NTNU
Norwegian University of
Science and Technology

Energy-Efficient Adaptive Sensing in Low Power Wide Area Networks

Thomas Andersen

Master of Science in Communication Technology
Submission date: June 2018
Supervisor: Frank Alexander Krämer, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title: Energy-Efficient Adaptive Sensing in Low Power Wide Area Networks
Student: Thomas Andersen

Problem description:

Low Power Wide Area Network (LPWAN) technologies such as LoRa and NarrowBand IoT (NB-IoT) are emerging to meet new demands in the field of IoT. The new technologies promise low energy consumption with battery life exceeding ten years.

This thesis will investigate LPWAN technologies in the light of a temperature measurement application. The goal is to investigate the potential implications on node's energy consumption when adding some form of intelligence to a system, providing adaptive transmission capabilities. The state-of-the-art transmission scheme in IoT is having sensors sending measurements to a Fusion Center (server) at a fixed interval. This is a potentially energy expensive solution, especially if the transmission rate is high. Additionally, using a fixed transmission scheme has no way of guaranteeing that the amount of data sent has an actual value to the system. The hypothesis of this thesis is that, if we provide the Fusion Center with some intelligence based on models (predictions on future temperature), the energy consumption in end nodes can be reduced. Therefore, instead of sending data (potentially useless to the system) at a fixed interval, the nodes refrain from sending if the measured temperature matches that of the Fusion Center's prediction. There are two major expectations of such an approach: A Fusion Center with less, but higher value data, and end nodes using less energy than that of the state-of-the-art solution.

The hypothesis will be tested both against the theoretical possibilities of the LPWAN technologies, as well as in experiments with devices running LoRa and NB-IoT.

The key parts of this thesis consist of:

- Understanding the key factors comprising energy consumption in LPWAN technologies
  - What are the baseline energy consumption for a LPWAN device?
  - Can we tweak device's operational cycle (sleep, sense, compute, send, sleep) to reduce energy consumption?
  - Can we calculate the cost of sending a byte? Is ten bytes ten times more expensive?
  - Which factors are fixed and which could potentially be improved?
- Understanding the LPWAN energy modes (e.g. eDRX in NB-IoT)

    - How can we best utilize the modes to extend battery life of devices?
- Establishing an inexpensive way of sending data to sensor nodes
    - Can we piggyback the prediction models from Fusion Center to nodes on existing packets?
    - What form should the prediction have to reduce the cost of sending it to nodes?
- Making a test system consisting of LPWAN nodes sending temperature measurements
    - What are the cost of sending at fixed intervals?
    - What are the cost of utilizing an adaptive transmission scheme?
    - Do we see an improvement in battery life? At what cost?

Responsible professor: Frank Alexander Kraemer, ITEM
Supervisor: Frank Alexander Kraemer, ITEM

Abstract

Low Power Wide Area Networks (LPWAN) is attracting attention from both research communities and the industry due to its low energy, long-range radio communication. Expectations of the capabilities of the LPWAN technologies, with LoRaWAN and NB-IoT being the two most prominent, are high. Standards organizations, such as the 3GPP, promise end-device battery life of more than ten years. However, little research has focused on investigating how this is achievable.

This thesis provides a study on the energy characteristics of LPWAN end-devices, focusing on the prominent factors leading to increased energy consumption. The research further investigates the potential implications of applying Adaptive Sensing techniques to the system.

From experiments with a Pycom Fipy LoRaWAN device, we show that energy consumption when transmitting uplink packets consists of two parts: an increasing linear factor related to increasing the packet payload size, plus a fixed cost independent of payload. Introducing accumulation of packets, or adaptive sensing, the fixed cost yields a potential reduction in end-device energy consumption of up to 9000 %, compared to using a fixed-rate transmission scheme. The results of the study also show that when adaptive sensing manages to reduce half of the transmitted packets, a seven-month increase in battery life is possible when using a 2400 mAh battery. This increase implies a 65 % extended battery life, even when sending only maximum payload packets.

Sammendrag

Low Power Wide Area Networks (LPWAN) tiltrekker seg oppmerksomhet fra både forskningsmiljøer og industrien på grunn av sin lav-energi, langdistanse radiokommunikasjon. Forventninger rundt egenskapene til LPWAN-teknologiene, med LoRaWAN og NB-IoT som de to mest fremtredende, er høye. Standardisingsorganisasjoner, som 3GPP, lover batterilevetid på over ti år for enhetene. Imidlertid har lite forskning fokusert på å undersøke hvordan dette kan oppnås.

Denne oppgaven undersøker energikarakteristikkene til LPWAN-enheter, med fokus på de fremtredende faktorene som fører til økt energiforbruk. Forskningen undersøker videre de potensielle implikasjonene ved å anvende Adaptive Sensing teknikker i systemet.

Fra eksperimenter med en Pycom Fipy LoRaWAN-enhet viser vi at energiforbruket ved overføring av pakker fra enhet til server består av to deler: En økende lineær faktor relatert til økning i pakkestørrelse, pluss en fiksert kostnad uavhengig av pakkestørrelsen. Ved å innføre akkumulering av pakker eller adaptive teknikker, fører den fikserte kostnaden til en potensiell reduksjon i energiforbruket på opptil 9000%, sammenlignet med å sende på et fastsatt intervall. Resultatene av studien viser også at dersom de adaptive teknikkene klarer å redusere halvparten av de overførte pakkene, er det mulig å øke batteriets levetid med syv måneder når man bruker et 2400 mAh batteri. Dette medfører en økning i batterilevetiden på 65 %, selv om man kun sender pakker med maksimal pakkestørrelse.

Preface

This Master's thesis is the final deliverable of the 5-year MSc program in Communication Technology at The Department of Information Security and Communication Technology, Norwegian University of Science and Technology (NTNU). The thesis is part of an on-going inter-disciplinary research project at the Faculty of Information Technology, and Electrical Engineering called Autonomous Resource-Constrained Things (ART). The research conducted for this master thesis consists of two parts: A pre-study conducted autumn 2017 and this thesis due June 2018. The research is by Thomas Andersen and supervised by Frank Alexander Kraemer at The Department of Information Security and Communication Technology, NTNU.

The initial research problem, forming the basis for the pre-study, and later this thesis was presented to NTNU by The Norwegian Public Roads Administration and Telenor. The pre-study with title *"Is an IoT approach to road condition monitoring feasible?"* focused on the potential of deploying small cost sensors along roads, monitoring the environment, as opposed to high-cost weather stations that are in use today. This thesis builds on the pre-study but with a broader perspective: an investigation into how we can build adaptive sensing applications within LPWAN.

Acknowledgments

First of all, I want to thank the NPRA and Telenor for presenting an exciting research project. Although the project has taken a more specialized form since the initial project for the pre-study, the task is in many ways the same.

I want to thank my supervisor Frank Alexander Kraemer for his guidance throughout the process of writing this thesis. I want to thank him for always being available to answer questions, for being genuinely interested in the research, for pushing me to write, for reading through all my work and giving constructive feedback. Lastly, for making me understand that *if it does not work, that is also an answer.*

I especially want to thank Ph.D. student Nattachart Tamkittikhun for his interest in and knowledge about my research. Although not any formally defined relationship with, or obligations towards my research existed,

he was always available for questions, and help in general. I especially want to put a focus on the fact that he more than once sat with me debugging code and electronics, often after midnight. Much of the energy consumption measurements would not have been possible if it had not been for his help.

Lastly, I am grateful for Sanna Granbo's support throughout the process, especially for leaving work early to look after our puppy Truls so I could focus on the thesis.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1
Introduction

There are high expectations of what the future of IoT should, and needs to be. IoT devices are expected to be deployed *anywhere*, be as self-sufficient[1] as possible, and provide users with up-to-date information about its sensing environment. Therefore, technologies providing low-cost operation and communication is required. Additionally, applications need to be designed to reduce the energy consumption of end-devices to a minimum. Industry, academia, and standards development bodies have devoted significant efforts to LPWAN in the last years [RKS17]. Low Power Wide Area Network (LPWAN) technologies such as LoRaWAN and Narrow-band IoT (NB-IoT) are emerging to meet new demands in the field of IoT. The new technologies promise long-range communication and low energy consumption with battery life exceeding ten years. However, under which operating condition this holds is unclear. Many IoT devices, such as sensors and actuators are required to be battery-operated. An investigation into the characteristics affecting the energy consumption of these devices is therefore crucial. Most of the published work within the field, however, only consider this topic to a limited extent, providing rough estimates on energy performance based on datasheets, not considering the realistic behavior of device hardware. Additionally, research tends to focus on comparing the different technologies based on features such as coverage, scalability, throughput and latency [MBCM18]. This thesis investigates the factors influencing energy consumption of LPWAN devices, both based on work done by other researchers, and experiments. We introduce the concept of adaptive sensing into the system as a means of intelligently trying to lower energy consumption of end-devices.

[1] The term self-sufficient here refers to the devices not needing battery changes or other management tasks. For instance, a device solely running on solar panels is defined as self-sufficient.

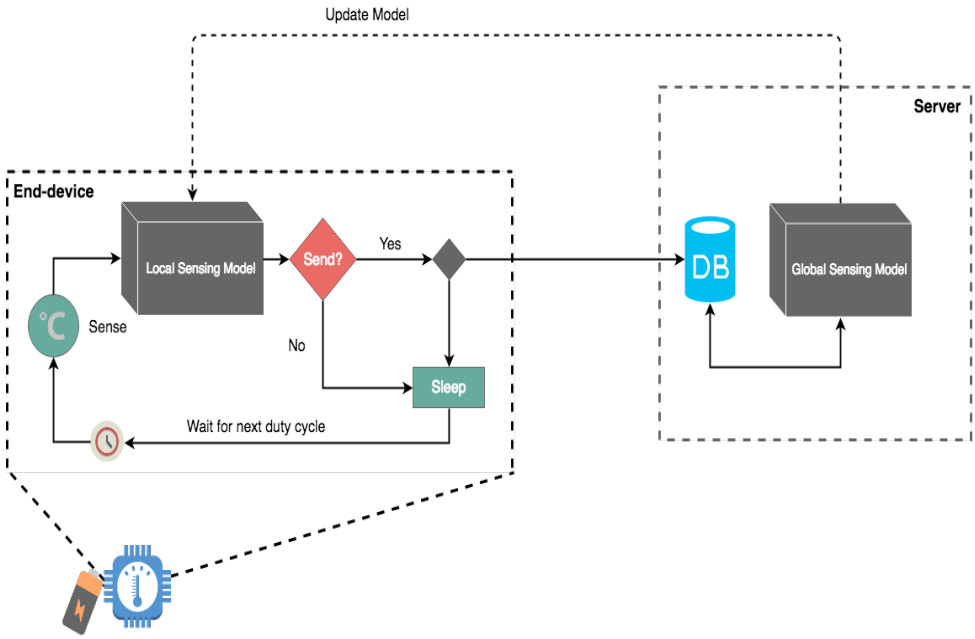

Figure 1.1: System schematics

Transmitting data from a sensor node to a server is often an expensive task when it comes to energy consumption. Nodes can be located far away from the server, making the cost of a single transmission high. Still, many applications rely on sensors providing frequent measurements throughout a day. Instead of statically defining a lower frequency for nodes to sense and transmit data, adaptive sensing introduces mechanisms into the system to dynamically change this at runtime. We can apply many different adaptive sensing techniques to a system (see section 2.5 for techniques relevant to this thesis). Although the techniques are different in the way they achieve the *adaptiveness*, the general goal is the same: save energy by preventing unnecessary tasks from happening, providing the server with "just enough" information and at the same time consuming the least amount of energy doing so. In other words, send data only when it is needed. Doing this requires knowledge about the environment that is to be measured. For instance, an environment application measuring temperature needs to know something about what it should expect when measuring, to be able to make the right decisions as to whether to send or not. Figure 1.1 shows a simplified system design schematics, exemplifying one way of introducing adaptive sensing into the system. We introduce adaptive sensing as a means of trying to lower the energy consumption of end-devices, and the design consists of two models: a local, and a

global sensing model. The *Global Sensing Model*, residing on the server, is in charge of modeling future data for all the nodes in the network. The results of the global sensing model can, for instance, be predictions of future sensor values for a given node, and are made using different techniques (e.g., statistics or machine learning). The global model bases its predictions on historical values from the sensor nodes, but can also include other data sources (e.g., weather forecasts). The *Local Sensing Model* is local to each end-device, and is based on the results of the global sensing model. The local sensing model is used to make decisions on whether the end-device should send data to the server or not. Additionally, the model can define application parameters such as device sleep time or sensing parameters (e.g., if the device should measure a single sensor value or multiple times and average the results). During application runtime, the model can be updated on a defined interval or upon request (by either end-devices or the server). The server updates the local sensing model of an end-device by sending a downlink message. The end-devices in the system design in Figure 1.1 has one primary goal: send data to the server as seldom as possible. When the global sensing model generates accurate predictions, the end-devices sends less often, leading to extending the lifetime of the system.

1.1 Motivation

To motivate our research, consider an environmental monitoring sensor network deployed along (potentially remote) roads. The sensor nodes should collect data about the environment (e.g., temperature, humidity and wind) and send data to a central server, potentially located far away. The central server calculates a prediction model for the data to be sensed in a location every hour and sends the prediction to the end-devices in that location. End-devices are required to use as little energy as possible, so instead of sending temperature data to the central server each sensing cycle, the nodes compares the measured temperature against the prediction model, provided by the sensor. If the prediction holds, the end-device do not need to send the data, since the server already knows the result. With a good prediction model, such a design could reduce the overall energy consumption of end-devices without limiting the hardware (e.g., reducing transmit power of radios). Such a system must be designed to operate under the following conditions and constraints:

Location: Design of end-devices should permit deployment in any location and terrain, leading to the need for batteries and possibly solar panels.

