Philip Winkler, BSc

Supporting Duty-Cycled LoRa Communication in Contiki

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dr. Carlo Alberto Boano

Institute for Technical Informatics

Graz, May 2019

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date

Signature

Philip Winkler, BSc

Supporting Duty-cycled LoRa Communication in Contiki

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

Masterstudium Information and Computer Engineering

eingereicht an der

Technischen Universität Graz

Betreuer

Ass. Prof. Dr. Carlo Alberto Boano

Institut für Technische Informatik

Graz, Mai 2019

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Datum	Unterschrift

Abstract

The rising popularity of the Internet of Things (IoT), and the increasing need to connect ubiquitous objects to each other and to the Internet has led to the development of several wireless technologies, in order to satisfy the requirement of different application domains. Especially popular are, nowadays, low-power wireless technologies enabling the creation of Low Power Wide Area Networks (LPWANs). One of these popular technologies is LoRa (Long Range). Thanks to its large transmission range and low power consumption, LoRa enables large-scale IoT applications such as air quality measurements in cities or smart water management in rural areas. LoRa gains its communication range potential due to its unique Chirp Spread Spectrum (CSS) modulation scheme, and furthermore allows developers to fine-tune the transmission performance and range capabilities by modifying several physical settings. To further enhance the energy efficiency and reliability of LoRa communications a duty-cycled Medium Access Control (MAC) protocol is necessary, in order to minimize the radio's active time while offering full communication functionality. Such protocol should also allow the runtime adaptation of LoRa's PHY settings in order to ensure a reliable communication despite the mobility of nodes or other external influences.

In this thesis we first introduce a port of a popular LoRa platform, the STM32L152 Nucleo-64 combined with the SX1272 from Semtech, to the well-known open-source operating system Contiki. We then design DeFiL-MAC, a duty-cycled efficient MAC protocol for LoRa and implement it for the Contiki OS. DeFiL-MAC is a protocol based on Time-Division Multiple Access (TDMA) that provides an efficient master-to-slave communication and that guarantees the duty-cycle regulations can be automatically met. DeFiL-MAC also provides a mechanism for the run-time adaptation of LoRa's physical layer settings based on the quality of the link between two devices. This allows to reduce packet loss and to find the best settings for each communication link, and hence to increase both the efficiency and the reliability of communications.

An experimental evaluation validates DeFiL-MAC's operations and shows that the efficiency and reliability of a deployed LoRa network is increased by applying DeFiL-MAC's adaptation of physical layer settings.

Kurzfassung

Die zunehmende Beliebtheit des "Internet der Dinge" (Internet of Things, IoT) und die zunehmende Notwendigkeit, allgegenwärtige Objekte miteinander sowie dem Internet zu vernetzen, hat zur Entwicklung mehrerer Funktechnologien geführt, um den Anforderungen verschiedener Anwendungsdomänen gerecht zu werden. Besonders beliebt sind heutzutage drahtlose Technologien mit geringem Stromverbrauch, die die Realisierung von Low Power Wide Area Networks (LPWANs) ermöglichen. Eine dieser populären Technologien ist LoRa (Long Range). LoRa ermöglicht dank des großen Übertragungsbereichs und des geringen Stromverbrauchs großflächige IoT-Anwendungen, wie Messungen der Luftqualität in Städten, oder intelligentes Wassermanagement in ländlichen Gebieten. LoRa nutzt das so genannte Chirp Spread Spectrum (CSS) -Modulationsschema und ermöglicht Entwicklern die Feinabstimmung der Übertragungsleistung und der Reichweitenfunktionen, indem verschiedene physikalische Einstellungen geändert werden. Um die Energieeffizienz und Zuverlässigkeit der LoRa-Kommunikation weiter zu verbessern, ist ein Medium Access Control -Protokoll (MAC) erforderlich, das die aktive Zeit des Funkgeräts minimiert und gleichzeitig die volle Kommunikationsfunktionalität bietet. Ein solches Protokoll sollte auch die Laufzeitanpassung der PHY-Einstellungen von LoRa ermöglichen, um trotz der Mobilität von Knoten oder anderen äußeren Einflüssen eine zuverlässige Kommunikation sicherzustellen.

In dieser Masterarbeit stellen wir zunächst den Port einer beliebten LoRa-Plattform, dem STM32L152 Nucleo-64 in Kombination mit dem SX1272 von Semtech, in das bekannte open-source-Betriebssystem Contiki vor. Des Weiteren präsentieren wir DeFiL-MAC, ein effizientes MAC-Protokoll für LoRa, das mittels einem Lastzyklus gesteuert wird, und implementieren es im Betriebssystem Contiki. DeFiL-MAC ist ein auf Zeitmultiplex-Vielfachzugriff (Time Division Multiple Access, TDMA) basierendes Protokoll, das eine effiziente Master-to-Slave-Kommunikation ermöglicht und gewährleistet, dass die Vorschriften für den Lastzyklus automatisch erfüllt werden. Weiters bietet DeFiL-MAC einen Mechanismus für die Laufzeitanpassung der Einstellungen der physischen Schicht von LoRa, basierend auf der Qualität der Verbindung zwischen zwei Geräten. Dies ermöglicht es, Paketverluste zu reduzieren und die besten Einstellungen für jede Kommunikationsverbindung zu finden, und somit sowohl die Effizienz als auch die Zuverlässigkeit der Kommunikation zu erhöhen.

Eine experimentelle Auswertung bestätigt die korrekte Funktion von DeFiL-MAC und zeigt, dass die Effizienz und Zuverlässigkeit eines eingesetzten LoRa-Netzwerks durch die Anpassung der Einstellungen der physikalischen Schicht durch DeFiL-MAC erhöht wird.

Acknowledgments

This masters thesis was conducted at the Institute for Technical Informatics at Graz University of Technology.

First and foremost, I would like to thank my supervisor Carlo Alberto Boano for his excellent support and aid during my work on this thesis. Without his guidance and feedback this thesis would not have been possible at its current extent. Ultimately, it were also his exceptional courses that got me interested in the topic of wireless sensor networks.

Furthermore, I would like to thank my partner, Julia, for her continuous encouragement, motivation and patience. Finally, I would like to thank my friends and family, especially my parents, for their guidance, backing and their immeasurable support. I would not be at this point in my life without them.

Graz, May 2019 Philip Winkler

Danksagung

Diese Diplomarbeit wurde am Institut für Technische Informatik an der Technischen Universität Graz durchgeführt.

In erster Linie möchte ich mich bei meinem Betreuer Carlo Alberto Boano für seine herausragende Unterstützung und Hilfe während der Verwirklichung dieser Masterarbeit bedanken. Ohne seine Ratschläge und Feedback wäre diese Masterarbeit so nicht möglich gewesen. Schlussendlich waren es auch seine außerordentlichen Vorlesungen die das Interesse an drahtlosen Sensor Netzwerken in mir geweckt haben.

Ich bedanke mich bei meiner Partnerin, Julia, für ihre Ermutigungen, ihre Geduld und ihr Verständnis. Abschließend möchte ich mich bei meinen Freunden und meiner Familie bedanken. Insbesondere danke ich meinen Eltern, ohne deren Rückhalt, Erziehung und Unterstützung ich bestimmt nicht an diesem Punkt in meinem Leben stehen würde.

Graz, im Mai 2019 Philip Winkler

Contents

List of Figures

List of Tables

List of abbreviations

6LoWPAN IPv6 over Low Power Wireless Personal Area Network.

ACK Acknowledgement.

ADR Adaptive Data Rate.

BLE Bluetooth Low Energy.

BW Bandwidth.

CAD Channel Activity Detection.

CCA Clear Channel Assessment.

CoAP Constrained Application Protocol.

CR Coding Rate.

CSMA Carrier-Sense Multiple Access.

CSS Chirp Spread Spectrum.

DeFiL-MAC Duty-cycled eFficient LoRa-MAC.

EWMA Exponential Weighted Moving Average.

FCC Federal Communications Commission.

FDD Frequency Division Duplex.

FDMA Frequency Division Multiple Access.

FIFO First In First Out.

FSK Frequency Shift Keying.

FSPL Free-Space Path Loss.

GFSK Gaussian Frequency Shift Keying.

GMSK Gaussian Minimum Shift Keying.

GPIO General Purpose Input/Output.

HAL Hardware Abstraction Layer.

HTTP Hyper Text Transport Protocol.

IDE Integrated Development Environment.

IoT Internet of Things.

IPv6 Internet Protocol version 6.

LLSEC Link Layer Security.

LPL Low Power Listening.

LPWAN Low Power Wide Area Network.

LPWANs Low Power Wide Area Networks.

MAC Media Access Control.

MF-TDMA Multiple Frequencies Time Division Multiple Access.

MSK Minimum Shift Keying.

OOK On-Off Keying.

OS Operating System.

PHY Physical Layer.

PRR Packet Reception Ratio.

RAM Random-Access Memory.

RDC Radio Duty Cycling.

RF Radio Frequency.

ROM Read-Only Memory.

RPL Routing Protocol for Low power and Lossy Networks.

RSSI Received Signal Strength Indicator.

SF Spreading Factor.

SNR Signal-to-Noise Ratio.

SPI Serial Peripheral Interface.

TCP Transmission Control Protocol.

TDMA Time Division Multiple Access.

UDP User Datagram Protocol.

Chapter 1

Introduction

In recent years, the demand for connecting ubiquitous objects and gadgets to the Internet has driven the upraise of the Internet of Things (IoT) and, with it, billions of connected wireless devices [9].

IoT application domains range from smart manufacturing (i.e., Industry 4.0 [10]) over personalized health care to home automation, and can hence have an immense amount of possible applications with largely diverse and unique requirements. Nevertheless, the employed IoT devices share some common basic restrictions: the demand for low cost and low size. As a result of those preconditions, further limitations have developed, such as no tethered connection for data exchange, constrained power supply, and computational power. Those requirements, together with the need for efficient operation, have triggered the development of several new wireless technologies.

While most of those technologies make use of the free-licensed 2.4 GHz ISM band (e.g., Bluetooth Low Energy (BLE), ZigBee and Z-Wave), the high frequency and link budget makes them unsuitable for use case scenarios that require data transmission over large distances or propagation of radio waves through several obstructions. LoRa [11] is a relatively recent wireless technology designed to address these limitations and to build Low Power Wide Area Networks (LPWANs). Commonly, LoRa and other Low Power Wide Area Network (LPWAN) technologies, such as SigFox [12] and Weightless [13], use a single-hop star network topology that is built around a central gateway node[1]. The latter is reachable by every sensor node, due to increased range capabilities, thus reducing communication and network complexity while increasing stability.

Furthermore, wireless communication technologies have been proven to be quite energy demanding. Hence, methods for power saving and run-time optimization, such as duty-cycling, and low-power Media Access Control (MAC) protocols [5] are commonly used in LPWANs as well as other devices establishing the IoT.

[1]The terms 'Gateway' and 'Master' will be used interchangeably within this thesis and are considered equivalent

1.1 Problem statement

LoRa and some alternative LPWANs solutions are already commercially used in a wide variety of applications [14] such as emission measurements in Nordic cities [15] and smart water management systems [16]. However, those novel technologies are mostly not standardized and distributed with proprietary software frameworks. Thus, they are often incompatible with existing systems. In the case of LoRa, two different types of hardware devices are needed: one of those being more expensive and powerful for it to function as gateways, others, more generic and cheap taking on the role of sensor nodes.

Furthermore, to provide a mechanism for conserving energy, the LoRa Alliance [17] released a specialized MAC protocol, LoRaWAN [17], that provides the user with a framework and the necessary tools to set-up a LoRa network. However, due to the fact that this is proprietary software, an end user can not fine-tune this framework nor the MAC protocol to fit the specific applications needs, or use a single type of hardware for both gateways and nodes[2]. Therefore, an open-source implementation that integrates LoRa-enabled devices into a more standardized framework would help further research on LPWANs, and help in integrating the LoRa technology into everyday usages, as well as provide a more flexible baseline for developers.

An example of the benefits of an open-source platform is the Contiki Operating System [18], an open-source operating system specifically crafted for the requirements of constrained IoT devices. Since its creation in 2003, it supported some of the research that led to the emergence of wireless sensor networks and its networking principles. For example, using Contiki, researchers from all over the world have developed routing protocols, e.g., the Routing Protocol for Low power and Lossy Networks (RPL) [19], or distinctive approaches to media access protocols, such as ContikiMAC [5].

A first goal of this thesis is hence the implementation of an open-source port for a LoRa device in the Contiki Operating System, which provides full compatibility within the Contiki framework and network stack. As a second objective, this thesis tackles the design and creation of a MAC protocol that allows for the free and adaptive tuning of LoRa's PHY parameters, so to increase the robustness and energy efficiency of LoRa-based networks, while providing added flexibility and granularity.

The main challenge of integrating a LoRa device into the Contiki Operating System (OS) is that the existing starting point platforms are deeply interwoven with their implemented radio driver. Therefore, preceding the implementation of a new radio driver, the existing radio driver and its components have to be stripped from its base platform implementation. Regarding the design of a MAC protocol, the main challenge is that LoRa is a communication technology made to communicate very little amount of data (in the order of 300bps up to around 6kbps). This in combination with the desired compliance to the duty-cycle regulations [20], results in packets that can have a large time-on-air and therefore long sleep times in between active phases. This makes synchronization between gateway and end-the nodes a very fickle process. Finally, both implementations should be compatible with the existing Contiki OS structure and be interoperable with other LoRa devices.

[2]The terms 'Slave' and 'Node' will be used interchangeably within this thesis and are considered equivalent

1.2 Contributions

The contributions of this thesis are threefold.

Integrating the SX1272 into Contiki. First, we provide an open-source port of a popular LoRa platform by integrating the STM32L152 Nucleo-64 from STMicroelectronics in combination with the Semtech SX1272 LoRa transceiver into the Contiki operating system. This port is kept compliant with the well-established Contiki architecture, as it uses and implements each of the required interfaces defined by Contiki.

Design of DeFiL-MAC. Second, we present DeFiL-MAC, a duty-cycled, efficient MAC protocol designed for LoRa devices supported by the Contiki operating system. DeFiL-MAC is designed to fulfil the basic concept of a low-power MAC protocol: increasing the efficiency of a communication system by introducing the duty-cycled operation of the radio module. Thus, the first key-feature of DeFiL-MAC is that it provides efficient operations, not only by reducing the active time of the radio to a minimum amount, but also by introducing a smart, adaptive mechanism to adapt the physical layer parameters of LoRa at runtime. The increase in efficiency regarding active time is accomplished by performing duty-cycling in combination with TDMA, thus, nodes only need to be active within their dedicated slot, enabling them to spend the remaining time in a sleep mode that consumes very little energy. A key feature of LoRa is that, by altering the physical settings, one can have a direct influence on the robustness and communication range of the system. We further introduce the ability for the MAC to automatically negotiate more reliable (higher energy consumption) or more conservative (less energy consumption) settings, based on the current communication performance at runtime. This alteration does not only have a direct influence on the energy consumed by a node, but also influences the time-on-air of a transmitted packet, which has a direct influence on the used energy. This change of settings establishes the second key feature of DeFiL-MAC: reliability. Due to the low data-rate of LoRa its packets can have a time-on-air within seconds: this, in combination with the 1% duty-cycle regulation [20] for the license-free sub-gigahertz radio frequency band, results in few packets with little data spread over a long period of time. Thus, packets have to be reliably received by the gateway, even more so because there are few of them. The adaptive switching of PHY parameters increases the provided reliability of a LoRa network, since it can detect changes in the communication performance and propose more reliable settings to avoid either losing a packet or the loss of a link. Being situated within the dynamic world of the IoT, we required our MAC to be scalable, which means it can autonomously handle nodes joining or leaving a network. DeFiL-MAC is fully open-source and embedded into the well-known Contiki operating system. We believe that this will help LoRa in gaining more popularity within the IoT community and to encourage developers to implement their own LoRa network which they can operate, customize and tune at full potential.

Experimental evaluation. Third, this thesis provides an experimental evaluation of the implemented duty-cycled MAC protocol on a LoRa network that is built with the hardware that was previously ported to the Contiki OS. We validate the operations of the protocol with a laboratory setup of three LoRa devices, two in the role of nodes and one as a centralized gateway, by allowing the gateway to establish a demo network and exchanging dummy data with its nodes. Furthermore, we evaluate the correct function and decreased

energy consumption of the active PHY switching mechanism by extending the experimental setup with a programmable attenuator, in order to decrease the communication performance at will.

1.3 Outline

The remainder of this thesis is structured as follows. Chapter 2 introduces the key features of LoRa, describing its principle of operation, why it is able to propagate data over large distances, and some of its applications. Additionally, the Contiki operating system and its main features are introduced. Chapter 3 presents the first part of this thesis and describes how we created a port for the STM Nucleo-64 in combination with a Semtech SX1272 radio, while listing related works and studies. Section 4 will provide a detailed overview on the design process of our proposed MAC protocol, as well as present details on the implementation and its functionality. The performance and evaluation of our protocol is discussed in Chapter 5. Finally, Chapter 6 concludes this thesis and offers an outlook on future work.

Chapter 2

Background

This chapter gives a short overview of LoRa, all other relevant software components and concepts used in this thesis, as well as the utilized hardware. Section 2.1 introduces LoRa technology, describing its key-features as well as its benefits, Section 2.1.1 briefly explains the modulation technique behind LoRa, while Section 2.1.2 will introduce the physical layer of LoRa. An introduction to the LoRaWAN MAC protocol can be found in 2.1.3. Section 2.2 gives an overview of the Contiki Operating System, including its network stack (Section 2.2.1), Protothreads (Section 2.2.2) and examples for built-in MAC protocols (Section 2.2.3). Section 2.2.3 will introduce the functionality of TDMA methods. In Section 2.3 details on the hardware used in this thesis are presented, providing details of the STM32L152 Nucleo-64 in Section 2.3.1 and the SX1272 in Section 2.3.2 respectively.

2.1 LoRa Technology

Published by Semtech in 2013, LoRa (derived as abbreviation of **Lo**ng **Ra**nge), is a proprietary, physical layer, radio modulation technology [21]. Differently from other technologies of the LPWAN family, LoRa uses a derivative of the CSS for its communication [22] [23]. The core features of the CSS will further be discussed in Section 2.1.1. Like most of its siblings, LoRa uses the license-free sub-gigahertz radio frequency bands, i.e., 868 MHz in Europe and 915 MHz in North America. Compared to other technologies, further key-features are: a low bandwidth and low battery usage, with the drawback of a fairly small data-rate. Figure 2.1 visualizes where LoRa would be placed among other radio technologies, when comparing bandwidth and theoretical transmission range.

As can be seen, sending messages over large distances comes at a price of limited data transmission, due to the low bandwidth.

If we would theoretically consider a perfect radio technology, it would include the following three crux properties:

- Low energy consumption;
- High data rate;
- Reliable, long range transmission.

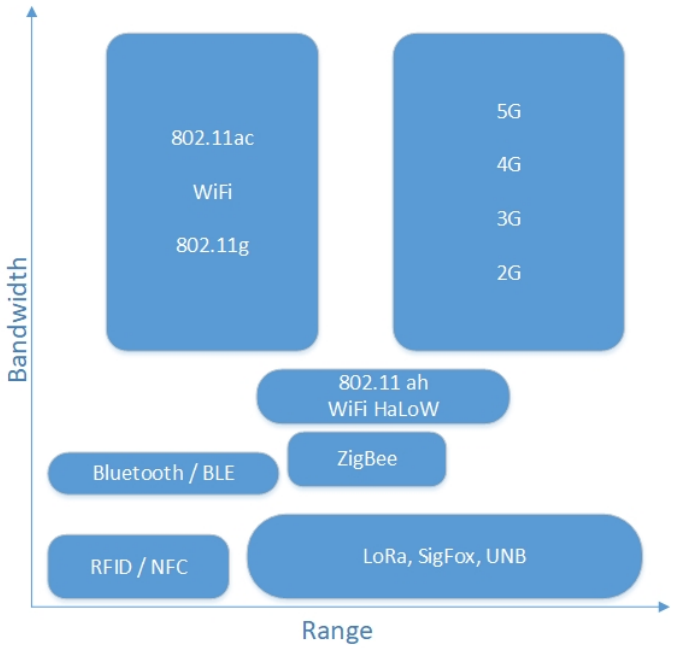

Figure 2.1: Range vs. bandwidth - comparison of several transmission technologies (adapted from [1]).

When it comes to real-world applications however, we are limited by several factors, mostly the laws of physics, thus being limited to only pick two of the ideal parameters presented above. One of those restrictive factors is the propagation of a radio wave through a given medium, from a sender to a receiver. This is specified by the so-called 'link budget' property of a communication system. The link budget is a special metric that considers all gains and losses in a given arrangement, from the transmitter, through a given medium (e.g., free space, cable, etc.) to the receiver. A simplified formula that expresses those relations is given by Eq. 2.1.

$$ReceivedPower(dB) = TransmittedPower(dB) + Gains(dB) - Losses(dB) \quad (2.1)$$

However, when taking a closer look at the formula, one needs to break down the total gains and losses into individual terms. For example, the total losses would consist of: various transmitter losses (e.g., connectors), path loss (commonly free space loss), miscellaneous losses (e.g., fading) and receiver losses, rendering the previous addressed formula much less trivial.

Furthermore, those losses are shown in Figure 2.2, when transmitting data over a given channel, e.g., free space: a portion of the transmission power of the sender (limited for the license-free band, Europe: 14 dBm) is lost due to the previously mentioned path loss and fading [22] [24]. Path loss is specified as the attenuation of an electrical wave as it travels through a given medium. Free-space path loss, being one of the more common loss factors in communication networks, is defined as the loss between two isotropic radiators in free

space. Despite its name, also includes a receiving antenna aperture component in the total attenuation [25] [26].

Eq. 2.2 depicts how the calculation for the Free-Space Path Loss (FSPL) is done for d in meters and f in Hertz. The important aspect of this formula is that it illustrates quite well how this loss scales squared to the distance a signal travels. Another factor that is shown in the formula is the absence of the used bandwidth of a signal. Although bandwidth is one of the key parameters of LoRa and any communication system, it has no direct dependency to the electromagnetic wave of a signal itself, but the antenna used in the system (higher bandwidth requires a smaller antenna), and thus have its impact through it.

$$\begin{aligned} \text{FSPL(dB)} &= 20\log_{10}\left(\frac{4\pi df}{c}\right) \\ &= 20\log_{10}(d) + 20\log_{10}(f) + 20\log_{10}\left(\frac{4\pi}{c}\right) \\ &= 20\log_{10}(d) + 20\log_{10}(f) - 147.55 \end{aligned} \tag{2.2}$$

where: d = Distance in Meters
f = Frequency in Hertz

As already mentioned, fading is another factor that is responsible for loss in a communication system. By definition, fading is referred to as an alteration of the signal strength caused by various variables. There are multiple forms of fading, caused by different influences, such as: multipath propagation (multipath-fading), weather (especially rain), temperature [8], or obstacles affecting the wave propagation (shadow fading / shadowing) [24].

