for different PHYs [34]. The authors defined two alternative timeslot structures allowing multiple packets transmissions to increase the throughput for higher data rate PHYs while maintaining a fixed slot duration. This approach is different from our slot bonding approach where only 1 packet is transmitted, but the slot length is adapted to the chosen PHY. Consequently, while our approach might be more complex to implement, it is more flexible and extensible to efficiently support a wider range of PHYs. Additionally, they developed a flexible Link Quality Estimation technique to dynamically switch between PHYs depending on the current propagation's characteristics of the link. Another interesting work is the approach of Brachmann *et al.* They proposed two multi-PHY designs: (i) a *multi-template* design where slower PHYs are scheduled to have logical slots spanning multiple real slots, and (ii) a *single-template* design where the slot size is based on the slowest PHY [111]. Their multi-template design is similar to our slot bonding approach presented in Chapter 5. The real hardware evaluation of Brachmann *et al.* applies their multi-template design by assigning management traffic to the slower, more reliable PHY while data is transmitted over the faster, less reliable PHY. As such, they do not not discuss a PHY selection mechanism and their results primarly focus on the advantages of using a multi-PHY in terms of range, channel utilization and synchronization accuracy. The TSCH slot bonding firmware implementation presented in this chapter is a combination of a new easy-to-configure implementation, developed in the latest version of the Contiki-NG repository, merged with insights from the open-source multi-PHY work from both Van Leemput *et al.* and Brachmann *et al.*

To the best of our knowledge, this work is the first to evaluate a parent and PHY selecting mechanism in a TSCH slot bonding, multi-hop network deployed on a real hardware testbed.

6.3 Heuristic Parent and PHY Selection

In this section, we propose a computationally efficient heuristic algorithm that allows TSCH slot bonding nodes to make a parent and PHY selection. The heuristic is a step towards a distributed selection mechanism that could be integrated into the RPL or any other routing protocol. First, we re-iterate how parent nodes are normally selected in a single PHY network and afterwards explain the motivation for this heuristic by briefly referring back to our work in Chapter 5. Then, we present the heuristic itself. We conclude with providing insight in how this heuristic could be integrated into RPL.

6.3.1 RPL Parent Selection

In 6TiSCH networks, it is the RPL routing protocol that organizes the network paths along which data is transmitted from the originating node to the network sink [45]. In Section 2.3.4, we already gave an overview of RPL, but here we will explain the important concepts related to RPL parent selection.

RPL organizes the network nodes in a tree topology which is rooted at the sink node (i.e., a border router). Each node in the RPL routing tree can have zero or more child(ren), and one or more parent(s). A node that has no children, is a leaf node, and the node in the tree that does not have any parents, is the root node of tree. When data has to be transmitted to the sink (or root) of the network, it is forwarded on the link between every node and its parent along the path from the originating node towards the root. The location of the node in this routing tree, is determined by its *Rank* value. The node calculates this value itself and broadcasts this information in so-called DIO messages. To calculate this Rank value, but also to select the preferred routing parent, a node applies an Objective Function (OF) that uses one or more routing metrics to approximate the distance of the node to the root. The minimal 6TiSCH profile mandates to use the default OF0 [36, 46]. More specifically, when a node applies OF0 to select a parent among its set of neighbors, it uses a so-called *step_of_rank* value which represents the link properties to the specific neighbor. That *step_of_rank* becomes part of a normalized *rank_increase* value that is added to the Rank value received via routing broadcast messages (i.e., DIO messages) from each neighbor. The neighbor with the lowest resulting Rank value is picked as (new) preferred parent (only if the difference with the rank of the previous parent exceeds a parent switching threshold). The minimal 6TiSCH profile defines this *step_of_rank* as a function of the ETX (i.e., the estimated number of needed transmission attempts for one successful transmission) towards that neighbor. An alternative to OF0 is the MRHOF OF [70], which selects routes that minimize a metric, while using hysteresis to reduce churn in response to small metric changes. It works with additive metrics along a route, and the metrics it uses are determined by the metrics that the RPL DIO messages advertise. In the absence of advertised metrics, MRHOF also uses ETX to make parent selections.

6.3.2 Motivation

In Chapter 5, we defined a GA that acted as a centralized scheduler for multi-PHY TSCH networks by making parent, PHY and slots selections for each node in the network. This GA allowed us to implement the slot bonding problem and find solutions in near-optimal fashion. As such, it was used for the analysis of the proposed multi-PHY TSCH slot bonding technique. Due to its time complexity (see Section 5.5.3) and its centralized approach, it is not feasible to use it for real-time parent and PHY selections in TSCH networks.

Therefore in this chapter, we present a computationally efficient heuristic that allows nodes to make parent and PHY selections in a multi-PHY 6TiSCH

Algorithm 8 Procedure to map the preferred PHY per possible parent

```
1:  procedure MAPPHYPERPARENT(n, δ)
2:      m_n = map()                          ▷ for node n, map possible parents to PHY
3:      for p ∈ P_n do                       ▷ loop over every possible parent of node n, in P_n
4:          m_reliable = None                ▷ most reliable PHY to parent p
5:          for m ∈ M_p do                   ▷ find most reliable PHY
6:              if reliability(m) > reliability(m_reliable) then
7:                  m_reliable = m
            m_n[p] = m_reliable
8:          for m ∈ M_p do                   ▷ find possible faster PHY
9:              if rate(m) > rate(m_n[p]) then
10:                 if reliability(m_reliable) − reliability(m) ≤ δ then
11:                     m_n[p] = m
12:     return m_n
```

network. The algorithm is a step towards a distributed approach that can be integrated in RPL OFs, taking into account the ETX and a PHY characteristic, i.e., the data rate that is translated into the number of slots needed to bond together to transmit a packet.

6.3.3 Heuristic

The proposed heuristic aims to optimize the overall PDR of the network. Its inputs are a reliability threshold value (i.e., δ), the data rates of the available PHYs and for each node, the link reliability values for all PHYs to all the node's possible parents. The output of the heuristic is a preferred parent and a PHY, to connect to that parent, for each node in the network. To achieve its goal, the heuristic first selects, for each node, a preferred PHY for each of its possible parents. Afterwards, taking into account those PHYs for each possible parent, the heuristic minimizes the number of allocated slots from each node to the root. Therefore it considers the bonded slot length necessary for each PHY and the link reliability to that possible preferred parent. Both steps of the heuristic are described in Algorithms 8 and 9 and discussed in detail below.

In the first step of the heuristic, each node in the network, except the root (i.e., each node n in the set N_0), assigns a preferred PHY to each possible parent in its set of possible parent nodes. To do so, it considers the trade-off between reliable and/or fast PHYs as we have seen in the allocation analysis of Section 5.6.5 that both metrics are important when picking a parent. This PHY selection procedure is shown in Algorithm 8 (and this procedure is called in Algorithm 9 at lines 2 and 3). Given node n, this algorithm first searches, for every possible parent of n (i.e., in set P_n), for the most reliable PHY, $m_{reliable}$ from node n to that parent p. Afterwards, based on the known data rate of each possible PHY, the fastest PHY is selected that is within a given threshold (i.e., the parameter δ)

Algorithm 9 Heuristic to find parents and PHYs

1: **procedure** ASSIGNPARENTANDPHY(α, δ)
2: **for** $n \in N_0$ **do** ▷ for every n, assign the preferred PHY per parent
3: m_n = MAPPHYPERPARENT(n, δ)
4: converged = false
5: **while** ¬ converged **do** ▷ stop when no node changed parent anymore
6: converged = true
7: $x_0 = 0$ ▷ score of root initialized to 0
8: **for** $n \in N_0$ **do**
9: $p_n = None$ ▷ preferred parent of node n
10: $x_n = None$ ▷ score to preferred parent of n
11: **for** $p \in P_n$ **do**
12: **if** $x_p \neq None$ **then** ▷ only if possible parent has a score
13: $ETX = \frac{1}{reliability(m_n[p])}$ ▷ $ETX = \frac{1}{reliability}$
14: $x = x_p + ETX \cdot s_{m_n[p]}$ ▷ s_m is bonded slot length of PHY m
15: **if** $x_n = None \vee x < x_n$ **then**
16: $x_n = x$
17: $p_n = p$
18: converged = false ▷ there was a change
19: **return** $\{p_0, p_1, \ldots, p_{|N_0|}\}, \{m_0[p_0], m_1[p_1], \ldots, m_{|N_0|}[p_{|N_0|}]\}$

of the most reliable PHY to that parent. Formally, this means that for the selected PHY m to a possible parent the link reliability l_m is such that $l_m \geq l_{max} - \delta$ with l_{max} being the highest possible reliability among all PHYs supported by the node to that possible parent. Choosing a small δ value will lead to more reliable links between a node and its parent, but also leads to less free space in the schedule as PHYs with lower data rates need more time to transmit a packet and thus need more regular slots to be bonded together. Choosing a large δ allows less reliable PHYs to be chosen, and assuming that less reliable PHYs have faster date rates that need a shorter bonded slot to transmit a packet (this assumption is largely confirmed by the PDR results of the different IEEE 802.15.4g modulations in [17]), this will lead to more free space in the schedule. Specifically, if $\delta = 0$, the heuristic always chooses the most reliable PHY for each possible parent, while when $\delta = 1$, the heuristic always chooses the PHY with the fastest data rate. The value of δ should thus be determined empirically and can be network, application or even node-specific.

Having selected a PHY for each possible parent for each node n, from line 5 onward in Algorithm 9, the heuristic searches for the preferred parent p_n of every node n. The heuristic stops when every node has selected a preferred parent. Every node n is assigned a so-called *score* x_n and the score of the root is 0, assigned at line 7. The algorithm tries to minimize this score for each node as it represents the expected number of regular slots (i.e., not *bonded* slots) needed

to transmit a packet along the path from the node to the root. For every node n, it loops over every possible parent p of node n and if that parent was already assigned a score x_p, it calculates a new score x for node n. On line 13, this score is calculated by using the ETX value for the PHY selected for that possible parent. Afterwards on line 14, that ETX is multiplied with the bonded slot length of that PHY (i.e., s_m, the number of regular slots that need to be bonded together to transmit a packet, using PHY m), and added to the score x_p of the parent. If it is the first time the score is assigned to the node n or the score is smaller than the previous x_n, this possible parent becomes the new preferred parent (see line 15-17). By using the ETX and the bonded slot length for the score calculation, and by looking for the smallest parent score x_p, the heuristic actually aims to pick a preferred parent with a path to the root that minimizes the number of allocations in the schedule. As we have seen in the results of Section 5.6 (of the previous chapter) that more free space leads to higher PDR values, this way the heuristic tries to optimize the space in the schedule so every node can make sufficient allocations to successfully (re-)transmit their packets. On line 18, we denote that there was a change in topology as this may affect other scores of other nodes. As such, the heuristic should not converge yet and keep running until no changes happen. Finally, when converged, the heuristic returns the sets of chosen parents and PHYs for each node.

An example of the heuristic is given in Figure 6.1. Node 3 uses the heuristic to choose a preferred parent between the nodes 0, 1 or 2 and a PHY to that parent. First, it selects a PHY per parent, with $\delta = 0.1$. Afterwards, it calculates the x score using the score from each possible parent, x_p, the ETX value to that parent and the bonded slot length for the preferred PHY to that parent, s_{PHY}, to select node 2 as its new preferred parent (using preferred PHY PHY1).

6.3.4 RPL Integration

While the proposed heuristic is defined from a centralized point of view (and it can be used as a centralized scheduler), it is clearly a step towards a distributed parent and PHY selection mechanism. Here, we provide an explanation on how the different inputs of the heuristic can be obtained and how the heuristic can be integrated into an RPL OF in a distributed way (i.e., utilizing only information that is locally available at each node).