Energy constraints: End-devices should be energy-aware, and use as little energy as possible. Energy-awareness implies usage of adaptive sensing techniques, as opposed to fixed interval sensing and data transmissions. End-devices have two goals: provide the system with sensor data and maximize system lifetime.

Unattended operation: A sensor network consists of a potentially large number of sensors, covering large geographical areas. End-devices should, therefore, require little or no manual configuration and management. Deployment of sensors outdoors also yields considering the dynamics of the environment. Lastly, end-devices needs to enable remote updating of run-time parameters, and possibly device firmware.

1.2 Research Goals

There are two goals in this research: understanding the factors influencing energy consumption of LPWAN devices and how to reduce them. The idea that sending data less often should result in lower energy consumption is somewhat intuitive. However, we do not know to what extent this is true. We also do not know what the best mechanisms for reducing energy consumption is. Is it better to try to send less often in itself, or should we focus on aggregating measurement data to send larger packets when first sending?

To reach the research goals, we ask the following overall research question:

RQ1: What are the main factors driving energy consumption of LPWAN devices, and what is the effect of using adaptive sensing schemes as a means of reducing overall energy consumption?

The proposed research looks at techniques for reducing the energy consumption of sensor nodes at the application layer, by using the underlying already available techniques of the LPWAN technologies. Although improvements possibly could be made to improve the existing LPWAN technologies, this is not the focus of this thesis.

1.3 Thesis Structure

We structure the thesis in the following way: Chapter 2 presents the background material for this thesis, with an introduction to Low Power Wide Area Networks in general, before presenting two of the most prominent technologies; LoRaWAN and Narrowband IoT (NB-IoT). We continue with a presentation of Adaptive Sensing, focusing on some of the relevant sub-branches within the area. Chapter 2 concludes with a presentation of energy consumption models for LPWAN. In chapter 3, we present the research methodology used in this thesis. The chapter also includes a presentation of the equipment used in the experiments as well as an introduction to the experiment design. In chapter 4, we present experiments and preliminary results. Chapter 5 discuss the findings from the experiments in light of the background material, before chapter 6 concludes this thesis.

Chapter 2
Background and Related Work

Sensor networks periodically collect context information on the physical environment from remote terrains and send the data back to a server for further processing [HEE13]. The server is generally located far away from the nodes. Limited energy resources are the primary factor limiting the lifetime of deployed sensor nodes. Therefore any development of application in this domain needs to take the issue of energy efficiency into consideration. Sensor networks discussed in this thesis are applications where data is not expected to change dramatically or critically in small time frames.

This chapter first presents a general overview of Low Power Wide Area Networks before looking into two of the most prominent technologies, LoRaWAN and NB-IoT. The technologies are presented in a general way, focusing on details relevant to this thesis. The chapter concludes with an introduction to adaptive sensing and some of the relevant methods of the field, before presenting energy consumption models for LPWAN.

2.1 Low Power Wide Area Networks

Low Power Wide Area Networks (LPWANs) is a novel communication paradigm, made to complement traditional cellular and short-range wireless technologies in addressing the diverse requirements of IoT applications. LPWA is a generic term for a group of technologies that enable wide area communications at lower cost points and better energy consumption [SWH17] and is one of the fastest growing spaces in IoT. LPWAN technologies can be used *anywhere* and *anytime* to sense and interact with their environment instantly [RKS17]. Proprietary LPWAN technologies are already being rolled out in Norway and are attracting attention from developers for many reasons. One being its low power, long range, and low-cost communication characteristics [MBCM18]. Another reason is that the technologies fill a gap in the IoT landscape. Short-range wireless networks like ZigBee and Bluetooth are very energy-efficient and used in many applications. However, the technologies are

not applicable to scenarios with sensors distributed over large geographical areas. The range of these technologies is limited to a few hundred meters at best [RKS17]. Cellular technologies like 3G/4G, on the other hand, provide wide area coverage, but their design and features (e.g., signaling, handover between base stations and high data rates) makes them too energy demanding for resource-constrained devices. LPWAN provides communication for up to 10-40 km in rural areas and 1-5 km in urban areas, is highly energy efficient (10+ years of battery life) and is cost-effective in regards to hardware purchases [MBCM18]. These features, however, comes at the expense of low data rates, which therefore makes the technologies not suitable to all application scenarios. Applications needing multi-year battery lifetime while sending small amounts of data over long distances a few times per hour is the intended targets. In figure 2.1 we see the data rate and transmission range of different radio technologies. The transmission range of LPWAN surpasses those of the other radio technologies but is best fit for applications made to send just a tiny amount of data. Many LPWAN technologies have arisen in the last couple of years, both in the licensed and unlicensed frequency bands, and approximately seven billion IoT/M2M devices are predicted to be connected to the Internet using LPWAN within 2025 [RKS17]. As many different hardware manufacturers and software companies are making proprietary solutions for the different technologies, the need for standardization in the LPWAN space is apparent. Several well-known standard developing organizations such as European Telecommunications Standard Institute (ETSI) [ETS], Third Generation Partnership Project (3GPP) [3GPa], Institute of Electrical and Electronics Engineers (IEEE) [IEE], and the Internet Engineering Task Force (IETF) [IET] are working towards the open standards for LPWA technologies [RKS17]. In Norway, research and development have been made to implement LoRaWAN and NB-IoT by the country's two largest telcos, Telenor and Telia Sonera. LoRaWAN has come the longest regarding deployment, but both Telia Sonera and Telenor have begun testing on NB-IoT. Telenor is planning a nationwide launch of NB-IoT in autumn of 2018.

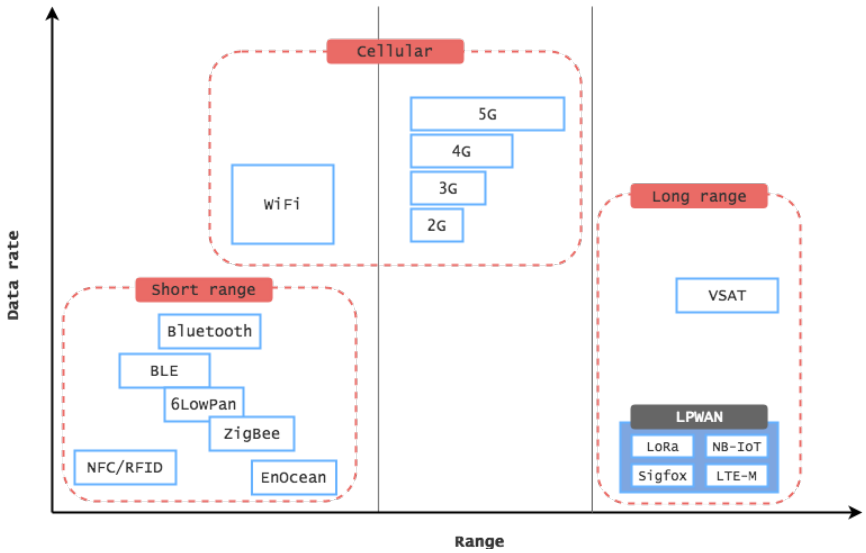

Figure 2.1: Required data rate vs. range of radio communication technologies. Adapted from [MBCM18] and [SWH17]

The following presentations of NB-IoT and LoRaWAN provides a general overview of the two technologies. Many details (e.g., the internals of the network) are left out. The reason for this is that most of the internals of the LoRaWAN and NB-IoT networks are not configurable by the end-devices, and are therefore not needed to understand how to achieve energy efficiency in the respective networks. A general overview, and what makes the two network different, however, will be presented. We also show a comparison of some of the differences between the two technologies.

2.2 LoRaWAN

2.2.1 LoRaWAN General Overview

LoRaWAN is a long-range, low power wireless communication technology based on a one-hop radio system [CMVG17]. LoRaWAN operates in the unlicensed Industrial, Scientific and Medical (ISM) band [AVT+17] and has a star-of-stars topology, composed of end-devices, gateways and a central server (see Figure 2.2). There is no one-to-one association between end-devices and gateways, so multiple gateways often receive data transmitted from one end-device. All gateways forward all packets from end-devices to the network server, which filters out duplicate packets, using only the *best* received packet. The topology improves the ratio of successfully received messages, as many gateways may try to forward packets from end-devices [MBCM18]. Additionally, the topology reduces the need for handover, since multiple gateways receive the messages. LoRaWAN defines the architecture of the system and the communication protocol, while LoRa defines the physical layer modulation scheme used for message transmissions [SWH17]. LoRa is a proprietary modulation scheme based on Chirp Spread Spectrum Modulation (CSS) [SWH17] used between end-devices and LoRaWAN gateways. Gateways communicate with the network server using IP based protocols.

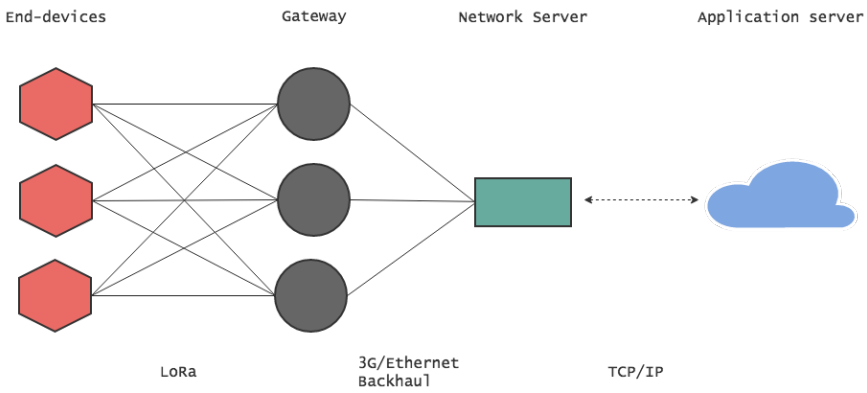

Figure 2.2: LoRaWAN System Architecture. Inspired by [CMVG17]

The LoRaWAN specification defines three functionality classes: Class A, Class B, and Class C [SWH17]. When choosing which class to use for any given system implementation, there is a trade-off between latency of sent packets and energy consumption (battery life) of end devices. All LoRaWAN devices must support Class A, which allows bidirectional communication between an end-device and the

network server, scheduled by the end-device based on its needs. Downlink messages (from the server to end-device) for devices with Class A can only occur after an uplink transmission (see Figure 2.3). Class B is based on Class A but also supports additional downlink transmissions happening at a specified time. The idea is that a Class B device can listen for incoming packets at a predefined time, which can be useful, e.g., for device configuration. Class C allows downlink transmission to happen at any time, except when the end-device is transmitting an uplink message. Class C end-devices consumes more power but offers the lowest latency of the three classes. Both Class B and C are optional to implement for device manufacturers and are therefore not often seen used. All experiments in this thesis use the Class A functionality class. LoRaWAN end-devices have communication constraints, due to duty-cycle limitations posed by the ISM band. In Europe, the duty cycle limitation is 1% in the EU868 frequency band. In practice, this means that end-devices have a maximum transmission time of 36 seconds/hour in each sub-band [AVT+17].

2.2.2 Physical Layer

LoRa modulation is used to achieve better spectral efficiency and increased network capacity. Each LoRa symbol is composed of 2^{SF} chirps, where SF represents the corresponding spreading factor [SWH17]. There are six orthogonal SFs in the range of 7 to 12, which provide different Data Rates. Each data rate is directly proportional to the spreading factor. The higher the spreading factor (i.e., slower transmission), the longer the communication range [AVT+17]. Figure 2.1 shows the different data rates and spreading factor configurations available. The EU 868 ISM band defines three default channels for LoRaWAN. Each channel has a bandwidth of 125 kHz, use the LoRa modulation and must allow data rates in the range 0.3 kbps to 5 kbps (DR0 to DR5).

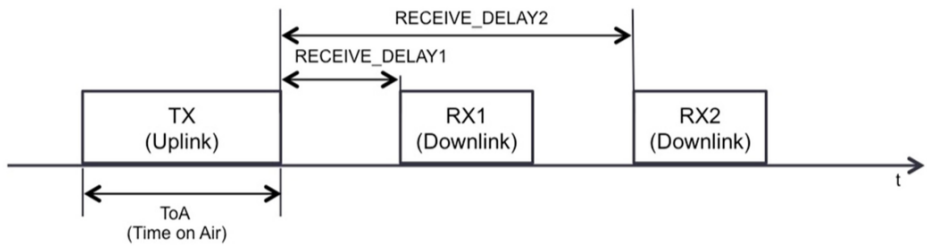

Figure 2.3: Basic transmission scheduling in LoRaWAN Class A functionality. Taken from [CMVG17]

Figure 2.3 shows the transmission scheduling in LoRaWAN for the Class A functionality class. Receive windows RX1 and RX2 are opened by the end-device to listen for downlink packets shortly after an end-device sends a successful uplink packet [MBCM18]. The default offset time, i.e., the time between the uplink packet, to RX1 and RX2 is opened is one and two seconds, respectively [AVT+17]. Downlink traffic destined for the end-device is only allowed in these reception windows for Class A devices. Packets not received, will have to wait until the end-device finishes a new uplink transmission.

Table 2.1: Data rates (DR) and related configuration for EU863-870 band channels. Taken from [CMVG17].