The sum of those losses increases when increasing the distance between the sender and the receiver. As stated previously, the link budget then basically specifies the amount of power that is available for a signal to be powerful enough to be detected by a receiver, which is referred to as 'receiver sensitivity'. If we take a look at the given formula for the receiver sensitivity, or the minimum amount of power that a signal needs to have in order to be correctly received, shown in Eq. 2.3, we can see another key feature of why LoRa can achieve such a long range. The receiver sensitivity scales with the used bandwidth, and since LoRa uses a relatively low bandwidth, the sensitivity of the receiver is higher, thus allowing signals to have a lower minimum power to be detectable. The typical values for LoRa receiver sensitivity and link-budget are illustrated by a simple calculation shown in Eq. 2.4.

$$\begin{aligned} P_{i,\text{min}} &= N_{\text{Floor}} + SNR_{\text{o,min}} + NF_{Rx} \\ &= -174dBm/Hz + 10\log\left(BW_{Rx}\right) + SNR_{o,min} + NF_{Rx} \end{aligned} \tag{2.3}$$

where: BW_{Rx} = used Bandwidth
$SNR_{o,min}$ = Signal-to-Noise ratio
NF_{Rx} = Receiver Noise Figure

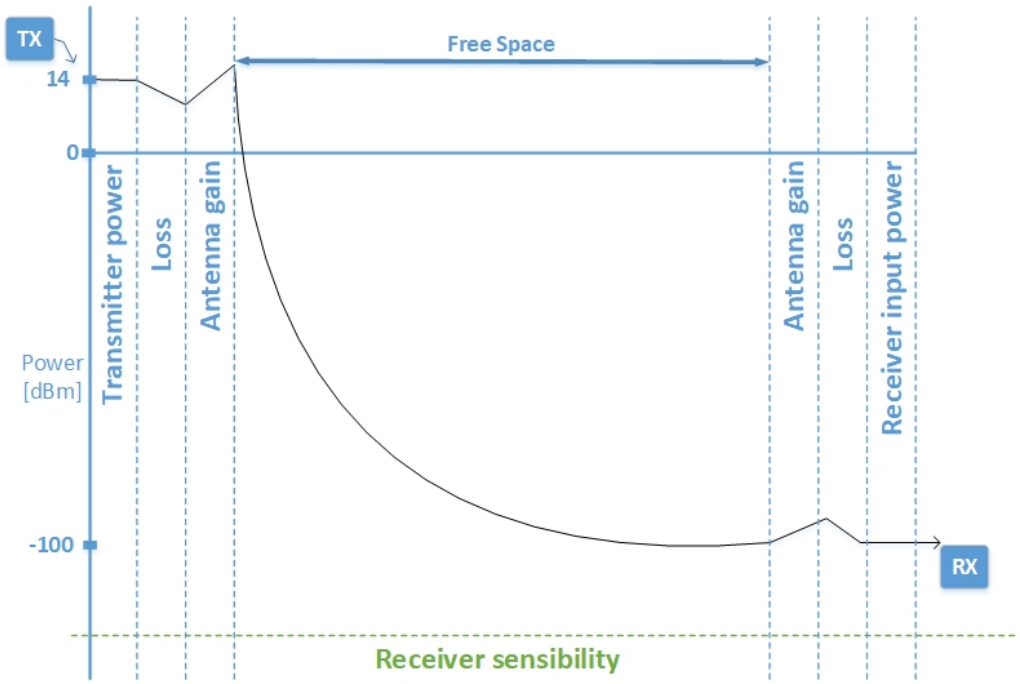

Figure 2.2: Transmission over free space, loss of power (adapted from [1]).

$$\begin{aligned} Sensitivity &= -174dBm + 10\log(125kHz) - 20 + 6dB = -137dBm \\ LinkBudget &= 14dBm + 137dBm = 151dBm \end{aligned} \tag{2.4}$$

where: $TXPower = 14\text{dBm}$
$Bandwidth = 125\text{kHz}$
$SNR = -20$
$NoiseFigure = 6\text{dB}$

As shown in Eq. 2.4, a LoRa receiver can typically receive signals that are 20 dB below the noise level, which results in a sensitivity of -137dBm. When transmitting with 14dBm, we can achieve a link-budget for that system of 151dBm. If we bring those values back into perspective of the free-space path loss, shown in Eq. 2.2, a FSPL of around 150dBm would correspond to a theoretical distance of roughly 800km. Indubitably, in real-world applications this calculation does not hold, due to several limitations and losses (some of which have been mentioned previously).

2.1.1 Chirp Spread Spectrum

As stated previously, LoRa uses a variation of the CSS [22] as modulation scheme. Originally developed in the 1940s, an early derivation of the chirp-spread spectrum was traditionally used for military applications, due to its long communication distances and interference robustness [27]. LoRa is its first low-cost implementation for commercial usage [27]. Figures 2.3, 2.4, and 2.5 illustrate the distinct difference between the frequency representation of

an OOK based modulation, a FSK modulated communication, and the CSS modulation used by LoRa.

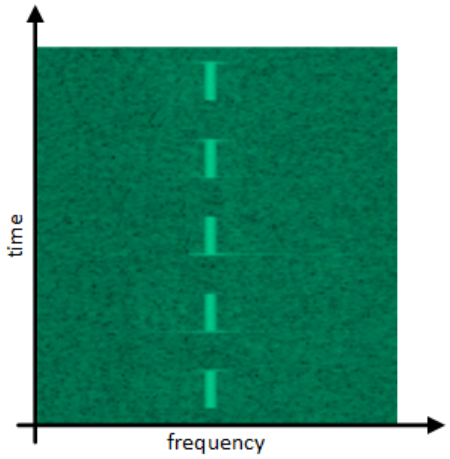

Figure 2.3: Frequency spectrum of OOK (adapted from [1]).

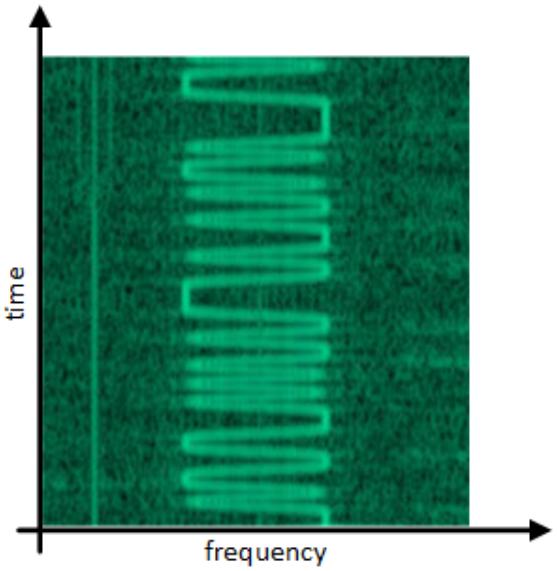

Figure 2.4: Frequency spectrum of FSK (adapted from [1]).

As indicated by the various frequency spectra, the chirp spread spectrum is considerably distinguishable from other modulation techniques. Further illustrated in Figure 2.6, LoRa modulation uses the unabridged available, fixed bandwidth per 'chirp', the latter being specified as one iteration of the frequency increasing from low to high. The encoding of information into such a signal is done by introducing gaps and 'jumps' into the sequence of chirps: this can be seen in Figure 2.6 and Figure 2.7 respectively.

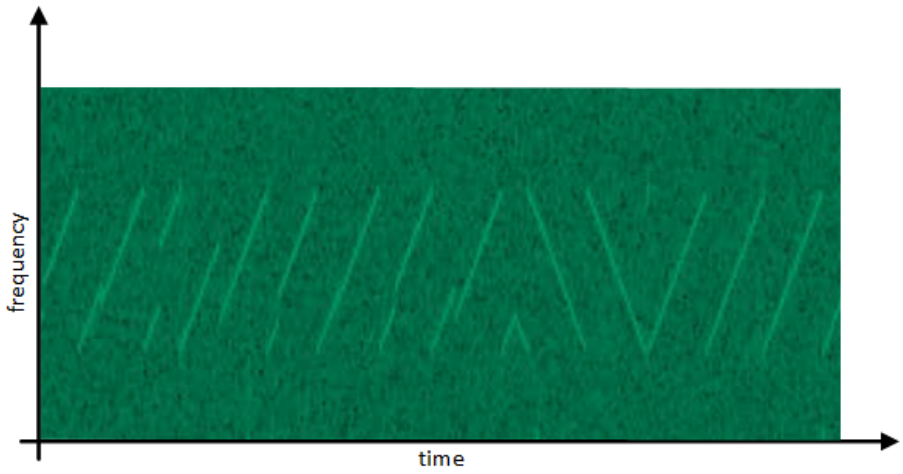

Figure 2.5: Frequency spectrum of a LoRa communication (adapted from [1]).

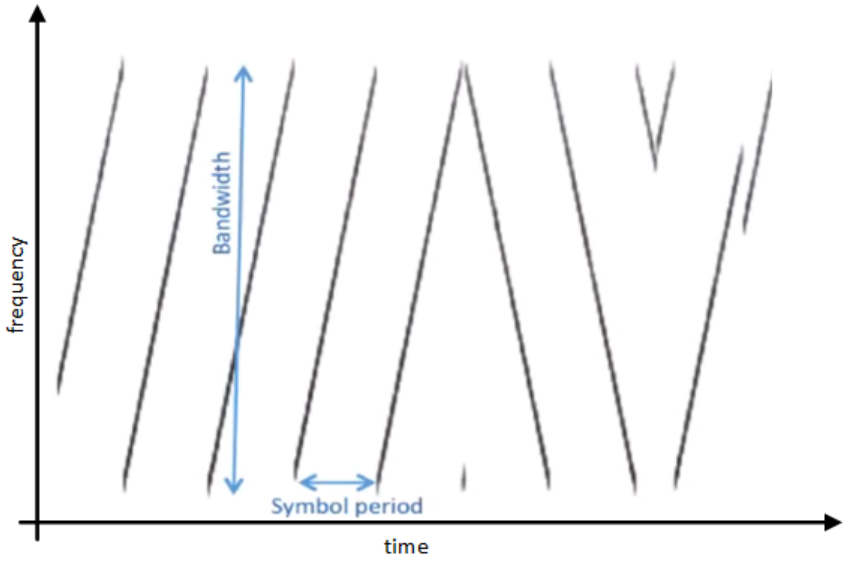

Figure 2.6: Bandwidth utilization with CSS (adapted from [1] and [2]).

The advantages of this unique modulation scheme become even more distinct when considering the demodulation of a signal, also when regarding the robustness against noise. The demodulation is done by combining the received chirps with generated reverse chirps on the receiver side. As displayed in Figure 2.7, the decoding process of a received LoRa signal is fairly straightforward. The receiver generates a signal that matches the PHY settings of the sender, the Spreading Factor (SF) of said signal plays a crucial role here and will further be described in Section 2.1.2, but in an inverse form. This can also be seen in Figure 2.7, those so-called 'inverse chirps' are generated by altering the specified frequency from high to low. Another important aspect is displayed in Figure 2.7: even if the signal is corrupted with noise on the channel (illustrated in orange), symbols can still be decoded correctly. Instead, generally, if there is noise present on a channel, it is at a specific frequency band (e.g., narrowband interference - horizontally) and not across a certain range of frequencies.

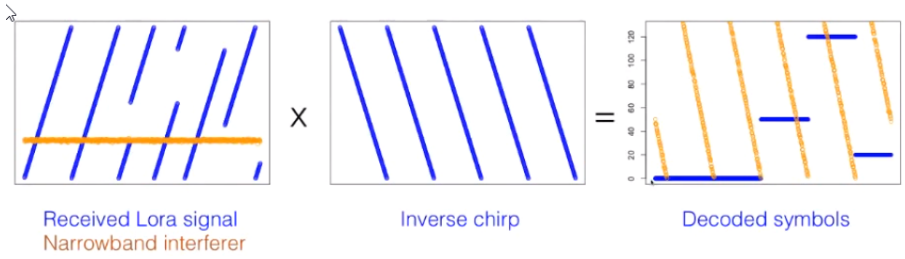

Figure 2.7: Demodulation principle of a LoRa signal with noise [1].

At the receiver, the inverse chirps are then generated and multiplied with the received signal, cancelling out the basic chirps with the inverse chirps. This leaves only the decoded symbols which get spread out over time. However, the noise that is interfering the received signal is spread over the whole spectrum. This way, certain types of noise do not have any influence on a received LoRa signal.

Telkamp [1] and Knight [2] list further advantages of the LoRa chirp spread spectrum in their analysis:

- Simple to implement: Nodes can be kept simple and there is no mismatch in capabilities between sink-nodes (generally prone to be more computational powerful) and normal nodes;
- Resistant to in- and out of-band interference;
- Resistant to multipath and fading;
- Doppler shift resistant (moving devices pose no added difficulties).

2.1.2 LoRa Physical Layer

Another key ability of LoRa is that its communication performance and range capabilities can be tuned by the programmer, depending on the current use-case of the setup. As for most low-power transceivers, carrier frequency and transmission power can be altered

for LoRa devices as well. However, LoRa's performance can be fine-tuned by varying a number of PHY settings, that is: bandwidth, spreading factor, coding rate, and carrier frequency. Those parameters are, among others, used to achieve a more robust transmission by lowering the receiver sensitivity to sustain an increased data-rate [8]. A listing of the crucial PHY parameters and their impact on a LoRa communication system is given in Table 2.1.2.

Parameter	Possible values	Impact
Bandwidth	125kHz to 500kHz	Higher bandwidths allow for transmitting packets at higher data rates (1 kHz = 1 kcps), but reduce receiver sensitivity and communication range.
Spreading Factor	7 to 12	Bigger spreading factors increase the signal-to-noise ratio and hence radio sensitivity, augmenting the communication range at the cost of longer packets and hence a higher energy expenditure.
Transmission Power	2dBm to 14dBm	Higher transmission powers reduce the signal-to-noise ratio at the cost of an increase in the energy consumption of the transmitter.
Coding Rate	4/5 to 4/8	Larger coding rates increase the resilience to interference bursts and decoding errors at the cost of longer packets and a higher energy expenditure.

Table 2.1: Summary of LoRa's configurable settings and their impact on communication performance [8].

Traditionally, in LPWAN systems, the bandwidth used scales with the amount of data to be sent. For LoRa or other technologies that use CSS, the used bandwidth is usually fixed at start up and will not be adapted to the transmitted data.

Generally, the reach of a radio can be increased by either increasing the transmission power or the energy per bit. With the transmission power being a limiting component, LoRa uses spreading factors to alter the modulation rate and the energy per bit, thus being able to control the range. The effect that a change of the spreading factor has on a sequence of chirps can be seen in Figure 2.8.

As shown in Figure 2.8, for SF = 7 we are able to have the most chirps within a given time frame. This directly corresponds to the amount of data that possibly can be sent. When observing SF = 8 we can see, that it is exactly half as fast as SF = 7, furthermore SF = 9 is half as fast as SF = 8. This can be directly applied to all following spreading factors up to SF = 12.

The direct relationship between spreading factor and available bit rate can be even more exemplified when observing the Eq. in 2.5 for the symbol rate Rs and the bit rate Rb. A further observation from both formulas is that the spreading factor has a direct influence on the bits per symbol, thus increasing or decreasing the time it takes to send a symbol. A

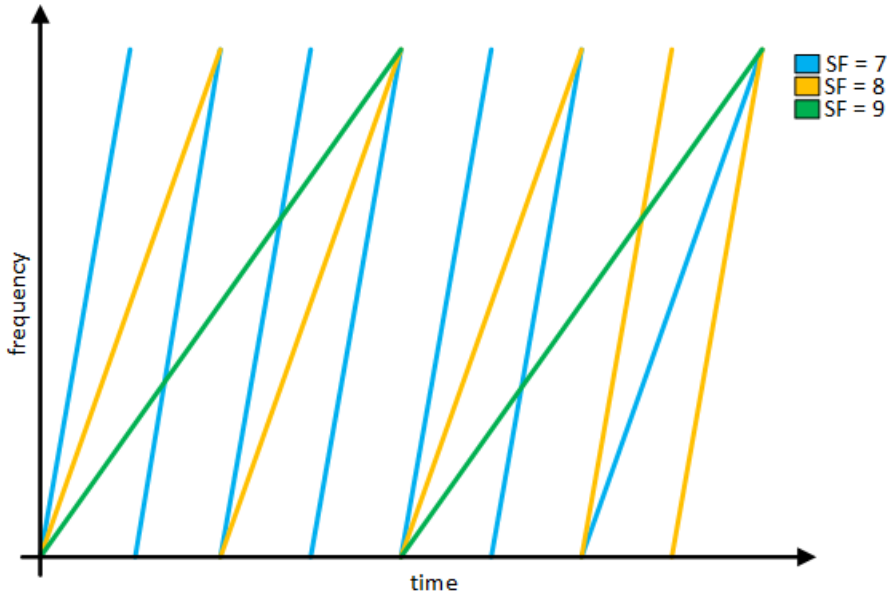

Figure 2.8: Basic functionality of spreading factors [1].

resulting 'rule of thumb' regarding the relationship between range and spreading factor is that the slower you send, the further away you can theoretically send.

$$\begin{aligned} R_s &= \frac{BW}{2^{SF}} \\ R_b &= SF \cdot R_s = SF \cdot \frac{BW}{2^{SF}} \end{aligned} \tag{2.5}$$

where: SF = Spreading Factor
BW = Bandwidth.

2.1.3 LoRaWAN: Protocol and Architecture

Released in 2015, through a collaborative effort by the LoRa-Alliance, LoRaWAN defines the communication protocol and system architecture for the network while the LoRa physical layer enables the long-range communication link. The protocol and network architecture have the most influence in determining the battery lifetime of a node, the network capacity, the quality of service, the security, and the variety of applications served by the network [17] [28].

LoRaWAN is capable of operating a network in a star topology, since each node can transmit directly to one or multiple gateways, given that, usually, nodes are not coupled with one specific gateway. LoRaWAN's gateways have powerful radios, capable of receiving and decoding up to fifty multiple concurrent transmissions [29]. Furthermore, LoRaWAN specifies data rates from 300bps up to 5.5kbps and supports, among others, (secure) bi-directional communication [28], [30].

Another feature of LoRaWAN is that it specifies several classes for its end-devices. In particular, LoRaWAN specifies three separate classes to address the different requirements of various applications:

- Class A
  - Battery powered
  - Small payload
  - End-device initiate communication (uplink)
  - Unicast Messages
- Class B
  - Low latency
  - Unicast and Multicast messages
  - Receive periodic Beacon from gateway
  - Transmission at fixed intervals
- Class C
  - No latency
  - Transmission at any time, initiated by gateway
  - End-device is constantly receiving
  - Unicast and Multicast messages

In their introducing article about LoRaWAN, the LoRa-Alliance further specifies those device classes by referring to their individual properties regarding battery life and their influence of network latency, this can be seen in Figure 2.9.

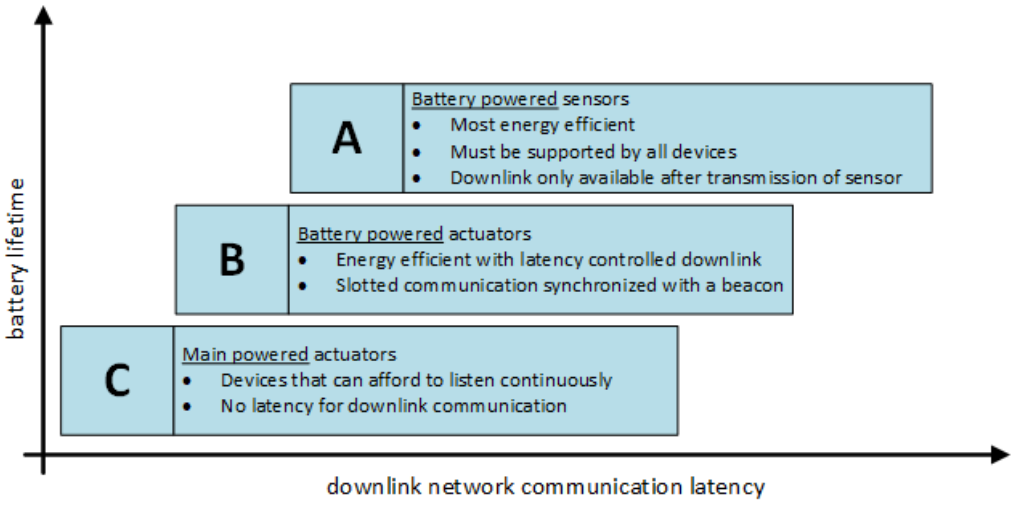

Figure 2.9: Further properties of LoRaWAN's classes regarding battery life and latency, adapted from [3].

Furthermore, LoRaWAN is provided with a feature called 'adaptive data rate': if enabled, LoRaWAN will automatically manage the SF for each end-device [3]. Basically, here the protocol decides whether to change the spreading factor or not, based on a look-up

table that holds the available spreading factor values and a corresponding Signal-to-Noise Ratio (SNR) threshold for said value. If the SNR value is above or below a given threshold, the protocol orders a change of spreading factor.

Limitations. Although LoRaWAN provides users with a solid framework for a LoRa network, it has some drawbacks. One major hindrance is that it is maintained and developed solely by the LoRa-Alliance. Therefore, by not providing an open-source implementation, the majority of developers and users are left out in the further evolution of the protocol. While also having positive effects, another drawback is that LoRaMAC requires specialized hardware that operates as a gateway for a network. While those gateways are much more powerful, in respect to computational power, than a common sensor node within the network, they are also much more expensive(e.g., Gateway: Dragino LG02, ∼75€[31]/ LoRa node: Adafruit Feather M0 with RFM95, ∼29€[32]). Furthermore, LoRaWAN is not suited for real-time applications and use cases that require the transmission of large amounts of data, mostly due to its low data rate.

As described previously, LoRaWAN also implements Adaptive Data Rate (ADR), a mechanism to change transmission settings based on the current communication performance. While being useful, this technique only implements a few possible configurations, thus yielding little granularity and leaving behind energy optimization possibilities [33].

2.2 Contiki OS

The Contiki OS is an open source lightweight operating system developed especially for constrained IoT devices that is retailed since 2011 as the "Open Source OS for the Internet of Things" [18]. Contiki itself is written in the C programming language and is fully functional with approximately 10kB Random-Access Memory (RAM) and 30kB Read-Only Memory (ROM), while still providing its users with a full network stack supporting Hyper Text Transport Protocol (HTTP), Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), and other operating system features. A more detailed overview of the network stack will be given in Section 2.2.1. Among others, the Contiki operating system also supports some low-power communication protocols such as Constrained Application Protocol (CoAP), RPL, and IPv6 over Low Power Wireless Personal Area Network (6LoWPAN) [18]. Another feature that differentiates Contiki from other operating systems for memory constrained devices, is the introduction of protothreads, whose main advantages will be introduced in Section 2.2.2.

2.2.1 The Contiki Network Stack

As already stated, the Contiki OS comes with several functionalities, including a fully functional network stack consisting of four layers (top-down): Network layer, Medium Access Control layer, Radio Duty Cycling layer and Radio layer. An outline of the network stack is illustrated in Figure 2.10.

The network layer implements high level protocols such as Internet Protocol version 6 (IPv6), UDP and RIME [34]. The MAC layer is responsible for data retransmission in case of lost packets. The Contiki OS provides two default implementations: Carrier-Sense Multiple Access (CSMA), an implementation of a protocol that detects collisions of

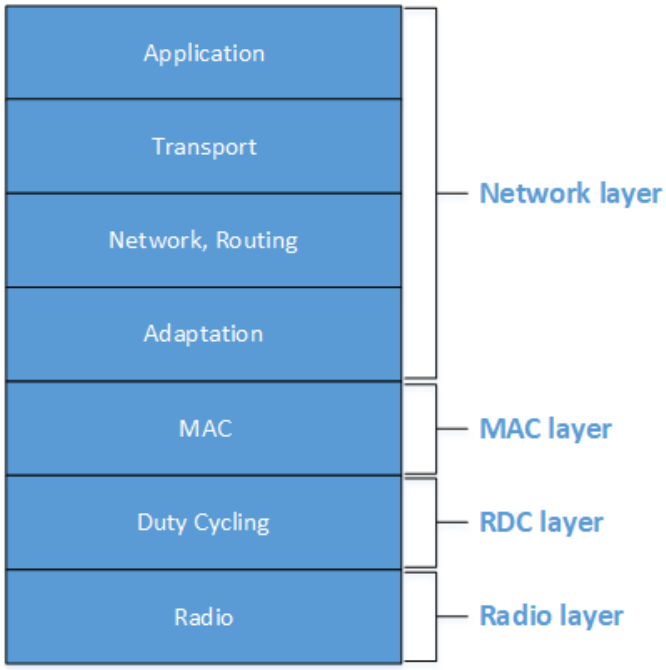

Figure 2.10: Network stack within of Contiki OS.

packets and re-transmits them accordingly, and `nullmac`, which is an implementation that basically provides no additional retransmission functionality and passes the unmodified information on to the Radio Duty Cycling (RDC) layer, providing similar functionality as an unsynchronized ALOHA protocol [35].

The RDC layer manages duty cycling of the radio, thus deciding when to turn off the radio, as well as provide sleep and wake-up functionalities. Contiki provides some built-in RDC drivers, such as `ContikiMAC` [5], `X-MAC` [4], and `nullrdc`, which provides no duty-cycling functionality. At the bottom of the network stack is the framer and the radio layer: the latter implements functionality such as transmission and reception of radio packets, while the framer is responsible for parsing the transmission data. Furthermore, the radio layer possesses the ability to trigger an interrupt if a packet was received correctly. It also performs some low-level functions such as handling the initialization procedure.

2.2.2 Protothreads

Commonly, on memory-constrained devices, processes and threads are implemented by an event-driven architecture. One benefit of such an approach is that since there are no multiple threads or processes running concurrently, they do not need an individual stack, thus conserving valuable memory space by sharing the same stack.

Another benefit is that event-driven systems can get away with not implementing locking mechanisms, e.g., semaphores, because there should be only one instance of an event handler running at a given time.

However, most of those systems based on events are non-intuitive to work with or

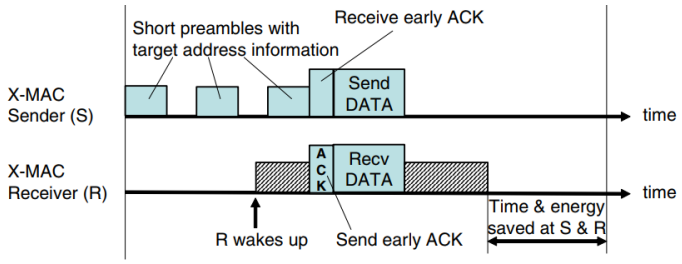

Figure 2.11: Operation principle of X-MAC, retrieved from [4].

write code for, as it can be hard to express code as a state machine. Some applications simply can not be expressed within a pure event-based mechanism. This is where multi-threaded approaches have an advantage. The Contiki OS combines the advantages of both approaches within so-called 'protothreads'. Those protothreads are based on an event-driven system with preemptible threads using an event-driven kernel and a user library providing multi-threading functionality. The Contiki kernel dispatches events to run processes by calling the polling handler of the respective process. Each called polling handler runs to completion and cannot be preemptively stopped by the kernel [36] [18].