The inputs of the heuristic are the data rates of the different PHYs, the different link reliability values (or $ETX = \frac{1}{reliability}$) and the score values of possible parent nodes. The data rates of the different PHYs and the related number of regular slots that need to be bonded together to transmit a packet, s_m, are known by every node beforehand as they are characteristic for each PHY. The different reliability values could be approximated by mapping neighbor RSSI values to PDR values, based on PHY measurements carried out in the

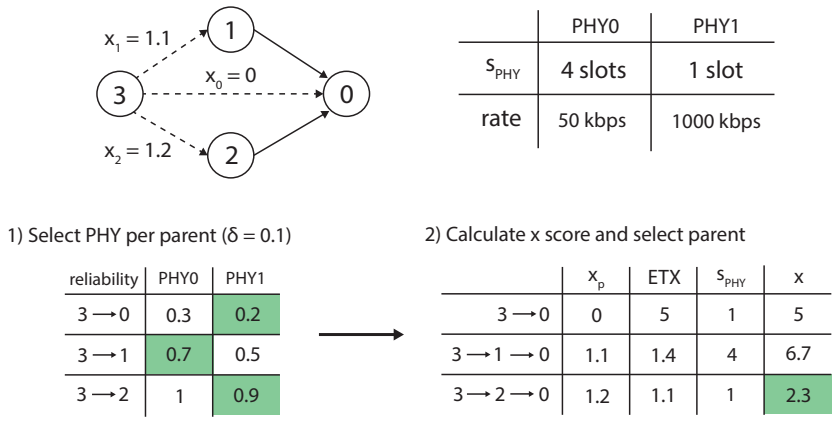


Figure 6.1: Example of node 3 using the heuristic to select its parent and PHY. First, the node selects the PHY per parent (with threshold $\delta = 0.1$) and afterwards the node calculates the x score to select parent 2.

environment where the network is being deployed. For example, Contiki-NG[1] has the option called `LINK_STATS_INIT_ETX_FROM_RSSI` to estimate the ETX from RSSI values [73]. This could be extended to support multiple PHYs. Alternatively, it is also possible to learn the link reliability over time. The link reliability can be defined as the number of ACKs over the number of transmission attempts, which can then be turned into an $ETX = \frac{num\_tx}{num\_ack}$, as it is done in Contiki-NG (i.e., option `LINK_STATS_ETX_FROM_PACKET_COUNT`) but also in the 6TiSCH OpenWSN implementation[2]. When no transmission attempts are available for a certain possible parent, a default ETX value can be chosen (e.g., representing an average link quality). In this case where link reliability is learned over time and a node has to find a fast and reliable PHY for the current preferred parent (as proposed in Algorithm 8), a node can apply a Link Quality Estimation (LQE) technique as proposed by Van Leemput *et al.* to iterate through the different PHYs [34]. The necessary score values x_p of possible parents can be distributed throughout the network using the Metric Container option in RPL DIO messages, similar to the metrics distributed for the MRHOF OF [45, 70].

Using those inputs, the MRHOF OF can be used to further integrate the heuristic, as that OF is designed to minimize the path cost based on additive metrics (that are turned into Rank values), i.e., the score values used in the heuristic. Subsequently, using MRHOF, the node will select (with hysteresis) a parent that requires less defaults slots to be scheduled along the path towards

[1] `https://github.com/contiki-ng/contiki-ng/blob/develop/os/net/link-stats.c`

[2] `https://github.com/openwsn-berkeley/openwsn-fw/blob/develop/openstack/02b-MAChigh/neighbors.c`

the root, as this is the goal of the proposed heuristic.

6.4 Slot Bonding Implementation

This section gives an overview of our proof-of-concept TSCH slot bonding implementation in the Contiki-NG firmware. The implementation is publicly available[3]. First, we discuss the development platform. Second, we discuss the different PHYs that were considered and the possible channel allocations. Finally, we discuss the used TSCH timing values used in the implementation.

6.4.1 Platform

The slot bonding proof-of-concept is implemented in the latest version[4] of the Contiki-NG firmware [73]. Contiki-NG is an open-source, cross-platform operating system for IoT devices with support for the 6TiSCH stack. The implementation is developed on the Zolertia hardware platform, more specifically the RE-Mote (revision B) platform featuring the Texas Instruments CC1200 sub-GHz low-power transceiver[5]. As the Contiki-NG firmware is ported to many different platforms, also the slot bonding implementation can be used on different hardware.

6.4.2 PHYs

The implementation supports the 2 sub-GHz PHY configurations shown in Table 6.1, i.e., the 2-GFSK modulation with 200 kHz channel spacing and 50 kbps data rate and the 4-GFSK modulation with 1667 kHz channel spacing and 1000 kbps data rate. PHY layers in the sub-GHz spectrum are used because of the better propagation characteristics compared to the 2.4 GHz spectrum [123]. Both CC1200 radio configurations are by default available in the Contiki-NG repository. The implementation can be extended to support more PHY radio configurations. As indicated in the work of Van Leemput *et al.* and as seen in Table 6.2, taking into account the strict regulation of the sub-GHz spectrum, 44 200 kHz and 4 1667 kHz channels are possible (with frequency overlap) in the 7 sub-GHz bands divided over the European Union 863-870 MHz and 873-920 MHz spectra [34, 124]. Such a channel allocation with frequency overlap actually avoids wasting bandwidth and allows for a rich frequency diversity. However, the frequency overlap can also result in inter-PHY interference and possible duty cycle saturation for certain frequency bands. The alternative is allocating channels for the different PHYs without frequency overlap (thus reserving different parts of the wireless spectrum for different PHYs), with the consequence of limiting the frequency diversity.

[3] `https://github.com/imec-idlab/tsch-slotbonding`
[4] `https://github.com/contiki-ng/contiki-ng`, commit 5fdfc98
[5] `http://www.ti.com/product/CC1200/datasheet/`

Table 6.1: The used PHYs configurations, together with their configured time slot length and number of bonded regular slots.

Modulation	Bandwidth (kHz)	Data rate (kbps)	Time slot length (µs)	Bonded nr. of slots
2-GFSK	200	50	36000	4
4-GFSK	1667	1000	9000	1

Table 6.2: Possible channel allocations for 200 kHz and 1000 kHz channels in the European sub-GHz frequency band, as listed by Van Leemput *et al.* [34].

Start Freq.	End Freq.	Bandwidth	200 kHz	1000 kHz
863 MHz	868 MHz	5000 kHz	25	3
868 MHz	868.6 MHz	600 kHz	3	0
868.7 MHz	869.2 MHz	500 kHz	2	0
869.4 MHz	869.65 MHz	250 kHz	1	0
869.7 MHz	870 MHz	300 kHz	1	0
874 MHz	874.4 MHz	400 kHz	2	0
917.4 MHz	919.4 MHz	2000 kHz	10	1
			44	**4**

6.4.3 Timing Values

As introduced in Chapter 5, TSCH slot bonding bonds multiple *regular* time slots into a bonded slot that is long enough to transmit or receive a frame of 128 bytes (125 data bytes, 2 CRC bytes and 1 byte frame length), given the data rate of the selected PHY. As such, each bonded slot is tailored to the requirements of the specific PHY and thus has the advantage of limiting the waste of airtime resources.

As described in Section 6.4.2, we use the 2 available CC1200 PHYs with data rates of 50 kbps and 1000 kbps. The differences between the two PHYs and their data rate is reflected in the total time slot lengths and the timing values for the different states within a time slot, as defined in the radio configurations of these PHYs. Brachmann *et al.* list these different timing values and how they can be determined [111]. The largest difference between the 2 PHYs is reflected in the transmission length of the data frame due to the difference in data rate (i.e., in kbps), as the transmission length equals $\frac{128\times8}{datarate}$ milliseconds. Using the timings as defined in their original radio configurations, the total time slot lengths are 31.46 ms and 5.808 ms for 50 kbps and for 1000 kbps respectively. However, we also have to include extra time at the start of the slot to reconfigure the radio for the appropriate PHY. Similar to Brachmann *et al.*, we add 3 ms for this

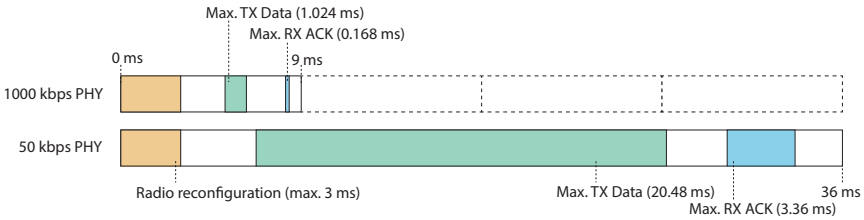


Figure 6.2: Illustration of the slot timings for the 1000 kbps and 50 kbps (bonded) slots.

reconfiguration per time slot, bringing the total time slot lengths to 34.46 ms and 8.808 ms. As slot bonding requires bonding multiple slots together to form a longer time slot for PHYs with slower data rates, the total time slot length of the fastest PHY should be a common divisor of the time slot lengths of the slower PHY(s). Therefore, we extended the slack time at the end of both time slots to end up with total lengths of 9 ms and 36 ms, as shown in Table 6.1. This means that the *regular* time slots in the TSCH schedule have a length of 9 ms (meaning that the TSCH ASN is incremented every 9 ms) and are used by the 1000 kbps PHY. When allocating a slot for the 50 kbps PHY, 4 regular slots of 9 ms will be bonded together. Figure 6.2 illustrates the differences between the time slots of the 2 chosen PHYs.

6.5 Evaluation

In this section we evaluate the proposed parent and PHY selection heuristic. First, we present our experiment setup and methodology. Afterwards, we use the 6TiSCH simulator to compare the heuristic against the near-optimal approach, proposed in Chapter 5. Finally, we evaluate the heuristic on a real hardware office testbed using the proposed slot bonding implementation.

6.5.1 Experiment Methodology & Setup

We first describe the experiment methodology and setup for the simulation experiments, afterwards we describe the testbed experiments.

6.5.1.1 Simulator Experiments

In the simulation experiments, we compare the proposed parent and PHY heuristic of Section 6.3 to the near-optimal GA scheduler defined in Chapter 5. The experiment process of the GA is similar to the 4-step process used in Section 5.6.1. First, the network topologies were generated by the 6TiSCH simulator that was introduced in Chapter 2 [83]. Second, those topologies were fed to the GA that solves the slot bonding problem in near-optimal fashion. Both the simulator and

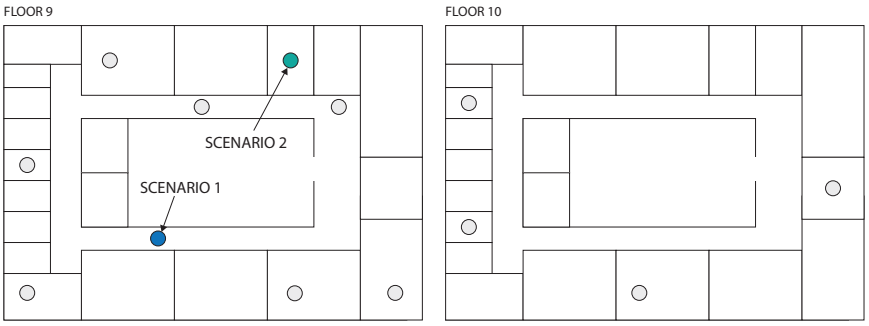


Figure 6.3: The two floors of the hardware testbed with the 13 used nodes illustrated at their locations. We considered 2 scenarios of 12 nodes: scenario 1 includes the 11 grey nodes and the blue node while scenario 2 includes the 11 grey nodes and the green node.

the GA are publicly available[6]. Third, the best possible solution found by the GA, that includes the selection of a parent, PHY and a number of slots for each node, was used as input for the algorithm defined in Section 5.5.2 to generate a TSCH schedule for all nodes in the network. Fourth, the resulting scheduling solution is used as a centralized schedule to make static schedule allocations in the TSCH experiments using the 6TiSCH simulator. For a more elaborate description and illustration of this experiment setup, we refer the reader back to Section 5.6 and Figure 5.6 in Chapter 5.