Data Rate (DR)	**Configuration**			**Physical Bit Rate (bit/s)**
	Modulation	**Spreading Factor (SF)**	**Bandwidth**	
0	LoRa	SF12	125 kHz	250
1	LoRa	SF11	125 kHz	440
2	LoRa	SF10	125 kHz	980
3	LoRa	SF9	125 kHz	1760
4	LoRa	SF8	125 kHz	3125
5	LoRa	SF7	125 kHz	5470
6	LoRa	SF7	250 kHz	11000
7	FSK	50 kbit/s		50000
8-15	Reserved for future use			

2.2.3 MAC Layer

The LoRaWAN protocol defines three basic MAC messages: Join message, Confirmed Data message and Unconfirmed Data message [CMVG17]. Join messages is exchanged between the end-device and the server when the end-device wants to join the network. Confirmed Data Messages are data messages requiring ACK messages returned from server to the end-device, much like TCP for IP based protocols. Due to the duty-cycle restrictions of LoRaWAN mentioned earlier, infrequent use of Confirmed Data Messages is recommended. We can compare unconfirmed Data Messages to UDP, where an end-device sends a message into the network without having any way of knowing if the intended party received the message. We only use unconfirmed messages in the experiments in this thesis.

2.3 Narrowband IoT

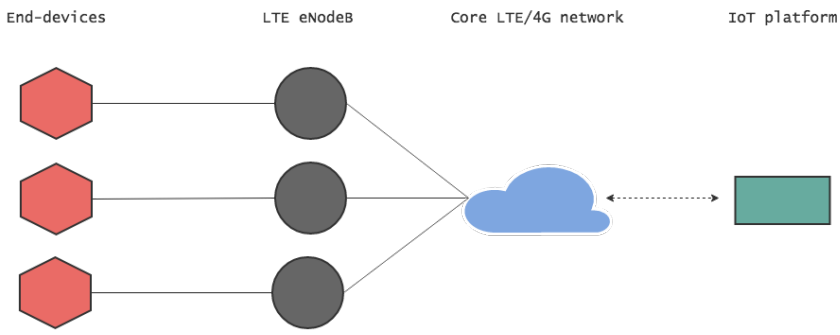

Figure 2.4: A high level view of the NB-IoT architecture.

2.3.1 General Overview

NB-IoT is a new LPWAN technology standardized by The 3rd Generation Partnership Program (3GPP) as a part of release 13. In Figure 2.4 we present a high-level view of the NB-IoT architecture. The NB-IoT communication protocol is based on the LTE protocol [MBCM18] but is a new air interface for IoT [SWH17]. NB-IoT development focused on reducing device cost and enhancing end-device battery life, removing many of the features available in LTE (e.g., handover and dual connectivity). The technology reuses the same licensed frequency bands used by LTE, meaning end-devices require SIM cards issued by an operator to connect to the networks. The NB-IoT core network uses the evolved packet system (EPS) but introduces two optimizations for the cellular Internet of Things (CIoT): the User Plane CIoT and the Control Plane CIoT [SWH17]. NB-IoT allows connection of 100 000 devices per cell, and cell access procedure is similar to the one used in LTE. The existing E-UTRAN network architecture and backbone can be reused, with only software upgrades to the existing LTE infrastructure [MBCM18]. These software upgrades enable NB-IoT to coexist with LTE and GSM under the 700 MHz, 800 MHz and 900 MHz licensed frequency bands. NB-IoT is optimized for small and infrequent data messages [SWH17], which means that it is energy efficient for end-devices. NB-IoT data rates are limited to 200 kbps for downlink and 20 kbps for uplink messages

[MBCM18]. For modulation, NB-IoT uses Quadrature phase-shift keying (QPSK), with OFDMA for downlink and SC-FDMA for uplink traffic [SWH17].

There are three frequency band deployments in NB-IoT: standalone, guard-band and in-band (see Figure 2.5). NB-IoT utilizes GSM frequency with a bandwidth of 200 kHz between guard bands of 10 kHz for standalone operation while using un-used guard band and resource blocks of the LTE carrier for guard-band and in-band operations, respectively.

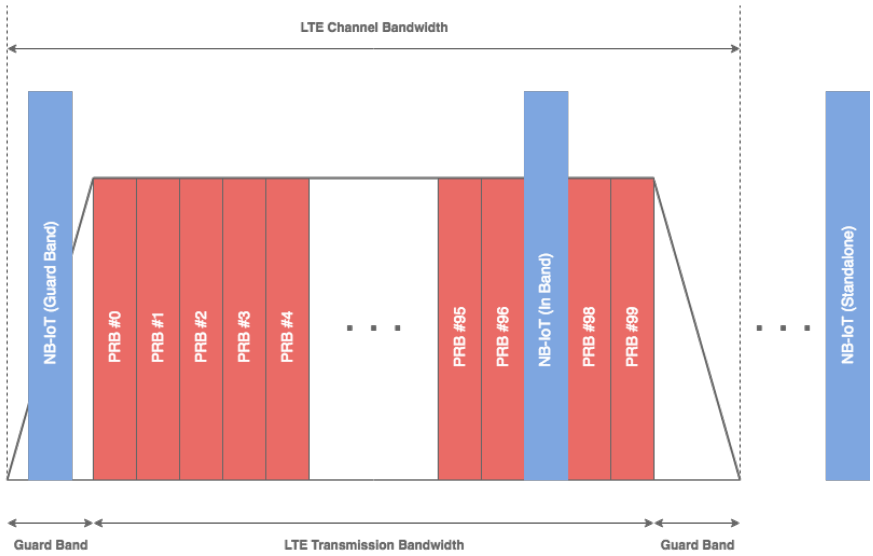

Figure 2.5: NB-IoT operation modes. Adapted from [LL17].

2.3.2 Operating Modes

Figure 2.6, shows the three power saving modes that can be used to reduce energy consumption on end-devices: discontinuous reception (DRX), extended discontinuous reception (eDRX) and power saving mode (PSM) [LL17]. In *Discontinuous Reception* (DRX), the UE modem deactivates during a specific period negotiated with the base station (BS). The end-device turns off the modem and reactivates it only during a specific time to monitor the downlink channel. If the BS has a packet destined for the end-device, transmission of the packet is delayed until the end-device reactivates the modem. This transmission scheme is in many ways similar to class A of the LoRaWAN protocol. In figure 2.6 (a), we see the UE activating the modem and listening for paging information from the BS for 1 ms before, deactivating the modem and going back to sleep. The maximum DRX period of an LTE network is 2.56 seconds, due to user experience considerations. Since energy consumption of IoT devices is more

important than immediate response, *extended Discontinuous Reception* (eDRX) was introduced as a way of enabling the UE to extend the time between monitoring paging messages from the BS (see figure 2.6 (b)). The eDRX period is extendable to as much as 175 minutes. Many IoT applications do not require end-devices sending packets frequently or being able to receive a packet from a server at any given time. *Power Saving Mode*[1] (PSM) (see figure 2.6 (c)) was introduces as a means of further lowering the energy consumption of end-devices. PSM turns off the access stratum all together and enables deep sleep mode on end-devices over longer periods of time. NB-IoT uses the Tracking Area Update (TAU) message for negotiating and deciding the PSM interval between the BS and the UE. The maximum PSM interval is in the range of several months.

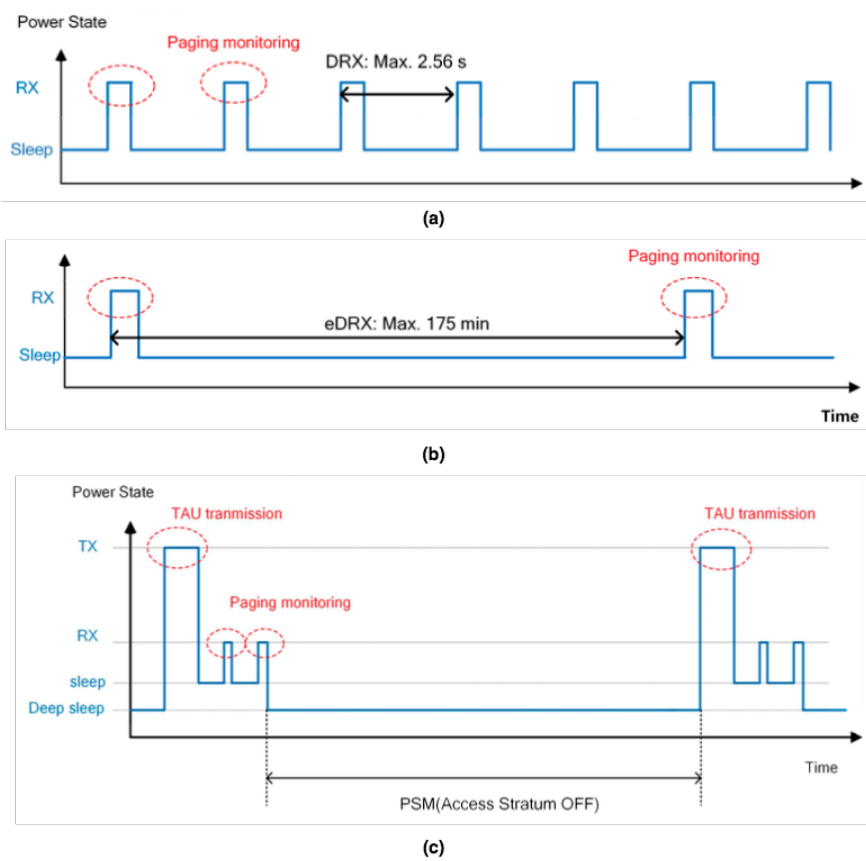

Figure 2.6: User equipment (UE) power state: (a) discontinuous reception (DRX); (b) extended discontinuous reception (eDRX); (c) power saving mode (PSM). Adapted from [LL17].

[1] In some literature referred to as Power Saving State (PSS).

2.4 Comparison Between LoRa and NarrowBand IoT

In the following, we show a brief comparison between LoRaWAN and NB-IoT. We include Figure 2.7 because it graphically shows how the two technologies focus on and achieve different factors related to IoT. We will not, however, discuss the details about the specifics regarding these factors, since it is out of the scope of this thesis[2]. We will, however, remark that the *battery life* from figure 2.7 is much lower for NB-IoT due to synchronous communication, signaling with base stations and QoS handling. Table 2.2 summarizes the most important technical details of the two technologies.

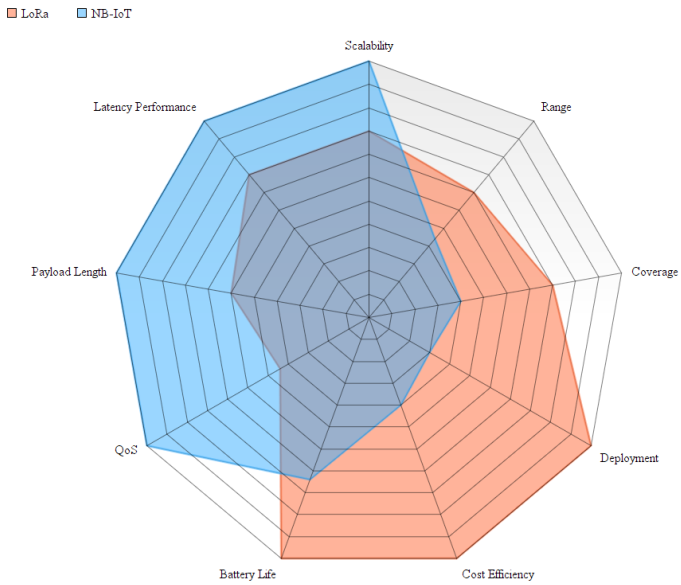

Figure 2.7: Advantages of LoRaWAN and NB-IoT regarding IoT factors. Note that in the diagram, LoRa is used instead of LoRaWAN. Mekki et al. use the term LoRa in the diagram that inspired this figure. It does not, however, make sense to compare LoRa, a physical modulation scheme, with NB-IoT. We used a web-based diagram maker to generate this diagram, and we discovered the error too late to remake it. Adapted from [MBCM18].

[2]The interested reader should consult [MBCM18] and [SWH17] for more detail.

Table 2.2: Comparision between LoRaWAN and NB-IoT. Adapted from [MBCM18].

	LoRaWAN	**NB-IoT**
Modulation	CSS	QPSK
Frequency	Unlicensed (868 MHz in Europe)	Licensed LTE bands
Bandwidth	250 kHz and 125 kHz	200 kHz
Maximum data rate	50 kbps	200 kbps
Bidirectional	Yes / Half-duplex	Yes / Half-duplex
Maximum messages/day	Unlimited (however, limited by the duty-cycle restrictions of the ISM band)	Unlimited
Maximum payload length	242 bytes	1600 bytes
Range	5 km (urban), 20km (rural)	1 km (urban), 10 km (rural)
Interference immunity	Very high	Low
Authentication and Encryption	AES 128	LTE encryption
Adaptive data rate	Yes	No
Handover	Not needed (broadcast)	Single base station
Localization	Yes (TDOA)	No (under specification)
Allow private network	Yes	No
Standardization	LoRa-Alliance	3GPP

2.5 Adaptive Sensing

Wireless sensor nodes have limited resources and often rely on batteries to supply energy. It is therefore important to be aware of, and manage the use of energy resources. The main goal of a sensor network is to provide the server with data gathered from its environment. The sensor nodes typically go through a duty cycle which consists of different states. First, the node wakes up from its sleep phase and enters the active phase. In the active phase, nodes gather data from sensors, process and send the data to the server. Finally, the node returns to the sleep phase. The duty cycle repeats as long as the device is active. For a given hardware configuration there is not much to do with the energy consumption of the different states. Radios for instance, often allow changes to the transmit power, which when lowered can reduce energy consumption. However, this is often not desired since it implies limiting the transmission range of the sensor node. One approach to lowering the energy consumption of nodes without limiting the capabilities of the node itself is to reduce the number of radio transmissions. In other words, we need to limit how often the node sends data to the server. It is trivial that limiting the most energy consuming duty cycle state[3] will save energy, but this comes at a cost. When saving energy this way, we lower data granularity, leading to less knowledge of the environment we are sensing. Adaptive sensing is a set of techniques that can be exploited to mend for these challenges. The basic idea is to save energy by preventing unnecessary tasks from happening. Figure 2.8 shows a taxonomy of the different techniques available. The nodes in blue are techniques applicable to the scenarios we discuss in this thesis. A presentation of the relevant techniques follows.