2.2.3 MAC protocols within the Contiki OS

This Section briefly describes some of the MAC protocols that are built into the current Contiki versions.

Nullmac. The first, and most rudimentary MAC protocol a developer can chose to use is `nullmac`, located at `../core/net/mac/nullmac.c`. Indirectly implied by its name it does not really provide any MAC functionality at all. Its main duty is to call the corresponding RDC layer functions, such as turning the radio on and off, thus providing the user with pass-through functionality.

X-MAC. The implementation for this MAC protocol within the Contiki OS is called `CX-MAC` and is located at `../core/net/mac/cxmac.c`. Despite the different name, which simply inherits the affiliation to Contiki, the fundamental idea is the same as in the original `X-MAC`, namely: employing a strobed preamble approach by transmitting a series of short preamble packets [4], and waiting for the reception of an Acknowledgement (ACK) packet as a response to said preamble before transmitting data. A visual representation on the operation of X-MAC can be seen in Figure 2.11. Furthermore, "it allows conserving of energy by truncating those strobed preamble messages, at both the transmitter and receiver and allows for lower latency. Non-target receivers which overhear the strobed preamble can go back to sleep immediately, rather than remaining awake for the full preamble as in conventional Low Power Listening (LPL) approaches" [4].

ContikiMAC. Introduced by Dunkels in 2011, `ContikiMAC` (located at: `../core/net/mac/contikimac` within the Contiki OS) makes use of the concept of periodical wake ups as already used in e.g., `X-MAC` and combines it with a Clear Channel

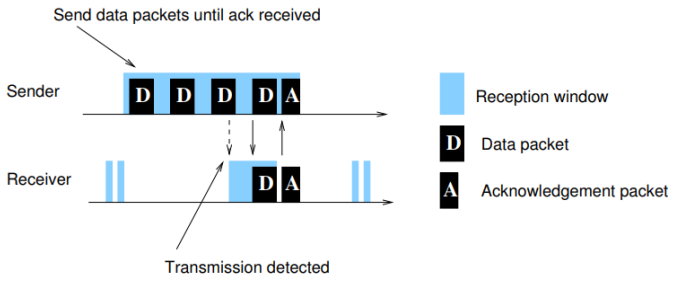

Figure 2.12: ContikiMAC's operation principle, nodes mostly sleep and use periodical checks to detect radio activity, retrieved from [5].

Assessment (CCA) mechanism as well as a transmission phase lock [5]. The basic functionality of this protocol is illustrated in Figure 2.12 and evolves around the idea that a sender repeatedly sends its data messages until it receives an ACK message from the receiver. Moreover, a CCA is used to get information about the current channel: if the current measurement of the Received Signal Strength Indicator (RSSI) is above a certain threshold there is activity on the given channel which lets the receiver make a decision about if he is required to stay awake for packet reception. On the other hand, it allows the receiver to briefly go back to sleep if currently no activity is detected. Furthermore, `ContikiMAC` makes use of an approach called 'transmission phase-lock', this allows for a sender to learn the wake up phases of a receiver. With this knowledge the sender may reduce its periodically transmitted data packets to be synchronized with the wake up phase of the receiver, which directly relates to a lesser number of packets that have to be transmitted [5].

TDMA in the Contiki OS Commonly used in systems with limited resources, e.g., wireless cellular radio systems, such as 2G [37], TDMA is a well-known classical accessing control mechanism. While not being included by default in more recent releases of Contiki, its implementation is located at `../core/net/mac/tdma_mac` for older versions such as 2.6. This protocol within Contiki implements the general idea behind TDMA, a visual representation of this function principle can be seen in Figure 2.13. The main function of TDMA is to split a limited resource (e.g. carrier frequency) into individual frames, that are repeated periodically. Those frames are then split into individual slots that each are assigned to a single user. However, a user is not limited to one slot per frame, in some cases a user might be assigned with multiple slots to prioritize its transmissions. Within their slot, users can carry out its communications. This arrangement into slots allows for multiple participants to use a limited resource, without interfering with each other or risking the loss of data due to transmission collisions [37] [38]. There are a plethora of use cases where TDMA or a combination of TDMA with another channel accessing method is applied (e.g., GSM and IS-136 [39] combine TDMA with other channel accessing methods, such as Frequency Division Multiple Access (FDMA) and Frequency Division Duplex (FDD)). For example: Hong et. al. propose a TDMA based approach to suit the requirements of underwater sensor networks [40]. Even in neural networks TDMA can be applied: Mennes et. al. propose the use of a Multiple Frequencies Time Division Multiple

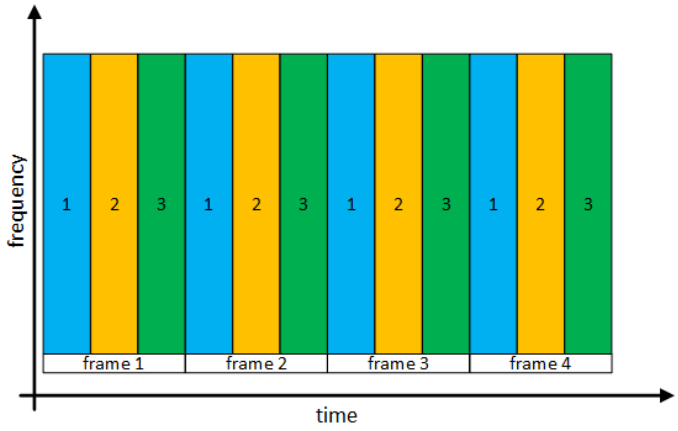

Figure 2.13: Rough functionality of a TDMA protocol, frames are divided into slots for each individual participant.

Access (MF-TDMA) approach to realize a scheduler for collaborative wireless networks [41].

2.3 Hardware

This section will give a more detailed view on the hardware platforms used in this thesis. The STM32L152 Nucleo-64 by STMicroelectronics is presented in Section 2.3.1, while Section 2.3.2 will introduce the Semtech SX1272.

2.3.1 STMicroelectronics STM32L152 Nucleo-64

The base device used in this thesis is the STM32L152 Nucleo-64 from STMicroelectronics. Figure 2.14 shows the layout of the device. Equipped with an ARM Cortex M3 processor that operates at 32 MHz, 512KB of flash memory and 80KB SRAM, the Nucleo provides sufficient computational power as well as memory space to operate as a base for an IoT application. Additionally, this device provides a reset and a user button, a LED, as well as an embedded ST-Link debugger/programmer. This combination of debugger and programmer allows for almost effortless programming of the Nucleo.

If this platform is used in combination with the STM32-Workbench environment [42], the developer has the potential to debug the developed programs on the Nucleo itself and step through his program by using breakpoints and other debugging tools. This System Workbench toolchain is an Eclipse-based Integrated Development Environment (IDE) provided by a third party not affiliated to STMicroelectronics under the name Ac6 [43], which is a service company providing training and consultancy on embedded systems [42].

Furthermore, one of the major advantages of the Nucleo is that it is equipped with standardized Arduino headers. The pin-out of those headers is shown in Figure 2.15. This is particularly convenient, since the LoRa transceiver we want to use in combination with the Nucleo, is the SX1272 from Semtech. The layout and pin-out of the SX1272 is

shown in Figure 2.16: as for it also makes use of Arduino connectors, thus making the interconnection of both devices effortless.

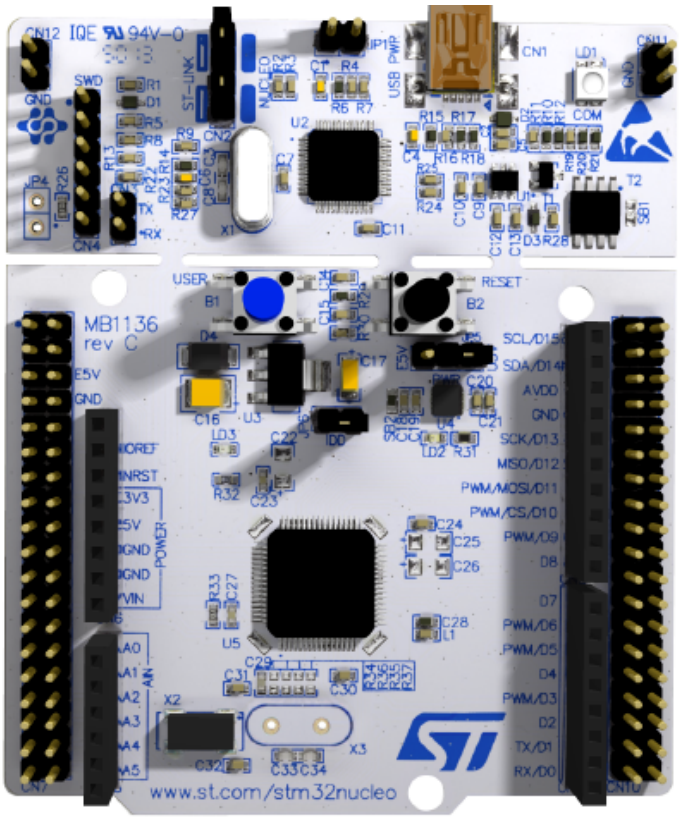

Figure 2.14: Nucleo-64 STM32L152 layout, from [6].

Another asset of the Nucleo and the Semtech SX1272, which will be described more in detail in Section 2.3.2, is that they are supported by the MBED operating system, which also provides an online compiler environment for quick code development and deployment [44]. The latter was used for additional verification and testing purposes of our implementation in Contiki, which will be described in detail in Section 3.

2.3.2 Semtech SX1272

For this thesis the Semtech SX1272 LoRa transceiver is used. This radio is equipped with both a standard FSK and a LoRa modem, which can be operated independently. Depending upon the selected mode, either conventional OOK, FSK modulation, Gaussian Frequency Shift Keying (GFSK), Minimum Shift Keying (MSK), Gaussian Minimum Shift Keying (GMSK) or LoRa spread spectrum may be employed [45]. The high sensitivity of up to -137dBm, combined with an integrated power amplifier that operates at +20dBm, yields a maximum link budget of 157dB, which is optimal for any application requiring long range and robustness [45]. In this thesis, the traditional FSK modem was neglected, while the LoRa functionality was fully implemented.

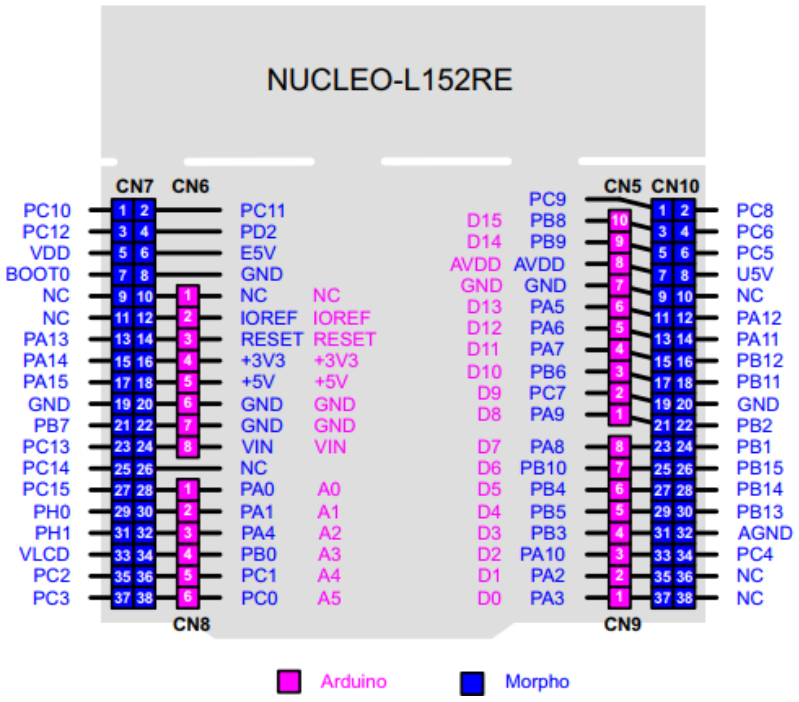

Figure 2.15: Arduino and Morpho headers of the Nucleo, from [6].

- Some other key features of the SX1272 are [45]:
  - Low RX current of 10mA;
  - 127 dB Dynamic Range RSSI;
  - Automatic Radio Frequency (RF) Sense and Channel Activity Detection (CAD);
  - Built-in temperature sensor and low battery indicator.

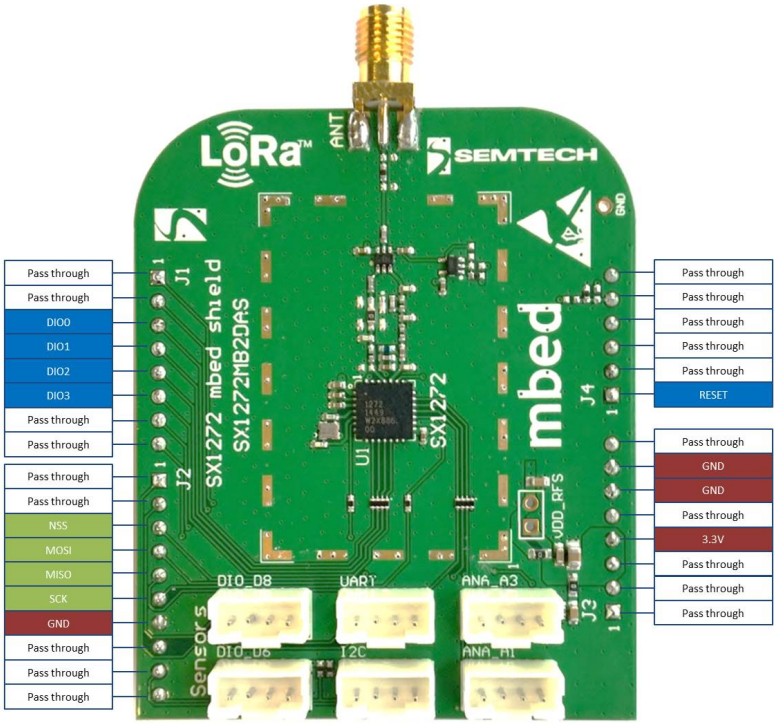

Figure 2.16: Pin layout of the Semtech SX1272, from [7].

Chapter 3

Porting LoRa to the Contiki OS

This chapter presents the process of porting a popular LoRa platform to the Contiki operating system, which was the first step of this thesis. Section 3.1 gives a short overview of related work in regard of porting LoRa to Contiki. Section 3.2 briefly introduces the used software and the required versions of each component. In Section 3.3 we provide more detailed information on the code itself and what parts had to be replaced in the Contiki OS architecture. We further give some reasoning on why we decided to create a separate port.

3.1 Related Work

This section presents work related to this thesis, particularly in context with our port of the SX1272 based LoRa platform to Contiki.

In 2016, Aerts [46] presented an approach to integrate LoRa into the Contiki operating system. The authors use the LoRaMAC project on GitHub [47] as a base. This project is administered and maintained by the LoRa-Alliance and provides a LoRaWAN compliant base system for several hardware platforms. It also includes preconfigured project files for a few Integrated Development Environments, such as KDevelop [46]. In their publication, Aerts further presents a guide, listing several important steps when porting new hardware platforms to the Contiki OS.

The hardware used to validate the results presented in their approach is the Zolertia Z1 sensor node as well as a LoRaMote. The latter is a demo platform equipped with a SX1272 radio chip. Furthermore, the LoRaMote platform is fitted with various sensors which provide a variety of application [48].

However, this implementation has some drawbacks: the source project is only compatible to a few platforms and it implements only the end device of a LoRaWAN network. Therefore, a gateway node has to be obtained by other means. Furthermore, LoRaWAN itself is not open source and thus does not allow for adaptation by a multitude of developers.

3.2 Software

The foundation of our implementation was a clone of the official Contiki OS, version 3.0, since this release already provides support for the Nucleo-64 STM32L152 which is, as described in Section 2.3.1, one of the hardware platforms this Thesis is based on. However

in this Contiki version the Nucleo is only supported in combination with the Spirit1, a different radio module for the Nucleo-64. For the development, Visual Studio Code in combination with the ARM compiler `arm-none-eabi-gcc`, version 4.8.4, was used. As an additional tool for correctly flashing and verifying our device the 'STM32 ST-LINK Utility' by STMicroelectronics was used [49].

3.3 Implementation

In this section we describe the main tasks that had to be accomplished in order to port the SX1272 radio in combination with the Nucleo-64 to Contiki. As stated in Section 3.1, Aerts et. al. [46] proposed a port for the LoRaMote platform to Contiki. However, since this port uses LoRaWAN as its MAC layer application, it did not fit our requirements since it is closed source and provides little freedom to the developer. Furthermore, we desired to use a single type of hardware for both gateway and nodes, which was not possible with a LoRaWAN port. Finally, the used device in [46] did not match any candidate for our use case, thus it was not an optimal starting point for our approach. As already described in Section 3.2, a fork of Contiki version 3.0 was the perfect foundation, supplying the basic functionality of our desired platform where we could build our radio driver upon. Considering this version included an official port of the Nucleo in combination with the Spirit1, a low-power RF transceiver by STMicroelectronics, intended for RF wireless applications in the sub-1 GHz band [50]. The main challenge however, that came up while building upon this version of Contiki, was stripping the unwanted code segments from the essential base parts of the Nucleo implementation, since those were deeply interwoven.

Figure 3.1 shows the general layout of the Contiki directory tree and is used to facilitate understanding where files need to be altered or added in the porting process.

Separating Nucleo and Spirit1 code. As a first step, we altered the existing Nucleo and Spirit1 code, such that the Nucleo and its functions would be available without depending on any Spirit1 code, to obtain that goal, we took the provided example code, located at `./contiki/platform/stm32nucleo-spirit1` and `./contiki/examples/stm32nucleo-spirit1` and made changes such that the code would work just with the Nucleo, discarding all the Spirit1 exclusive code. Towards this goal, we disconnected the SX1272, since it was not needed at this early stage. Since the Spirit1 code is deeply interwoven into the basic configuration of this platform implementation, finding all related code parts turned out to be a cumbersome task, mainly of commenting out or removing unwanted code, recompiling and checking for functionality on the device. Besides the Makefile (`Makefile.stm32nucleo-spirit1`) and all the obvious Spirit1 related files within `./platform/stm32nucleo-spirit1`, the main files that needed to be modified were: `../stm32cube-prj/Src/stm32l1xx_it.c/.h`, `../stm32cube-prj/Src/spirit1_appli.c/.h`,
as well as `./platform/stm32nucleo-spirit1/st-lib.h` within the Spirit1 platform structure.

We also had to change Contiki-specific code, mainly the `platform-conf.h`, `hw-config.h` and `contiki-conf.h` files, specifying the desired protocols, such as `nullmac_driver`, and other platform-specific parameters. Since we aimed for full functionality of the Nucleo, including the LEDs on the board and the button sensor, the callback for the button event had

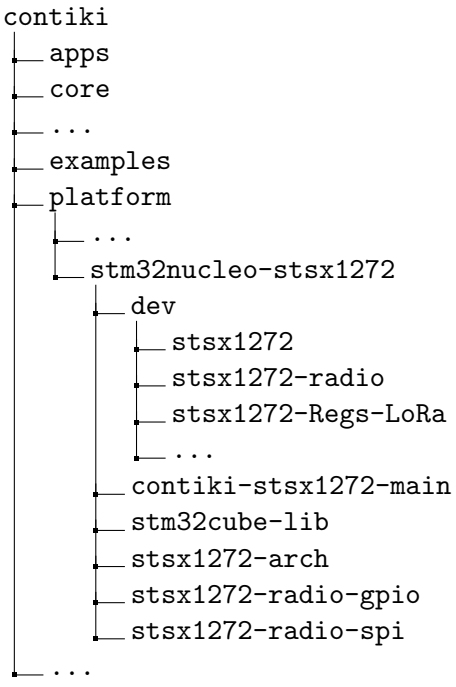

Figure 3.1: Contiki's directory structure with the SX1272 implementation.

to be rewritten, because for the Spirit1 it was implemented within `../spirit1_appli.c`, which, for our approach, is no longer included. At last, a small basic test program was written to assure the correct operation of the Nucleo.

Implementation of the SX1272 radio driver. After that, the integration of the SX1272 radio and thus the implementation of a radio driver was started. Subsequently, after communication with the SX1272 via Serial Peripheral Interface (SPI) and General Purpose Input/Output (GPIO) interfaces was possible, we handled the transmission and reception of packets, hence also providing interrupt handling.

After that, we continued with the integration of the SX1272 radio. For this task we first specified the general structure of the radio-related source files and where they would fit within the Contiki OS structure. Looking at the existing Contiki network stack, illustrated in Figure 2.10, we decided that, as a first step, exchanging the radio layer with the LoRa radio would be sufficient. Thus, we used the default configuration within `contiki-conf.h`: `#define NETSTACK_CONF_RDC nullrdc_driver` and `#define NETSTACK_CONF_MAC nullmac_driver`, which performs no duty cycling optimization, nor carries out retransmissions. For the network layer, we decided to use the existing RIME stack (Figure 2.10) functionality.

With this knowledge, we decided upon a prototype structure for newly generated and SX1272 radio related files within Contiki as shown in Figure 3.1.

The first file that needs to be adapted is `./contiki-stsx1272-main.c`, since it handles the start-up of the device. Within the main routine, one can observe that GPIO and SPI interfaces are initialized right after the Hardware Abstraction Layer (HAL). Thus, we started

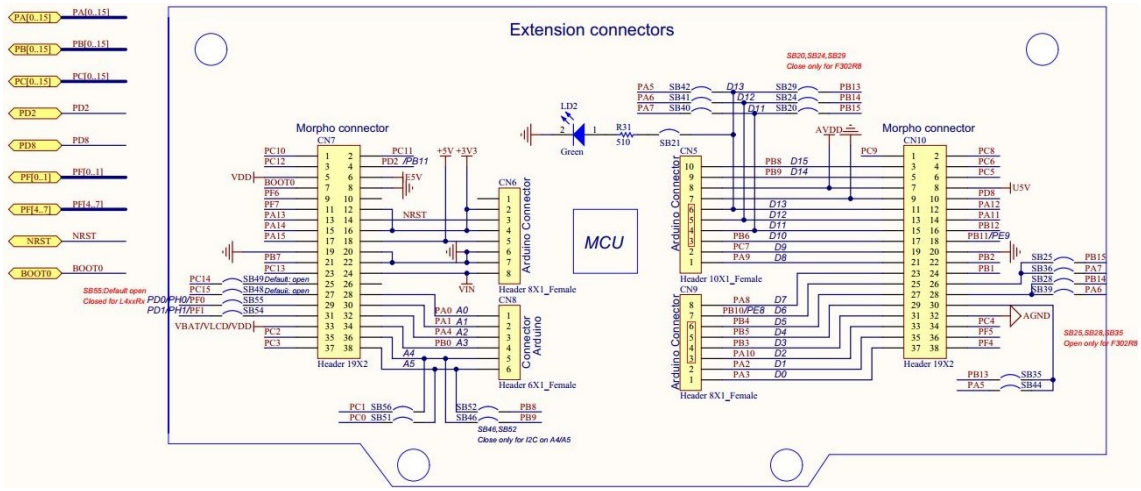

Figure 3.2: Extension connectors of the Nucleo-64, from [6].

to port the GPIO and SPI interface of the SX1272. Defining the files `sx1272-radio-gpio` and `stsx1272-radio-spi`. These files embed the GPIO pin/port configuration to interconnect the Nucleo with the radio device. The relevant GPIO pins and ports can be found when looking at the extension connectors of the Nucleo-64, specified in the user manual [6]. An illustration of the extension headers can be found in Figure 3.2, with the relevant pins marked with a red rectangle.

Another important part of the GPIO files is the definition of how specific events (rising or falling edge) on the individual pins should be handled. In the case of the SX1272, we defined that rising edge events on GPIO pin 0 and GPIO pin 1 should be handled as an interrupt and thus be managed by an interrupt-handler, implemented in `../dev/stsx1272-radio`. An example of how this is specified can be found in Listing 3.1.

Listing 3.1: Example GPIO pin/port configuration.

```
1 #define RADIO_GPIO_0_PIN                GPIO_PIN_10
2 #define RADIO_GPIO_0_EXTI_MODE      GPIO_MODE_IT_RISING
3 ...
4 RadioGpioInit(RADIO_GPIO_0,             RADIO_MODE_EXTI_IN);
```

The next step was to configure the SPI interface, in order to enable write and read functionality to and from the radio registers of the SX1272. Again, the basic definition of pins and ports can be found in `stsx1272-radio-spi.h`, while `stsx1272-radio-spi.c` implements further SPI functionality, such as writing and reading from specific addresses. A peculiarity of the SX1272 radio is the First In First Out (FIFO) data buffer that is used in LoRa mode to write data when sending or receiving data. Thus, we needed to extend the standard `spi_read` and `spi_write` functionality by an adding the `readFifo()` and `writeFifo()` functions, which basically issue a read/write to the SPI on address 0. All of the SPI functions mentioned above are then used by `../dev/stsx1272`, implementing dedicated SX1272 read/write functions (`STSX1272Write()`, `STSX1272Read()`, `STSX1272WriteFifo()` and `STSX1272ReadFifo()`) in order to keep the general structure of the project. Thus, strictly radio related code is placed within `../dev/...`, whereas code files that also implement Nucleo functionality are placed in the parent folder `../stm32nucleo-stsx1272`.