The proposed heuristic only makes parent and PHY selections and does not allocate (bonded) slots. Therefore, using the network topology created by the 6TiSCH simulator in the first step of the experiment process, we let the proposed heuristic make the parent and PHY selections and fix these selections, and afterwards let the GA find the best slot allocation solution for those parent and PHY choices. The reason for keeping the GA scheduler for the slot allocations, is that we want to fairly compare the PHY and parent selection, without having influence from (sub-optimal) slot allocation. This implementation is publicly available[7].

6.5.1.2 Testbed Experiments

The proposed heuristic is also validated on the *imec Wireless OfficeLab*, using the TSCH slot bonding proof-of-concept discussed in Section 6.4 [125]. The testbed is located in an office environment spanning different floors of 805 m^2 separated, with reinforced concrete, with a total of 150 sensor nodes of which a subset of 13 nodes was used as illustrated in Figure 6.3. The deployed sensor nodes are the Zolertia RE-motes [82].

[6] `https://github.com/imec-idlab/tsch-slotbonding-ga-simulator`
[7] `https://github.com/imec-idlab/tsch-slotbonding`

The testbed evaluation was a 3-step process as illustrated in Figure 6.4. First, using the slot bonding implementation, we monitored the link reliability between all nodes, in both directions, for the 50 kbps and 1000 kbps PHYs (as listed in Section 6.4.2). Every link was monitored for 300 s, transmitting a packet of 127 bytes every second, and its reliability was defined as the average $\frac{num\_ACKs}{num\_TX}$ of 5 consecutive observations, each over a time span of 60 s. The monitored link reliability values are publicly available[8]. Links between nodes that did not connect or became disassociated during the experiment, were assigned a reliability value of 0. We considered 2 different scenarios, each with 12 nodes as illustrated in Figure 6.3. It turned out that in normal conditions, the 1000 kbps PHY was able to connect (almost) all nodes with a very good reliability. Therefore, to better emulate an industrial environment in which the reliability of different network links can heavily vary and/or some nodes can not connect with certain PHYs, we used different transmission powers for the 2 PHYs and disabled a part of the 1000 kbps PHY links in both scenarios. For the 50 kbps PHY, the transmission power was set to 14 dBm, while for the 1000 kbps PHY it was set to 0 dBm. Additionally, we disabled uniformly at random 70 % of the 1000 kbps PHYs links of each node (i.e., assigning links a reliability value of 0). The goal of the monitoring step was to determine the link reliability values of all nodes to each other for each PHY, as these will be used by the heuristic to make the parent and PHY selections. While this monitoring step simplified our evaluation of the heuristic, in an integrated solution this could be replaced with run-time statistics as explained in Section 6.3.4. As such, afterwards these reliability values were fed, together with the slotframe information and the selected set of nodes (including a randomly selected root node), to the GA scheduler. In case of the heuristic, the GA only decided the slot allocations as the heuristic determined the parent and PHY selection for each node. Finally, the resulting network topologies with selected parents and PHYs and the TSCH schedules were configured on all selected nodes in the hardware testbed and the results were monitored for 300 ms.

The slot bonding implementation that we deployed on the hardware testbed has a channel allocation of 3 channels of 200 kHz and 2 channels of 1667 kHz for the 50 kbps and 1000 kbps PHY respectively, without frequency overlap in the 863-868 MHz band, as shown in Figure 6.5. We have limited the number of channels to speed up the network bootstrap process. We considered 2 different slotframe lengths of 261 ms (i.e., 29 slots of 9 ms) and 423 ms (i.e., 47 slots of 9 ms). The nodes can actually only use 17 and 36 slots respectively in each slotframe: we included 8 slots to facilitate 2 shared bonded slots (of 4 regular slots each) that use the 50 kbps PHY and padded the resulting length with unused extra slots to become a prime number of slots (to achieve pseudo-random channel hopping). The shared slots are placed at the beginning of each slotframe and used to broadcast the TSCH EB messages that advertise the network (every 7 seconds). We disabled the 6P layer and applied the Contiki-NG

[8] `https://github.com/imec-idlab/officelab-reliabilities`

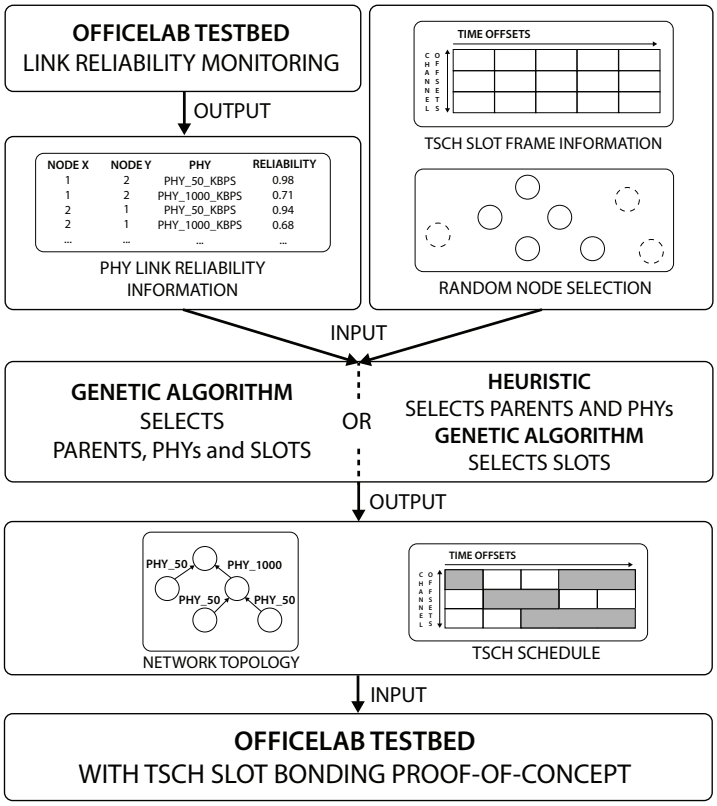


Figure 6.4: Flow diagram illustrating the testbed evaluation process. The testbed link monitoring information is used as input for the heuristic (with the GA for slot allocations) or the GA, and the outcomes of those schedulers is loaded on the sensor nodes in the testbed.

NullNet option, meaning there is no network or any higher layer functionality. All network topologies and schedule allocations are statically configured at the MAC layer, using the parent and PHY selections and the TSCH schedule provided by the heuristic and/or GA. The queue size is set to 8 packets. The maximum number of allowed transmission opportunities per packet, is set to 4, as advised by the 6TiSCH Minimal Configuration [36]. Every node transmits a packet of 127 bytes every slotframe.

The GA configuration (i.e., operators, number of generation and mutation and cross-over probabilities) is the same as described in Section 5.6.1.2. However, the GA (and the proposed heuristic) now use the 50 kbps and 1000 kbps PHYs. Subsequently, the feasibility algorithm that determines the TSCH sched-

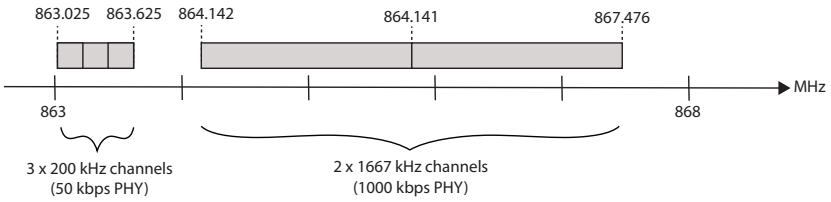


Figure 6.5: The channel allocation for the 50 kbps and 1000 kbps PHYs in the 863-868 MHz band, as used in the evaluation. The former uses 3 channels of 200 kHz while the latter uses 2 channels of 1667 kHz.

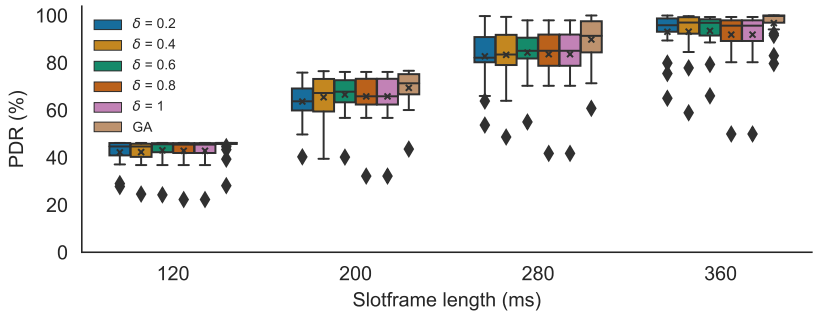


Figure 6.6: PDR simulation results for different δ values for different slotframe lengths, comparing the heuristic with the GA scheduler. The *x* denotes the mean value.

ule, defined in Section 5.5.2, was extended to be able to allocate bonded slots in the non-overlapping channels reserved for the respective PHYs. Additionally, no interference between any of the nodes was allowed.

The simulator results are averaged over 20 iterations, while the testbed results are averaged over 12 iterations.

6.5.2 Simulator Results

First, we compare the heuristic for different δ values to the GA scheduler in TSCH simulation experiments with 14 nodes. The results are shown in Figure 6.6. The PDR results are compared for different slotframe lengths of 120 ms, 200 ms, 280 ms and 360 ms, and δ values ranging from 0.2 to 1.0. When $\delta = 1.0$, the heuristic will pick for each possible parent the fastest available PHY. It is clear that the performance of the heuristic approximates the performance of the GA for all slotframe lengths. The PDRs of the GA scheduler were 44.6 % (120 ms), 69.4 % (200 ms), 89.8 % (280 ms) and 96.8 % (360 ms). While the performance of all δ values is very similar for all slotframe lengths, when $\delta = 0.6$

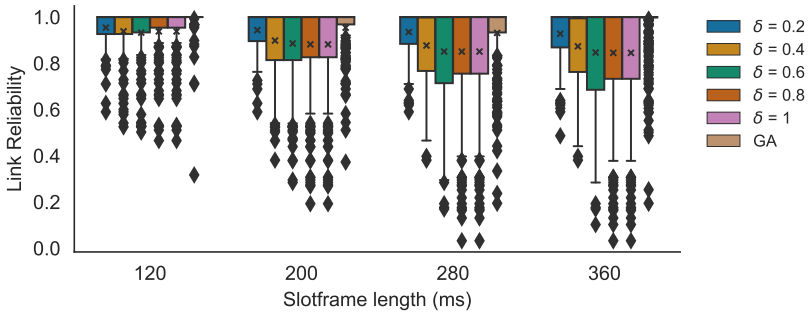


Figure 6.7: Mean link reliability values for different slotframe lengths, for different δ values of the heuristic. The x denotes the mean value.

the PDRs were the highest with 43.1 % (i.e., a decrease of 3.4 % of the GA's performance), 66.6 % (i.e., a 4 % decrease), 84.3 % (i.e., a 6.1 % decrease) and 93.5 % (i.e., a 3.4 % decrease) respectively. Figure 6.7 shows the reliability values for all links for the different δ values (with the mean reliability value denoted by an x). As expected (see Algorithm 8), the results show that with increasing δ values, the average link reliability value decreases. This decreasing reliability has an effect on the number of propagation failures, the number of needed retransmissions and consequently the number dropped packets (because of too many retransmission attempts). However, faster PHYs also need less transmission time and thus shorter bonded slots (i.e., less regular slots per bonded slot), leaving more free space in the schedule for extra retransmission slots or for other nodes to make slot allocations. As such, in this case the best trade-off between using shorter/longer bonded slots and lower/higher reliability values was found when $\delta = 0.6$. This was confirmed by $\delta = 0.6$ consistently achieving the lowest total number of packet drops, including the dropped packets because of exceeding the retransmission threshold, but also the packets that were dropped immediately at the transmitter because there were not enough allocated slots to empty the queue on time.