2.5.1 Duty Cycling

Duty cycling consists of sensor nodes being active only for the time it needs to sense its environment and then returning to sleep immediately after [AAFR09]. Two complementary approaches can be used to achieve duty cycling: *Topology control* and *Power Management*. Topology control is when we exploit redundancy in nodes either based on energy levels of sensor nodes or the data they provide and adaptively selecting only a subset of all nodes to be active [ACDFP09]. In *Adaptive Sensing for Environmental Monitoring*, Arici et al. [AA04] propose an adaptive sensing algorithm for environmental monitoring based on a two-tier network topology, where top-tier nodes gather, process and send sensor node data of bottom-tier nodes to the server. The scheme generates clusters of sensor nodes and classifies nodes as redundant when a cluster produces low variance measurements. A similar two-tier approach can be seen in Hady et al.'s *Intelligent Sleeping Mechanism for wireless sensor networks*

[3]This notion is used throughout this thesis. However, this is not applicable to all cases. See Alippi et al.'s *Energy Management in Wireless Sensor Networks with Energy-Hungry Sensors* [AAFR09] for more detail.

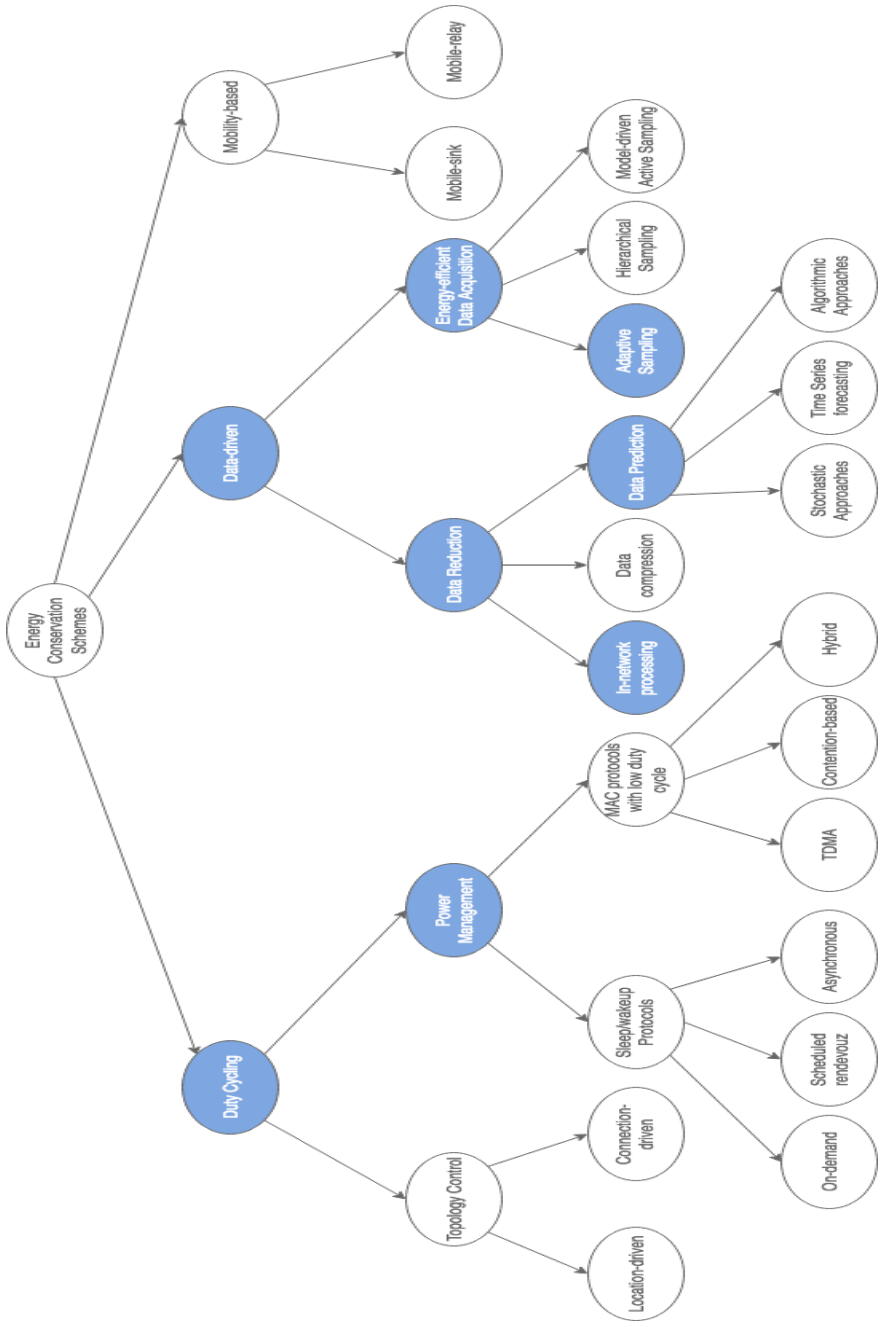

Figure 2.8: Taxonomy of adaptive sensing strategies. Adapted from [ACDFP09]

[HEE13]. Top-tier nodes receive data from all nodes in its cluster, accumulated and average the data and send it to the server. The server analyzes the data, and based on some set threshold decides which cluster(s) should sleep for the remainder of the round. Both pieces of research require having an intelligent server with an overview of the entire system that makes network resource allocation decisions based on the information provided by sensor nodes. These types of duty cycle schemes will not be investigated further in this thesis, however, if we were to scale up our experiments, combinations of different techniques could be investigated.

Power Management with regards to duty cycling in this thesis mostly deals with sleep/wake-up protocols. More generally, this means entering and exiting deep sleep on devices through application code using the hardware APIs of the devices.

2.5.2 In-network Processing

Data reduction schemes address the case of unneeded samples [ACDFP09]. In-network processing relates to performing data aggregation techniques on sensor nodes. The techniques will be application-specific, but can, for instance, relate to calculating the average over a set of measurement data. In-network processing could be used to exploit the findings in section 4.4, where we show that aggregation of data could reduce the energy consumption of end-devices by a factor of 0.6133 mJ for each aggregated packet.

2.5.3 Data Prediction

Data Prediction refers to building an abstraction, or a model of a sensed phenomenon [ACDFP09]. The model describes the data evolution, and the goal is to predict the sensor values within certain levels of error (dynamically or statically defined for each application). A successful prediction indicates that knowledge of the sensed environment is present in the system without the sensor nodes needing to send data. When the model prediction fails to meet the allowed level of error, sensor nodes send data. Failed predictions could also lead to the model needing to be updated. Data prediction leads to reduced information sent by nodes, hence less communication. What the Data Prediction model is, or should be will not be discussed further in this thesis. Data Prediction will rather act as a black box providing the system with a model prediction, on which sensor nodes decides on whether to send data or not. For the interested reader, see [WLG+11] for an application of prediction-based data aggregation using Combined Grey model and Kalman Filter Data Aggregation (CoGKDA) or [LBSB07] for an algorithm enabling adaptive model selection for time series predictions.

2.5.4 Adaptive Sampling

Energy-efficient data acquisition aims at reducing the energy spent by the sensing subsystem [ACDFP09]. Adaptive sampling can, for instance, reduce the number of packets a node needs to send or how often a node senses by using information about the environment of the node. More specifically by exploiting spatiotemporal correlations between data. Environmental data, like temperature, has natural variations that can be used to know something about future values or to make a deduction of temperature on one node based on data from another node. In Willett et al. [WMN04], *Backcasting* is proposed as an adaptive sensing scheme where a small subset of wireless sensors initially communicate their sensor data to a server, forming an estimate of the environment. Based on the estimate, the server activates a portion of available nodes to achieve the desired level of accuracy in the collected data. The key idea is that the estimate can detect a correlation in the sensed environment, indicating that many sensors may not need to be activated to achieve the desired level of accuracy.

2.6 Modeling the Energy Consumption of End-devices

Tamkittikhun et al. experimented with an estimation model for the energy consumption of sensors nodes, in *Energy Consumption Estimation for Energy-Aware, Adaptive Sensing Applications* [THK17]. The research focuses on identifying the distinct states that an end-device goes through during a duty-cycle. Measurements of average power consumption in each state, conducted prior to deployment, is used to estimate the total energy consumption for end-devices at runtime. The research showed that the total energy consumption E of a duty cycle for a given sensor node could be estimated at runtime by:

$E_{duty-cycle} = \sum_{i=1}^{I} P_i * \Delta T_i$

where P_i is the power consumed in state i and Δt_i is the time spent in state i.

This approach is a very intuitive way of making an energy consumption estimation. For all states, measure the time used in the state and multiply it by the power that state consumes. At runtime, this only requires the use of timers which calculates the Δt_i for each state. For this approach to work, however, it requires identifying the power consumption in each state at design time. We also need to have a system that operates more or less, in the same way, each duty-cycle. So if we make changes to the system, we need to know how this affects the energy consumption of the node.

2.6.1 Modeling the Energy Consumption of LoRaWAN

A similar approach is seen in the analytical models of end-device current consumption and lifetime, in Casals et al.'s *Modeling the Energy Performance of LoRaWAN*

[CMVG17]. The same approach as in Tamkittikhun et al.'s work is used to calculate the average current consumption. However, Casals et al. takes it one step further and introduces T_{notif} to the equations as the period between duty cycles, to be able to estimate system lifetime. A larger T_{notif} means the device sleeps for a longer period. The formula for the average current of an end-device is:

$I_{avg} = \frac{1}{T_{notif}} \sum_{i=1}^{I} I_i * T_i$

The article further presents the system lifetime as:

$T_{lifetime} = \frac{C_{battery}}{I_{avg}}$

where $C_{battery}$ refers to the battery capacity in mAh. The calculation for $T_{lifetime}$ results in an estimate for the lifetime of an end-device in hours, given the current and time used in each state of the duty cycle.

The article also defines the energy consumed by the end-device as:

$E = I_{avg} * V * T_{notif}$

where V denotes the voltage of the end-device.

2.6.2 Modeling the Energy Performance of NB-IoT

Chapter 7.3.6.4, *Energy Consumption Evolution* and chapter 5.4, *Energy Consumption Evolution Methodology* of the 3GPP release 13 [3GPb], provides calculations for the battery life of NB-IoT devices. In release 13, NB-IoT initially was designed to use FDMA as channel access for both uplink and downlink. The design has later been altered to use OFDMA in downlink and SC-FDMA in uplink, making the calculations in Release 13 invalid, due to increased complexity. Despite this, we can still get a general idea of the energy consumption. As with the formulas in 2.6 and 2.6.1, the idea is to identify the different states the device goes through and multiply each states power consumption with the time used in that state.

From Release 13 we have that, *Energy per data report* is

$e_1 = E_{Tx} + E_{Rx} + E_{Idle\,tasks}$

$E_1\,(Joules) = \frac{e_1}{1000000}$

Additionally, the *Energy consumed per day* is:

$E_2 = E_{per\,report} * Reports_{per\,day} + E_{PSS\,per\,day}$

$e_2\,(WattHours) = \frac{E_2}{3600}$

This leads to:

$Days\,of\,battery\,life(D) = \frac{Battery\,Capacity}{e_2}$

$Years\,of\,battery\,life\,(Y) = \frac{D}{365}$

Release 13 only provides values for 4 of the 11 defined parameters for the equations. However, the authors still manage to summarize the results for the battery life of NB-IoT device in two tables (with integrated PA and external PA). How they came to these conclusions is however unclear. The table for the external PA is reproduced in Table 2.3. Only one of the two tables is shown here for brevity since the difference between the values in the two tables is small.

Table 2.3: Battery lifetime estimates for NB-IoT with external PA [3GPb].

	Battery Life (years)		
Packet size, reporting interval	Coupling loss = 144 dB	Coupling loss = 154 dB	Coupling loss = 164 dB
50 bytes, 2 hours	22.8	11.5	2.7
200 bytes, 2 hours	18.8	6.3	1.7
50 bytes, 1 day	36.1	31.9	18.1
200 bytes, 1 day	35.1	26.8	13.4

Chapter 3
Methodology and Design

3.1 Scientific Method

The scientific method used in this thesis is a combination of different methodologies. We use both quantitative (e.g., confirming or dismissing hypothesis) and qualitative methods (e.g., observing a phenomenon for a device). Fundamental research techniques were used to gather information, acquire new knowledge and study the background material comprising this work, whereas the methodology used in the experiments is a combination of design science and lean development.

Design science has a strong focus on evaluation and iteration of research questions [Wie14]. The build-and-evaluate loop in the methodology is a good match for research in software engineering. The goal or the objective of the software engineering process is often widely known and documented, but how to achieve the goal in the best possible way is not a trivial task. In software engineering, there is much talk about lean development, test driven development and different approaches to breaking big tasks into smaller ones. There are many different reasons for this. The three most prominent reasons are more control (smaller tasks), evaluation/test-ability of the code (easier evaluation of small pieces of code) and iteration (faster to redo and improve a small task). The same applies to this thesis. The primary goal of any research, at least in computer science, is to solve a problem that has yet to be solved or to provide a more effective solution to an existing problem. To do so, we need to know what the problem is and know what a solution might be. We do not yet need to know how to find the solution, nor if our proposed solution is the best one. The design science methodology provides guidelines for conducting an iterative research process. The build-and-evaluate loop either brings us closer to a possible solution or further away. Either way, we gained knowledge of the problem in question.

Design science also embodies concepts from other research methodologies such as empirical research, but in a way more suited for experiments in computer science. The methodology enables both quantitative and qualitative research through knowledge

building and evaluation of the research results.