While testing and verifying the functionality of the SPI communication, an interesting pitfall could be observed. For debugging reasons, our first implementation included toggling the LED, if the user button was pressed. After that, we could observe that a SPI reset was triggered. This was due to the issue that the GPIO pin/ port configuration of the LED on the Nucleo is the same as the SPI reset, which can be seen when looking at the extension connectors in Figure 3.2. To fix this issue we had to remove the LED functionality from the current version of the code.

The final step was to implement the general radio functionality, where some parts were reused from the MBED interface [44]. This is done within the file `stsx1272radio.c`. This file implements all necessary radio functions, such as: initializing the radio, preparing a packet for dispatching (`stsx1272_prepare()`), sending a packet (`stsx1272_send()`), or configuring the interrupt handler (`stsx1272_radio_interrupt_handler()`) that gets called upon packet reception and that notifies the upper layers. The aforementioned functions responsible for transmitting a packet were implemented to be compliant with the standard Contiki `NETSTACK_RADIO` nomenclature. A list of the structure of a device driver for a radio in Contiki can be found in the following itemization [36]. Furthermore, an example function (`stsx1272_init()`) can be seen in Listing 3.2.

- int (* init)(void);

  Initializes the radio hardware.

- int (* prepare)(const void *payload, unsigned short payload_len);

  Prepares a packet to be sent by the radio.

- int (* transmit)(unsigned short transmit_len);

  Sends the previously prepared packet.

- int (* send)(const void *payload, unsigned short payload_len);

  Prepares & transmits a packet.

- int (* read)(void *buf, unsigned short buf_len);

  Copies a received packet to Contikis input buffer.

- int (* channel_clear)(void);

  Performs a Clear Channel Assessment (CCA) to find out if another device is currently transmitting.

- int (* receiving_packet)(void);

  Checks if the radio driver is currently receiving a packet.

- int (* pending_packet)(void);

  Checks if the radio driver has just received a packet.

- int (* on)(void);

  Turns the radio on.

- int (* off)(void);

  Turns the radio off (or go to into Low Power Mode).

Listing 3.2: SX1272 Init function.

```
1 int stsx1272_init(void)
2 {
3   STSX1272_Current_Config.Settings.on_off = 1;
4   stsx1272_on();
5   process_start(&radio_process, NULL);
6   return RADIO_RESULT_OK;
7 }
```

3.3.1 PHY Settings Configuration

As already stated in Section 2.1, the four main configuration parameters of a LoRa radio are: carrier frequency, spreading factor, bandwidth and coding rate. All of those parameters can be set with the function `radio_result_t stsx1272_set_value(radio_param_t param, radio_value_t value)`, respectively. This implementation provides an added value, since each relevant parameter can be tuned by the programmer to fit the current use-case. This flexibility can be used to influence the communication range or robustness. A short example for parameter configuration can be seen in Listing 3.3.

Listing 3.3: Fully configurable LoRa parameters.

```
1 NETSTACK_RADIO.set_value(RADIO_PARAM_LR_BANDWIDTH, LORA_BANDWIDTH); // 500
    kHz
2 NETSTACK_RADIO.set_value(RADIO_PARAM_LR_DATARATE, LORA_SPREADING_FACTOR);
    // [SF7..SF12]
3 NETSTACK_RADIO.set_value(RADIO_PARAM_LR_CODERATE, LORA_CODINGRATE);
4 ...
```

Chapter 4

MAC Layer Design

This chapter describes the steps taken in order to design and implement a MAC layer protocol for our previously ported LoRa devices within the Contiki operating system. In Section 4.1 the requirements of the sought MAC protocol will be elucidated, while Section 4.2 will showcase the design of the DeFiL-MAC protocol. Section 4.3 will present the detailed implementation of this protocol and its integration into the Contiki operating system.

4.1 Requirements

The following paragraphs list the requirements for the sought low-power MAC protocol, as well as emphasize the reasoning behind them.

Efficiency. First and foremost a low-power MAC protocol should be efficient, as it is one of the primary factors why such protocols are used within the IoT and its devices. Thus, there is the need for the design to be efficient in terms of power consumption, which directly relates to it being efficient in terms of the amount of time a node spends in its 'active' state, since generally a node requires substantially less energy when it is in its 'sleep' state. Therefore, the MAC is designed to be **duty-cycled**. Moreover, since a master-slave approach is commonly used for LoRa and slave-to-slave communication is usually not intended, the use of a TDMA based approach is a natural fit. Furthermore, those concepts have proven to be fairly efficient in terms of active time, considering a participant has only to be active within its dedicated slot [51] [52].

Reliability. A major aspect of LoRa and LPWAN applications is that, due to its long range capabilities, they can be deployed in hardly accessible terrains and/or spread far apart from each other, which increases the difficulty for physical error handling at the node directly, thus further enhancing the need for reliable transmissions. The importance of transmission reliability is emphasized by the factor that the employed radio technology uses a small bit rate, thus only sending little data at a time, often with a substantial amount of delay in between. For this reason, it is important to achieve a high transmission reliability to minimize delays due to retransmissions, which can be high due to the duty cycle regulations [20].

Another requirement, or sub-requirement, that is directly tied to the first specification for the protocol being efficient, is that LoRa's special properties should be used capitalized to its full potential. The unique ability to fine-tune the physical parameters (as explained in detail in Section 2.1.2) of the radio technology, not only to have a direct influence on the communication and range performance but also altering the power consumption of the system, must not be left unused. Therefore, a defined requirement is to implement an adaptive mechanism to capitalize on the previously stated benefits.

Scalability. Since applications within the IoT are rarely static and have a fixed number of participants, it is also required for our designed MAC to support a flexible amount of members. Furthermore, to enhance its scalability, it is required that the protocol is able to autonomously handle nodes leaving/joining the network.

Compliance to the duty-cycle regulations. As already briefly mentioned in previous sections, when using the license-free sub-gigahertz radio frequency bands,(i.e., 868 MHz in Europe and 915 MHz in North America) users have to be compliant to some regulations defined in the European Standard (Telecommunications Series) [20] for Europe and by the Federal Communications Commission (FCC) for North America. That regulation states, that a device within that frequency band can only be actively sending for 1% of the time, 99% of the time it can only be inactive (e.g., listen or sleep). As this regulation applies to both gateways and nodes, this limiting factor has to be taken into consideration while designing the MAC protocol in order to stay compliant. Furthermore, this regulation limits the amount of nodes that can join a DeFiL-MAC network, since no more nodes are allowed to join if the current members already make use of the 1% active time.

Open-source. One of the features we aim to achieve is to develop a MAC protocol for the open source community. Amongst other benefits, it also brings more attention to the field of LPWANs and LoRa itself, by encouraging developers to set up their own efficient and versatile LoRa network by operating it with DeFiL-MAC.

4.2 DeFiL-MAC: Design

As already discussed in Section 2.1.3 there is an existing MAC protocol for LoRa networks. Therefore, it is not the lack of existence of a MAC protocol why we decided to implement our own, but much more the drawbacks that come with the existing solutions, as well as the desire to specify our individual requirements.

Thus, in this thesis we present DeFiL-MAC. Its name derives from its key features: DeFiL-MAC is a Duty-cycled and eFficient MAC protocol for LoRa. Since a master-slave approach is commonly used for LoRa, and slave-to-slave communication is usually not intended the use of a TDMA based approach is a natural fit. Already defined in the requirements, we desired to provide DeFiL-MAC to the open-source community, and thus implementing DeFiL-MAC within the popular and well-known open-source operating system Contiki was a perfect match.

The starting point of our MAC protocol was to take some of the advantages that LoRaWAN provided and combining them with a conventional duty-cycled, TDMA - MAC

design. Figure 4.9 illustrates where within the Contiki network stack our MAC and its components should be placed, a more detailed overview of the implementation will later be given in Section 4.3. We also decided, that the MAC should handle turning on and off the radio, i.e. do the work of the RDC layer, which makes an implementation within the RDC-layer redundant, thus keeping the built-in `nullrdc` in place (which provides no functionality to switch the radio on or off at all), while still providing radio duty cycling functionality within the MAC-layer.

Generally, there are two different participants in the composition of this MAC: a master (aka. gateway) and a slave (aka. node). The hardware used for gateway and node can be identical, however the master node is assumed to not be battery powered, thus energy consumption is not the focus for those node types.

For energy efficiency reasons, and the assumption that the master is not battery powered, most of the computations as well as the decisions are done on the master node. A master node is considered to be always on, thus being either in receiving, transmitting or idle state. Figure 4.1 illustrates the operation of DeFiL-MAC by an example with three participants. A master node, that is the central director of the whole LoRa network and in this case, two slave nodes.

The master is responsible for the coordination of the network and also for processing the data received by the single nodes. As we can see in Figure 4.1, the time frame where every communication within the protocol happens is called a 'SuperSlot', the length of this SuperSlot can be programmed by the developer in order to suit the needs for the application. Furthermore, duty cycle regulations state that the radio can only be active (i.e., actively transmitting) for 1% of the time. In our application this is also expressed with the SuperSlot, so the total active radio time is calculated by calculating 1% of the SuperSlot, e.g., for a SuperSlot of five minutes, the radio can be active for three seconds, referred to as 'ActiveSlot'. Directly dependent on that ActiveSlot is how many nodes can be supported within the current network and how much time each node gets to transmit its data to the gateway. The amount of time the gateway uses to advertise by sending so called 'beacon' messages to nodes is a fixed parameter, specified by the 'BeaconSlot' - parameter.

Join process and data exchange. If a node wants to join the network managed by a given gateway, it has to respond to a received beacon message, which basically is a very short, broadcast message that is always sent with the most reliable physical settings possible, which simultaneously consume the most energy. The communication steps of such a join process can be seen in Figure 4.2.

In the example overview, Figure 4.1, Node 2 is already part of the network where Node 1 is trying to acquire an available slot. A situation where multiple nodes try to join the network at the same time is illustrated in Figure 4.3.

Only one node can join per cycle: in particular, the node from which the join request message is received first by the master, is accepted into the network. If a free slot is still available, a confirmation message containing the assigned slot will be immediately sent to the node, which is required to confirm the reception by sending an empty message back to the master. If a nodes join request is not accepted by the master, it will back off and try again upon reception of the next beacon. If, for whatever reason, the node is not accepted a second time, it will back-off and try to join the network again on every $2^n th$ received

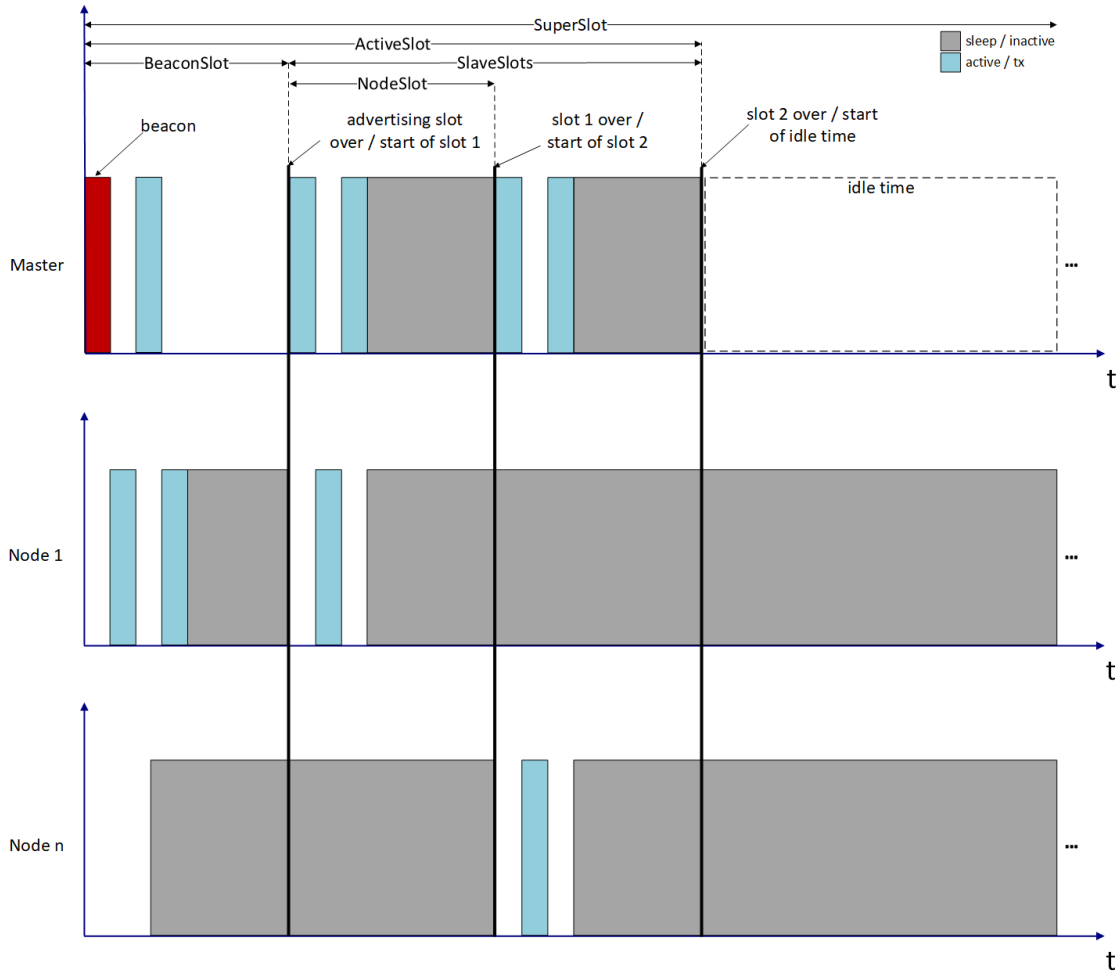

Figure 4.1: General layout of DeFiL-MAC. A beacon message is represented by a red rectangle, while blue depicts a message transmission. Grey areas illustrate idle or sleeping phases of the nodes.

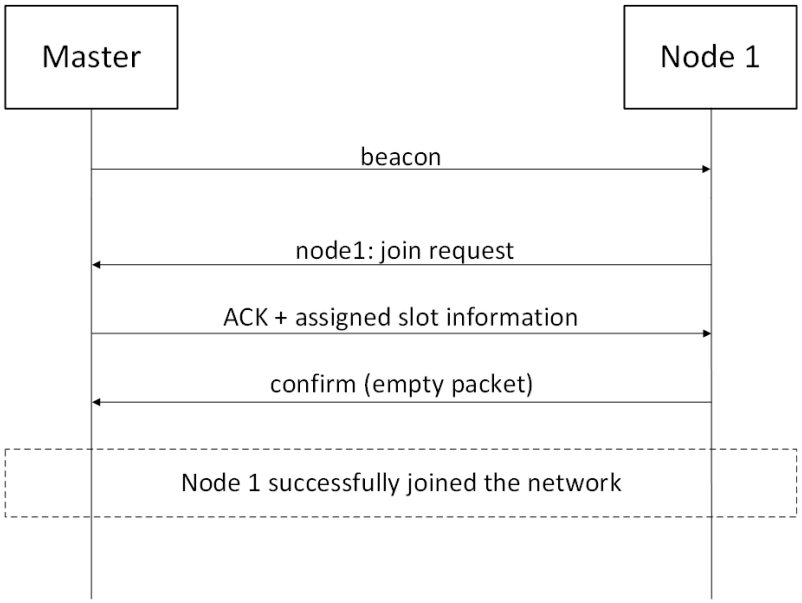

Figure 4.2: Communication process in which node attempts to join a network. The acknowledgement also contains information about the slot a node has been assigned to.

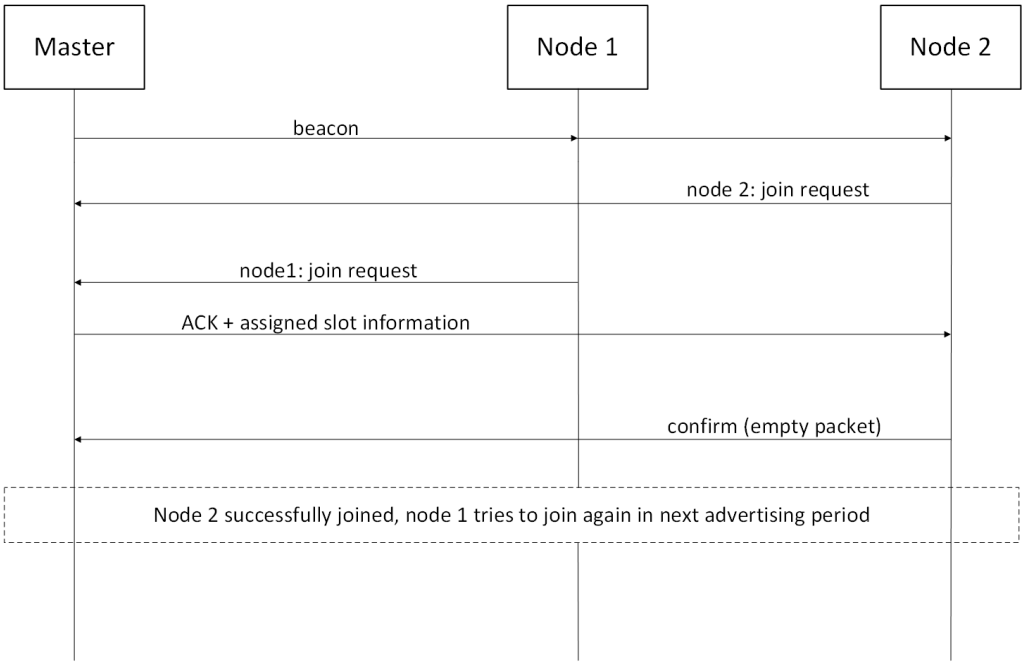

Figure 4.3: Operation of DeFiL-MAC when multiple nodes try to join the network at the same time. Node 2 is accepted into the network, while Node 1 will try to join again upon reception of the beacon within the next cycle.

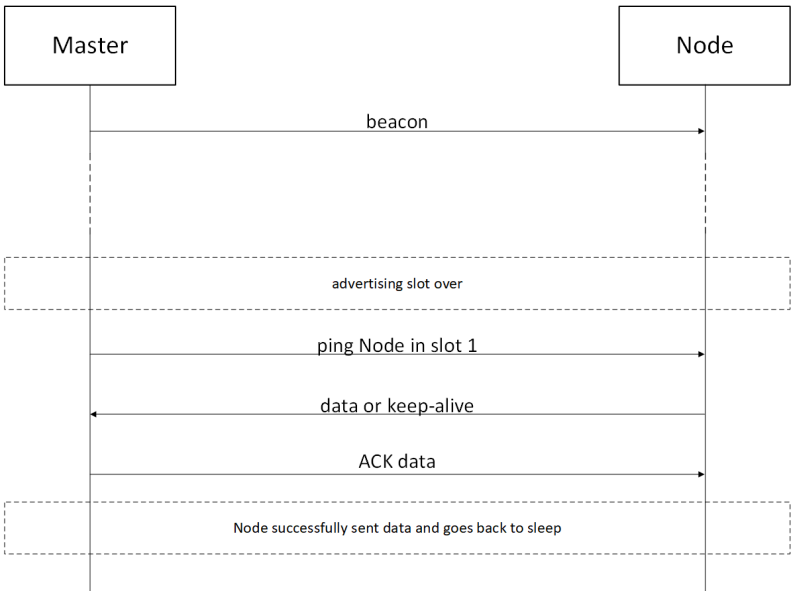

Figure 4.4: Communication steps between master and slave node when successfully exchanging data.

beacon. Furthermore, as each node has an unique identification number within the network, this is used to offset the sending of a join request after the reception of each of the $2^n th$ received beacons. This avoids collisions as well as blocking the master node for requests of other nodes in the occasion that the downlink (messages from master to slave) is faulted.

A node is classified by an unique identifier (e.g. a node ID) within the network, which is on the one hand used by the master to direct its messages to certain nodes, as well as by all other nodes within the network to distinguish between messages relevant and irrelevant to them. Generally, all messages within the network can be received by all nodes operating on the same PHY settings and all nodes that join a network use the most reliable settings as their default, this matter will be of vital importance when we introduce variable PHY settings for nodes, which will be introduced later.

Once a node has joined the network, it has its fixed slot where it can send data to the master. The detailed communication process can be seen in Figure 4.4, as soon as the gateway finishes advertising the network within the BeaconSlot it sends a specific message to the nodes in their corresponding slots, this message is called a 'ping' and confirms to the slave that the gateway is ready to receive and process its data.

If a ping is received by a node and the id contained in the message corresponds to its own, the node sends either data or a keep-alive message to the gateway. Each node has a queue for outgoing messages: if this queue is empty a keep-alive message is sent, in order to inform the gateway of its existence. After correctly receiving the message from the slave, the master responds with a short ACK message. After reception of said message, the slave will return to sleep. In the case that the slave does not receive an ACK or the ACK is lost, it will try to resend the data to the master, within its current slot. If this resending of data is enabled, DeFiL-MAC has to assume that a node has to use the maximum configured resend tries to receive an ACK. What has to be taken into consideration here is, that

enabling resending of data and increasing this resend count will increase robustness of the transmission, as it is freely configurable by the developer. However, it will also heavily increase the slot time per node, which will reduce the amount of nodes that can be supplied within the network.

The master keeps track if a slave does not respond to a ping, and after a set amount of missed pings, the node will be removed from the network, freeing up the slot for a new node. The amount of pings that have to be missed is also fully configurable by altering the `NODE_NO_RESPONSE_MAX` parameter. The same procedure is also pursued if a node wants to leave its current network: within its slot, the node sends a specific message to the gateway which then removes the node from its internal list and frees up the slot. The gateway does not confirm this request by sending an ACK, since in the worst case if the request to leave is lost, the node still leaves the network and will not respond to future pings from the gateway, thus will be removed after `NODE_NO_RESPONSE_MAX` missed pings.

A successful data exchange between gateway and node is illustrated in Figure 4.4, after that, if there are more nodes in the network the same procedure is repeated initiated by a ping of the master. If no more nodes are present within the network the master will idle for the remaining time of the SlaveSlot (see Figure 4.1), before starting to advertise the network again by sending its beacon message. All slaves already in the network will wake up to receive this beacon message and each beacon in the following cycles, in order to synchronize their timers, e.g., adjust for possible clock drift, and verify the correct delay to the starting time of their individual slot. For this purpose, a timestamp of the last received beacon is compared to the timestamp of the currently received beacon. As the target delay between two beacons is a known parameter, a slave node can compute the given offset to its master and thus adapt its sleep cycle accordingly.

A state chart for gateway and node can be seen in Figure 4.5 and Figure 4.6 respectively, to further fathom the functionality and states each type of node can go through.

Adaptation of physical layer settings. With the basic functionality of DeFiL-MAC described, we want to introduce another of its key features, which is the ability to automatically adjust PHY settings depending on the current performance of the communication.

Basically, the gateway of the network is responsible for the coordination of this additional functionality. It keeps track of the current communication performance with each individual node. For this purpose, each time the gateway receives a message from one of its network participants, it calculates the current RSSI and SNR values as well as the Packet Reception Ratio (PRR) for said node. RSSI and SNR values are extracted from the SX1272 register map, by reading `RegPktRssiValue` and `RegPktSnrValue` respectively. The PRR is estimated by the gateway by dividing the number of messages sent to a specific node by the amount of received messages from that node.

Additionally, each time a slave sends its data to the gateway, it includes the accumulated RSSI and SNR values from the gateway's messages, since those parameters are dependent on the direction (e.g., uplink can be flawless, while downlink has interferences). A more detailed description on how those values are accumulated and how they are computed into a representative value will be presented in Section 4.3.