6.5.3 Testbed Results

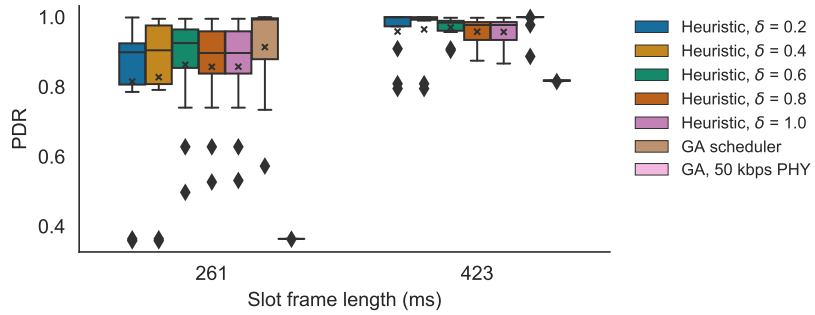
Here we evaluate the performance of the heuristic on the testbed. First, we select an appropriate δ value for the heuristic by comparing theoretical PDR values using the monitored reliability data. Afterwards, we look at the PHY allocations for the different slotframe lengths. Finally, we compare the performance of the heuristic (with the selected δ) to that of the near-optimal GA scheduler using our proposed slot bonding implementation on the testbed.

6.5.3.1 δ Selection

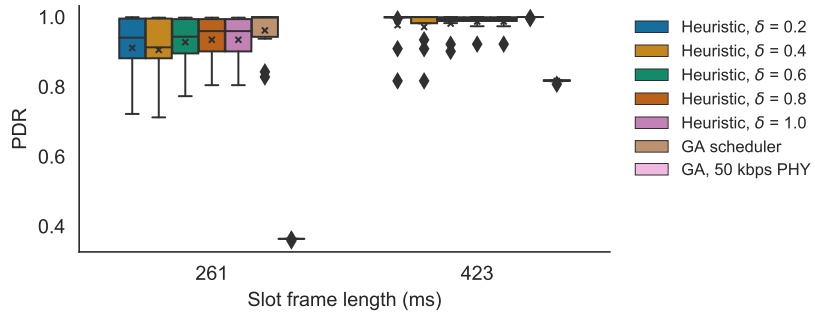
First, we search for the optimal δ values for the heuristic, using the monitored link reliability values of the testbed. We used the expected number of delivered packets calculation to calculate the expected PDR values, defined in Section 5.4.1. Figure 6.8 show the comparison between different δ values, for the 2 network scenarios as discussed in Section 6.5.1.2, each for the 2 different slotframe lengths 261 ms and 423 ms. It is clear from the results that the 261 ms slotframe is saturated, with all average PDR values being below 1. For scenario 1, with both the 261 ms and the 423 ms slotframe, the best performing δ is 0.6 with the PDRs being 0.86 and 0.97, while the PDRs of the GA were 0.91 and 0.99. So similar to the simulator results, the heuristic finds the best trade-off between using fast PHYs with shorter bonded slots and lower reliability values when $\delta = 0.6$. When $\delta > 0.6$, the lower reliability actually starts deteriorating the overall PDR. For scenario 2, the heuristic actually performs best when $\delta = 0.8$ with the average PDRs being 0.93 and 0.98 compared to the 0.96 and 1.0 PDRs of the GA. So again, the performance of the heuristic is very close to that of the centralized GA scheduler. The results show that the optimal δ is different for scenario 1 (i.e., $\delta = 0.6$) and scenario 2 (i.e., $\delta = 0.8$) (although the differences between the PDR results are small for both δ values in both scenarios). This is explained by the fact that the enabled 1000 kbps PHY links in scenario 1 showed lower link reliability values, when compared to the values of the enabled links in scenario 2 (as explained in Section 6.5.1.2, a random subset of the 1000 kbps links was disabled in each scenario). Therefore, in scenario 1, a $\delta > 0.6$ affects the PDR results more severely than in scenario 2 as it introduces more less reliable links in the topology. Figure 6.9 shows the average number of times per iteration each PHY was used for a link for the GA scheduler and the heuristic with increasing δ values for scenario 2 for both slotframe lengths. We do not show the PHY allocations of scenario 1 as these were similar to those of scenario 2. Only links of nodes for which a packet could reach the root (i.e., there is at least one allocated slot for every link along the path towards the root) are shown. While the the heuristic clearly prefers the faster 1000 kbps PHY (i.e., if the link is reliable enough, as defined in Algorithm 8), the GA also seems to focus on allocating 50 kbps PHYs. In case of the heuristic, when δ increases, the number of 1000 kbps PHYs also increases and the number of used 50 kbps PHYs decreases. When comparing both slotframes, it is clear that for the longer slotframe length of 423 ms, it is possible to allocate more 50 kbps PHYs because there is more free space, leading to higher PDR values, as observed in Figure 6.8.

The PHY allocation and theoretical PDR results show that the heuristic generally performs well in terms of PDR but that the optimal δ value is dependent on the specific environment. Simultaneously, the results also clearly indicate that the heuristic does not necessarily need the most reliable links to perform well, but can benefit from using more faster and less reliable PHYs that leave more free space for possible extra slots for retransmissions or other nodes' allocations.

Additionally, Figure 6.8 also shows the benefits of using multiple PHYs as



(a) Scenario 1 with 12 nodes.



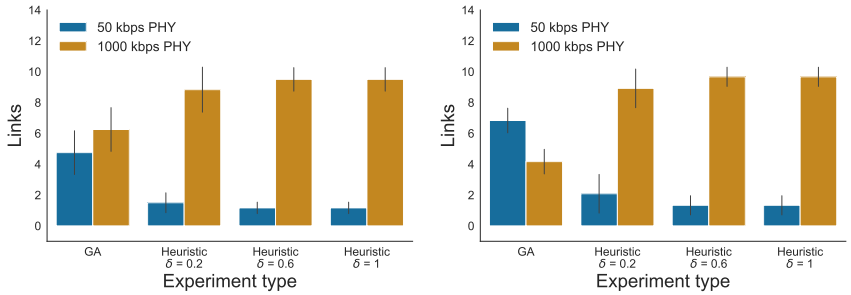
(b) Scenario 2 with 12 nodes.

Figure 6.8: PDR values for different δ values for 261 ms and 423 ms slotframe lengths, comparing the heuristic with the GA scheduler and the GA scheduler that can only use the 50 kbps PHY.

when only allowing the 50 kbps PHY results in PDRs of 0.36 and 0.82 for scenario 1 and PDRs of 0.36 and 0.82 for scenario 2, for the 261 ms and 423 ms slotframe respectively. This is explained by the fact that the 50 kbps PHY requires bonded slots with a length of 4 regulars slots, thereby occupying a lot of space in the schedule per transmission attempt and not allowing sufficient free space for extra slots or other nodes to allocate slots. We do not show the results for only using the 1000 PHY as that did not allow for fully connected topologies.

6.5.3.2 Testbed Validation

We have carried out experiments using the nodes of scenario 2 on the hardware testbed, using the proposed slot bonding implementation. Figure 6.10 shows the results for the heuristic (with $\delta = 0.8$), the multi-PHY GA and the single-PHY GA that can only use the 50 kbps PHY.



(a) 261 ms slotframe length. (b) 423 ms slotframe length.

Figure 6.9: Average number of PHY allocations per iteration for the GA scheduler and the heuristic with different δ values, for scenario 2. The error bars in the figures represent the standard deviation.

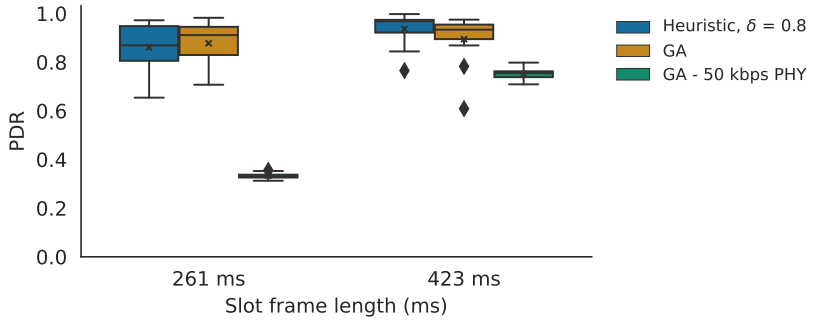


Figure 6.10: The PDRs values for the testbed experiments, showing the results for the heuristic, the GA and the GA that can only use the 50 kbps PHY. The x denotes the mean value.

In general, it is observed that the testbed results approximate the theoretical values shown in Figure 6.8b. As expected, with the longer slotframe length, the mean PDR values also increase compared to the shorter slotframe length. More specifically, in the case of the 261 ms slotframe length, the GA performs better than the heuristic with PDRs of 0.88 and 0.86 respectively, meaning there is only 2.3 % decrease in performance. The GA that can only use the 50 kbps PHY is outperformed by its multi-PHY counterparts as it only achieves a PDR of 0.33 (i.e., the GA and the heuristic increase on this performance with 166.7 % and 160.6 % respectively). When using a 423 ms slotframe length, the heuristic actually performs slightly better than the GA with PDRs of 0.94 and 0.88, and both again outperform the single-PHY GA that has a PDR of 0.75 (i.e., the GA and the heuristic increase on this performance with 25.3 % and

17.3 % respectively). The heuristic performing better than the multi-PHY GA is explained by the observation of link reliability values during these multi-hop experiments which differed from the reliability values observed during the monitoring phase (on which the parent, PHY and slot allocation was based). More specifically, the monitoring results showed better performance mainly for the 50 kbps PHY compared to the values observed during the multi-hop experiments. As the GA more often uses 50 kbps PHY than the heuristic does, the PDRs of the GA got more negatively affected as a result. As such, these results actually motivate the integration of the heuristic in a routing protocol's objective function with real-time link reliability monitoring to dynamically adapt to varying link conditions, as discussed in Section 6.3.4. Overall, the results clearly show the efficiency of the proposed parent and PHY selection heuristic.

6.6 Conclusion

This chapter further explored the support of multiple PHY layers in a single IEEE 802.15.4e TSCH slot bonding network. More specifically, we have investigated parent and PHY selection in such a multi-PHY network. We first proposed a computationally efficient PHY and parent selection heuristic that can be integrated into a distributed routing protocol. The performance of this heuristic was initially evaluated via simulation and theoretical results, by comparing it against the near-optimal, but computationally complex centralized scheduler proposed in Chapter 5. The results confirmed that the heuristic is capable of approximating the PDR results of the centralized scheduler. Second, we also proposed an easy-to-configure TSCH slot bonding implementation in the latest version of the Contiki-NG IIoT operating system. Subsequently, the slot bonding implementation was used to confirm the PDR performance of the proposed heuristic in a real sensor testbed. In this testbed experiment, the proposed heuristic was at most 2.3 % worse than the GA in terms of PDR. Both approaches outperformed the single-PHY solution, increasing PDR results up to more than 160 %.