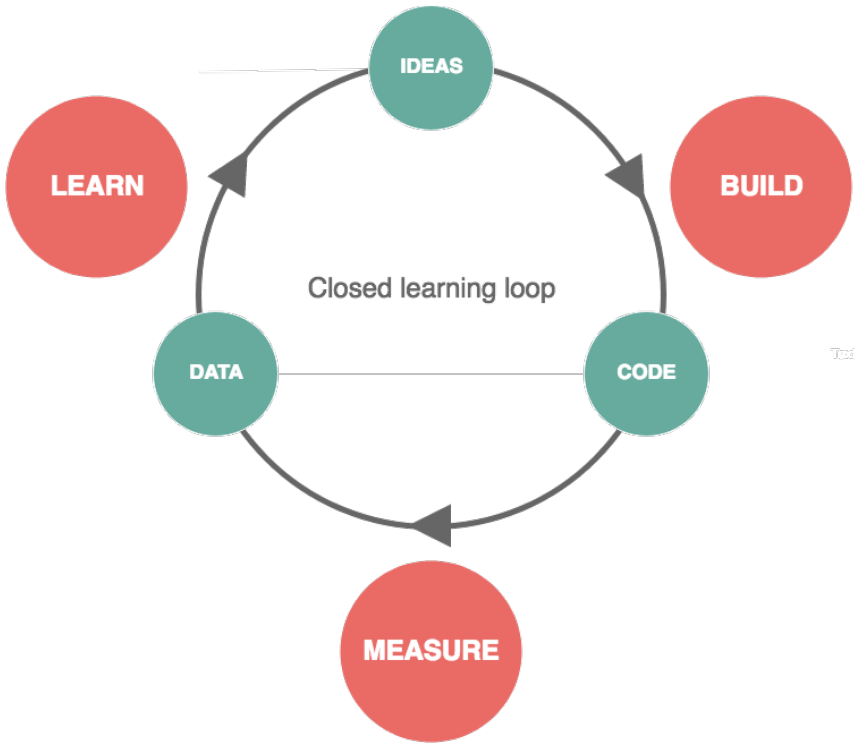

Figure 3.1: Closed learning loop. Adapted from [Spa16]

Figure 3.1 depicts the lean development process used in the experiments. The process is iterative and generally starts with some knowledge or assumptions of the artifacts that are to be studied. Knowledge or assumptions form ideas that we need to test. The realization of the ideas happens in the build phase. The code from the build phase forms the basis for the measure-phase. From the data generated in the measurements, we get new knowledge of the studied artifacts. The iterative loop proceeds as long as an answer to our questions is not found or, our solution could be improved. We used this iterative process throughout the experiments, both in understanding how the hardware and software of the devices could be used to perform the experiment themselves and to improve the experiment once a working solution was up and running.

3.2 Equipment and Test Architecture: *What, how and why*

This section describes the different hardware and software used in experiments in this thesis. The section aims at describing what each equipment is for, how we use it and why. As a general note, the acquiring of the equipment used for experiments have mostly been based on what is already available in the lab. Where it is appropriate, we discuss the potential limitations of the chosen equipment. Figure 3.2 shows the test architecture and the equipment used in the experiments in this thesis.

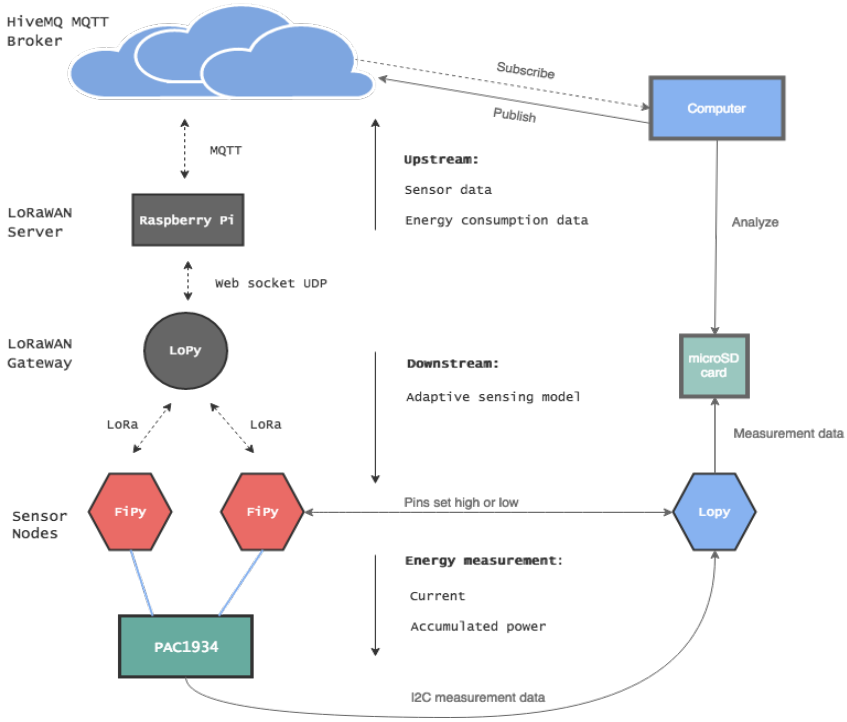

Figure 3.2: Test architecture

3.2.1 Pycom Fipy

The Fipy (red hexagons in figure 3.2) is the subject of the experiments. The Fipy runs all Micropython [1] code to connect to the LoRaWAN network, send and receive messages. The board comes equipped with LoRa and LTE radios making it possible to connect to both NB-IoT and LoRaWAN networks. We chose the Fipy because of the ease of writing, uploading and managing code on the device. Since Pycom

[1] MicroPython is a lean and efficient implementation of the Python 3 programming language that includes a small subset of the Python standard library and is optimized to run on microcontrollers and in constrained environments.

uses Micropython as the API programming language, prototyping on the device is fast. Powering the Fipy is done in two different ways throughout the experiments: USB and Vin (see Chapter 4). Some experiments use a DS18X20 temperature sensor connected to the Fipy for prototyping and testing of an adaptive sensing model. In other experiments, for instance, when testing for linearity of packet sizes against energy consumption, random bytes were sent. The LoRaWAN setup for the Fipy was done using the Pycom LoRa API with the following parameters:

Mode: LoRa.LORAWAN
Region: LoRa.EU868
Device class: LoRa.CLASS_A
Transmit Power: 14 (2-14 possible)
Bandwidth: LoRa.BW_125KHZ
Spreading Factor (SF): 7
Data Rate: 5 (5470 bps, see Table 2.1)

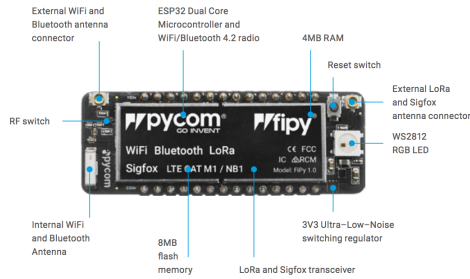

Figure 3.3: Pycom Fipy [Pycd]

In mode *LORAWAN* the only configurable parameter is device class (and three other parameters not applicable to the experiments in this thesis). The above parameters, which are the default values for the Pycom LoRa implementation, are therefore used in all experiments. We use device class A in all experiments.

3.2.2 Pycom Expansion Board

The Pycom Expansion Board is a development board acting as a shield for both the Fipy and the Lopy. The Expansion Board enables additional hardware features for the different modules connected to it, such as battery power, USB power and serial communication, microSD card storage as well as easy prototyping with jumper cable connections. The serial communication over USB has been used extensively throughout the experiments, both for uploading code to the boards and to monitor logs from the devices at runtime.

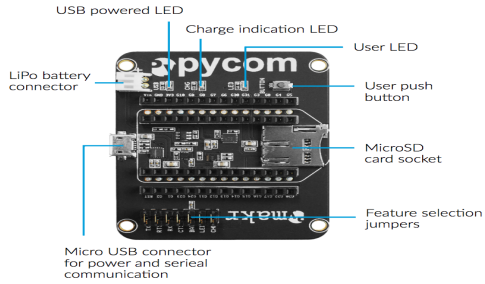

Figure 3.4: Pycom Expansion board [Pycb]

3.2.3 Sodaq NB-IoT Shield

A Sodaq NB-IoT shield connected to a Crowduino M0 development board has been used to try to establish a connection to the TeliaSonera NB-IoT network, with code from the TeliaSonera Github page [Tel18]. A successful connection has not been established, despite testing multiple different locations, firmware versions and

different versions of the code from GitHub. Engineers from TeliaSonera provided help and guidance in the process of trying to make it work, still without success. Telenor also provided resources to establish a connection towards their LTE network, also without success. Experiments with NB-IoT will therefore not be part of this thesis as a result of the networks still being in their infancy with regards to deployment.

3.2.4 Pycom Lopy

The Pycom Lopy served two different purposes in the experiments. One Lopy acted as the LoRaWAN Gateway forwarding LoRa packets between the end-devices and the LoRaWAN Server (grey circle in figure 3.2). The gateway was set up using the LoRaWAN Nano-Gateway implementation from Pycom [Pyca]. The Nano-Gateway establishes a connection to the LoRaWAN server by connecting to the Raspberry Pi WiFi hotspot (see section 3.2.6). The Gateway forwards any packet received in either direction without any filtering. The LoRaWAN Server filters out any packets not destined for devices belonging to the private network.

The reason for using the Nano-Gateway implementation was based on simplicity since it required only minor changes to network parameters to work. However, the fact that the Lopy only has one antenna makes it a far from ideal device acting as a Gateway. Having only one antenna makes sending and receiving messages at the same time impossible; thus collisions happen more often. Since the LoRaWAN network by default is public[2], and the gateway will receive incoming messages from all near-by devices, the chance of receiving packets from other devices while uplink or downlink traffic to our end-devices are happening, is highly possible. Class A devices also limit downlink messages to right after the reception of a successful uplink message. A collision happening while attempting to send a downlink message to a device, results in a lost message. In future experiments, different hardware for gateways should be considered.

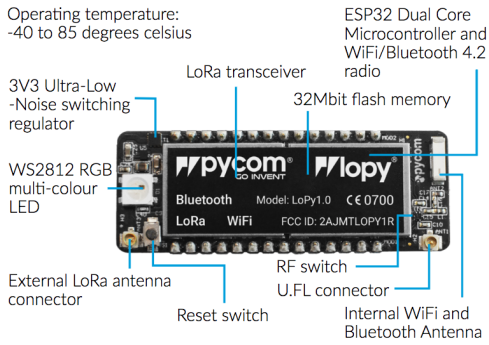

Figure 3.5: Pycom Lopy [Pyce]

Another Lopy device was used for conducting energy measurement experiments (blue hexagon in figure 3.2). This Lopy was both connected to the Fipy and the

[2]All access points (gateways) in LoRaWAN are by definition public and will receive all packets sent from devices in reach of the gateway. However, a private network allows for filtering out data from devices not registered by the server.

PAC1934 Power Monitor (see section 3.2.5). The connection to the PAC1934 was used to get higher precision measurements. Instead of visually reading data from the graphical user interface of the PAC1934, we established an I2C connection between the devices. The python API of the PAC1934 was used to get measurement data from the PAC1934 to the Lopy. See section 3.2.5 for further descriptions of this process. The connection between the Lopy and the Fipy was made to programmatically start and stop measurements at runtime, instead of doing this manually. A connection between two devices was made using three GPIO pins using jumper cables (see section 3.3.2 for a more thorough description of this process). The pins can be set to high (pin value of 1) or low (pin value of 0). Initially, all pins were instantiated to 0. The Fipy set individual pins to 1 when changing state.

3.2.5 Microchip PAC1934 DC Power Monitor Evaluation Board

The PAC1934 (green rectangle in figure 3.2) measures and monitors different energy consumption metrics such as current, power, and energy. In figure 3.6, the PAC1934 is shown with three boxes highlighting the different pins and features used in the experiments. Following is an explanation of the three boxes:

1) Jumper J1 in Box 1 measures input voltage from a connected device. The jumpers marked J6, and J7 needs to be placed to the left (shunt), making the current flow through the circuit and allowing measurements. When placed in the right position (as in figure 3.6) the channel is in demo mode. The shunt resistor R1 has a value of 0.004 Ω. R1 is R_{Sense} in equations that follows. In most experiments, the devices measured are treated as black boxes, meaning that we look at the device as a whole and not individual components or duty cycle states within the device. These experiments were conducted by connecting the PAC1934 to a desktop computer in the IoT-lab. Measurements were started and stopped manually by using the PAC1934 software. This method gives us the big picture and can answer questions regarding for instance accumulated energy of a device over time or power consumed in one duty cycle. However, looking at the device as a black box is in many cases not precise enough. In the top right corner of box 1, J30 provides a 5V output supply. The use of these jumpers was key to getting the deep sleep mode to work on the Fipy (see section 4.2).

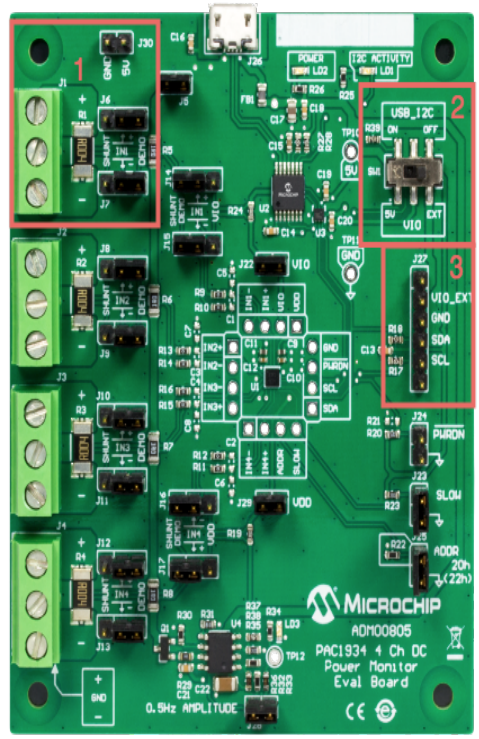

Figure 3.6: Microchip PAC1934 DC Power Monitor Evaluation Board [Mic]

2) The USB_I2C switch (seen in the top of figure 3.6) indicates whether the PAC1934 should write measurement data to the USB port or I2C. By using I2C, we gain more control over the experiments, can start and stop measurements programmatically at runtime, thus higher precision measurements are possible. See section 3.3.2 for a description of this process. For USB measurements we use the graphical user interface application on the desktop computer in the lab, while for I2C measurements, the PAC1934 Python API on a Lopy.