The entire decision and computation if and when to change the PHY settings is kept at the gateway, since, as already mentioned, it is assumed to be more computational powerful and to have no limitation to its power supply. This decision is usually based upon the

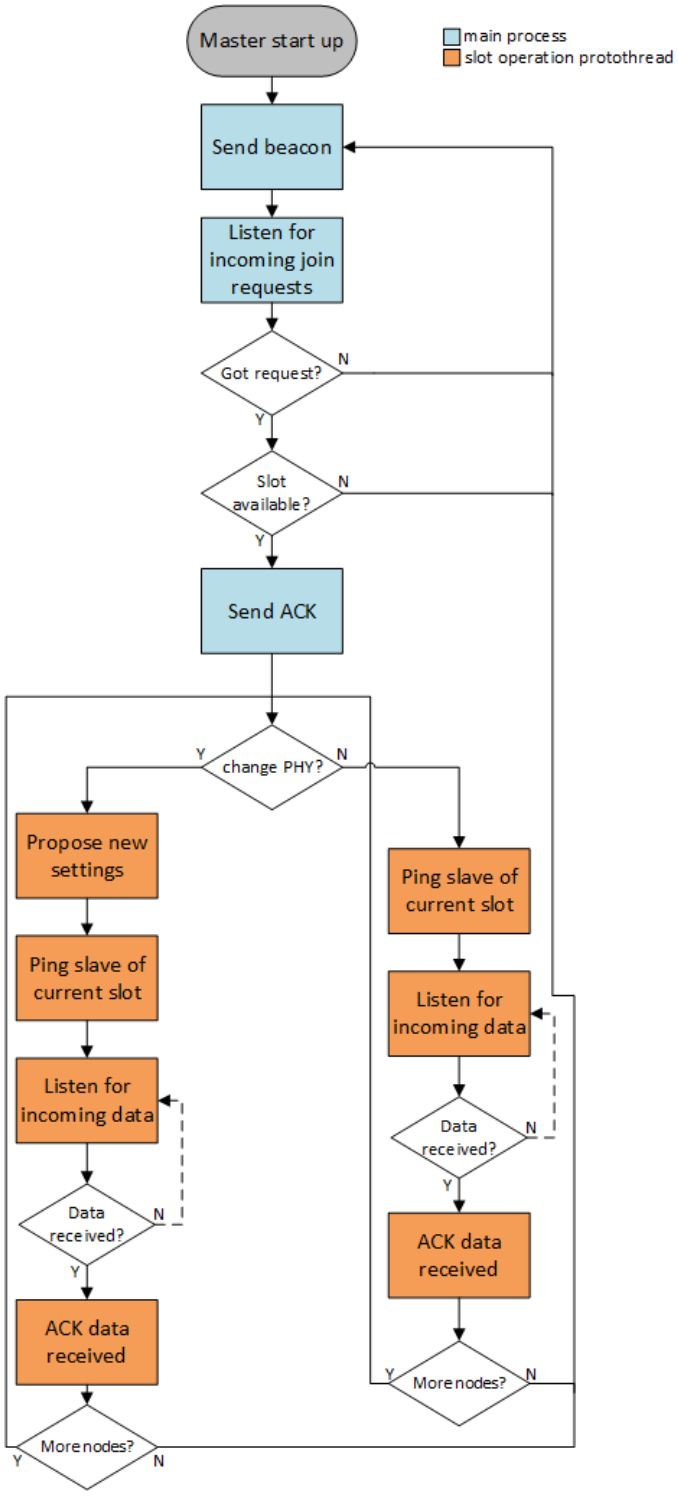

Figure 4.5: State chart of a gateway.

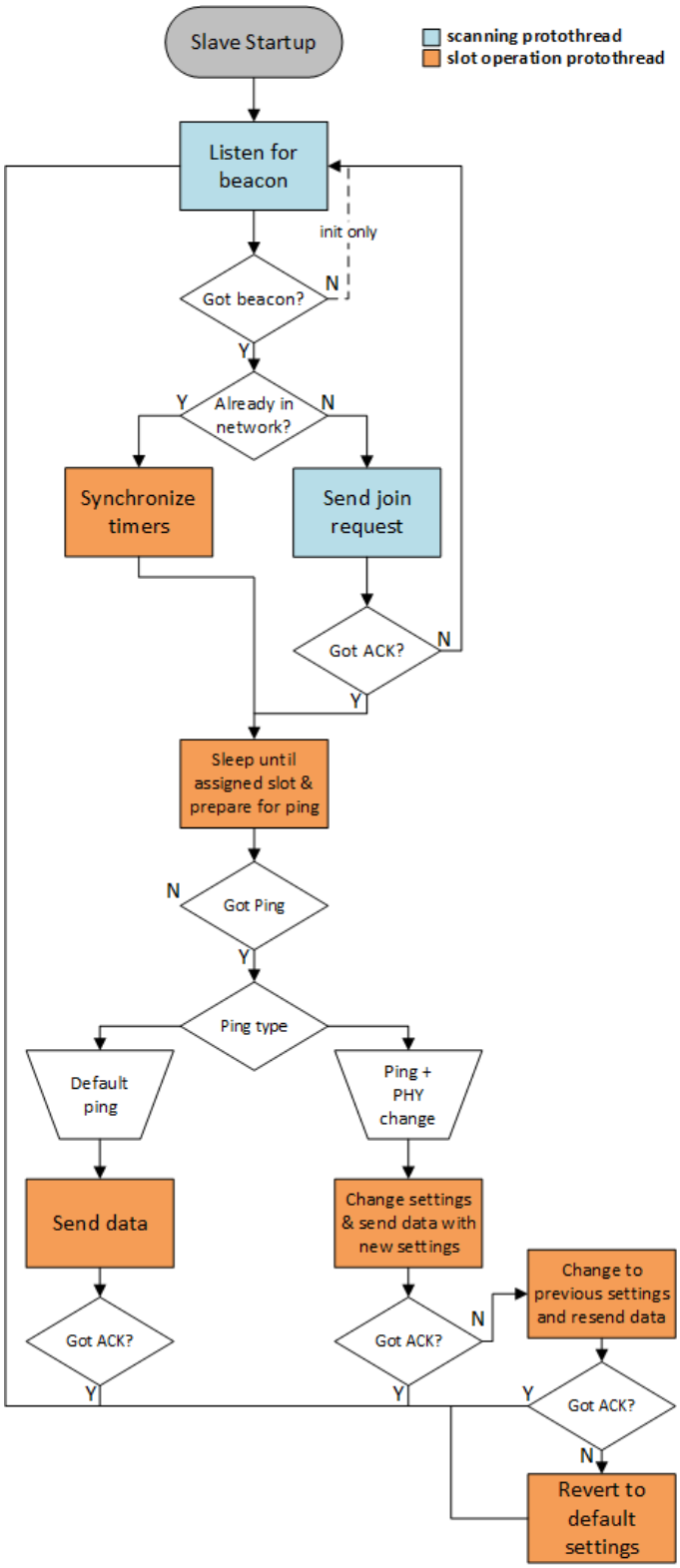

Figure 4.6: Different states a slave node cycles through while operation.

amount of packets that has been received from a specific node, and if the RSSI and/or the SNR is below/above a programmed threshold. This decision to change the PHY settings of a node can either be to improve energy efficiency, by proposing faster, less reliable settings, or to increase robustness at the cost of energy by proposing more reliable settings. If the gateway decides a change of parameters is suitable for an individual node, it sends a special ping message (see Figure 4.1) to the affected node. This ping has a distinct message header and contains, beside the id of the targeted node, an id of the proposed PHY settings. Those ids and their related settings are freely programmable by the developer. However, for evaluation purposes, those settings and their ids are defined by energy consumption from high to low, and will be explained more in detail in Section 4.3, which will also give overview of all message types available in the MAC protocol.

Figures 4.7 and 4.8 further emphasize the process of the PHY switch. After the successful reception of the ping message, the node changes its physical settings to the ones proposed by the master node before sending its data. In the meantime, the gateway changes its settings as well, expecting incoming data on the newly communicated settings. If it correctly receives the packet, it saves that this slave is now using the proposed settings and sends an ACK message to the node, before continuing with the next node in the list. Upon reception of the ACK message, the slave also saves that the master expects data from it on the proposed settings and goes back to sleep. Thus, both participants have an up to date information about the current settings they agreed on. This process can be seen in Figure 4.7. If, after the PHY change, data from the slave gets lost, the slave tries to revert its settings to the previous ones and resends the data. This case is displayed in Figure 4.8. The gateway also reverts its settings to the ones used by the slave before a PHY change was proposed, after not receiving data on the newly proposed ones for a certain amount of time (defined by the slot time that is available per node, see Figure 4.1). If the data then is received correctly, both nodes will save the currently used settings as the expected ones. In the event that still no data is received, thus no ACK is being sent by the master, both nodes will know to revert their settings to the ones defined by default (the most reliable settings), losing the progress towards becoming more energy efficient, but increasing reliability of the protocol because the node is not lost from the network. In both cases, if the change of settings is successful or unsuccessful, the master will try to re-evaluate its decision for that individual node after the minimum amount of packets for a PHY change have been exchanged again.

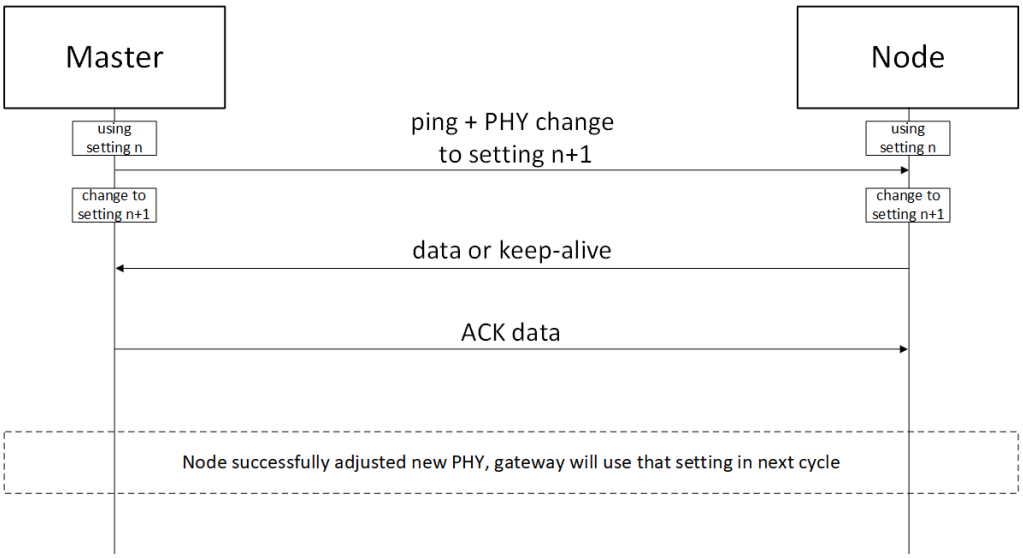

Figure 4.7: Communication sequence between gateway and node, successful adaptation of PHY settings.

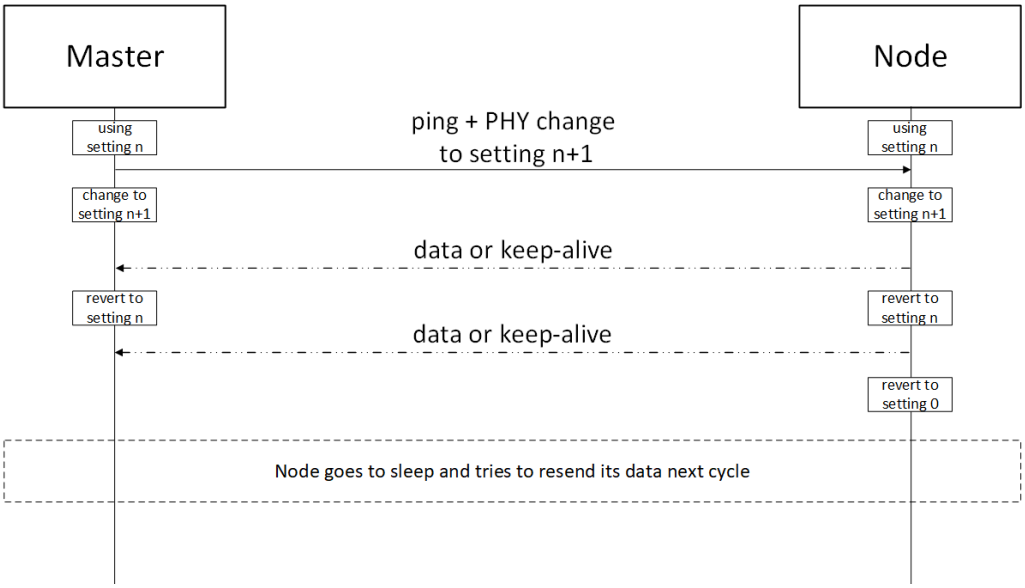

Figure 4.8: Failed PHY switch, communication between master and slave node.

Design Challenges. As already previously stated DeFiL-MAC applies the concepts of TDMA, which implies that the timings when nodes are waking up or between node slots need to be fairly precise. Thus, the main challenge while designing DeFiL-MAC was to coordinate and synchronize this timing among all nodes as well as the central gateway node. If, for example, a node misses a beacon or its dedicated slot by waking up 200 ms too late, it can lose its slot within the network or miss its opportunity to optimize its

physical settings. Hence, the difficulty lies in finding the trade off between waking up early to avoid missing vital packets and achieving the sought efficiency by waking up as late as possible. One possible solution would have been to always include the current timestamp of the master's clock with each message transmitted to allow slaves to synchronize to it. However, this would further reduce the data rate, allowing for only very little data packets. Thus, this solution was not chosen. Instead, we use the known timings of each slot and the known delay between beacons to estimate the timing offset to the gateway. By improving this offset with each received beacon, we can iteratively minimize the time a node needs to stay awake.

4.3 DeFiL-MAC: Implementation

In this section we present our DeFiL-MAC and give a detailed explanation of the implementation process within the Contiki operating system.

Integration into Contiki. The general directory structure of Contiki was already briefly mentioned in Section 3.3 in Figure 3.1, the source files for our MAC were placed within the 'core' folder of Contiki, a more explanatory view is shown in Figure 4.10. Furthermore, in Figure 2.10 the provided protocol stack within the Contiki OS was shown, Figure 4.9 gives a more explicit view on which protocols were exchanged for the integration of DeFiL-MAC.

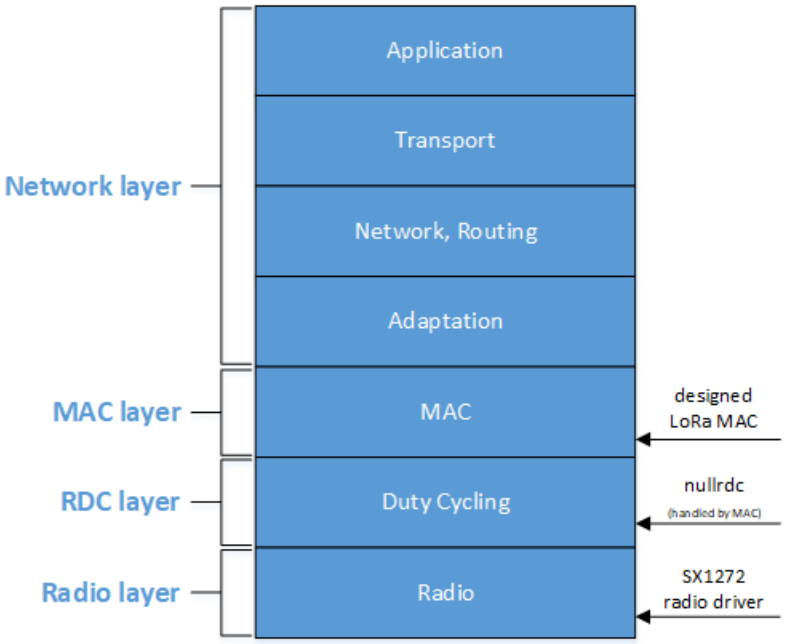

Figure 4.9: Network protocol stack of the Contiki OS, illustrating where implementations have been made.

For a MAC protocol to be to be compliant with the Contiki MAC layer driver interface, a few functions are mandatory to implement, those are:

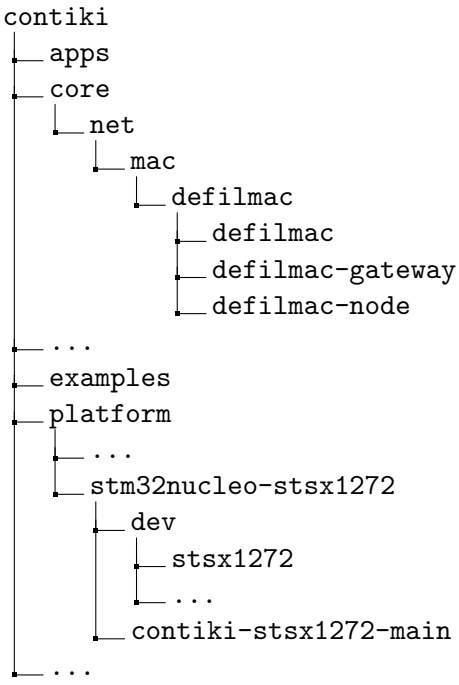

Figure 4.10: Contiki's directory structure with the rough layout of the MAC protocol implementation.

- static void init(void);

  Initializes the MAC driver.

- static void send_packet(mac_callback_t sent, void *ptr)

  Sends a packet and calls the 'sent' callback function.

- static void packet_input(void)

  Informs the upper layers (e.g. Link Layer Security (LLSEC), or Network Layer) of an incoming packet.

- static int on(void)

  Turn the MAC layer on.

- static int off(int keep_radio_on)

  Turn the MAC layer off.

- static unsigned short channel_check_interval(void)

  Returns the channel check interval, expressed in clock_time_t ticks.

All of those functions and their intended functionality have been implemented within the `defilmac.c` file, as it serves as the centralized originator for the further implementations.

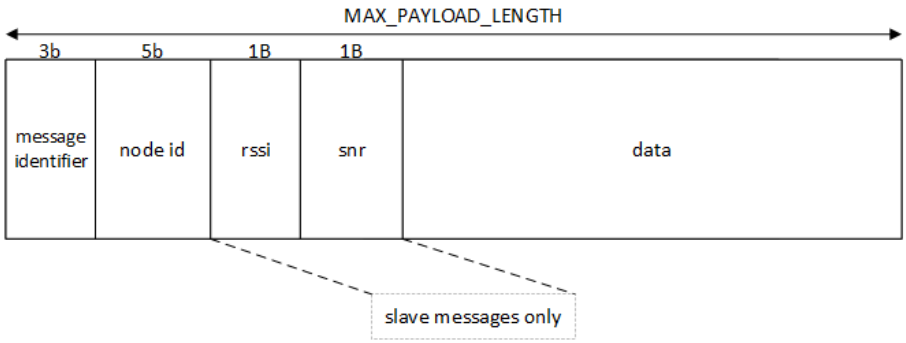

Figure 4.11: Message format used within the DeFiL-MAC.

DeFiL-MAC message structure and message types. Since for LoRa all messages are broadcasts and can be received by all nodes on the same physical settings as the originator, it is important to identify if a specific message is addressed at a certain node. For that reason, all nodes within the network are assigned a unique identifier, and if a message is sent from the master to a specific node this identified is included within the header of the message. If a message is received by a node, it can definitely distinguish if this directive (e.g., a ping) is targeted for it. Furthermore, there are several specific message identifiers at the start of a transmitted message, again to simplify the identification of the purpose of the message. The general structure of such messages used within DeFiL-MAC is illustrated in Figure 4.11. A table of all the messages used for communication in DeFiL-MAC is shown in Table 4.3. It can be observed that messages from gateway to node are encoded with a starting zero in their identifier, e.g., `0b001` marks a message type ping from master to a slave node. Messages sent from an end node to its gateway are flagged with a leading one in their encoding, e.g. `0b100` is the identifier for a join request from node to gateway within the advertising period.

Initialization phase. As soon as Contiki is within its start-up routine, and DeFiL-MAC is selected, the `init` function is called. Before the MAC protocol is switched on, there are several things that have to be verified within this `init` function. One of the main conditions that has to be established is, whether the duty cycle regulations and the necessary network timings can be met with the parameters set by the developer. This is done within the function `calculateDutyCycleSpecs()`. Among other parameters, this function takes the defined 'SuperSlot' duration to calculate other mandatory slot lengths and further network statistics (see Figure 4.1). Furthermore, it is at this point where it is decided how many nodes can be supported by DeFiL-MAC under the assumption of the default (most reliable) physical settings as well as the other currently programmed parameters, such as the SuperSlot duration, the duration of the BeaconSlot, or if the retransmission of failed data packets is enabled and if this is the case, how many retransmissions have to be accounted for in the worst case.

Another important part of those calculations is to forecast a 'worst case' prediction on the time-on-air of a packet with the maximum data length (the LoRa Alliance advises to limit the payload to a maximum of 59 bytes for low data rates e.g., $SF = 11$ [17]) allowed,

Encoding	Message identifier	Description
0b000	MSG_BEACON	Beacon message, used for advertising the network and signaling nodes that the join period is now.
0b001	MSG_PING	Message used to ping a specific node, requesting data or a simple reply from it.
0b010	MSG_PING_PHY	Request data from a slave and order a change of PHY settings. Contains the ID of the proposed settings.
0b011	MSG_ACK	Signals the node that the most recent transmission was received correctly by the gateway.
0b100	MSG_JOIN_REQUEST	Sent by the node to request a slot within a network. Sent directly after reception of a beacon.
0b101	MSG_SLAVE_DATA	Message containing the data collected by the node.
0b110	MSG_SLAVE_KEEPALIVE	Message used to confirm reception of an ACK from the gateway, or in place of data, if no data has yet been acquired.
0b111	MSG_SLAVE_LEAVE	Message used by a slave node if it desires to leave its current network.

Table 4.1: Different types of messages exchanged by gateway and node.

and thus, derive the minimum slot length for each slave from this value. This calculation is illustrated in Eq. 4.1, Eq. 4.2, and Eq. 4.3 respectively, which are derived from [53]. Equation 4.1 determines how many symbols are needed to transmit a certain amount of payload, where Eq. 4.2 calculates the amount of time it takes to transmit one symbol. Finally, in Eq. 4.3 the time-on-air for an entire packet, including its preamble can be estimated. As shown, the duration per transmitted symbol is directly dependent on some of the PHY settings used, such as Coding Rate (CR), SF and bandwidth.

$$\text{payloadSymbNum} = 8 + \max\left(\text{ceil}\left(\frac{8PL - 4SF + 28 + 16 - 20\text{H}}{4(SF - 2DE)}\right)(CR + 4), 0\right) \quad (4.1)$$

where: PL = number of payload bytes
SF = used spreading factor
H = 0 if header enabled, H = 1 if header disabled
DE = 0 low data rate optimization
CR = used coding rate

$$T_{sym} = \frac{2^{SF}}{BW} \tag{4.2}$$

where: SF = used spreading factor
BW = used bandwidth

$$\begin{aligned} T_{\text{payload}} &= payloadSymbNum \cdot T_{sym} \\ T_{\text{packet}} &= T_{\text{preamble}} + T_{\text{payload}} \end{aligned} \tag{4.3}$$

Furthermore, T_{packet} is used in combination with the programmed SuperSlot duration, as well as the BeaconSlot duration to determine how many slaves can be supported within the current DeFiL-MAC network, and how much time each individual slave is assigned to send its data. This calculation can be seen in Eq. 4.4. If either the duty-cycle regulations cannot be met, or there is not enough time for one node within the network, the `init` function will stop, and return an error to the user since a reasonable functionality of DeFiL-MAC is not possible with the programmed parameters.

$$\begin{aligned} T_{\text{totalActive}} &= T_{\text{SuperSlot}} \cdot 0.01 \\ T_{\text{totalSlaves}} &= T_{\text{totalActive}} - T_{\text{BeaconSlot}} \\ T_{\text{minPerNode}} &= \lceil T_{\text{packet}} + \Delta_{safety} \rceil \\ N_{\text{MaxNodes}} &= \lfloor \frac{T_{\text{totalSlaves}}}{T_{\text{minPerNode}} \cdot resendCount} \rfloor \end{aligned} \tag{4.4}$$

where: $T_{\text{SuperSlot}}$ = programmed duration of the SuperSlot
$T_{\text{BeaconSlot}}$ = programmed duration of the BeaconSlot
Δ_{safety} = time added for safety, up to 250ms
$resendCount$ = amount of retries for data transmission

If those initial checks have been concluded successfully, two lists are initialized, a 'packet queue' - list (`packets_to_be_sent`) as well as a 'slave member' - list(`nodeList`). The packet queue list is necessary in the case that no network has yet been established, but a program is actively calling the send method. If a program within Contiki issues a call to `NETSTACK_MAC.send_packet()`, `send_packet()` within the DeFiL-MAC main file (`defilmac.c`) will be called. As we only want to send packets if our node has already joined a network, this `send_packet()` will check if the node has an active network that it is part of. Furthermore, since we can not send data whenever it is ready, but only within our dedicated slot, the packet is added to a packet queue. The size of this queue can be defined by the developer and consists of `N` packets with the length of `MAX_PAYLOAD_LEN` each. If a node is allowed to send data, it will then send the first packet in the queue.

If there are currently no packets in this queue the node will send an empty 'keep alive' message (message identifier `0b110 - MSG_SLAVE_KEEPALIVE` 4.3) to inform the master of its presence within the network. The 'slave member' list, however, contains a list of all slave nodes currently in a network and is actively used by the master do communicate with its nodes and implements some basic list functionality such as: `createAndInsertAtHead()`, `createAndInsertAtTail()`, `getNodeById(uint8_t searchId)` or `removeNodeFromList(uint8_t Id)`. A member object of this list and its variables can be found in Listing 4.1.