Chapter 7

Conclusion

7.1 Summary and Contributions

This thesis focused on the application of the IoT in industry, also called IIoT. More specifically, I focused on the improvement of IEEE 802.15.4e TSCH that has become an increasingly popular wireless solution for IIoT applications. Its combination of a time-synchronized approach and frequency diversity has been proven to be highly reliable and capable of low-power operation. Despite its proven track record, still a lot of research has to be carried out to investigate how these networks can be optimized for IIoT applications. The contributions in this book aimed at answering 3 major research questions related to this topic:

1. **How do we precisely characterize the TSCH energy consumption?** An accurate energy consumption model for both the sub-GHz and 2.4 GHz frequency bands, using state-of-the-art IIoT hardware, has been presented. The model includes an elaborate and up-to-date set of time slots and states and accurately models variable packet sizes. The accuracy of the model was experimentally verified by comparing the measurements to the modelled values. For both the comparison between modelled and measured time slots and a small-scale network comparison experiment, the average error was less than 3 %. Additionally, the energy model allowed us to compare the energy consumption of sub-GHz and 2.4 GHz frequency bands in a simulation environment. The presented model allows researchers to quickly characterize the energy consumption of new network topology formations, new TSCH scheduling approaches or higher-layer algorithms/applications that should satisfy the low-power operation requirements of IIoT.

2. **How do we achieve low-latency communication in a TSCH network?** Minimizing the communication delay while maintaining the TSCH low-power operation is an important requirement for (time-critical) IIoT ap-

plications. Therefore, I formally stated the problem of minimal-latency scheduling of recurrent transmissions, such as sensor data with a fixed reporting interval. Afterwards, I presented a new TSCH distributed scheduling approach, called Recurrent Low-Latency Scheduling Function (ReSF). This scheduling approach minimizes the latency of recurrent traffic in multi-hop networks while keeping the energy consumption to a minimum. ReSF builds a minimal-delay path from source to root and activates the recurrent cells on this path only when traffic is expected, and deactivates them immediately afterwards. Numerous simulation results showed the significant latency improvement - up to 80 % in static traffic scenarios - of this approach over traditional TSCH scheduling approaches. The presented scheduling approach is thus proven capable of improving the responsiveness of IIoT networks.

3. **How do we use multiple PHYs in TSCH to increase the reliability?** While TSCH inherently introduces frequency diversity to enhance its reliability, its performance is limited by the characteristics of the chosen PHY layer. Therefore, I presented a method to use multiple PHYs within a single TSCH network, called *slot bonding*. The slot bonding approach allows improving the network's number of delivered packets by efficiently adapting the allocated resources to each PHY's requirements. First, I formally described the complex slot bonding problem. Afterwards, a near-optimal approach was implemented to solve and analyse the problem and, in combination with TSCH network simulation, prove the scalability advantage of slot bonding in terms of packet delivery ratio. Additionally, a heuristic was provided, that could be integrated in a routing protocol's distributed objective function, to replace the computationally complex near-optimal approach. It allowed slot bonding nodes to select an appropriate parent and PHY to optimize the network's number of delivered packets. Using a slot bonding enabled TSCH network on a real hardware testbed, it was shown that the heuristic performed at most 2.3 % worse than the near-optimal centralized scheduler in terms of delivered packets. These results contributed to the proof that the industrial network's overall reliability could clearly benefit from using the slot bonding approach to introduce multiple PHY layers in a single network to adapt the PHY layer to the link's propagation characteristics.

As such, the answers to these research questions, as investigated in this thesis, have contributed to the further improvement of the domains of **energy consumption**, **responsiveness** and **reliability** when applying IEEE 802.15.4e TSCH in an IIoT context.

7.2 Future Work

While this thesis offers several contributions towards an energy-efficient, responsive and reliable IIoT, research on this topic is still ongoing and several open issues remain to be solved.

- The popularity of TSCH and 6TiSCH drives the implementation of the 6TiSCH stack into a plethora of new hardware platforms. The presented **energy model** can be **extended to support** these **other platforms**. While maintaining the same level of accuracy is a burdensome task (as numerous measurements have to be done to include all states and time slots), one can simplify the work by focusing on states that are expected to consume the most energy at the cost of a slight decrease in accuracy. Furthermore, the consumption of the CPU and radio does not necessarily have to be measured as theoretical (but less accurate) values can be found in the data sheet of the manufacturer. Such an extension would allow future research to compare different platforms to each other when evaluating a new algorithm or IIoT application.

- As investigated and confirmed in this thesis, using multiple PHYs in the same TSCH networks seems a promising direction to further improve the network's reliability. Therefore, it also important to further research the **energy consumption** impact of these **new PHY layers**. Different data rates lead to different transmission and reception state durations and different modulation computational complexities lead to different current consumption. Updating the presented energy model such that it is able to characterize the energy consumption of the most important PHYs of the IEEE 802.15.4g amendment (e.g., SUN-FSK, SUN-OQPSK and SUN-OFDM) would profoundly help new multi-PHY TSCH research.

- Autonomous scheduling approaches show interesting advantages for TSCH networks as they make resource negotiation traffic unnecessary by using available routing information. **ReSF**, the proposed distributed scheduling function, manages to significantly decrease network latency, might also profit from **an autonomous approach**. An interesting first step towards this goal might be the Escalator scheduling function that manages to daisy-chain its autonomous resources [64]. By additionally taking into account the recurrent information and distributing it in routing management traffic, no extra management traffic would be necessary and the energy consumption can be further decreased by deactivating autonomous resources when they are not needed.

- The merits of using TSCH slot bonding for multi-PHY networks was proven in Chapter 5. However, the slot bonding analysis primarily focused on one particular SUN-OFDM option for which the different MCSs (i.e., the PHYs) needed the same bandwidth. When combining even more PHYs from the

IEEE 802.15.4g amendment, with different bandwidth requirements, this assumption will not hold anymore. Therefore, we also briefly introduced the concept of **channel bonding** that can bond different channels together so also PHYs needing wider channels can be supported. The **efficiency** of this channel bonding approach should be **investigated**. It should be compared to the approach used in the implementation of Chapter 6 that assigned different parts of the available frequency band to the different PHYs without overlap (resulting in less available wireless spectrum per PHY).

- The **heuristic** introduced in Chapter 6 allows slot bonding nodes to select a parent and PHY. Such a selection strategy should be integrated with existing or new **scheduling approaches** and **routing protocol objective functions**. However, to achieve an optimal performance, such an integration should carefully consider the trade-off between the current state of the node (e.g., amount of traffic, battery state and free schedule space) and the different characteristics of each possible PHY (e.g., achievable throughput, energy consumption and resource duration). In addition, while combining different PHYs with different transmission lengths, a node must closely monitor that it complies with the frequency band duty cycle regulations.

Appendices

Appendix A

TSCH Energy Modeling Results

The content of this appendix is partially based on:

- Glenn Daneels, Esteban Municio, Bruno Van de Velde, Glenn Ergeerts, Maarten Weyn, Steven Latré, and Jeroen Famaey. "Accurate Energy Consumption Modeling of IEEE 802.15.4e TSCH Using Dual-Band OpenMote Hardware." In Sensors, no. 2 (2018): 437. [Impact Factor: 3.031]

A.1 Introduction

This appendix lists additional research results obtained in Chapter 3. As such, the different states within a time slot and their resulting CPU and radio states are listed for different time slots in Section A.2, the durations of different states within different time slots are shown in Section A.3 and Section A.4 compares the measured current drawn over time with the model for different time slots.

A.2 Time Slot States

This section lists the different states and accompanying CPU and radio states for the `RxDataTxAck`, `TxData`, `RxData`, `RxIdle`, `Sleep` and `TxDataRxNoAck` time slots in Tables A.1 to A.6 respectively. These different time slots are discussed in detail in Section 3.3.

Table A.1: States in a RxDataTxAck slot.

State	CPU State	Radio State
RxDataOffsetStart	Active	Sleep
RxDataOffset	Sleep	Sleep
RxDataPrepare	Active	Idle
RxDataReady	Sleep	Idle
RxDataListenStart	Active	Idle
RxDataListen	Sleep	Listen
RxDataStart	Active	RX
RxData	Sleep	RX
TxAckOffsetStart	Active	Idle
TxAckOffset	Sleep	Sleep
TxAckPrepare	Active	Idle
TxAckReady	Sleep	Idle
TxAckDelayStart	Active	Idle
TxAckDelay	Sleep	TX
TxAckStart	Active	TX
TxAck	Sleep	TX
RxProc	Active	Sleep
Sleep	Sleep	Sleep

A.3 Time Slot State Durations

The duration of the different states within different time slots are listed in this section. The state durations for the RxDataTxAck, TxData, RxData, RxIdle, Sleep and TxDataRxNoAck time slots are shown in Tables A.7 to A.12 respectively. Section 3.4.2 discusses these results in detail.

A.4 Slot Measurements and Model Comparison

Figures A.1 and A.2 show the current drawn over time according to both the measurements and the model, for the TxData, RxData, RxIdle and Sleep time slots, when using the CC2538 and CC1200 radio respectively. These results are discussed in detail in Section 3.5.1.

Table A.2: States in a TxData slot.

State	CPU State	Radio State
TxDataOffsetStart	Active	Sleep
TxDataOffset	Sleep	Sleep
TxDataPrepare	Active	Idle
TxDataReady	Sleep	Idle
TxDataDelayStart	Active	Idle
TxDataDelay	Sleep	TX
TxDataStart	Active	TX
TxData	Sleep	TX
TxProc	Active	Sleep
Sleep	Sleep	Sleep

Table A.3: States in a RxData slot.

State	CPU State	Radio State
RxDataOffsetStart	Active	Sleep
RxDataOffset	Sleep	Sleep
RxDataPrepare	Active	Idle
RxDataReady	Sleep	Idle
RxDataListenStart	Active	Idle
RxDataListen	Sleep	Listen
RxDataStart	Active	RX
RxData	Sleep	RX
RxProc	Active	Idle
Sleep	Sleep	Sleep

Table A.4: States in a RxIdle slot.

State	CPU State	Radio State
RxDataOffsetStart	Active	Sleep
RxDataOffset	Sleep	Sleep
RxDataPrepare	Active	Idle
RxDataReady	Sleep	Idle
RxDataListenStart	Active	Idle
RxDataListen	Sleep	Listen
RxProc	Active	Sleep
Sleep	Sleep	Sleep

Table A.5: States in a `Sleep` slot.

State	CPU State	Radio State
`SleepStart`	Active	Sleep
`Sleep`	Sleep	Sleep

Table A.6: States in a `TxDataRxNoAck` slot.

State	CPU State	Radio State
`TxDataOffsetStart`	Active	Sleep
`TxDataOffset`	Sleep	Sleep
`TxDataPrepare`	Active	Idle
`TxDataReady`	Sleep	Idle
`TxDataDelayStart`	Active	Idle
`TxDataDelay`	Sleep	TX
`TxDataStart`	Active	TX
`TxData`	Sleep	TX
`RxAckOffsetStart`	Active	Sleep
`RxAckOffset`	Sleep	Sleep
`RxAckPrepare`	Active	Idle
`RxAckReady`	Sleep	Idle
`RxAckListenStart`	Active	Idle
`RxAckListen`	Sleep	Listen
`TxProc`	Active	Sleep
`Sleep`	Sleep	Sleep

Table A.7: State durations in the RxDataTxAck time slot with a total length of 15 ms and s being the packet size in bytes.

State	Duration (µs)	
	CC2538	**CC1200**
RxDataOffsetStart	126	126
RxDataOffset	1567	1567
RxDataPrepare	38	676
RxDataReady	969	331
RxDataListenStart	17	58
RxDataListen	1283	1242
RxDataStart	17	15
RxData	$(3+s)\times 32-17$	$(3+s)\times 32-15$
TxAckOffsetStart	$126+(s\times 0.91)$	$362+(s\times 8.439)$
TxAckOffset	$3443-(s\times 0.91)$	$2810-(s\times 8.439)$
TxAckPrepare	153	930
TxAckReady	518	77
TxAckDelayStart	17	58
TxAckDelay	349	369
TxAckStart	16	15
TxAck	880	881
RxProc	94	135
Sleep	$5308-(s\times 32)$	$5267-(s\times 32)$

Table A.8: State durations in the TxData time slot with a total length of 15 ms and s being the packet size in bytes.