3) When the USB_I2C switch is set to I2C, GPIO pins marked J27 are used for gathering measurement data. The two bottom GPIOs (SCL and SDA) are handling data transfers, while GND and VIO_EXT need to be connected to GND, and a voltage output port on the device that should receive the data.

The PAC1934 reads the different electrical metrics and returns them in a binary representation to either the USB or I2C interfaces. When the USB_I2C switch is set to USB, the application running on the desktop computer in the lab handles all the calculations, returning the measurement in a human-readable format. When the switch is set to I2C, all calculations need to be performed manually on the binary data gathered by calling the Python API.

The PAC1934 datasheet specifies the formulas used by the application for the measurement calculations [Mic17]. The same formulas were used for calculations in experiments using the I2C interface of the PAC1934.

$$I_{max} = \frac{FSR}{R_{SENSE}} = \frac{100mV}{0.004\Omega} = 25A$$

where FSR indicates the Full-Scale Range of the measurements in bidirectional mode ($\pm$ 100mV) and R_{Sense} is the value of the input port resistor (R1 in box 1 in figure 3.6). I_{max} puts a bound on the highest current, and ultimately the range of currents (I_{Sense}) that can be measured by the PAC1934. The measurements in this thesis only consider unidirectional positive currents (FSR = 100mV).

$$I_{Sense} = FSC * \frac{V_{SENSE}}{2^{16}}$$

This means that we will be able to measure currents in the range of

$$I_{Sense} = 25A * \frac{V_{Sense}}{2^{16}} = \begin{cases} 25A * \frac{1}{2^{16}} = 381\mu A \\ 25A * \frac{2^{16}}{2^{16}} = 25A \end{cases}$$

where I_{Sense} is the actual bus current and V_{Sense} is the voltage value read from the V_{Sense} result registers on the PAC1934. See Algorithm 3.1 for the Python implementation of the formula for I_{Sense}.

The range when using 0.004 Ω as value for R_{Sense} could be a potential problem. Many devices do not draw more than approximately 50 μA when in deep sleep mode. We need to monitor the results when experimenting with deep sleep and find a way to validate the results. A potential fix to this problem is to introduce an external resistor in series with R_{Sense}. The introduction of an external resistor would make I_{max} smaller, ultimately leading to a shift in the range of currents that it is possible to measure. For instance, by introducing a 0.1 Ω resistor in series with R_{Sense}, we get I_{max} to be approximately 1 A, making it possible to measure currents down to $1/2^{16} = 15$ μA. This range should be more than low enough to capture deep sleep currents of any device. Experiments of the deep sleep mode on the Fipy needs to consider the possibility that the PAC1934 returns false values due to the range of I_{max}, and ultimately add external resistors to try to fix the problem.

Algorithm 3.1 Python implementation of formula for I_{Sense}

```
def getCurrent(registerValue):
    firstbyte = registerValue[0]
    secondbyte = registerValue[1]
    firstbyte = firstbyte<<8    #binary left shift 8 places
    Vsense = first | second     #concatinate the two bytes to
                                #one variable

    return 25 * (V_Sense/(2**16))

addr = i2c.scan()[0]                    #get the I2C address
data = i2c.readfrom_mem(addr, 19, 2) #read two bytes of data
                                #from current register

I_Sense = getCurrent(data)              #calculate the current
```

3.2.6 Raspberry Pi 3

The Raspberry Pi (grey rectangle in figure 3.2) serves two purposes. The first purpose is to act as a WiFi hotspot for the LoRaWAN gateway to connect to and send UDP packets through, towards the LoRaWAN Server.

The second purpose of the Raspberry Pi is to run the Lorawan-server itself. Lorawan-server is an Open-source LoRaWAN Server that integrates both the network server and the application server of the LoRaWAN protocol architecture [Got18]. Additionally, the server allows for making custom integrations with other web services through web sockets or MQTT. Lorawan-server provides a graphical user interface in the form of a website running locally on the Raspberry Pi, making it very easy to manage the network, adding gateways and end-devices, and monitor traffic. The reason for implementing the Lorawan-server as a private network, as opposed to using existing infrastructure and gateways (e.g., the one set up by Telenor on the roof of the Elektro building) is twofold: connectivity and control. When trying to connect to a LoRaWAN network from the IoT lab through *public* gate-

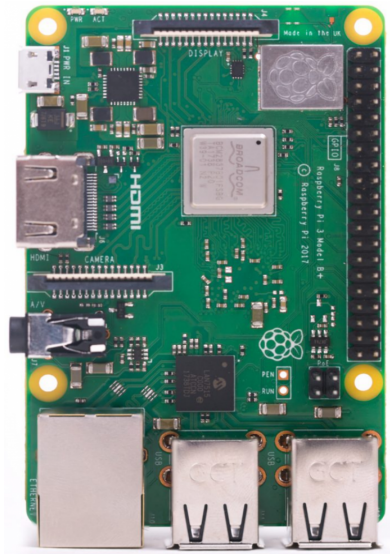

Figure 3.7: Raspberry Pi [Ras]

ways, establishing a connection was often not possible. However, for connection attempts outdoors or from locations with a more direct route towards the gateways, a connection was established immediately. The distance to the nearest antenna from the IoT lab was most likely the reason for these connectivity issues. The second reason for implementing a private LoRaWAN network is because of the added control. Registration of devices and gateways with the LoRaWAN server is easy, and most importantly packets can be tracked through every network element, making debugging and finding faulty elements easier.

3.2.7 HiveMQ MQTT Broker

The HiveMQ MQTT broker (depicted as clouds in figure 3.2) is an enterprise MQTT broker. However, unencrypted message transmissions are open for everyone to use free of charge [Hiv]. The reasons for using MQTT is because of its simplicity both in setup and usage. Additionally, the LoRaWAN Server supported easy setup of an MQTT integration, by just specifying server address, port, and topics for publishing, and subscribing. MQTT was used to send messages from, and to end-devices in a standard MQTT setup (see figure 3.8). Uplink messages from end-devices (temperature measurements or random bytes) were forwarded to a subscribing client, by the LoRaWAN server publishing the message to the topic. For downlink traffic towards end-devices, the LoRaWAN server subscribes to an MQTT topic. For messages published to this topic (model data), the LoRaWAN server forwards the messages to the end-device.

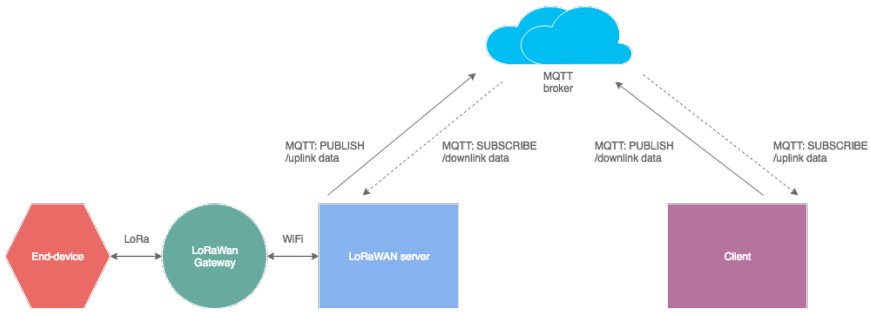

Figure 3.8: MQTT test architecture

3.2.8 Computer

A desktop computer running Windows 10 on the IoT lab served two purposes: To start and monitor the LoRaWAN gateway running on the Pycom Lopy and to run the graphical user interface of the Microchip PAC1934. We conducted the black box experiments of the Fipy on the desktop computer. The monitoring of the

gateway Lopy was necessary since the gateway often crashed or stopped working. By monitoring the gateway's behavior through the USB serial communication from the Lopy, problems were rapidly discovered and fixed. One particular error happened quite often, and the cause has yet to be found. However, a simple reboot of the Lopy fixed the error, so we did not investigate this further. Another computer (blue rectangle in figure 3.2) was used to write and upload Micropython code to the devices as well as interacting with the MQTT broker through publishing and subscribing to message topics.

3.3 Data Collection Methods

Experiments use two data collection designs: black box testing and a novel state machine design. The following section describes these two designs, why and how we use them. Note that it would be possible to implement a solution using only the device that is the subject of the tests (except for the PAC1934, monitoring the energy consumption). However, this would mean processing the measurement data internally on the same device that is the test subject, potentially corrupting the measurements. This approach is therefore not used in any of the experiments.

3.3.1 Black Box Experiment Design

A black box test design tests functional or non-functional properties of a component or system without reference to its internal structure [Fun17]. It is often used by software designers when the internal structures of the software are unknown, or when testing the results of running the software is more important than how the software produces these results. Black box tests are used in this thesis for instance in testing hypothesis (e.g., average idle current in section 4.1) and doing accumulative tests (e.g., testing for linearity of energy consumption against packet size, as in section 4.3. Figure 3.9 shows the black box design used in the experiments. A USB cable running between the *Test-subject* and the *Observer* was split, and the *Tester* is responsible for measuring the voltage drop across the USB cable. We analyzed the results of black box tests through visual inspection of graphs produced by the PAC1934 software running on a computer in the lab (see section 3.2.5).

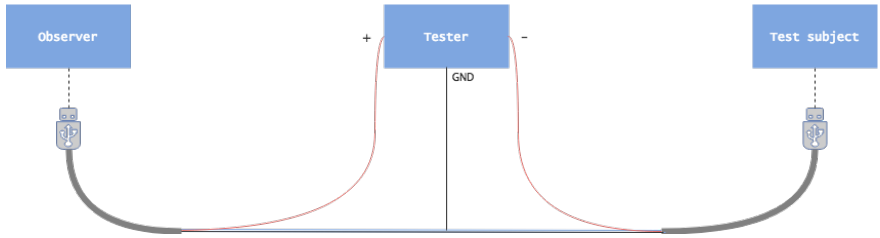

Figure 3.9: Black Box Test Design

3.3.2 A Novel Experiment Design for Higher Precision Measurements

Black box experiment design is good for getting the big picture, and to confirm or dismiss assumptions we may have about different phenomena. However, since they mask internal details, they give little or no information about specifics. One approach to getting more information of internal structures (e.g., what it cost to send a data packet) would be to conduct comparative measurements. First, we measure the system in a steady state configuration (e.g., while the system is idle and not sending packets) to get a baseline comparison. Then we configure the system to execute the artifact we want to study while we measure the effects. The two measurements are compared, and the deviation between the two is our result. While being an intuitive and quite straightforward process, in many cases this still is not precise enough. We want to focus only on one specific phenomenon and remove everything that is not a part of this. This, however, is not necessarily easy. We need to start the measurement at the exact moment a device enters a given state of the duty cycle and, terminate the process when exiting the state. However, measurements need to be conducted from the outside (from another device) both to eliminate the possibility of the device consuming more energy because it measures itself and, to have more control over the process. To achieve this, we came up with a state machine configuration for the experiments in this thesis. The general idea is that the device subject of the experiment sends events when entering and exiting different states. The observing device responsible for conducting the measurements listens to and captures these events. The observer starts and stops the measurements exactly when receiving the different events from the measuring device. This design allows higher precision measurements that programmatically can be started and stopped without needing any manual labor.

Figure 3.10 shows the experiment design as a state machine configuration. When the Fipy turns on, the Lopy waits for the booting process to complete. When completed, the Fipy sets the first pin to 1. Next, the Fipy runs all initialization code (e.g., connecting to the LoRaWAN network) and prepares for sending packets. The Lopy

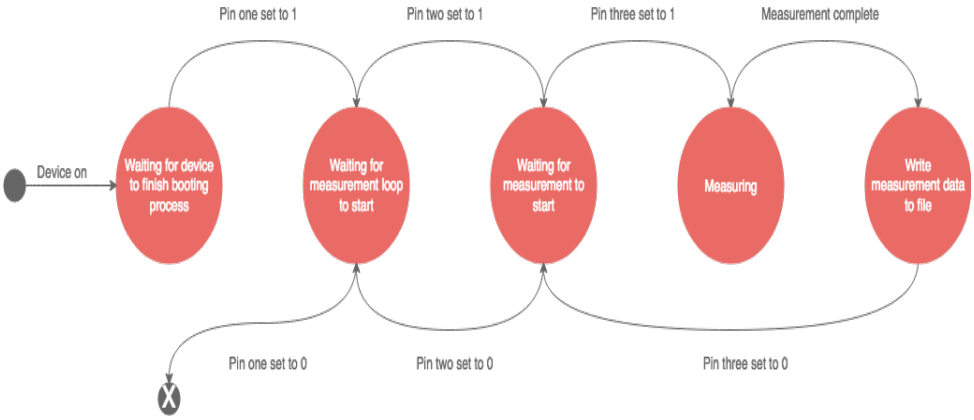

Figure 3.10: State machine of measurement process

waits for the measurement loop to start. When everything is set up, the Fipy sets its second pin to 1. Now the Lopy is ready to measure and is waiting for an individual measurement to start. When the Fipy sets its third pin to 1, it enters the state that is the subject of the experiment (e.g., sending or receiving packets). The Lopy then starts measuring some energy consumption metric (e.g., current) by calling the python API of the PAC1934. The Lopy continues to measure until the third pin is 0. The collected data is then written to file on an SD card connected to the Lopy. Finally, the Lopy returns to waiting for a new measurement to start. This measurement process is looped a specified number of times until the second pin is 0. The first pin is set to 0 to finish the program. As opposed to the "black box" experiment design, this design enables higher precision measurements targeted at specific states of the duty cycle.