Listing 4.1: A member object of the LoRa node list.

```
1  struct loraNode
2  {
3    uint8_t _id;       // unique identifier
4    uint16_t _slot;      // assigned slot
5    uint8_t _phySettings;   // currently used settings
6    int16_t _rssi;       // accumulated rssi value
7    int8_t _snr;       // accumulated snr value
8    int16_t _Mrssi;    // accumulated rssi value of master to slave connection
9    int8_t _Msnr;      // accumulated snr value of master to slave connection
10   uint16_t _pktCount;    // number of received packets
11   uint16_t _txPktCount;    // number of packets sent to
12   uint8_t _miaCnt;     // number of cycles node is missing for
13   uint8_t _prr;        // packet reception rate
14   struct loraNode *next;  // next node in network
15   struct loraNode *prev;  // previous node in network
16 };
```

The final step within the initialization sequence of the MAC is a call to the `on` function, which will handle the further operation.

Main process flow. As illustrated in Figure 4.10, and when recalling Figures 4.5 and 4.6, the main functionality of DeFiL-MAC is split into two separate structures: one for the gateway and one for the node. Depending on whether the current compilation is done for a slave node or for a gateway node, the corresponding processes and threads are started. This differentiation can be made by setting the `IS_MASTER` flag within `project-conf.h` of the current Contiki project (e.g., `../examples/stm32nucleo-stsx1272/defilmac-demo`) to either `true` or `false`.

Basic functionality of the main master process. For the master node, the main process (`defilmac_master_process`) handles the basic functionality of the gateway such as advertising the network by sending beacon messages at the programmed delay, handling the reception of possible join requests and assuring the correct sleep cycles in order to be compliant with the duty-cycle limitations. The beacon messages are always sent with the most reliable PHY settings possible, to maximize the probability for its reception amongst all nodes.

Within its advertisement phase and after a beacon has been sent, the master waits for a specified amount of time, if the `pending_packet` flag gets set by the radio driver. If this flag is set, the master handles reception of the packet and verifies if the latter has the correct message format of a join request (see Table 4.3). If those checks are successful, the master verifies if there is still space available in the network by checking if the counter `CurrentNodesInNetwork` is smaller than `MaxNodesInNetwork`. On success, the requesting node will be added to the previously initialized list. The master's main process will then spawn a separate protothread (`m_slot_operation_pt`) that handles the direct communication with each individual node, making sure to return control to the main process for the purpose of sending beacons and handling join requests. The behaviour and main states as well as decisions of this protothread are illustrated within Figure 4.5, where the main process is represented by blocks in blue and the protothread by blocks in orange. At this stage no more nodes are allowed to join the network until the next advertising

period. This is done to ensure that the master is not blocked by a node requesting access to the network within another node's communication slot.

Master: Slotted Operation Protothread. After being spawned by the main process, this protothread retrieves the previously set up list of all nodes currently within the network. As this list is sorted by assigned slots in an ascending fashion, the node in the first occupied slot is always head of the list. Thus, the slot operation protothread grabs the head of the list and assures that the correct delay for its individual ping is set correctly. This is especially important if we consider that a node in slot 0 has left the network, therefore slot 0 is not allocated and the first slot that expects its ping is in slot 1 so the gateway needs to wait for the exact delay until it can ping the node in slot 1, as otherwise this node will be not listening and thus synchronization is lost. After this, the protothread verifies that the PHY settings of the current node match those used on the master. If not, possibly the ping can not be received by the node. As already illustrated in Listing 4.1 the list of nodes contains the value of the settings that are currently used on the slave node: as this list is always kept up to date while communicating with the slave, the master can relatively easily verify the use of the correct settings. In the event of detecting a discrepancy between the settings used on the gateway and the node, the gateway makes sure to change its own parameters before sending a ping, as this changing of parameters takes less than 50ms it has virtually no effect on the remaining timings. This behaviour gains increased importance as soon as the adaptation of the physical settings is introduced, where, there could be cases where each node uses different individual settings.

Runtime adaptation of PHY settings. The decision if a change of parameters is issued for a specific slave is done by the master. For the master to be able to make that decision it calculates and stores the link metrics, RSSI, SNR and PRR, for that specific node. As RSSI and SNR values can be very volatile, they are combined into their average values by using the Exponential Weighted Moving Average (EWMA), it is depicted in Eq. 4.5.

$$S_t = \begin{cases} Y_1, & t = 1 \\ \alpha \cdot Y_t + (1-\alpha) \cdot S_{t-1}, & t > 1 \end{cases} \tag{4.5}$$

where: Y_1 = initialization value at time t
α = degree of weighting decrease, constant factor between 0 and 1.

The advantage this method brings is that, with the α parameter, the developer can have a direct influence on the behaviour of the averaging process. By selecting an α that is closer to 1, the averaged sum values more recent measurements higher and is hence more reactive. Thus, within our use case, a higher α (e.g., in the range of 0.6 to 0.8) can be used to react to things that have an influence on the link quality between master and node (e.g., a moving obstruction in front of a node) without losing the general benefits of an average sum, such as the bad influence of outliers. Naturally, the more packets are exchanged between the master and a node, the more precise this calculation gets, thus the minimum amount of packets that need to be transferred for the master to decide about a change of the physical settings can be freely set by the developer. In our current implementation of DeFiL-MAC we used a value of α to be 0.8, wielding satisfactory results.

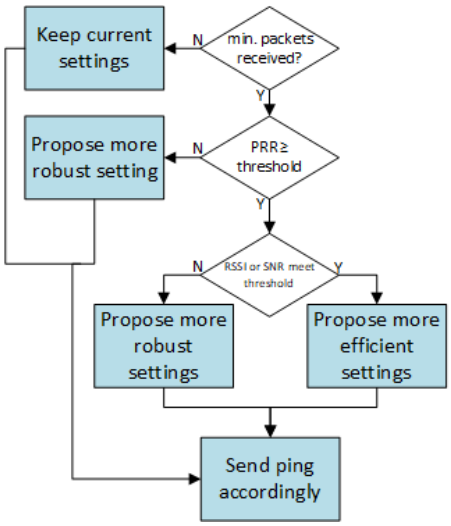

Figure 4.12: Decision graph of the master node, evaluating if a PHY change should be performed.

After the master has received the defined amount of packets (can be chosen by the developer by altering the `MIN_PACKETS_TO_PHYCHANGE` define) from the current node and the averaged values for RSSI and SNR as well as the PRR have been calculated, the decision if a PHY change should be ordered is made. This judgement is furthermore based on whether the parameters are above or below their respective defined thresholds, PRR being the most decisive. For evaluation purposes we required a threshold of 95% for the PRR, the RSSI threshold was set to -70 dBm, while SNR was required to be 5 dB. Thus, for a more conservative setting to be proposed a PRR above 95%, a RSSI larger than -70 dBm and a SNR larger or equal to 5 dB is required. RSSI and SNR can be directly read from the SX1272 register map, by reading the registers `RegPktRssiValue` and `RegPktSnrValue` respectively. For those values to be representative, the following formulas have to be applied:

$$SNR[dB] = \frac{\text{Snr[twos complement]}}{4} \tag{4.6}$$

where: Snr = value in `RegPktSnrValue`

$$\text{RSSI[dBm]} = \begin{cases} -139 + Rssi_{reg}, & SNR \geq 0 \\ -139 + Rssi_{reg} + SNR \cdot 0.25, & else \end{cases} \tag{4.7}$$

where: $Rssi_reg$ = value in `RegPktRssiValue`

Figure 4.12 illustrates a more detailed decision graph on the proposal of PHY settings at the gateway.

Basically, the master checks the PRR value for the current node: if this value is below a set threshold the master will propose more robust settings. If the PRR is above the threshold, RSSI and SNR will be observed, and based on their relation to the programmed threshold more robust or more conservative settings will be proposed. A table of the physical settings used within this thesis and the reasoning why they were chosen will be given at a later part of this section.

The master then either sends a general ping to the current slave, or a ping that includes the id of the newly proposed settings. If new settings were proposed, the master switches to the new settings and expects data from the slave. If there was no proposal, the master expects to receive the data on the current settings of the slave. If, after a defined period of time there is no response from the slave, a counter (`_miaCnt`, see Listing 4.1) will be incremented: if this counter reaches a threshold, the current node will be eliminated from the network, assuming it has left or has no more data to send. The expelled node will then have to rejoin the network after the reception of one of the beacons in the following cycles. If there was a proposal for new physical settings, the master will listen for data on those new settings. However, if there is no response the master will switch back to the previous settings and listen for data again. If data is received on the previous settings, those settings are saved as the current ones and the communication performance is evaluated again at a later point in time. If there is no answer from the slave, the counter is increased again and the slave is expected to send its data on the most reliable settings in the next cycle. In both cases, if the master receives a response from the slave, he sends back an ACK message, to confirm the reception. With this message the slot of the current slave is over and the master continues with the next node in the list, if there are any remaining. If no more nodes are in the network, the protothread hands back control to the main process, for sending the beacon and handling join requests, as the protothreads tasks are fulfilled.

Basic functionality of the slave process. Similar to the structure of a gateway node, a slave node has a main process (`defilmac_slave_process`) and two protothreads (`defilmac_scan` and `s_slot_operation`) that handle the functionality (see 4.6. Differently from the master's main process, the slave's main process is only responsible for starting the protothreads. The code snippet for the main slave process can be seen in Listing 4.2.

Listing 4.2: Main process of a node.

```
1  PROCESS_THREAD(defilmac_slave_process, ev, data)
2  {
3    static struct pt s_listen_pt;
4    static struct pt s_slot_operation_pt;
5
6    PROCESS_BEGIN();
7
8    printf("DeFiLMAC: Slave Main process started.\r\n");
9
10   PT_INIT(&s_listen_pt);
11   PT_INIT(&s_slot_operation_pt);
12
13   while (1)
14   {
15     while (!defilmac_is_joined)
16     {
```

```
17        /* Start scanning, attempt to join when receiving beacon message */
18        PROCESS_PT_SPAWN(&s_listen_pt, defilmac_scan(&s_listen_pt));
19      }
20      printf("Starting slot operation for slave node\r\n");
21      PROCESS_PT_SPAWN(&s_slot_operation_pt, defilmac_slave_slot_op(&
            s_slot_operation_pt));
22
23      printf("We need to re-synchronize.\r\n");
24    }
25    PROCESS_END(); // not reached
26  }
```

As illustrated, an important flag of this process is `defilmac_is_joined`: this flag is set to `true` if the node has successfully received a beacon and has joined a network. In the case that a node is removed from its network or has lost synchronization to its master, this flag is be set to `false`, forcing the node to start scanning for a beacon again and begin the join process anew.

The first protothread that is started as soon as initialization is done is `defilmac_scan`. The latter is responsible for putting the node into its active receiving mode, thus permanently listening on the most reliable settings (settings with id 0, see Table 4.3) for a beacon. For the reception of this initial beacon, the node has to stay permanently awake. If a beacon has been received, this protothread will send a join request message to the gateway node and await a response. Furthermore, it will save the timestamp at reception of that beacon, which is used to set the timer for the expected next beacon as well as for clock correction. If there is no response, the node will assume that it either did not get an available slot, or its message has been lost. It will try to join the network again upon reception of the next beacon. However, a node will not try indefinitely to join upon receiving a beacon: there is an exponential back-off in place to avoid that a single node blocks the master in the advertisement phase each cycle. Thus, a node will try to join the network immediately after the reception of the first two beacons: after that it will only try after each 2^nth received beacon, including an offset that is based on the unique node id. If, however, a node receives a slot in a network, it will receive an ACK message that includes the id for its slot from the master. To finish the join process, the node will send a message containing only the message identifier but no payload to the master, confirming the reception of the ACK and the slot id. A slave node will furthermore use the same list of nodes (implemented in `../mac/defilmac/nodelist.c`) as the master, however it will only add one entry to it, which is the master node. Upon receiving any message from the master, it will update the link metrics within that entry as well as keep track of the current PHY settings of the master. If, by chance, a node receives a message from another node, it will be immediately discarded, as the message's identifier is different for master and slave nodes. Thus, a node will ignore any message with an identifier that starts with a leading 1 e.g., 0b100 (see Table 4.3). After this, the node will enter its sleeping phase, as control is shifted to the `s_slot_operation` protothread. The amount of time a node sleeps before waking up to receive its ping, is determined by the assigned slot id. A node calculates its sleeping time as shown in Eq. 4.8:

$$T_{sleep} = T_{recv\_ack} - T_{recv\_beacon} + T_{beacon\_slot} - T_{safety} + assignedID \cdot T_{node} \tag{4.8}$$

where: T_{recv_ack} = Time at ACK reception
T_{recv_beacon} = Time at beacon reception
T_{beacon_slot} = Duration of the whole advertising phase
T_{node} = Duration of the slot per node
T_{safety} = Safety offset, early wake up

Synchronization. Note that T_{safety} is set to be a constant factor at initialization. However, each cycle every node, also the ones that have already joined a network, wake up to receive the beacon. This is done in order to ensure the correct delay until the start of each individual node slot and also to refine the safety offset value to be closer to the true offset to the clock of the master node, since clock drifting [54] [55] has to be taken into the equation, especially with the large delays with LoRa (e.g., delay between beacons of around 30 minutes or more). Thus, at each beacon, the slave will calculate a new offset to the masters clock and from that derive a more refined safety timing. For this purpose the timestamp at each beacon reception is saved, and compared to the timestamp at the reception of the previous beacon. Since the programmed time between two beacons is also known by the slave, it can calculate an estimation of the offset to the master node. Under the consideration of a safety timing (in our current DeFiL-MAC implementation set to roughly 500ms) a slave node can then alter the initial constant factor to be closer to the real offset to the master's clock. By refining this value over time we gain a good estimation of the true offset. Hence, it allows for a longer sleep time, resulting in a positive effect regarding overall efficiency. More information on this refinement and its results will be given in Chapter 5.

Slave: Slotted Operation Protothread. After its sleeping period, the node will wake up and immediately switch into receiving mode, expecting a ping. If, after a defined amount of time, the node does not receive a ping, it will assume that its timing is out of synchronization with the master and goes back to sleep until the next beacon, trying to re-synchronize. The slave also uses the `_miaCnt` counter to keep track of the missed pings: if this counter reaches a set threshold, or the next beacon is also missed it will leave the network and try to rejoin again, handing control back to the main process, `defilmac_slave_process`.

If a ping is successfully received within the reception window, depending on the type of ping received, the node will execute two different scenarios. In the case that the ping is a general ping with no additional information (message identifier `0b001 - MSG_PING`), the node will send the first packet within its `packets_to_be_sent` queue. If there are not yet packets within this list, the node will send a message with identifier `0b110 - MSG_SLAVE_KEEPALIVE` to avoid increasing the counter for missed responses. After transmission, it will listen for an ACK from the master, and, upon reception of said message, it will return to sleep mode, until the next beacon. If no ACK is received the node will attempt to send the same data packet again in the next cycle. If however the optional resending of packets is enabled (by setting the `RESEND_COUNT` define to > 1), a node will attempt to send its data again within its dedicated slot. It should however be noted that this option, while increasing reliability of transmission, is very costly in terms of available node slots within the network, because each slot has to account for the worst case, which is

that a node has to send its data for `RESEND_COUNT` amounts of time. Usually the minimum slot length is calculated by assuming the time-on-air for the maximum payload length with the most reliable (largest time-on-air) PHY settings, including some safety time, however with resending enabled, that worst case time estimation has to be multiplied with `RESEND_COUNT` increasing the reserved time per node substantially. Therefore, a developer can decide if he rather have less nodes in the network but receive data more reliable, or account for more nodes tolerating the possible data loss.

If the node receives a ping that includes the order to change its PHY settings (message identifier `0b010 - MSG_PING_PHY`, Table 4.3) it will change to the new settings before sending a data packet from its queue. Besides the data, a node always includes the link quality statistics within its data message, since those are used by the master to make a decision if a change of settings is feasible. To achieve a good estimation of the quality of the transmission, RSSI and SNR are again averaged using the EWMA, as described in Eq. 4.5. The general behaviour then is the same as for the general ping, as the node waits for an ACK. If the data is confirmed by the master, the node will save the new settings as being expected and go back to sleep until the next beacon. If no ACK is received, and resending is enabled, it will switch back to its previous settings before sending data again. This procedure has to be performed to avoid using different settings on master and slave node, which would cause the slave to lose synchronization with the master and thus being removed from the network. As soon as the data is then confirmed by the master, the slave accepts the current settings and enters the sleep cycle. If then the data still is not confirmed by the master, the node will assume some more issues, at which point it will roll-back its settings to the default settings with id 0 (most reliable) and go to sleep until the next beacon, since this timing is still known. While this roll-back does lose the progress of establishing more efficient settings, it does avoid the removal of the node from the network which can be more painful than losing the progress of the physical adaptation. Furthermore, the link quality measurements are kept in storage, which is beneficial when starting the adaptation process again since there is a plethora of link quality data available.

Selection of proposed parameters. For verification and evaluation purposes, we defined and introduced ten possible configurations of settings, where each of the relevant PHY parameters of LoRa is represented. A representation of the selected combinations for physical parameters can be seen in Table 4.3. Setting with id 0 represents the most reliable transmission parameters possible, which is also the default setting for the beacon, since it is desired that the probability for receiving a beacon is as high as possible and furthermore even nodes that are the furthest away should be able to receive a beacon and thus join the network. The remainder of settings and their individual composition of physical layer parameters can be freely chosen, however a setting with a higher identification is considered to be more energy conserving. Table 4.3 displays the selection of combinations we chose in order to achieve a good overview of the performance and behaviour of DeFiL-MAC.

Each consolidation of parameters is encoded into a two byte hexadecimal number, by shifting each binary representation of every relevant parameter to its designated position. The un-encoded parameters and each of their specific position for setting with id 0 can be seen in Table 4.3.

The reasoning behind why those settings were chosen that way can be seen when referring to Figures 4.13 and 4.14 respectively. Figure 4.13 illustrates how BW, SF and

Setting ID	TX Power	Spreading Factor	Coding Rate	Bandwidth	Encoding
0	14 dBm	12	4/8	125 kHz	0xA30E
1	14 dBm	12	4/8	250 kHz	0xA70E
2	14 dBm	12	4/8	500 kHz	0xAB0E
3	14 dBm	11	4/7	500 kHz	0x8A0E
4	14 dBm	10	4/7	500 kHz	0x6A0E
5	14 dBm	9	4/7	500 kHz	0x4A0E
6	13 dBm	8	4/7	500 kHz	0x2A0D
7	13 dBm	7	4/7	500 kHz	0x0A0D
8	11 dBm	7	4/6	500 kHz	0x090B
9	9 dBm	7	4/5	500 kHz	0x0809

Table 4.2: Combinations of physical parameters chosen for verification. Setting with id 0 is considered as default and is therefore used initially by each node.

SF	BW|CR	reserved	power
1010	00|11	0000	1110

Table 4.3: Un-encoded composition of physical parameters for setting with id 0.

CR scale in comparison to the estimated time-on-air, receiver sensibility and link budget respectively. Transmission power has no direct influence on each of the measurements in Figure 4.13 and is therefore not represented, however it will be examined independently in Figure 4.14 in relation to the estimated power cost when transmitting. The first row of plots in Figure 4.13 compares each of the parameters that have an impact on the estimated time-on-air. When examining those plots, it can be observed that spreading factor and bandwidth reduce the time-on-air by a substantial amount. While the coding rate does have an influence on the time-on-air, compared to the other two settings it is relatively small. Thus, in settings 0 to 2 (see Table 4.3) we first lower the used Bandwidth and then Spreading Factor in addition to coding rate, in order to achieve a noticeable drop in time-on-air. However, this reduction in time does come at a prize, as shown in the remaining two rows of Figure 4.13, increasing the bandwidth or the spreading factor does have an influence on the available receiver sensibility (lower is better) as well as the available link budget (higher is better) for the communication. Thus, the remaining settings 3 to 9 in Table 4.3, are a combination of altering the various settings, where the setting with id 9 is the one with the largest potential energy saved. As discussed earlier, the programmed transmission power does not have a direct influence on any of the metrics displayed in Figure 4.13, however it does directly influence the consumed power at transmission, this can be seen in Figure 4.14. This comparison reveals, that while keeping the programmed power as high as possible, the most potential saving can be achieved when using a power of 13 or 9.

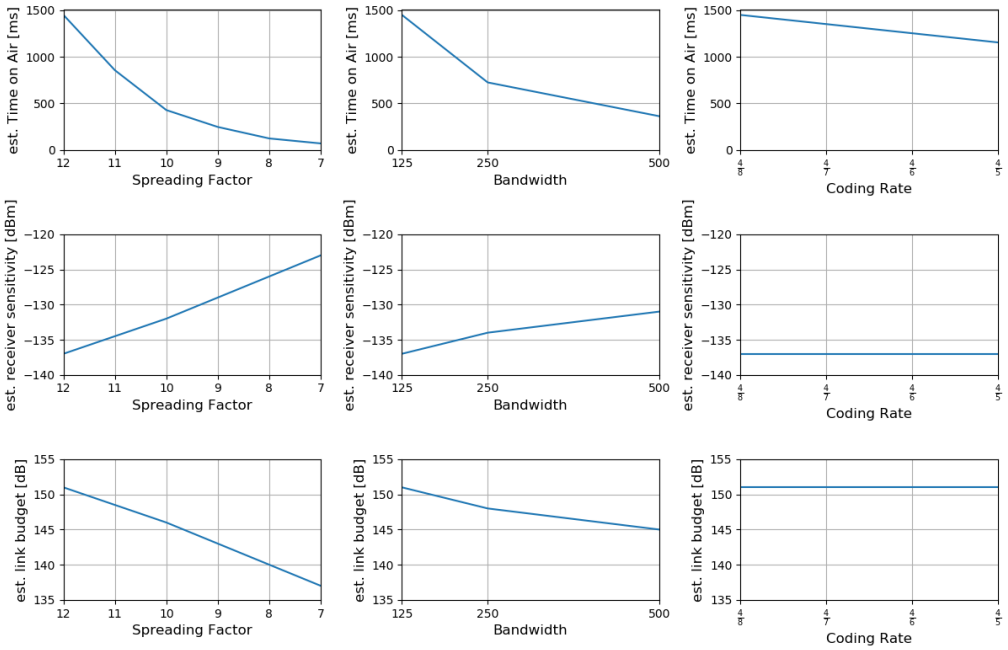

Figure 4.13: Scaling of each individual PHY parameter in regard to time-on-air, receiver sensibility and link budget.

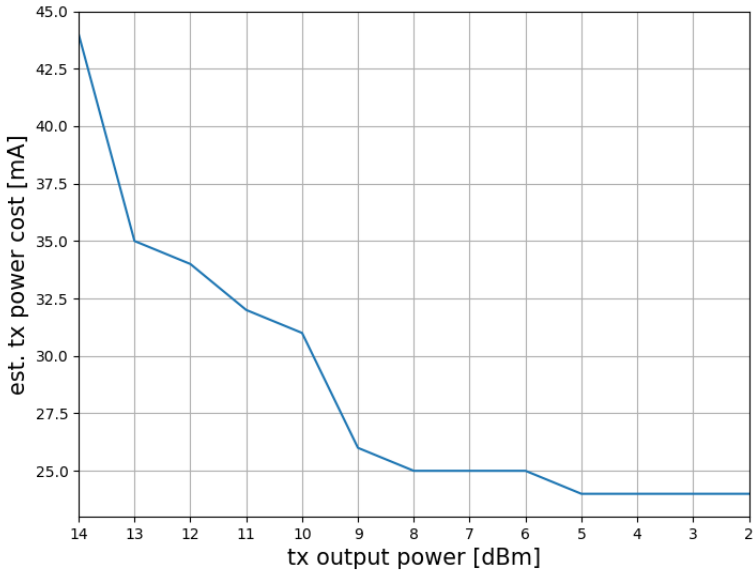

Figure 4.14: Transmission cost in relation to output power.

4.4 Limitations

The designed protocol presented in this thesis has the following limitations:

Worst-case calculations. At the initialization phase of DeFiL-MAC the current programmed parameters for the SuperSlot duration and the BeaconSlot duration are utilized to compute how much active time is available for the nodes in the network. Those parameters are fixed and have to be known beforehand by the gateway as well as all nodes, which limits the flexibility and the ability to adapt to network changes at runtime.

Furthermore, the amount of slaves that can be supported in a DeFiL-MAC network and the amount of time that is available per node for its transmissions is calculated under the assumption that the most reliable physical settings are used. Since those settings also use the most time-on-air this is a 'worst-case' estimation. Thus, the calculated timings are kept generous, which does not make the best use of the unused time. This becomes particularly clear if resending of data is enabled. In this case the worst-case amount of retries for data transmission has to be accounted for, which increases the time per node significantly and on the other hand reduces the number of nodes that can be supported by DeFiL-MAC.

Possible solutions for improving some of these limitations will be given in Section 6.1.