State	Duration (µs)	
	CC2538	**CC1200**
TxDataOffsetStart	105	105
TxDataOffset	1515	1454
TxDataPrepare	$60+(s\times 0.875)$	$738+(s\times 8.152)$
TxDataReady	$1954-(s\times 0.875)$	$1276-(s\times 8.152)$
TxDataDelayStart	17	58
TxDataDelay	349	369
TxDataStart	16	16
TxData	$(3+s)\times 32-16$	$(3+s)\times 32-16$
TxProc	72	109
Sleep	$10{,}832-(s\times 32)$	$10{,}795-(s\times 32)$

Table A.9: State durations in the RxData time slot with a total length of 15 ms and s being the packet size in bytes.

State	Duration (µs)	
	CC2538	**CC1200**
RxDataOffsetStart	126	126
RxDataOffset	1567	1567
RxDataPrepare	38	676
RxDataReady	969	331
RxDataListenStart	17	58
RxDataListen	1283	1242
RxDataStart	17	15
RxData	$(3+s)\times 32-17$	$(3+s)\times 32-15$
RxProc	$198+(s\times 0.91)$	$488+(s\times 8.439)$
Sleep	$10{,}706-(s\times 31.09)$	$10{,}416-(s\times 23.561)$

Table A.10: State durations in the RxIdle time slot with a total length of 15 ms.

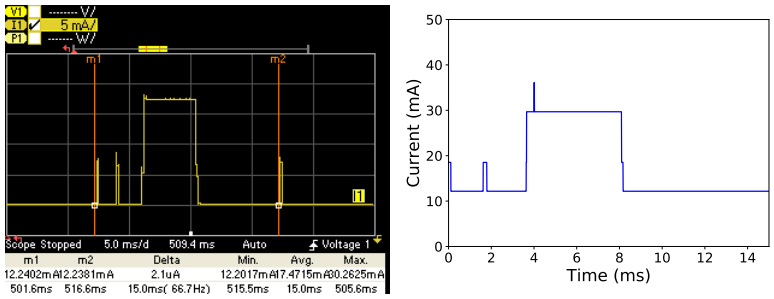
State	Duration (µs)	
	CC2538	**CC1200**
RxDataOffsetStart	126	126
RxDataOffset	1567	1567
RxDataPrepare	38	676
RxDataReady	969	331
RxDataListenStart	17	58
RxDataListen	2583	2542
RxProc	25	118
Sleep	9675	9582

Table A.11: State durations in the Sleep time slot with a total length of 15 ms.

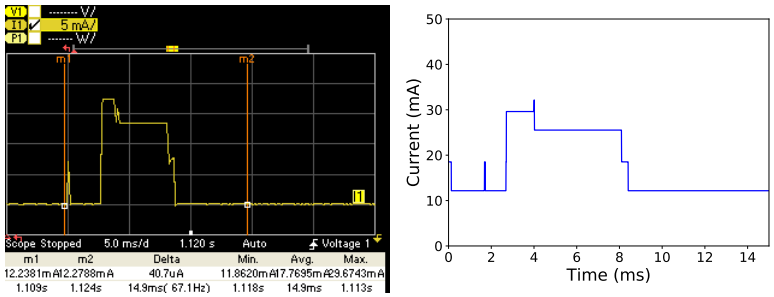
State	Duration (µs)	
	CC2538	**CC1200**
SleepStart	57	57
Sleep	14.943	14.943

Table A.12: State durations in the `TxDataRxNoAck` time slot with a total length of 15 ms and s being the packet size in bytes.

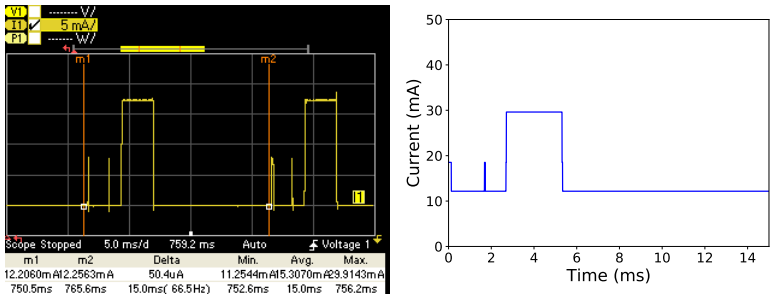
State	Duration (µs)	
	CC2538	**CC1200**
`TxDataOffsetStart`	105	105
`TxDataOffset`	1515	1454
`TxDataPrepare`	$60+(s\times 0.875)$	$738+(s\times 8.152)$
`TxDataReady`	$1954-(s\times 0.875)$	$1276-(s\times 8.152)$
`TxDataDelayStart`	17	58
`TxDataDelay`	349	369
`TxDataStart`	16	16
`TxData`	$(3+s)\times 32-16$	$(3+s)\times 32-16$
`RxAckOffsetStart`	32	75
`RxAckOffset`	3769	3116
`RxAckPrepare`	38	587
`RxAckReady`	267	328
`RxAckListenStart`	17	58
`RxAckListen`	983	942
`TxProc`	44	137
`Sleep`	$5754-(s\times 32)$	$5661-(s\times 32)$



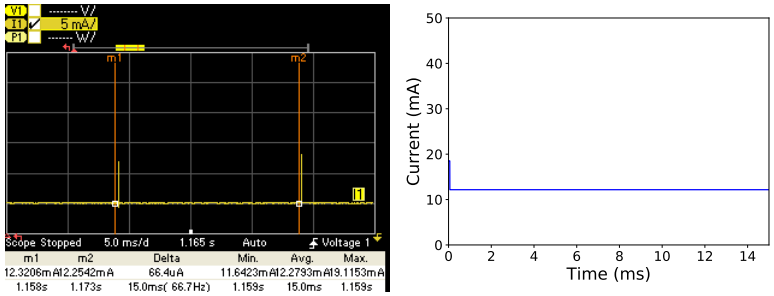
(a) `TxData` time slot.



(b) `RxData` time slot.

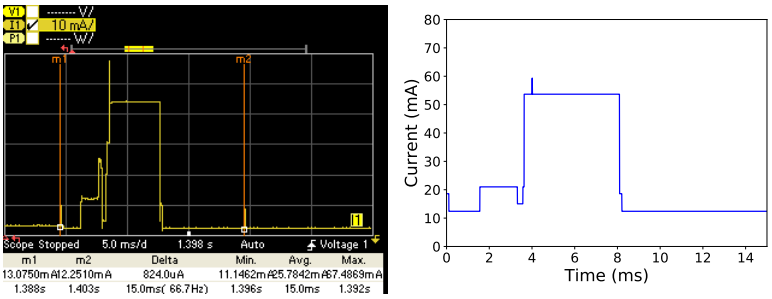


(c) `RxIdle` time slot.

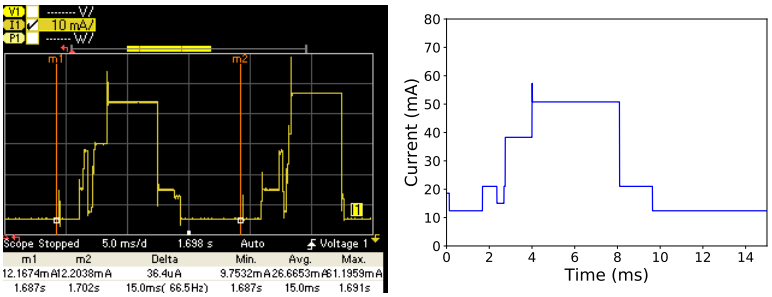


(d) `Sleep` time slot.

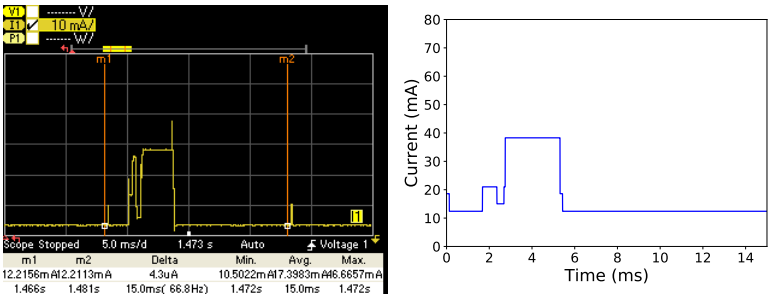
Figure A.1: Measured (left, between vertical lines m1 and m2) and modeled (right) current comparison for each time slot when using the CC2538 radio.



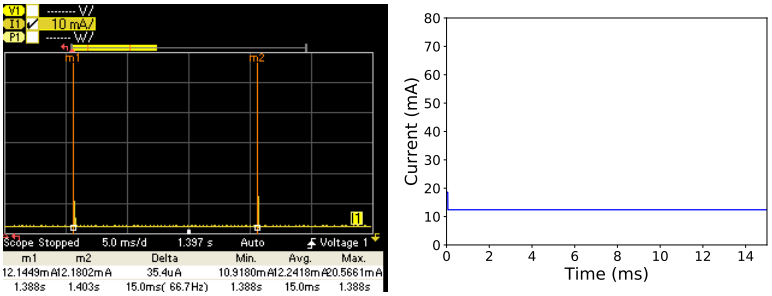
(a) `TxData` time slot.



(b) `RxData` time slot.



(c) `RxIdle` time slot.



(d) `Sleep` time slot.

Figure A.2: Measured (left, between vertical lines m1 and m2) and modeled (right) current comparison for each time slot when using the CC1200 radio.

Bibliography

[1] D. Sarangan, "2019 Update—Total Internet of Things (IoT) Device Forecast, 2017-2025," Frost & Sullivan, Tech. Rep., 2019.

[2] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelfflé, "Vision and challenges for realising the internet of things," *Cluster of European Research Projects on the Internet of Things, European Commision*, vol. 3, no. 3, pp. 34–36, 2010.

[3] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & information systems engineering*, vol. 6, no. 4, pp. 239–242, 2014.

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

[5] A. Seferagić, J. Famaey, E. De Poorter, and J. Hoebeke, "Survey on Wireless Technology Trade-Offs for the Industrial Internet of Things," *Sensors*, vol. 20, no. 2, p. 488, 2020.

[6] *Sigfox Connected Objects: Radio specifications*, Sigfox Standard v1.5, February 2020. [Online]. Available: https://build.sigfox.com/sigfox-device-radio-specifications

[7] L. Alliance, "Lorawan™ specification v1.1," Lora Alliance, Tech. Rep., 2017. [Online]. Available: https://lora-alliance.org/resource-hub/lorawantm-specification-v11

[8] *Wireless communication network and communication profiles - WirelessHART*, IEC Standard 62 591:2016, March 2016.

[9] 3GPP, "Narrowband Internet of Things (NB-IoT), Technical Report TR 36.802 V1.0.0, Technical Specification Group Radio Access Networks."

[10] *Bluetooth Core Specification*, Bluetooth Special Interest Group Standard v5.2, December 2019. [Online]. Available: https://www.bluetooth.com/specifications/bluetooth-core-specification/

[11] *IEEE Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 2: Sub-1GHz License Exempt Operation*, IEEE Std., May 2017.

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

[13] L. Doherty, W. Lindsay, and J. Simon, "Channel-specific Wireless Sensor Network Path Data," in *2007 16th International Conference on Computer Communications and Networks*. IEEE, 2007, pp. 89–94.

[14] K. Pister and L. Doherty, "TSMP: Time Synchronized Mesh Protocol," *IASTED Distributed Sensor Networks*, vol. 391, p. 398, 2008.

[15] T. Watteyne, L. Doherty, J. Simon, and K. Pister, "Technical Overview of Smartmesh IP," in *2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE, 2013, pp. 547–551.