The state machine could be implemented with two and not three pins: one for controlling all initialization and the second for handling each measurement. However, in cases where we wanted to loop the process of measurements (e.g., sending larger, and larger packets) the extra pin helped handle this.

In algorithm 3.2 and 3.3 simplified versions of the Python code for the test-subject and observer are presented. The code shows the measurement process and the communication between the two entities and has been simplified to highlight the general idea more than the actual implementation.

Algorithm 3.2 Simplified Python code for test subject

```
p_one, p_two, p_three = 0 #initialize pins to low
def togglePin(pin):
    pin.toggle() #change value of pin low/high (0 / 1)

#Booting process finished:
togglePin(p_one) #p_one = 1

#All initializing (e.g., radios) finished:

togglePin(p_two) #p_two = 1

for (i in range(numberOfTimesToMeasure)):
    togglePin(p_three) #start measurement
    sendPacket()
    togglePin(p_three) #stop measurement
    sleep(someTime)

togglePin(p_two) #All measurements finished

togglePin(p_one) #terminate measurement process
```

Algorithm 3.3 Simplified Python code for observer

```
#booting finished -> p_one = 1

timer = Timer.Chrono()

while(p_one == 1):
  while(p_two == 0):
      sleep(someTime) #waiting for p_two to become 1
  while(p_two == 1):
      while(p_three == 0):
          sleep(someTime) #waiting for p_three to become 1
      while(p_three == 1):
          timer.start()
          data = readDataFromPAC1934()
          current = calculateCurrent(data)
      timer.stop()
      timeSpent = timer.read()
      writeToFile(timeSpent, current)
      break
  break
```

Chapter 4
Experiments

The following chapter presents experiments of energy consumption of LPWAN devices. Figure 3.2 in Chapter 3 showed the architecture of the experiments and the communication between the different devices. Section 3.2 described what purpose each device serves, why and how the equipment is used. Because of difficulties with establishing a connection to an NB-IoT network, only LoRaWAN devices will be the subject of the experiments.

To be able to build an adaptive sensing application some essential elements need to be in place. First of all, we need a device that is in a state where it consumes as little energy as possible. One aspect of achieving low energy consumption is making sure that the device has turned off all unwanted peripherals and radios. Additionally, we need a device that sleeps as much as possible and consumes minimal energy doing so, meaning support for deep sleep mode is essential. In all the following experiments, we use standard settings in the LoRaWAN protocol, unless specified otherwise. A lab setting with just a small distance of around 4-5 meters between the gateway and the end-device were used. However, since the devices used the maximum transmit power (default for Pycom devices in LoRaWAN mode), one could argue that the results in many ways could apply to a more real-world setting outside of the lab.

From the problem description we have the following questions:

- What is the baseline energy consumption for an LPWAN device?
- Can we calculate the cost of sending a byte? Are ten bytes ten times more expensive?

To try to answer these questions, we conducted a series of experiments. Experiment 1 investigates the average idle current of the Fipy device. This experiment is needed to get an understanding of how the device operates when no program is running.

Experiment 2 investigates the deep sleep mode on the Fipy, which is the most crucial state for achieving low energy consumption of end-devices over time. All proceeding experiments seek to investigate energy consumption characteristics of the Fipy, and how to reduce them.

4.1 Average Idle Current of the Fipy

Hypothesis: The average current consumed by the FiPy in idle state with no application running is per the Pycom Fipy specifications 62.7 mA [Pycd].

Method: In this experiment, we will investigate the average current consumption of the FiPy in an idle state with all radios turned off and no application running. The experiment will be conducted using the *black box* method described in section 3.3.1. The current will be measured through the PAC1934 and visually inspected through the board's graphical user interface running on a PC in the lab. The FiPy is attached to a Pycom Expansion board connected to a MAC with a USB cable. The USB cable has been modified to enable measuring the current drawn through it. Since we at this point only are interested in measuring the current for the board as a whole, and not individual components on the board itself, such a black box setup is sufficient. Additionally, since the goal of the experiment is to confirm that the board is operating as the specifications indicate, visual inspections are sufficient, and no exact measurements are needed. The positive, negative and ground terminals of the USB cable from the MAC, was connected to the respective terminals of the PAC1934. Additionally, a grounding cable connected the PAC1934 and the Fipy. A picture of the setup is seen in figure 4.1.

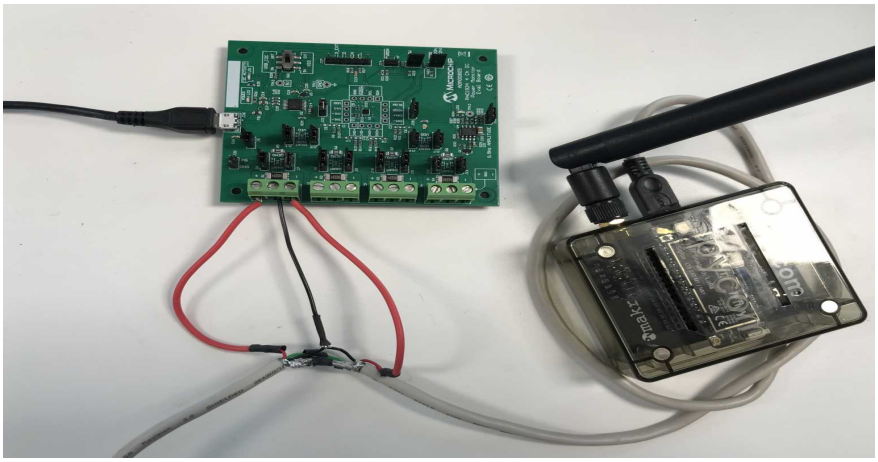

Figure 4.1: Experiment 1 setup

Results: Confirming or dismissing the hypothesis should not be that hard, but it turned out to be an actual design problem requiring a substantial amount of digging through specifications, device documentation, and forums. Pycom has designed the Fipy to have all radios on by default (except LoRa and SigFox requiring instantiation when needed). By default, the device, therefore, has both WiFi, Bluetooth, LTE, LED light and an FTP server active. By analyzing the current consumption in this state, we saw that the device drew a current of about 260 mA on average while idle. This result was very unexpected and is an exceptionally high current for an IoT device. Per the specification of the board, the idle (no radios) current consumption should be on average 62.7 mA. Instantiating all radios and then turning them off lead to the current dropping from around 260 mA to around 70 mA, which is per the specification[1].

At the time of writing, the design leading to this high current has yet to be fixed, and to achieve the idle (no radios) current of about 70 mA, manually turning off all radios are required. Since fixing this design in the firmware of the device is out of the scope of this thesis, the process of deinitializing all radios was implemented in the code in the method turnOffAllUnwantedRadios() (see algorithm 4.1) by using methods exposed through the Pycom API. The code snippet in algorithm 4.1 also deinitializes the server used for establishing an FTP connection on the board. The effects of this are minor, but since Server is in the network package of the device, this was also turned off.

Algorithm 4.1 Python method: Turning of all radios

```
def turnOffAllUnwantedRadios():
    wlan = network.WLAN()
    wlan.deinit()
    server = network.Server()
    server.deinit()
    bluetooth = network.Bluetooth()
    bluetooth.deinit()
    lte = network.LTE()
    lte.deinit()
```

4.2 Average Deep Sleep Current of the Fipy

Pycom has for a long time struggled with implementing deep sleep on their devices. There have been reports of devices consuming currents in the range of 50-100 mA

[1]It should also be noted that the LTE module on the Fipy will not turn off unless there is a sim card inserted into the sim card slot of the device

in a deep sleep. However, the company states that the problem is only affecting first generation devices such as the Lopy and not newer devices like the FiPy. The company has released a deep sleep shield that connects to the devices, to fix these deep sleep problems. The shield is responsible for handling going into, and waking up from the deep sleep, either by a timer expiring, or some interrupt has happened (e.g., some pin put high, or user pressed reset button). According to Pycom, this shield is unnecessary on the FiPy. To enter deep sleep, the only thing needed is a call to machine.deepsleep(time). The parameter *time* (in milliseconds) could be left out, leading to the device sleeping indefinitely until some external interrupt event happens. Given the problems, Pycom has had with deep sleep and the fact that low energy consumption in deep sleep mode is the most crucial aspect to achieving long device lifetime, we investigate deep sleep on the Fipy here.

Hypothesis: The average Fipy deep sleep current is in the μA range.

Method: This experiment uses the *Black box* design method, as in experiment 1.

Results: When entering the deep sleep state on the FiPy from the idle (no radios) state, the device current consumption rises from about 70 mA to over 200mA, before dropping to a steady current of around 170 mA for the duration of the time in deep sleep mode. When waking up from a deep sleep, the device again turns on all radios. In figure 4.2 a screen dump of this behavior taken from the graphical interface of the PAC1934 can be seen. The drop after the last peek is a call to turnOffAllUnwantedRadios() (see algorithm 4.1).

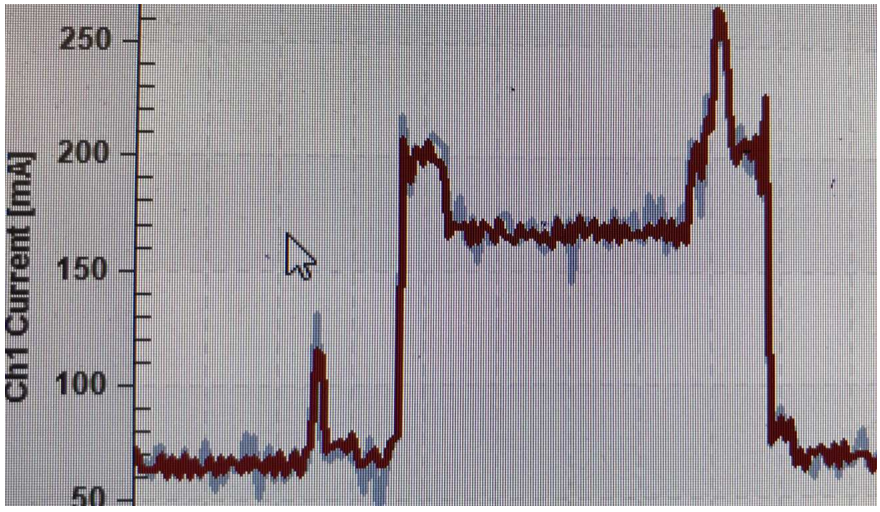

Figure 4.2: Deep sleep current

The observed deep sleep current is massively exceeding the expectations that the current consumption should be in the μA range. There could be multiple different reasons as to why the current consumption is this high. Pycom might have implemented the functionality wrongly, maybe rushing out the code to try to fix the deep sleep problem explained earlier. Another possibility is that our test setup is not capable of measuring currents that small, and the program running on the PAC1934 returns false values, as mentioned at the end of section 3.2.5. However, when posting about the problem on the Pycom forum, several other developers were having the same issue. After talking directly with Pycom engineers, it became clear that two things were not working as expected: deep sleep and the device *preparing for* deep sleep, related to the 170 mA and 200 mA currents respectively.

4.2.1 Fixing Deep Sleep

We found a partial fix for the deep sleep problem in the Pycom API documentation [Pycc]. For devices with an LTE modem, the RTS (request to send) and CTS (clear to send) jumpers need to be removed, since the LTE module uses them. This information was not provided in an easily accessible way by Pycom but hidden at the bottom of the documentation of the LTE module on the Pycom website. No information was made available under the documentation for deep sleep, which meant finding the cause took time. When removing the RTS/CTS jumpers, the current dropped to about 30 mA. A Pycom engineer provided tips through the Pycom forums on further reducing the deep sleep current. When powering the device through USB, the Expansion board draws extra current for the LED light and the USB-serial chip. A fix to this is to power the device through the Vin port with a jumper cable. The PAC1934 has an output voltage of 5 Volts available on the board (see box one in figure 3.6). By connecting a jumper cable from this voltage supply and adding a ground between the two devices, the deep sleep current dropped to a steady 12 mA. Given the potential problem with the measurement range of the PAC1934, we could not be 100 percent sure that the measured current was correct. A multimeter was therefore used to confirm the measurement of the PAC1934.

The deep sleep current is much higher than expected and practically too high to be used for any real-world deployment in LPWAN. However, the most important in our case is to know what the value is, not only that the value is in the μA range. Given the massive decrease in current draw and that further lowering it would require modifications to the firmware of the device, this is the lowest current currently available for the device.

4.2.2 Preparing for Deep Sleep

Figure 4.3: Fipy duty cycle. Peak shows device preparing for deep sleep

In Figure 4.3 we see a duty cycle of the Fipy device where the device wakes up, sends a packet, waits for a response in the two reception windows (RX1 and RX2) and finally goes to sleep. Right before the device goes to sleep the energy consumption of the device more than doubles for some seconds. This behavior was found to be a bug in the device after presenting the graphs of the duty cycle to Pycom engineers. The behavior may be the device preparing to enter deep sleep, according to the engineers. Pycom was not aware of this behavior, but was thankful for the contribution and are currently looking into the cause. Accumulative tests of energy will, therefore, be affected by this bug, since the peak in energy consumption happens every time right before the device enters deep sleep.

4.3 Relationship Between Payload Size and Energy Consumption

In this experiment, the aim was to investigate the correlation between the amount of data sent and the current drawn from the board. The goal was to try to find out if there was some linear correlation such that sending ten bytes draws ten times more current than sending one byte.

Research question: What is the relationship between packet payload size and energy consumption of end-devices?