Limited scalability. While the implementation of DeFiL-MAC is fully scalable for a large network of nodes, for this thesis we only had three total hardware devices available. Thus, with only two devices available to fulfil the role of slaves within a network, an evaluation for true scalability could not be accomplished. Nevertheless, the scalability of DeFiL-MAC was verified: both nodes join a network then their unique network id is reprogrammed and the nodes are reset, thus they join the network again with a different id. While this does work, the 'old' nodes are removed from the network after they miss a certain amount of pings.

Chapter 5

Evaluation

This chapter presents the evaluation of DeFiL-MAC. Section 5.1 validates the functionality of DeFiL-MAC by presenting test results as well as power measurements. Furthermore, Section 5.2 provides an evaluation on the reliability as well as efficiency of DeFiL-MAC.

5.1 De-FiL-MAC: Validation

This section presents and validates the correct functionality of DeFiL-MAC by first describing the experimental setup in Section 5.1.1. Section 5.1.2 illustrates and validates the correct communication between the master and a slave while Section 5.1.3 displays the compliance to the duty-cycle regulations as well as evaluates measured and calculated time-on-air for data packets within DeFiL-MAC.

5.1.1 Experimental Setup

For this validation, three STMicroelectronics STM32L152 Nucleo-64 in combination with the SX1272 were set up in a laboratory environment roughly 2 meters apart from each other. All of the devices used the default SMA antennas that the SX1272 is supplied with. One device was programmed to function as gateway, while the other two would take on the role of a slave node. To achieve precise measurements, one of the nodes was connected to a High Voltage Power Monitor by Monsoon Solutions Inc. (Type: AAA10, [56]). This, in combination with the corresponding software (PowerTool Version 5.0.0.25, by Monsoon Solutions Inc.), allowed us to measure the power consumption as well as observe the timings when the device was sending a packet or going into sleep mode. The master and the other nodes were directly connected to a PC, where with the help of a COM-port communication tool, the transmissions between the network participants could be observed and validated. Thus, all nodes had a wired power connection on the one hand for reproducibility and on the other to avoid failures due to depletion of batteries.

5.1.2 Master and Slave operation

This section shows the validation of the correct communication process between a master and a slave node. For this purpose, a demo application was set up, where the node uses a duty-cycle of 20% to avoid cycle times of around 20 minutes with a 1% duty-cycle, which

is furthermore caused by the high time-on-air when using the most reliable PHY settings. The SuperSlot duration is set to 120 seconds, which results in an active time of 24 seconds. Furthermore, the test application of a slave node is programmed to try to send a dummy data packet with 20 bytes of data every 15 seconds, however, as the node only is allowed to send data when within its slot, those packets are added to the packet queue and sent at the appropriate timing.

The behaviour of a slave node can be seen in Figure 5.1, which shows the joining process as well as the transmission of data between the slave and its master. In the first third of the figure, the slave node is initialized before it actively listens for the reception of a beacon. As soon as the beacon is received, the slave changes into transmitting mode and sends a join request to the master. This request is almost immediately confirmed by the master, since, as illustrated in Figure 5.1, after the join request, the node sends another very short message. This message is the confirmation of the received ACK including a slot id that was sent by the master. After this transmission, the node enters sleeping mode, which is illustrated by the lower power consumption than in the first third of the figure. After a short sleeping period, the node wakes up to receive its ping, which is the signal for the slave to send its data. Right after receiving the ACK from the master, that its data was received, the node will go back into sleep mode until the next beacon is due, which can be seen in the last third of Figure 5.1, where data is exchanged again, before the node goes back to sleep.

When comparing the measured power consumption in active listening and sleeping mode to the data sheet of the SX1272 [45], we can further validate the correct behaviour of the DeFiL-MAC slave: the supply current draw of the SX1272 when in receive mode is either 9.7 mA (for a used bandwidth of 125 kHz), 10.5 mA (for a used bandwidth of 250 kHz) or 12 mA (for a used bandwidth of 500 kHz), if multiplied with the output voltage of the Monsoon Power Monitor of 4.2V this results in an power consumption for the active listening of 40.74 mW, 44.1 mW and 50.4 mW, respectively ($Power = Voltage \cdot Current$). If we add this consumption to the base consumption of the Nucleo (236.86 mW, see Figure 5.5), the result is 280.96 mW, which conforms with the measured averaged consumption (surging in power consumption is due to LED on ST-LINK, thus average is used) in Figure 5.1, for a used bandwidth of 250 kHz and while in active receiving mode. If the radio goes to sleep mode we can observe a drop in power consumption, which is to be expected and can be verified with the measurements shown in Figure 5.5, as well as the data sheet of the SX1272 [45].

Both, the measured power consumption and the COM log verify the correct functionality of DeFiL-MAC communication between the master and slave.

A noticeable aspect of this power measurement is that power consumption is fairly high as well as constantly surging, even with the node in its sleep mode. However, both of those observations can be traced back to the integrated programmer/debugger combination (ST-LINK [49]) on the Nucleo. On the one hand, it is responsible for the increased energy demand and, on the other hand, it is equipped with a tricoloured LED that provides visual feedback of the ST-LINK communication. Since this LED is constantly blinking while the Nucleo is connected to the Power Monitor, it causes the power consumption to surge.

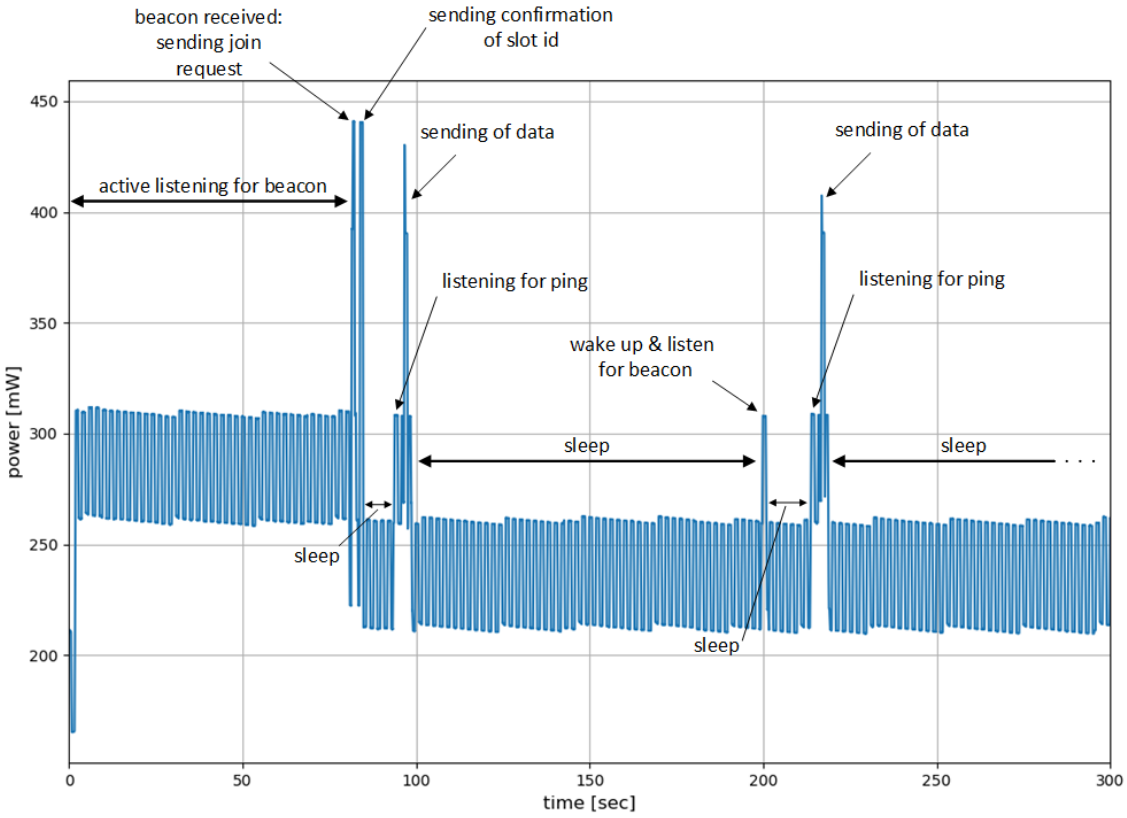

Figure 5.1: Functionality of a node in DeFiL-MAC.

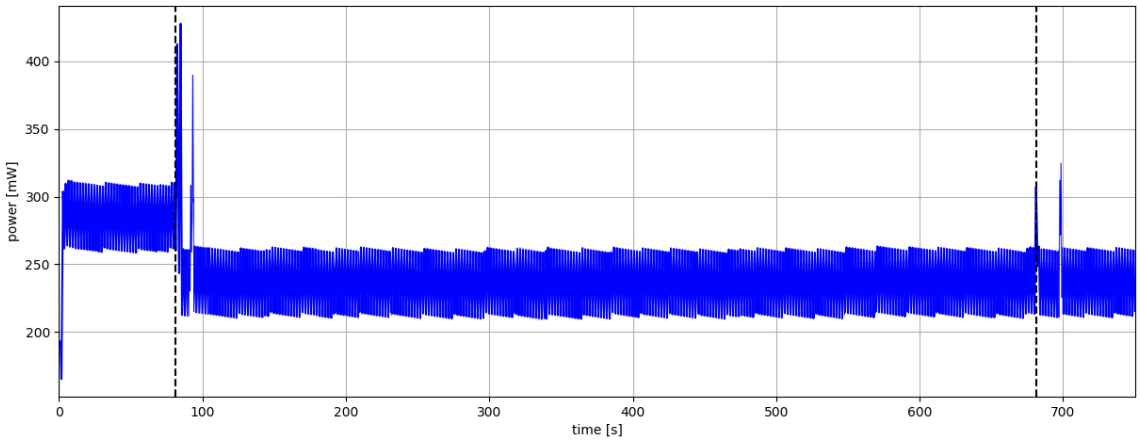

Figure 5.2: 1% duty-cycle of a slave communication, dashed lines mark the reception of a beacon.

5.1.3 Duty-cycle Regulations and Packet Time-on-Air

Duty-cycle. In a second evaluation setup, we validated the compliance to the duty-cycle regulations, therefore we altered the test application to use the required duty-cycle of 1%, with a programmed SuperSlot of 10 minutes. For this measurement, only two devices were used, one master and one slave node, due to the total available active time across all devices of only 6 seconds. Figure 5.2 illustrates the joining process and the exchange of two data packets between slave and master, measured at the slave node. As stated previously, the surging of the power consumption in the sleeping phases of the node is due to the blinking LED on the ST-LINK of the Nucleo.

The two dashed lines in Figure 5.2 mark the reception of a beacon, and thus the start of the total active time. Every transmission, from beacon reception to the sending of data and the reception of the ACK for the data, has to happen within this active time, which in this case is roughly 6 seconds. After that the slave sleeps and is inactive until the second beacon, its reception is marked by the second dashed vertical line in Figure 5.2. With a sleeping time of roughly 590 seconds and the active time of about 6 seconds, for a programmed SuperSlot duration of 600 seconds. Although there is a small discrepancy between the expected (594 second sleep / 6 second active) and the measured durations we can validate the compliance to the 1% duty-cycle regulation for this communication channel. The reason for this divergence is due to the extended length of this measurement and the error propagation of the time measurement within the Monsoon Power Monitor [56]. When testing with a larger duty cycle of e.g., 20% (as in Section 5.1.2) the timings were much more precise, as the measurement was not as prolonged.

Evaluating packet time-on-air. After evaluating the correct basic functionality of DeFiL-MAC, we validated that the calculated time-on-air estimations for our data packets were correct. For that purpose, we used a test application to send a data packet with 20 bytes of dummy data, which was repeated for each of the settings that were proposed in Table 4.3. Each measurement consists of the averaged time-on-air values of 30 sent

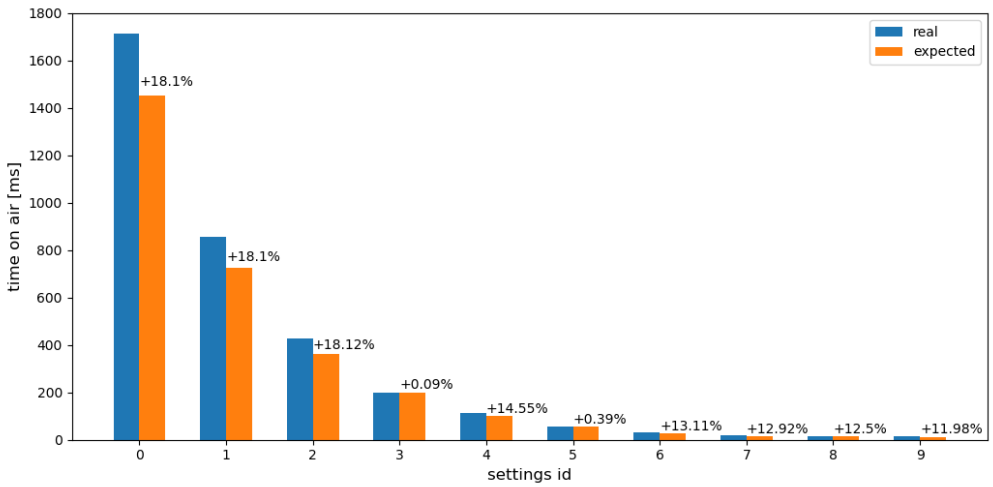

Figure 5.3: Comparison of calculated vs. measured time-on-air for a 20 bytes data packet with

messages. Figure 5.3 illustrates the comparison between measured values, which were acquired with the use of the Monsoon Power Monitor [56] and the calculated values. The calculations were done with Eq. 4.1 and Eq. 4.3 given in the data sheet of the SX1272 [45].

From Figure 5.3 it is noticeable that the difference between measured and calculated time-on-air increases with the total time-on-air of the packet, such, for example the difference for the most reliable settings is roughly 18% at an increase of 160 ms. However, for some fast settings e.g., setting with id 5, this discrepancy is below 0.4% which corresponds to roughly 3 ms. Nevertheless, as the calculated values only represent an estimation, a discrepancy between measured and calculated values is to be expected. Thus, for more reliable settings a safety timing of roughly 20% of the estimated time-on-air needs to be accounted for when estimating the amount of time a node needs for packet transmission while for faster settings a safety timing of around 15% of the estimated time-on-air is sufficient.

5.1.4 Synchronization

As already stated previously, DeFiL-MAC is based on TDMA, which requires fairly precise timings to avoid wasting energy due to long, unnecessary listening periods instead of sleeping phases. For this evaluation, two nodes were set up with a test program that would execute the default behaviour of DeFiL-MAC: join a network, listen for each periodical beacon, send data within the assigned slot, and then sleep until the next beacon. The delay between two beacons was specified to be 2 minutes. As already discussed in Section 4.2, a slave uses the periodical beacons to adjust the time at which it will wake up to receive the next beacon as well as the ping from the master. For the purpose of adjusting this wake up time, the slave compares the time stamp of the previously received beacon with the current one, from this difference, the new sleeping time until the next beacon is estimated. Starting with a programmed wake up time of 2 seconds earlier to the expected reception of

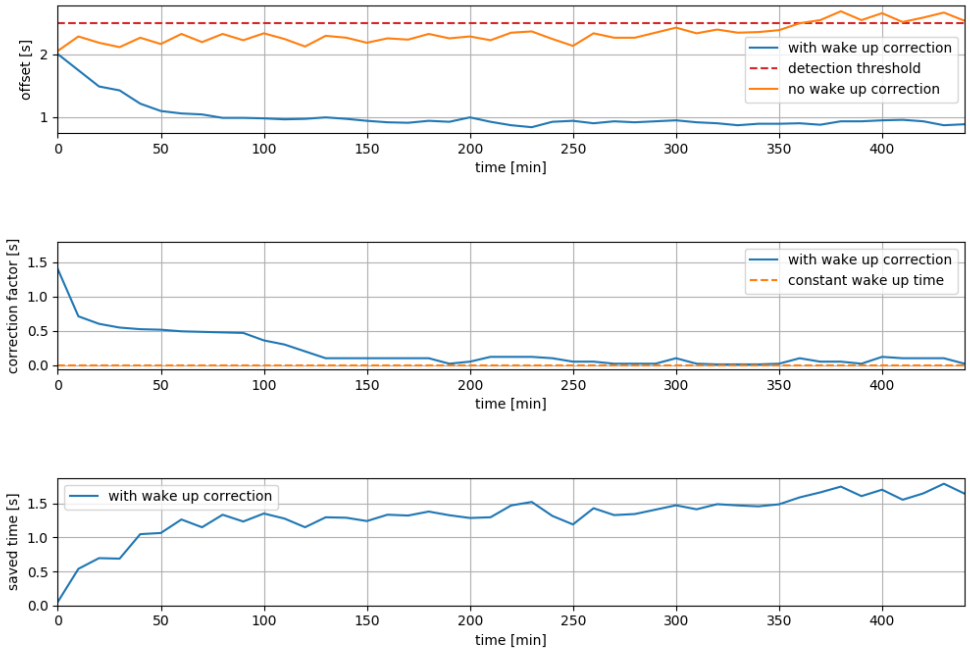

Figure 5.4: Wake up (clock drift) offset correction.

a beacon. This value will gradually be improved, and thus the sleeping time will increase. However, in our measurements we determined that a minimum time of 400 ms before the actual reception of a beacon is required for constant and reliable reception. Thus, in the first graph of Figure 5.4, the offset converges roughly to a value of 400 ms. This time is needed in order to guarantee that the node has woken up correctly and switched into receiving mode, as well as give the radio process enough time to prepare the reception of a packet. Since this behaviour can be observed the better the longer DeFiL-MAC is running, this measurement was done over a time period of roughly 7 hours. In this setup only one of the two nodes uses this wake-up adjustment: the other node uses a fixed programmed offset of 2 seconds before every beacon.

An illustration of this measurement can be seen in Figure 5.4. The topmost graph shows the delay between the wake up of a node and the actual reception time of a beacon. The node without the correction is illustrated by the orange line, while the node that is correcting its wake up is displayed by a blue line. The dashed line depicts the detection threshold of the beacon, if a node's offset between waking up and beacon reception is above that threshold the beacon can not be detected any more.

As illustrated in the first graph of Figure 5.4, both nodes start with a delay between wake up and actual beacon reception of roughly 2 seconds, the node that applies the correction (blue line) is able to extend its sleeping cycle noticeably. Not only is this having a positive effect on the energy consumption, but this correction is furthermore used to account for possible clock differences between the master and its nodes, by keeping a constantly improving synchronization of the beacon and wakeup timings. The effect of

this clock drifting can be observed in the last third of the first graph: the node that uses constant wake up timings loses synchronization with its master, since the delay between wake up and beacon reception became longer than the listening phase of the slave.

The second graph within Figure 5.4 illustrates how the correction factor is altered over time, while this factor is fairly large at the beginning of the measurement, it improves and adapts over time, stabilizing at around 150 ms.

Finally, the third graph of Figure 5.4 illustrates the time that is effectively saved by choosing to adapt the wake up time versus using a constant time to wake up before the reception of a beacon. Considering that the Nucleo with the SX1272 has a combined energy consumption of roughly 234.18 mJ/s in sleep mode, while in its active listening state it has an energy consumption of roughly 283.4 mJ/s, this saved time translates directly into saved power. Considering our measurement, the average time a node can stay longer in sleep mode saving is roughly 1.3 seconds, which translates to a power consumption of 502.814 mJ with sleep time correction and 566.8 mJ without correction, prior to each beacon, in addition to the reliability gained by not losing the synchronization to the master, after long run times of the network.

5.2 De-FiL-MAC: Reliability and Efficiency

This section evaluates the gained reliability and efficiency by using DeFiL-MAC. In Section 5.2.2 we present the measured energy consumption of the devices with DeFiL-MAC while using the different PHY settings presented in Table 4.3. Finally, Section 5.2.3 evaluates the benefits of the switching of PHY parameters at run time.

5.2.1 Experimental Setup

For the evaluation of the energy consumption of each combination of settings from 0 to 9 (see Table 4.3), a slave node was programmed with a test application. In this application the slave is programmed to use one of the 10 proposed combinations of PHY settings, and then send a packet of 20 bytes of data every 5 seconds for a total test duration of 5 minutes. This resulted in 60 total packets sent for each of the different settings. Furthermore, the slave node was connected to the Monsoon Power Monitor [56], to measure its power consumption.

The evaluation in Section 5.2.3 required a slightly different experimental set up. In order to be able to simulate different network conditions and link qualities, a programmable attenuator was added to one of the slave nodes. The programmable attenuator was a RCDAT-8000-90 by Mini Circuits [57], which can be connected to the SMA antenna of the slave. If the attenuator is connected via USB to a PC, a user can freely program the attenuation between 0dB and 90dB with a resolution of 0.25 dB, as well as program a sequence of attenuations that should be applied. Furthermore, the master and another slave node were connected to the PC, which allowed for data and transmission log collection. The Contiki application for both slave nodes was programmed to join the network of a master and send data packets with 20 bytes of dummy data. The master node's application is programmed to evaluate the link quality of a slave connection every 5 packets received by the slave (`MIN_PACKETS_TO_PHYCHANGE` = 5), with a threshold for RSSI of -70 and a PRR of 95%. Usually, the SNR would also be taken into consideration for this decision,

however the programmable attenuator only allowed a reproducible influence on the RSSI values, thus SNR was left out of the equation.

5.2.2 Energy consumption of individual PHY settings

The results of the measured energy consumption of each of the proposed settings can be seen in Figure 5.5. In addition to the measurement of each of the settings, we also measured the power consumption of the Nucleo platform on its own, without the SX1272 connected. This can be seen in the top graph of Figure 5.5, in comparison to the power consumption of the Nucleo with the SX1272, the combination of both in sleep/idle state, as well as the power consumption of the device in its active receiving mode for PHY setting 0. As shown, the power consumption of the SX1272 in idle/sleep mode is relatively small at roughly 1 mW, as soon as the radio is switched into listening mode, the power consumption is increased to roughly 46.5 mW, which conforms with the expected consumption of 44.1 mW specified within the data sheet [45]. The small discrepancy in power consumption is to be expected due to the Nucleo and the ST-LINK. The relatively high base power consumption of the Nucleo is a result of the integrated ST-LINK programmer/debugger [49].

The second graph in Figure 5.5 illustrates the measured power consumption of the device in transmission mode, while using each of the proposed settings in Table 4.3. Each of the measurements consists of 60 sent packets over the duration of 5 minutes. The illustrated values represent the average consumed power during that time period. The data sheet [45] specifies expected values for transmission current draw to be 28 mA for a programmed power of 13 dBm, which results in a power consumption of 117.6 mW for the SX1272. Since our measurement in Figure 5.5 is done over a duration of 5 minutes, and the idle time between packets is also a part of the measurement, those values can not be directly compared to the expected values in the data sheet. Thus, this measurement has to be seen as a comparison between the individual settings. Furthermore, it is clearly visible, that more efficient settings require more energy. This increased energy demand is to be seen in combination with the longer time-on-air, that was measured and illustrated in Figure 5.3 , which points out the increased overall efficiency for settings with a higher id. However, when measuring the peak power consumption in transmission mode for a programmed power of 14 dBm, as illustrated in Figure 5.1, and subtracting the power consumption of the Nucleo, the measured power consumption of the SX1272 (at roughly 122,51 mW) is within the expected range of the values specified in the data sheet.

5.2.3 Adaptation of PHY parameters at runtime

For the evaluation of DeFiL-MAC's performance we established the experimental setup as described in Section 5.2.1. In total, two slightly different experiments were conducted.

In the first experiment, the programmable attenuator was set up to execute four different scenarios, which are illustrated by the four dashed vertical lines in each sub plot of Figure 5.6. Furthermore the figure is split into five individual graphs for their measurements respectively. Each tick on the x-axis represents a decision of the master node, that is made after `MIN_PACKETS_TO_PHYCHANGE` received packets, if a switch for the node in question is suitable. The first graph illustrates the measured RSSI values of the slave node with the

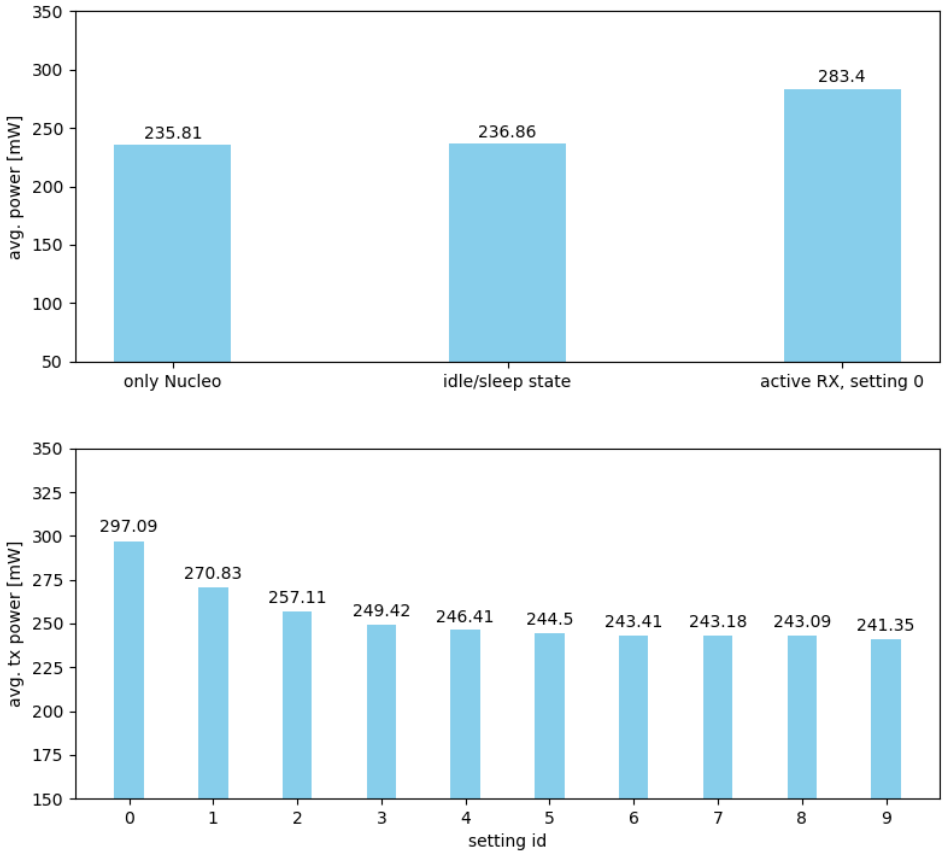

Figure 5.5: Measured energy consumption of the different physical settings applied in DeFiL-MAC, as well as base energy consumption of the Nucleo and the SX1272.