[16] ISA, *Standard ISA-100.11a-2011 Wireless Systems for Industrial Automation: Process Control and Related Applications*. ISA, 2011.

[17] J. Muñoz, T. Chang, X. Vilajosana, and T. Watteyne, "Evaluation of IEEE802.15.4g for Environmental Observations," *Sensors*, vol. 18, no. 10, p. 3468, 2018.

[18] P. Tuset-Peiró, R. D. Gomes, P. Thubert, and X. Vilajosana, "Evaluating IEEE 802.15.4g SUN for Dependable Low-Power Wireless Communications In Industrial Scenarios," *Sensors (available on Preprints)*, vol. 20, 2020.

[19] M. Rady, Q. Lampin, D. Barthel, and T. Watteyne, "No Free Lunch—Characterizing the Performance of 6TiSCH When Using Different Physical Layers," *Sensors*, vol. 20, no. 17, p. 4989, 2020.

[20] G. Daneels, E. Municio, B. Van de Velde, G. Ergeerts, M. Weyn, S. Latré, and J. Famaey, "Accurate Energy Consumption Modeling of IEEE 802.15.4e TSCH Using Dual-Band OpenMote Hardware," *Sensors*, vol. 18, no. 2, p. 437, 2018.

[21] G. Daneels, B. Spinnewyn, S. Latré, and J. Famaey, "ReSF: Recurrent Low-latency Scheduling in IEEE 802.15.4e TSCH Networks," *Ad Hoc Networks*, vol. 69, pp. 100–114, 2018.

[22] G. Daneels, S. Latré, and J. Famaey, "Efficient Recurrent Low-Latency Scheduling in IEEE 802.15.4e TSCH Networks," in *2019 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*. IEEE, 2019, pp. 1–6.

[23] G. Daneels, C. Delgado, S. Latré, and J. Famaey, "Towards Slot Bonding for Adaptive MCS in IEEE 802.15.4e TSCH Networks," in *Proceedings of the IEEE International Conference on Communications*. IEEE, 2020.

[24] G. Daneels, C. Delgado, R. Elsas, E. De Poorter, S. Latré, C. Blondia, and J. Famaey, "Slot Bonding for Adaptive Modulations in IEEE 802.15.4e TSCH Networks," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[25] G. Daneels, D. Van Leemput, C. Delgado, S. Latré, E. De Poorter, and J. Famaey, "Parent and PHY Selection in TSCH Slot Bonding Networks," *To be submitted*, 2021.

[26] T. Watteyne, S. Lanzisera, A. Mehta, and K. S. Pister, "Mitigating Multipath Fading through Channel Hopping in Wireless Sensor Networks," in *2010 IEEE International Conference on Communications*. IEEE, 2010, pp. 1–5.

[27] *IEEE Standard for Low-Rate Wireless Networks*, IEEE Std. 802.15.4-2015, 2016.

[28] S. Jeong, J. Paek, H.-S. Kim, and S. Bahk, "TESLA: Traffic-Aware Elastic Slotframe Adjustment in TSCH Networks," *IEEE Access*, vol. 7, pp. 130 468–130 483, 2019.

[29] D. Stanislowski, X. Vilajosana, Q. Wang, T. Watteyne, and K. S. Pister, "Adaptive Synchronization in IEEE802.15.4e Networks," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 1, pp. 795–802, 2013.

[30] T. Chang, T. Watteyne, K. Pister, and Q. Wang, "Adaptive Synchronization in Multi-hop TSCH networks," *Computer Networks*, vol. 76, pp. 165–176, 2015.

[31] B. Martinez, X. Vilajosana, and D. Dujovne, "Accurate Clock Discipline for Long-term Synchronization Intervals," *IEEE Sensors Journal*, vol. 17, no. 7, pp. 2249–2258, 2017.

[32] *IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 3: Physical Layer (PHY) Specifications for Low-Data-Rate, Wireless, Smart Metering Utility Networks*, IEEE Std. 802.15.4g-2012, 2012.

[33] R. D. Gomes, P. Tuset-Peiró, and X. Vilajosana, "Improving Link Reliability of IEEE 802.15. 4g SUN Networks with Adaptive Modulation Diversity," in *31st IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2020)*. IEEE, 2020.

[34] D. Van Leemput, J. Bauwens, R. Elsas, J. Hoebeke, W. Joseph, and E. De Poorter, "Adaptive Multi-PHY IEEE 802.15.4 TSCH in Sub-GHz Industrial Wireless Networks," *Ad Hoc Networks*, vol. 111, p. 102330, 2020.

[35] P. Thubert, "An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4," Internet Engineering Task Force, Internet-Draft draft-ietf-6tisch-architecture-29, Aug. 2020, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-6tisch-architecture-29

[36] X. Vilajosana, K. Pister, and T. Watteyne, "Minimal IPv6 over the TSCH Mode of IEEE 802.15.4e (6TiSCH) Configuration," RFC 8180, May 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8180.txt

[37] M. Vučinić, J. Simon, K. Pister, and M. Richardson, "Constrained Join Protocol (CoJP) for 6TiSCH," Internet Engineering Task Force, Internet-Draft draft-ietf-6tisch-minimal-security-15, Dec. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-6tisch-minimal-security-15

[38] T. Chang, M. Vučinić, rf Vilajosana, S. Duquennoy, and D. Dujovne, "6TiSCH Minimal Scheduling Function (MSF)," Internet Engineering Task Force, Internet-Draft draft-ietf-6tisch-msf-18, Sep. 2020, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-6tisch-msf-18

[39] Q. Wang, X. Vilajosana, and T. Watteyne, "6TiSCH Operation Sublayer (6top) Protocol (6P)," RFC 8480, Nov. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8480.txt

[40] G. Montenegro, J. Hui, D. Culler, and N. Kushalnagar, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," RFC 4944, Sep. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4944.txt

[41] P. Thubert and J. Hui, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks," RFC 6282, Sep. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6282.txt

[42] P. Thubert and R. Cragie, "IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Paging Dispatch," RFC 8025, Nov. 2016. [Online]. Available: https://rfc-editor.org/rfc/rfc8025.txt

[43] P. Thubert, C. Bormann, L. Toutain, and R. Cragie, "IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing Header," RFC 8138, Apr. 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8138.txt

[44] P. Thubert, E. Nordmark, S. Chakrabarti, and C. E. Perkins, "Registration Extensions for IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Neighbor Discovery," RFC 8505, Nov. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8505.txt

[45] A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, T. H. Clausen, and T. Winter, "RPL: IPv6 Routing Protocol

for Low-Power and Lossy Networks," RFC 6550, Mar. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6550.txt

[46] P. Thubert, "Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL)," RFC 6552, Mar. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6552.txt

[47] M. Gupta and A. Conta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification," RFC 4443, Mar. 2006. [Online]. Available: https://rfc-editor.org/rfc/rfc4443.txt

[48] "User Datagram Protocol," RFC 768, Aug. 1980. [Online]. Available: https://rfc-editor.org/rfc/rfc768.txt

[49] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, Jun. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7252.txt

[50] G. Selander, J. P. Mattsson, F. Palombini, and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)," RFC 8613, Jul. 2019. [Online]. Available: https://rfc-editor.org/rfc/rfc8613.txt

[51] M. R. Palattella, N. Accettura, M. Dohler, L. A. Grieco, and G. Boggia, "Traffic Aware Scheduling Algorithm for reliable Low-Power multi-hop IEEE 802.15.4e Networks," in *2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC)*, Sept 2012, pp. 327–332.

[52] D. Dujovne, L. A. Grieco, M. R. Palattella, and N. Accettura, "6TiSCH Experimental Scheduling Function (SFX)," Internet Engineering Task Force, Internet-Draft draft-ietf-6tisch-6top-sfx-01, Mar. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-6tisch-6top-sfx-01

[53] ——, "6TiSCH 6top Scheduling Function Zero (SF0)," Internet Engineering Task Force, Internet-Draft draft-ietf-6tisch-6top-sf0-05, Jul. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-6tisch-6top-sf0-05

[54] M. R. Palattella, T. Watteyne, Q. Wang, K. Muraoka, N. Accettura, D. Dujovne, L. A. Grieco, and T. Engel, "On-the-fly Bandwidth Reservation for 6TiSCH Wireless Industrial Networks," *IEEE Sensors Journal*, vol. 16, no. 2, pp. 550–560, 2015.

[55] T. Chang, T. Watteyne, Q. Wang, and X. Vilajosana, "LLSF: Low Latency Scheduling Function for 6TiSCH Networks," in *2016 International Conference on Distributed Computing in Sensor Systems (DCOSS)*, May 2016, pp. 93–95.

[56] N. Accettura, M. R. Palattella, G. Boggia, L. A. Grieco, and M. Dohler, "Decentralized Traffic Aware Scheduling for multi-hop Low Power Lossy Networks in the Internet of Things," in *2013 IEEE 14th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, June 2013, pp. 1–6.

[57] E. Municio and S. Latré, "Decentralized Broadcast-based Scheduling for Dense Multi-hop TSCH Networks," in *Proceedings of the Workshop on Mobility in the Evolving Internet Architecture*, 2016, pp. 19–24.

[58] R. Soua, P. Minet, and E. Livolant, "Wave: a Distributed Scheduling Algorithm for Convergecast in IEEE 802.15.4e TSCH Networks," *Transactions on Emerging Telecommunications Technologies*, vol. 27, no. 4, pp. 557–575, 2016.

[59] T. P. Duy, T. Dinh, and Y. Kim, "Distributed Cell Selection for Scheduling Function in 6TiSCH Networks," *Computer Standards & Interfaces*, vol. 53, pp. 80–88, 2017.

[60] V. Kotsiou, G. Z. Papadopoulos, P. Chatzimisios, and F. Theoleyre, "LDSF: Low-latency Distributed Scheduling Function for Industrial Internet of Things," *IEEE Internet of Things Journal*, 2020.

[61] S. Anamalamudi, B. L. (Remy), M. Zhang, A. R. Sangi, C. E. Perkins, and S. Anand, "Scheduling Function One (SF1): hop-by-hop Scheduling with RSVP-TE in 6tisch Networks," Internet Engineering Task Force, Internet-Draft draft-satish-6tisch-6top-sf1-04, Oct. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-satish-6tisch-6top-sf1-04

[62] A. Morell, X. Vilajosana, J. L. Vicario, and T. Watteyne, "Label Switching over IEEE802.15.4e Networks," *Transactions on Emerging Telecommunications Technologies*, vol. 24, no. 5, pp. 458–475, 2013. [Online]. Available: http://dx.doi.org/10.1002/ett.2650

[63] F. Theoleyre and G. Z. Papadopoulos, "Experimental Validation of a Distributed Self-Configured 6TiSCH with Traffic Isolation in Low Power Lossy Networks," in *Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWiM '16. New York, NY, USA: ACM, 2016, pp. 102–110. [Online]. Available: http://doi.acm.org/10.1145/2988287.2989133

[64] S. Oh, D. Hwang, K.-H. Kim, and K. Kim, "Escalator: An Autonomous Scheduling Scheme for Convergecast in TSCH," *Sensors*, vol. 18, no. 4, p. 1209, 2018.

[65] M. Ramakrishna and J. Zobel, "Performance in Practice of String Hashing Functions," in *Database Systems For Advanced Applications' 97*. World Scientific, 1997, pp. 215–223.

[66] T. Chang, M. VučiniĆ, X. V. Guillén, D. Dujovne, and T. Watteyne, "6TiSCH Minimal Scheduling Function: Performance Evaluation," *Internet Technology Letters*, p. e170, 2020.

[67] D. Hauweele, R.-A. Koutsiamanis, B. Quoitin, and G. Z. Papadopoulos, "Pushing 6TiSCH Minimal Scheduling Function (MSF) to the Limits," in *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2020, pp. 1–7.

[68] K. Muraoka, T. Watteyne, N. Accettura, X. Vilajosana, and K. S. Pister, "Simple Distributed Scheduling with Collision Detection in TSCH Networks," *IEEE Sensors Journal*, vol. 16, no. 15, pp. 5848–5849, 2016.

[69] P. Levis, T. H. Clausen, O. Gnawali, J. Hui, and J. Ko, "The Trickle Algorithm," RFC 6206, Mar. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6206.txt

[70] O. Gnawali and P. Levis, "The Minimum Rank with Hysteresis Objective Function," RFC 6719, Sep. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6719.txt

[71] D. Barthel, J. Vasseur, K. Pister, M. Kim, and N. Dejean, "Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks," RFC 6551, Mar. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6551.txt

[72] T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wang, S. Glaser, and K. Pister, "OpenWSN: a Standards-based Low-power Wireless Development Environment," *Transactions on Emerging Telecommunications Technologies*, vol. 23, no. 5, pp. 480–493, 2012.

[73] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki- a Lightweight and Flexible Operating System for Tiny Networked Sensors," in *29th annual IEEE International Conference on Local Computer Networks*. IEEE, 2004, pp. 455–462.

[74] E. Baccelli, O. Hahm, M. Günes, M. Wählisch, and T. C. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," in *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2013, pp. 79–80.

[75] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer *et al.*, "TinyOS: An Operating System for Sensor Networks," in *Ambient intelligence*. Springer, 2005, pp. 115–148.

[76] T. Chang, P. Tuset-Peiro, X. Vilajosana, and T. Watteyne, "OpenWSN & OpenMote: Demo'ing a Complete Ecosystem for the Industrial Internet of Things," in *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 2016, pp. 1–3.

[77] X. Vilajosana, P. Tuset, T. Watteyne, and K. Pister, "OpenMote: Open-source Prototyping Platform for the Industrial IoT," in *International Conference on Ad Hoc Networks*. Springer, 2015, pp. 211–222.

[78] Crossbow. TelosB Data Sheet. [Online]. Available: https://www.willow.co.uk/TelosB_Datasheet.pdf

[79] Nordic. nRF51822 System on Chip. [Online]. Available: https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF51822

[80] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele *et al.*, "FIT IoT-LAB: A Large Scale Open Experimental IoT Testbed," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. IEEE, 2015, pp. 459–464.

[81] Contiki-NG. Contiki-NG, the OS for Next Generation IoT Devices. [Online]. Available: https://www.contiki-ng.org/

[82] Zolertia. 6LoWPAN hardware solutions for Internet-of-Things applications. [Online]. Available: https://zolertia.io/

[83] E. Municio, G. Daneels, M. Vučinić, S. Latré, J. Famaey, Y. Tanaka, K. Brun, K. Muraoka, X. Vilajosana, and T. Watteyne, "Simulating 6TiSCH Networks," *Transactions on Emerging Telecommunications Technologies*, vol. 30, no. 3, p. e3494, 2019.

[84] T. Watteyne, P. Thubert, and C. Bormann, "On Forwarding 6LoWPAN Fragments over a Multihop IPv6 Network," Internet Engineering Task Force, Internet-Draft draft-ietf-6lo-minimal-fragment-15, Mar. 2020, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-6lo-minimal-fragment-15

[85] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level Sensor Network Simulation with Cooja," in *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*. IEEE, 2006, pp. 641–648.

[86] S. Duquennoy, A. Elsts, B. Al Nahas, and G. Oikonomo, "TSCH and 6TiSCH for Contiki: Challenges, Design and Evaluation," in *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2017, pp. 11–18.

[87] A. Elsts, "TSCH-Sim: Scaling Up Simulations of TSCH and 6TiSCH Networks," *Sensors*, vol. 20, no. 19, p. 5663, 2020.

[88] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network Simulations with the ns-3 Simulator," *SIGCOMM demonstration*, vol. 14, no. 14, p. 527, 2008.

[89] A. Varga and R. Hornig, "An Overview of the OMNeT++ Simulation Environment," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. ICST (Institute for Computer Sciences, Social-Informatics and ..., 2008, p. 60.

[90] P. Huang, L. Xiao, S. Soltani, M. W. Mutka, and N. Xi, "The Evolution of MAC protocols in Wireless Sensor Networks: A Survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 1, pp. 101–120, First 2013.

[91] X. Vilajosana, Q. Wang, F. Chraim, T. Watteyne, T. Chang, and K. S. J. Pister, "A Realistic Energy Consumption Model for TSCH Networks," *IEEE Sensors Journal*, vol. 14, no. 2, pp. 482–489, Feb 2014.

[92] P. Tuset, X. Vilajosana, and T. Watteyne, "OpenMote+: a Range-Agile Multi-Radio Mote," in *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Graz, Austria: ACM, Feb. 2016, pp. 333–334. [Online]. Available: https://hal.inria.fr/hal-01239662

[93] B. Van de Velde, G. Daneels, and E. Municio, "OpenWSN firmware, CC2538 and CC1200 driver implementation," https://github.com/imec-idlab/openwsn-fw.

[94] Wireshark Foundation, "Wireshark," https://www.wireshark.org.

[95] D. D. Guglielmo, B. A. Nahas, S. Duquennoy, T. Voigt, and G. Anastasi, "Analysis and Experimental Evaluation of IEEE 802.15.4e TSCH CSMA-CA Algorithm," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 2, pp. 1573–1588, Feb 2017.

[96] G. Z. Papadopoulos, A. Mavromatis, X. Fafoutis, N. Montavont, R. Piechocki, T. Tryfonas, and G. Oikonomou, "Guard Time Optimisation and Adaptation for Energy Efficient Multi-hop TSCH Networks," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, Dec 2016, pp. 301–306.

[97] I. Juc, O. Alphand, R. Guizzetti, M. Favre, and A. Duda, "Energy Consumption and Performance of IEEE 802.15.4e TSCH and DSME," in *2016 IEEE Wireless Communications and Networking Conference*, April 2016, pp. 1–7.

[98] S. Labs, "User Manual, Starter Kit EFM32GG-STK3700," https://www.silabs.com/documents/public/user-guides/efm32gg-stk3700-ug.pdf, 2013.

[99] ——, "Simplicity Studio 4," https://www.silabs.com/products/development-tools/software/simplicity-studio, 2017.

[100] K. Technologies, "N6700 Modular Power System Family Data Sheet," http://literature.cdn.keysight.com/litweb/pdf/5989-6319EN.pdf, 2016.

[101] M. Series, "Guidelines for Evaluation of Radio Interface Technologies for IMT-Advanced," *Report ITU*, no. 2135-1, 2009.

[102] I. Gurobi Optimization, "Gurobi Optimizer Reference Manual," 2016. [Online]. Available: http://www.gurobi.com

[103] M. Raza, N. Aslam, H. Le-Minh, S. Hussain, Y. Cao, and N. M. Khan, "A Critical Analysis of Research Potential, Challenges, and Future Directives in Industrial Wireless Sensor Networks," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 39–95, 2017.

[104] C.-S. Sum, M.-T. Zhou, F. Kojima, and H. Harada, "Experimental Performance Evaluation of Multihop IEEE 802.15. 4/4g/4e Smart Utility Networks in Outdoor Environment," *Wireless Communications and Mobile Computing*, vol. 2017, 2017.

[105] J. Muñoz, E. Riou, X. Vilajosana, P. Muhlethaler, and T. Watteyne, "Overview of IEEE802.15.4g OFDM and its Applicability to Smart Building Applications," in *2018 Wireless Days (WD)*. IEEE, 2018, pp. 123–130.

[106] P. Tuset-Peiró, F. Vazquez-Gallego, J. Munoz, T. Watteyne, J. Alonso-Zarate, and X. Vilajosana, "Experimental Interference Robustness Evaluation of IEEE 802.15. 4-2015 OQPSK-DSSS and SUN-OFDM Physical Layers for Industrial Communications," *Electronics*, vol. 8, no. 9, p. 1045, 2019.

[107] P. Tuset-Peiró, F. Adelantado, X. Vilajosana, and R. D. Gomes, "Reliability through Modulation Diversity: Can Combining Multiple IEEE 802.15. 4-2015 SUN Modulations Improve PDR?" in *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2020, pp. 1–6.

[108] P. H. Gomes, T. Watteyne, and B. Krishnamachari, "MABO-TSCH: Multihop and Blacklist-based Optimized Time Synchronized Channel Hopping," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 7, p. e3223, 2018.

[109] R.-A. Koutsiamanis, G. Papadopoulos, T. L. Jenschke, P. Thubert, and N. Montavont, "Meet the PAREO Functions: Towards Reliable and Available Wireless Networks," in *IEEE International Conference on Communications (ICC)*, 2020.

[110] P. Thubert and G. Papadopoulos, "Reliable and Available Wireless Problem Statement," Internet Engineering Task Force, Internet-Draft draft-pthubert-raw-problem-statement-04, Oct. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-pthubert-raw-problem-statement-04

[111] M. Brachmann, S. Duquennoy, N. Tsiftes, and T. Voigt, "IEEE 802.15.4 TSCH in Sub-GHz: Design Considerations and Multi-band Support," in *2019 IEEE 44th Conference on Local Computer Networks (LCN)*. IEEE, 2019, pp. 42–50.

[112] R. Elsas, J. Hoebeke, D. Van Leemput, A. Shahid, G. Daneels, J. Famaey, and E. De Poorter, "Intra-Network Interference Robustness: An Empirical Evaluation of IEEE 802.15.4-2015 SUN-OFDM," *Electronics*, vol. 9, no. 10, 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/10/1691

[113] C. M. Grinstead and J. L. Snell, *Introduction to Probability*. American Mathematical Soc., 2012.

[114] S. C. Ergen and P. Varaiya, "TDMA Scheduling Algorithms for Wireless Sensor Networks," *Wireless networks*, vol. 16, no. 4, pp. 985–997, 2010.

[115] D. S. Johnson, J. K. Lenstra, and A. R. Kan, "The Complexity of the Network Design Problem," *Networks*, vol. 8, no. 4, pp. 279–285, 1978.

[116] J. H. Holland, "Genetic Algorithms," *Scientific American*, vol. 267, no. 1, pp. 66–73, 1992.

[117] M. Ojo, S. Giordano, G. Portaluri, and D. Adami, "Throughput Maximization Scheduling Algorithm in TSCH Networks with Deadline Constraints," in *2017 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2017, pp. 1–6.

[118] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT press, 1998.

[119] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary Algorithms Made Easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.

[120] B. Bellekens, L. Tian, P. Boer, M. Weyn, and J. Famaey, "Outdoor IEEE 802.11ah Range Characterization using Validated Propagation Models," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–6.

[121] M. Lacage and T. R. Henderson, "Yet Another Network Simulator," in *Proceeding from the 2006 workshop on ns-2: the IP network simulator*. ACM, 2006, p. 12.

[122] D. Dujovne, T. Watteyne, X. Vilajosana, and P. Thubert, "6TiSCH: Deterministic IP-enabled Industrial Internet (of Things)," *IEEE Communications Magazine*, vol. 52, no. 12, pp. 36–41, December 2014.

[123] M. Woehrle, M. Bor, and K. Langendoen, "868 mhz: a noiseless environment, but no free lunch for protocol design," in *2012 Ninth International Conference on Networked Sensing (INSS)*. IEEE, 2012, pp. 1–8.

[124] M. Saelens, J. Hoebeke, A. Shahid, and E. De Poorter, "Impact of EU Duty Cycle and Transmission Power Limitations for Sub-GHz LPWAN SRDs: An Overview and Future Challenges," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, p. 219, 2019.

[125] Wireless Testlab and OfficeLab. [Online]. Available: https://doc.ilabt.imec.be/ilabt/wilab/

Creative Commons