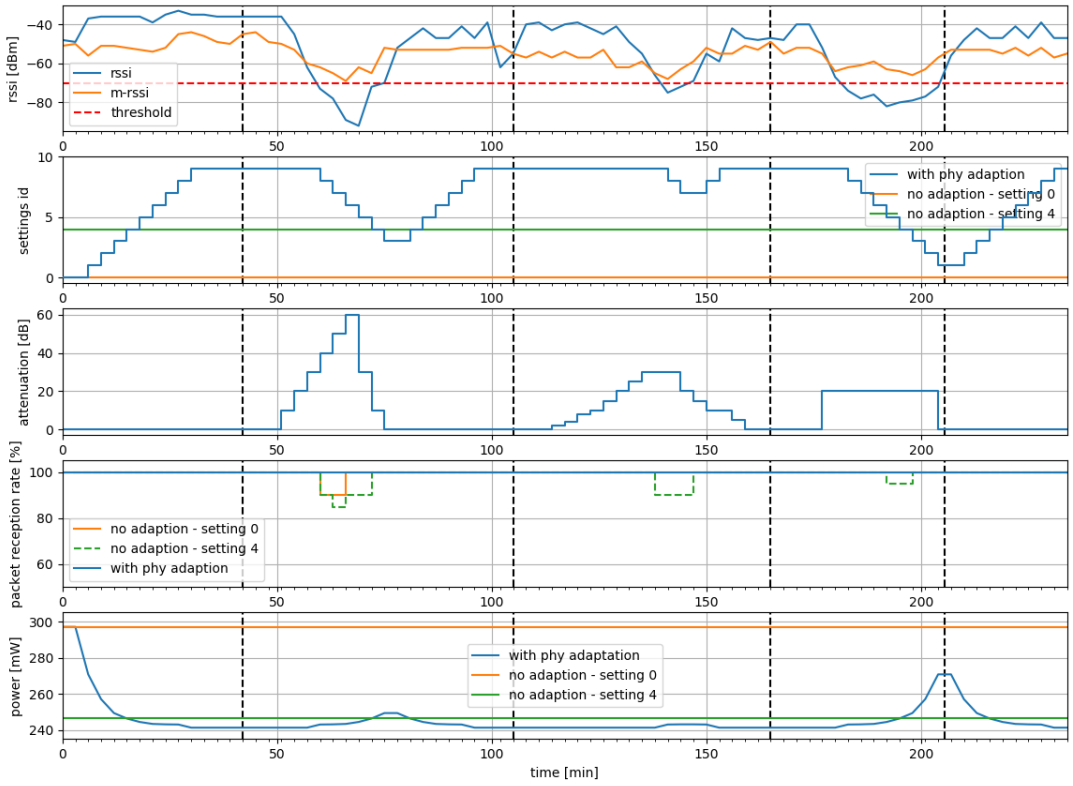

Figure 5.6: Measurement to evaluate the behaviour of DeFiL-MAC: a programmable attenuator is connected to a node to simulate changing link qualities and observe the run time adaptation of the physical settings.

programmable attenuator measured by the master. Those RSSI values are split into two separate measurements: `rssi` depicts the link quality from the slave to the master, while `m-rssi` is the quality of the link from master to slave. The dashed horizontal line in red illustrates the programmed RSSI threshold that is used in combination with the PRR to decide if a change of PHY settings is desirable.

The second graph of Figure 5.6 is an illustration of the settings id used by the slaves. The lines in orange and green depict the settings used by the node that is not applying the active PHY settings switch. In the third graph the behaviour of the programmable attenuator is shown, while the fourth graph illustrates the PRR of slave nodes that do not make use of the PHY setting adaptation in comparison to the one slave that actively changes its settings. Furthermore, the fifth graph is the averaged energy consumption of each of the nodes that utilize a constant setting, as well as the node with adaptive settings.

As already stated, the attenuator was programmed to go through four different scenarios. In the first phase (separated by dashed lines) of Figure 5.6 we can see that the attenuator was programmed to have no influence on the link quality at all, thus being operated at 0 dB. Since all the decisive parameters are within their respective thresholds the master node of the DeFiL-MAC network orders the node to gradually alter its currently used parameters up to the most efficient setting possible.

After reaching setting with id 9, in the second phase, the attenuation is gradually increased up to 60 dB. As illustrated, as soon as the RSSI measurements are below the programmed threshold, the master proposes different settings to the slave in order to increase robustness at the cost of a higher power consumption. If we compare those results to a node that uses a single setting (e.g., setting 0 or setting 4 as illustrated in Figure 5.6), the advantages of the active adaptation of PHY settings becomes evident: if the most reliable setting is chosen, the energy consumption is constantly high, even if the link quality does not require such robust settings. However, if the link quality suddenly decreases by a substantial amount (e.g., moving a node behind an obstruction), the loss of packets can not be fully prevented. If the node constantly uses a less robust setting, as depicted by the green lines in Figure 5.6, the difference in energy consumption to the node applying the PHY switch is not as bad. However, as illustrated in the fourth graph, more packets are lost in the transmission process. After reaching its peak attenuation at 60 dB, the attenuator is gradually lowered to 0 dB again. As the RSSI improves, the master proposes more efficient settings to the slave again.

In the third phase, the attenuation is progressively raised to a mid-range interference, and, again, as the RSSI drops below the programmed threshold, a PHY switch is proposed from the master. As this attenuation is not as large as in the second phase, there are less packets lost for the node with constant setting 4 and no packets lost for the node with setting 0.

Within the fourth phase, the attenuation is instantaneously raised to 20 dB and then kept at this level. This causes a similar behaviour to the one shown in the previous phases. As the RSSI dips below the threshold, the node that is ordered to change its PHY settings prevents the loss of packets by changing to more robust settings. Lastly, the attenuation is reduced to 0 dB, which causes the node to gradually switch to more efficient settings.

For the second experiment, the programmable attenuator was set to a constant attenuation of 40 dB, after an initialization phase. The measured results for this evaluation set up can be seen in Figure 5.7. The node with the attenuator is represented by the orange line in each graph respectively. As illustrated, after the phase where the attenuation was set to 0 dB, both nodes follow the same behaviour.

As the programmed threshold of -60 dBm is not reached, both nodes adapt their settings to be more energy efficient. Again, each of the ticks on the x-axis represents a decision by the master, after `MIN_PACKETS_TO_PHYCHANGE` were received by the respective node. After a certain time, the node with no attenuation has reached the most efficient setting possible. Since its position is fixed, the link quality is fairly constant, thus the node can operate with the proposed, highly efficient settings. The node that is connected to the attenuator, however, experiences a drop in its link quality to the master, which causes the RSSI to drop below the programmed threshold. At this point, the master proposes more reliable settings, in order to prevent packet loss or the loss of the node in general. This switch to more reliable and robust settings causes the link quality and RSSI measurement to be above the threshold, which instructs the master to propose a more efficient settings, after the next `MIN_PACKETS_TO_PHYCHANGE` number of packets were received. However, since the attenuation is still set to 40 dB, this switch to a less robust parameter causes the packet to not be received by the master. As this decreases the PRR, the master proposes a more reliable setting, causing the RSSI to climb above the threshold

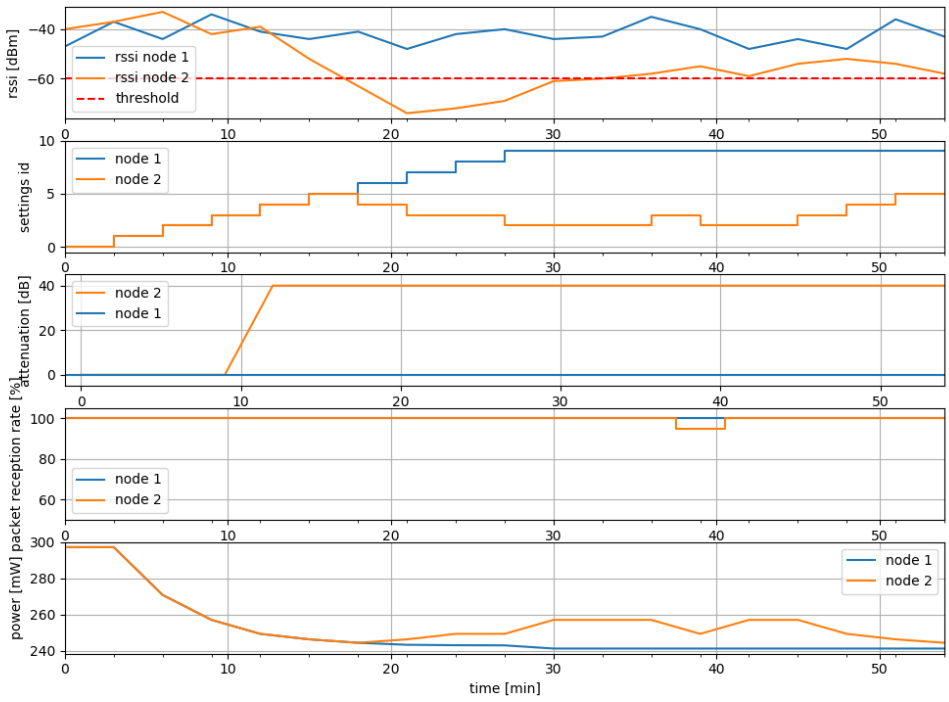

Figure 5.7: Behaviour of DeFiL-MAC's PHY setting adaptation when constant attenuation is applied: the aim to find the most efficient settings possible can cause loss of packets.

again. Since in this measurement the retransmission of packets is not enabled, the packet has to be retransmitted in the next cycle. If retransmission is enabled, this packet would be retransmitted by the slave after a switch to the more reliable settings. Thus, the cost for this attempt to change to a more efficient setting is an additional transmission with a more reliable setting.

As illustrated in Figures 5.6 and 5.7, the loss of packets can actively be prevented by observing the link metrics of a connection and the well-timed alteration of physical settings. Naturally, this is a trade off between power consumption and reliability, as it is the nature of an adaptive MAC protocol such as DeFiL-MAC to investigate more efficient settings over time. While the energy consumption can be lower when constantly using a less reliable setting, the loss of packets cannot be prevented.

Chapter 6

Conclusions and Future Work

In this thesis we present a port of the STM32L152 Nucleo-64 in combination with the SX1272 LoRa transmitter into the open-source Contiki operating system, and DeFiL-MAC, a duty-cycled and efficient MAC protocol for LoRa within Contiki, that provides a mechanism for the active alteration of LoRa's physical settings at runtime.

Port of a popular LoRa platform to the Contiki OS. The porting approach introduced in Chapter 3 integrates the STM32L152 Nucleo-64 from STMicroelectronics in combination with the Semtech SX1272 LoRa transmitter to the open-source Contiki operating system. Since this port is kept in compliance with the required interfaces within Contiki a developer is able to utilize the default instructions to communicate to the newly ported platform.

DeFiL-MAC: a duty-cycled and efficient MAC protocol for LoRa. In Chapter 4 we present DeFiL-MAC: a MAC protocol for the wireless transmission technology LoRa implemented in the Contiki operating system. Since for LoRa a master-slave approach is commonly used, and slave-to-slave communication is not intended, DeFiL-MAC is based on a TDMA approach. Moreover, by introducing duty-cycling, DeFiL-MAC provides efficiency, reliability as well as scalability while being compliant to the duty-cycle regulations for the license-free, sub-gigahertz, 868 MHz radio frequency band. Furthermore, to fulfil these key-features, and to capitalize on the ability to fine-tune the communication performance and range capabilities of LoRa by altering its physical parameters, DeFiL-MAC implements a mechanism to adapt the physical settings based on link quality. This adaptation allows DeFiL-MAC to avoid packet loss, while improving efficiency in terms of time-on-air and energy consumption by proposing more robust or more economical settings.

Evaluation. The evaluation in Chapter 5 showed that DeFiL-MAC can improve the efficiency of a network and also boost its reliability by applying the proposed adaptation of LoRa's physical layer settings.

6.1 Future Work

Future work revolves mostly around the improvement of DeFiL-MAC's efficiency, as well as addressing and providing possible solutions to the limitations specified in Section 4.4.

Revising limitations. One of the drawbacks that were briefly addressed in Section 4.4 was that at the initialization phase the SuperSlot and other durations have to be fixed and known by the master and all nodes in the network. A possible solution to this would be to use the master and its periodical beacons to advertise the current network statistics to its members as well as new nodes that intend to join the network. This would allow for more flexible timings, as the master can dynamically react to nodes joining and leaving the network.

Additionally, if resending is enabled DeFiL-MAC expects the worst-case amount of retries for data transmission, which in most cases is not needed, thus allocating time that is not used and therefore significantly decreasing the number of supported nodes. A solution to this drawback would be to: (i) disable retransmissions, which would reduce reliability, but could be suitable for networks where reliability is not as important as amount of nodes, or (ii) if a transmission was not successful, the master could propose some time (e.g., an unused slot) where the current slave can retry to send its data, if there are no free slots available, resending is not possible.

A further proposal to improve the utilization of available time is that nodes that are proposed faster settings could inform the master that they require less time for their data to be transmitted, freeing up some time within their slot for possibly other nodes. Moreover, this information could be used by a node itself: a node knows how long its slot is and how much data it wants to transmit, if faster settings are proposed the node might be able to transmit data twice within its slot, increasing its throughput.

However, as this would significantly increase the complexity as well as number of messages that need to be exchanged between gateway and nodes, the implementation was not considered for this thesis. Furthermore this behaviour would elicit new issues: nodes would need to request for a re-assignment of their slot time if they need to change to a slower, more reliable setting due to decreasing link quality.

Algorithm for proposing PHY settings. In the current implementation state of DeFiL-MAC, the decision if a change of PHY settings is suitable for a certain node is based on if a relevant link quality value is above or below a programmed threshold. In their work, Bor et. al. [33] present the implementation of an algorithm based approach, that allows for the discovery of the current optimal settings for a LoRa node. Thus, the implementation of a similar algorithm into DeFiL-MAC would possibly improve and optimize the adaptive PHY parameters selection.

Optimize listening time for initial beacon. In its current implementation, if a node has the intention to join a DeFiL-MAC network, it has to permanently listen for the initial beacon. As this listening mode consumes a fair amount of energy, this is not optimal and ideally the amount of time in this state should be kept as short as possible. Thus, a possible solution would be to modify the gateway's implementation such that it can use the currently open slave slots in its network to broadcast 'intermediate' beacons, which include

the current delay until the next beacon. If a node receives this intermediate beacon, it can use the provided information to go into sleep mode until the 'real' beacon is sent and then issue a request to join the network.

Bibliography

[1] LoRa, a course by Thomas Telkamp. https://www.thethingsnetwork.org. Accessed: 2016.

[2] Knight, Matthew and Seeber, Balint. Decoding LoRa: Realizing a Modern LPWAN with SDR. In *Proceedings of the GNU Radio Conference*, volume 1, 2016.

[3] Alliance, LoRa. What is LoRaWAN. *https://lora-alliance.org/resource-hub/what-lorawantm*, 2015.

[4] Michael Buettner, Gary V Yee, Eric Anderson, and Richard Han. X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 307–320. ACM, 2006.

[5] Adam Dunkels. The ContikiMAC Radio Duty Cycling protocol. 2011.

[6] STMicroelectronics. STM32 Nucleo-64 Board, User Manual. *Retrieved from: http://bit.ly/2iAbnxf*, 2014.

[7] ARM, https://os.mbed.com/components/SX1272MB2xAS/. MBED Operating System components.

[8] Cattani, Marco and Boano, Carlo Alberto and Römer, Kay. An Experimental Evaluation of the Reliability of LoRa Long-Range Low-Power Wireless Communication. *Journal of Sensor and Actuator Networks (JSAN)*, 6(2):7, 2017.

[9] Dave Evans. The Internet of Things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1(2011):1–11, 2011.

[10] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business & information systems engineering*, 6(4):239–242, 2014.

[11] LoRa. https://www.lora-alliance.org. Accessed: 2015-11-07.

[12] SigFox. http://www.sigfox.com. Accessed: 2015-11-07.

[13] Weightless open standard. http://www.weightless.org. Accessed: 2015-11-07.

[14] Khutsoane, Oratile and Isong, Bassey and Abu-Mahfouz, Adnan M. IoT devices and applications based on LoRa/LoRaWAN. In *Industrial Electronics Society, IECON 2017-43rd Annual Conference of the IEEE*, pages 6107–6112. IEEE, 2017.

[15] Dirk Ahlers, Patric Arthur Driscoll, Frank Alexander Kraemer, Fredrik Valde Anthonisen, and John Krogstie. A measurement-driven approach to understand urban greenhouse gas emissions in Nordic cities. NIK, 2016.

[16] Muhammad S Osman, S Yoyo, Philip R Page, and Adnan MI Abu-Mahfouz. Real-time dynamic hydraulic model for water distribution networks: steady state modelling. *Proceedings of the Sixth IASTED International Conference*, 2016.

[17] Alliance, LoRa. A technical overview of LoRa and LoRaWAN. *White Paper*, 2015.

[18] Dunkels, Adam and Schmidt, Oliver and Finne, Niclas and Eriksson, Joakim and Österlind, Fredrik and Tsiftes, Nicolas and Durvy, Mathilde. The Contiki OS: The operating system for the Internet of Things. *http://www.contikios.org*, 2011.

[19] Tim Winter, Pascal Thubert, Anders Brandt, J Hui, Richard Kelsey, Philip Levis, Kris Pister, Rene Struik, J Philippe Vasseur, and R Alexander. RPL: IPv6 routing protocol for low-power and lossy networks. Technical report, 2012.

[20] EN ETSI. 300 220-1. *European Standard (Telecommunications Series)*, 1, 2000.

[21] *IEEE Computer Society, (August 31, 2007). IEEE Standard 802.15.4a-2007. New York, NY: IEEE.*

[22] Berni, A and Gregg, WO. On the utility of chirp modulation for digital signaling. *IEEE Transactions on Communications*, 21(6):748–751, 1973.

[23] De Dominicis, Chiara Maria and Pivato, Paolo and Ferrari, Paolo and Macii, David and Sisinni, Emiliano and Flammini, Alessandra. Timestamping of IEEE 802.15. 4a CSS signals for wireless ranging and time synchronization. *IEEE Transactions on Instrumentation and Measurement*, 62(8):2286–2296, 2013.

[24] David Tse and Pramod Viswanath. *Fundamentals of wireless communication*. Cambridge University press, 2005.

[25] Syed Kamrul Islam and Mohammad Rafiqul Haider. *Sensors and low power signal processing*. Springer Science & Business Media, 2009.

[26] Warren L Stutzman and Gary A Thiele. Gain calculations for reflector antennas. *Antenna Theory and Design*, pages 433–436, 1981.

[27] Rashmi Sharan Sinha, Yiqiao Wei, and Seung-Hoon Hwang. A survey on LPWA technology: LoRa and NB-IoT. *Ict Express*, 3(1):14–21, 2017.

[28] Alliance, LoRa. LoRaWAN Specification. *LoRa Alliance*, 2011.

[29] Bor, Martin and Vidler, John Edward and Roedig, Utz. LoRa for the Internet of Things. 2016.

[30] Alliance, LoRa. A technical overview of LoRa and LoRaWAN. *White paper*, 2015.

[31] Dragino. Dragino LG02, Dual Channel LoRa Gateway. *Retrieved from: https://www.antratek.de*, 2019.

[32] Adafruit. Adafruit Feather M0 with RFM95 LoRa Radio. *Retrieved from: https://www.adafruit.com*, 2019.

[33] Martin Bor and Utz Roedig. LoRa transmission parameter selection. In *13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 27–34. IEEE, 2017.

[34] Adam Dunkels. Rime-a lightweight layered communication stack for sensor networks. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session, Delft, The Netherlands*, 2007.

[35] Norman Abramson. The aloha system: another alternative for computer communications. In *Proceedings of the November 17-19, 1970, fall joint computer conference*, pages 281–285. ACM, 1970.

[36] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *29th annual IEEE international conference on local computer networks*, pages 455–462. IEEE, 2004.

[37] David D Falconer, Fumiyuki Adachi, and Bjorn Gudmundson. Time division multiple access methods for wireless personal communications. *IEEE Communications Magazine*, 33(1):50–57, 1995.

[38] Guowang Miao, Jens Zander, Ki Won Sung, and Slimane Ben Slimane. *Fundamentals of mobile data networks*. Cambridge University Press, 2016.

[39] Roney Paul and KV Shah. An objective comparison of second generation cellular systems-GSM, IS-136 and IS-95. In *1997 IEEE International Conference on Personal Wireless Communications (Cat. No. 97TH8338)*, pages 510–514. IEEE, 1997.

[40] Lu Hong, Feng Hong, Zhong-Wen Guo, and Xiaohui Yang. A TDMA-based MAC protocol in underwater sensor networks. In *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–4. IEEE, 2008.

[41] Ruben Mennes, Miguel Camelo, Maxim Claeys, and Steven Latre. A neural-network-based MF-TDMA MAC scheduler for collaborative wireless networks. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2018.

[42] Ac6. System Workbench for STM32. *http://www.openstm32.org/HomePage*, 2019.

[43] Ac6. Assistance conseil. *www.ac6.fr*, 2019.

[44] ARM, https://os.mbed.com/compiler/. MBED, online compiler platform.

[45] Semtech. SX1272 Datasheet. *Retrieved from: http://www.semtech.com/images/datasheet/sx1272.pdf*, 2015.

[46] Jelle Aerts. *Integrating long range technology into the Contiki operating system framework*. PhD thesis, Masters thesis, Vrije Universiteit Brussel, 2016.

[47] LoRa Alliance. LoRa Network Node Code. *https://github.com/Lora-net/LoRaMac-node*, 2016.

[48] Semtech. LoRaMote User Guide. *Retrieved from: http://www.semtech.com/uploads/documents*, 2015.

[49] STMicroelectronics,http://www.st.com/en/development-tools/stsw-link004.html. STM32 ST-LINK Utility.

[50] STMicroelectronics. Spirit1 Datasheet. *Retrieved from: https://www.st.com/en/wireless-transceivers-mcus-and-modules/spirit1.html*, 2016.

[51] Sinem Coleri Ergen and Pravin Varaiya. TDMA scheduling algorithms for wireless sensor networks. *Wireless Networks*, 16(4):985–997, 2010.

[52] Julius Degesys, Ian Rose, Ankit Patel, and Radhika Nagpal. DESYNC: self-organizing desynchronization and TDMA on wireless sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 11–20. ACM, 2007.

[53] Semtech. SX1272 LoRa Modem: Designer's Guide. *Retrieved from: https://www.semtech.com/uploads/documents/LoraDesignGuide_STD.pdf*, 2013.

[54] Fikret Sivrikaya and Bülent Yener. Time synchronization in sensor networks: a survey. *IEEE network*, 18(4):45–50, 2004.

[55] Randolph Tjoa, Kim Loon Chee, PK Sivaprasad, SV Rao, and Joo Ghee Lim. Clock drift reduction for relative time slot TDMA-based sensor networks. In *2004 IEEE 15th International Symposium on Personal, Indoor and Mobile Radio Communications (IEEE Cat. No. 04TH8754)*, volume 2, pages 1042–1047. IEEE, 2004.

[56] Monsoon Solutions Inc. High Voltage Power Monitor. *https://www.msoon.com/powermonitor-support*, 2019.

[57] Mini Circuits. Programmable Attenuator RCDAT-8000-90. *https://ww2.minicircuits.com/pdfs/RCDAT-8000-90.pdf*, 2019.