Method: Manual black box tests were conducted to try to find if there is a linear relationship between energy consumption and payload size of transmitted packets. The method involved testing nine different combinations of payload sizes, each three

times. For the different payloads random bytes of different length were generated. This was done using os.urandom(*numberOfbytes*). This gives a random byte string of length *numberOfBytes*. The randomness was introduced to prevent potentially corrupting the results if caching somewhere in the devices were enabled. There was no indication in any material read that this was the case, but the method was used anyway as a means of gaining more control over the experiment. Each test was run for five minutes. The code written on the Fipy included blinking of the LED light two times to indicate that the test was about to start. When the last LED turned off, the device started transmitting packets. At this exact instance the *Start Acquisition* button (see figure 4.4) in the PAC1934 application running on the computer in the lab was pressed, making the application start recording the data. At the same time, we started a stopwatch on a mobile device. When the test had run exactly five minutes, the PAC1934 application was stopped and accumulated values for power was written down from the application. All tests consisted of sending 43 packets from the Fipy over the five minute period. When the device was not sending it was waiting to send by calling the Python method time.sleep(). The Fipy was using the standard LoRaWAN setting: data rate of 5, which corresponds to 125 kHz with a spreading factor of 7. This results in an indicative physical bit rate of 5470 bits/s. The method used is time consuming, potentially prone to errors and high variability of result due to manually starting and stopping the experiment. The PAC1934 application samples the registers for values 1024/min, and all tests consisted of more or less 300000 samples, meaning that the variance in the results could be low enough to at least get an indication of the relationship between the payload size and energy consumption of the device.

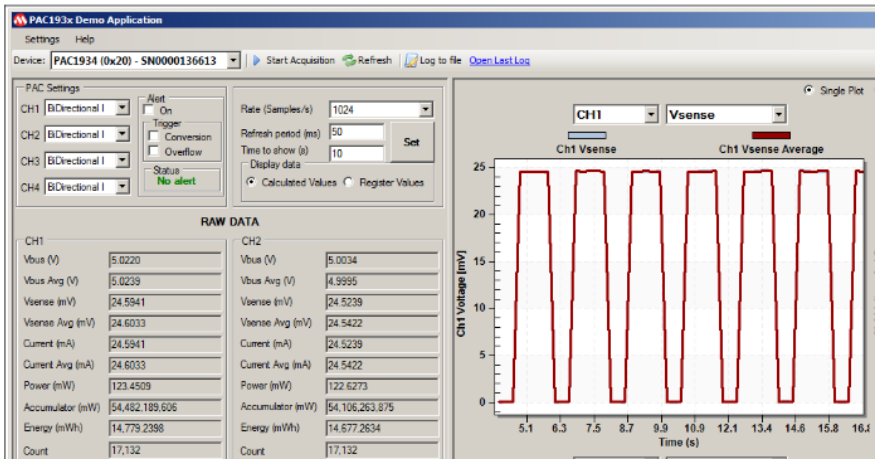

Figure 4.4: PAC1934 Graphical User Interface. The image is taken from [Mic17] and shows the application running in demo mode generating default waveforms.

Results: It is not easy finding the actual value of sending a byte. The reason for this is that for most of the time running the tests, the system is idle. The device sends 43 packets, and data from the tests show that sending a packet takes on average approximately 35 ms. One could divide the total experiment time into 35 ms chunks, finding the average accumulated power in each chunk. However, this would be a very unprecise method of finding the cost of each payload. Additionally, the 35 ms transmission time could be the time it takes for the application to send the packet through a socket and down the LoRaWAN protocol stack towards the radio. If the LoRaWAN protocol stack or the firmware on the device applies some form of internal threading, the time it takes to perform the packet transmission might be substantially higher than what is detectable from the application layer. Further analysis needs to be performed to use a method where we decide if setting a timer before the send method and reading the timer value after the method returns, is accurate enough to be used for calculations. This is discussed further in section 4.4.

In this experiment, the actual cost of sending a byte in itself is not that interesting. However, what happens when we increase the amount of data to send is. Figure 4.1 shows the results from the experiment.

Table 4.1: Manual test of linearity of payload size of sent packets

Payload Size (Bytes)	Accumulated Power (mW)	Energy (mJ)
0	99637.0	332.1
1	100239.9	334,1
2	100300.1	334.3
8	100491.0	335.0
16	105593.6	335.3
64	101162.0	337.2
128	102047.5	340.2
192	102998.6	343.3
242	103432.4	344.8

The measurements are based on an accumulation of the power over a period of five minutes by sampling the power register 1024 times per second. This is handled by the application (see *Accumulator (mW) under Raw Data in figure 4.4*). Therefore, the results have no direct indication of what it costs to send the different payloads. However, the only variable changing in the experiments is the payload length. By dividing the accumulated power by 300 seconds, we get the average accumulated power for one second. We first calculate the accumulated power when not sending any packets, and then subtract this value from all the measurements for the different payload sizes. From this, we get a good indication of the relationship we seek to find.

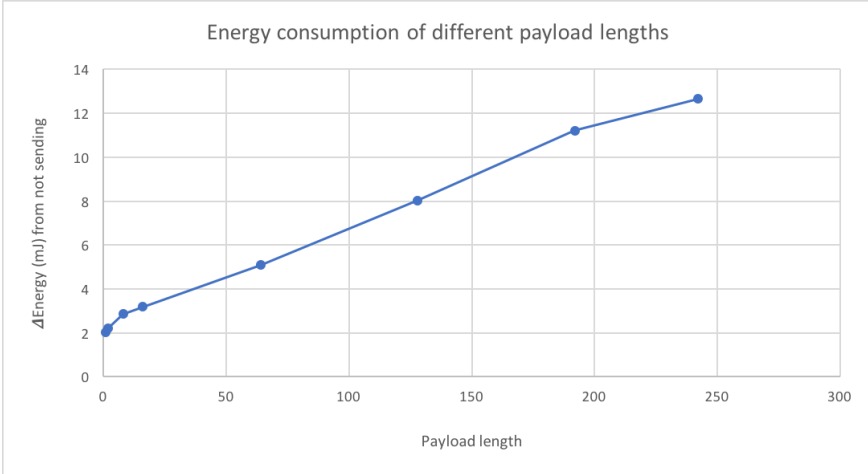

Figure 4.5: Relationship between payload size and accumulated power per second, when normalizing the power against the accumulated power of not sending

Figure 4.5 shows a clear linear relationship between the accumulation power per second and the different payload sizes. However, there is no easy way to find the actual cost of sending a byte or the actual difference between sending one byte and ten bytes. The test method also involved a substantial amount of manual labor, which is both time-consuming and prone to error. Therefore, we sought to refine this experiment by using the novel design method based on pins and a state machine configuration.

4.4 Relationship Between Payload Size and Energy Consumption (refined)

Research question: What is the relationship between packet payload size and energy consumption of end-devices?

Method: This experiment seeks to improve the results seen from the experiment in section 4.3, by using the state machine design presented in section 3.3.2. A figure of the hardware setup is seen in figure 4.6. The experiment consisted of sending 100 packets for ten different increments of payload sizes and writing the data to a CSV file for analysis. In section 4.3 we raised a concern about the potential inaccuracy introduced when measuring the energy consumed in the send method of the Pycom API. Before doing the tests, this concern was dismissed by sampling the power register of the PAC1934 right after the send method terminated. We saw a substantial drop

in the power consumption leading to a conclusion that the proposed method would provide us with the accuracy needed. For post-processing of data, the average and variance of all samples for a given payload size was calculated using functions in Excel. The graph and trend line in figure 4.7 were also generated using excel.

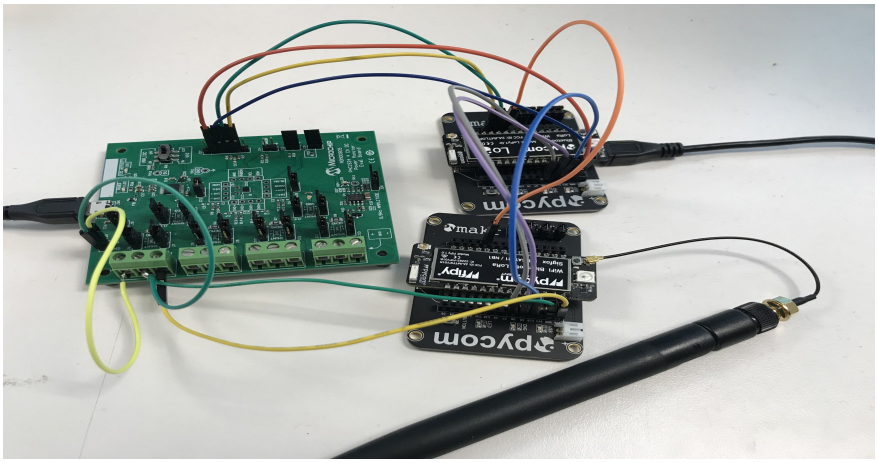

Figure 4.6: Experiment setup using jumper cables between pins

Results: Table 4.2 shows the averaged results for each payload size. Except for the values for 1-byte and 2-byte packets, we see an increase in energy when increasing payload sizes. The variance in the 100 samples is in the range of 0.2 to 1.0 μJ for all payload sizes, leading to high confidence in the accuracy of the measured values.

Table 4.2: Average energy consumption and variance of 100 sent packets of different payload sizes

Payload Size (Bytes)	Avg Energy (mJ)	Variance (mJ)
1	0.6152	0.0002
4	0.6140	0.0003
8	0.6158	0.0004
16	0.6208	0.0003
32	0.6283	0.0005
64	0.6400	0.0006
90	0.6513	0.0010
128	0.6656	0.0006
192	0.6973	0.0007
222	0.7077	0.0006

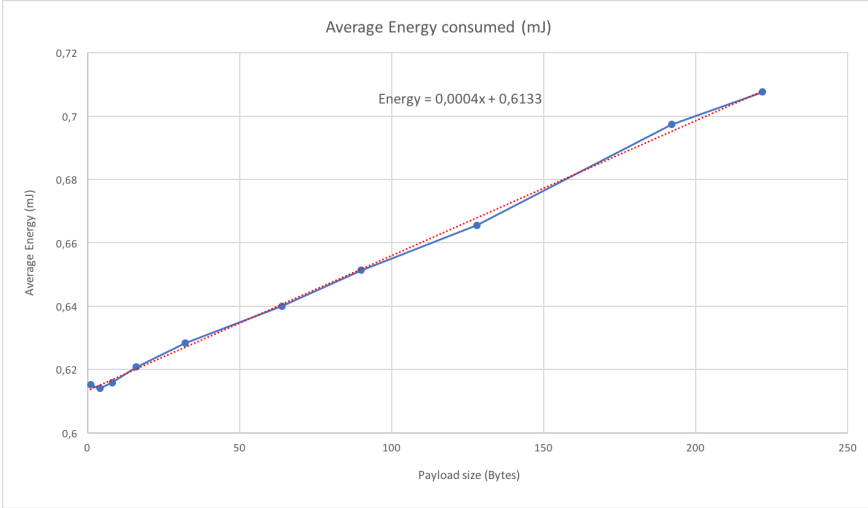

Figure 4.7: Average energy consumption for different payload sizes

From the formula for the trend line in Figure 4.7 we have that the energy consumption when sending, is a product of the packet size, plus some constant value. Therefore, there is always some start-up cost regardless of the packet size. So if we were to send two packets of ten Bytes, each transmission means consuming:

$$Energy = 2 * (0.004 * 10 + 0.6133) = 1.3066\, mJ$$

If we could store and hold the sensor value until the next transmission window, the energy consumed could be

$$Energy = 0.004 * 20 + 0.6133 = 0.6933\, mJ$$

See section 5.1 for a discussion of the potential implications of this.

4.5 Relationship Between Down-link Packet Payload Size and Energy Consumption

Research question: What is the relationship between downlink packet payload size and energy consumption of end-devices?

Method: This experiment uses the same method as in the experiment in section 4.3. Three different packet payloads (8, 16 and 23 bytes) were tested three times to get an indication of the effects on energy consumption when increasing down-link packet size. The experiment was only conducted three times with three different payload

sizes due to it being difficult to execute in a precise manner. The experiment used the same process as in section 4.3, by pressing the *Start Acquisition* button in the PAC1934 application and by starting a stop-watch. To send downlink packets to the end-device, the MQTT broker (see section 3.2.7) was used. A Python program running on a mac published random bytes of given payload length to a topic in which the LoRaWAN-server was subscribing. The LoRaWAN-server then forwarded the message to the end-device. The experiment was hard to execute in a precise manner due to packet collisions and the limited time frame in which end-devices accept incoming packets. Class-A LoRaWAN devices open up two short reception windows right after sending a packet. These windows gave us a short time slot to publish messages towards the MQTT broker, forward the packets from the broker to the Lorawan-server and finally send them the end-device. Most of the time this failed. In some cases, the Lorawan-server, or the gateway seemed to cache the packet, and forwarded it in the next reception window, other times this was not the case. Since the Lorawan-server also received (and then discarded) packet from other LoRaWAN devices, the reason that so few packets were received can be because of collisions of packets, or packets being dropped due to the system being half-duplex. However, no attempts were made to investigate if this was the case. We tested each payload length between five and ten times, but the results presented in this section consists of three successful runs of sending ten down-link packets for each of the three different payload lengths. The packets were confirmed successfully received by the end-device by monitoring the device's serial port, where the device printed incoming data.

Results: Although the sample size in this experiment is too small to give any complete answers, some indication of the relationship concerning our sample is clear. Figure 4.8 shows the amount of extra energy the device consumes when receiving a packet of a given payload length, compared to receiving a packet. The calculation consisted of subtracting the power value obtained in the experiment in section 4.3 from the accumulated power when sending. The difference between the two is then what accounts for receiving a packet. Based on our sample there is a linear increase in energy consumption when receiving larger packets. Additionally, there seems to be a start-up cost, which could lead to that accumulating downlink packets destined for an end-device could save energy.

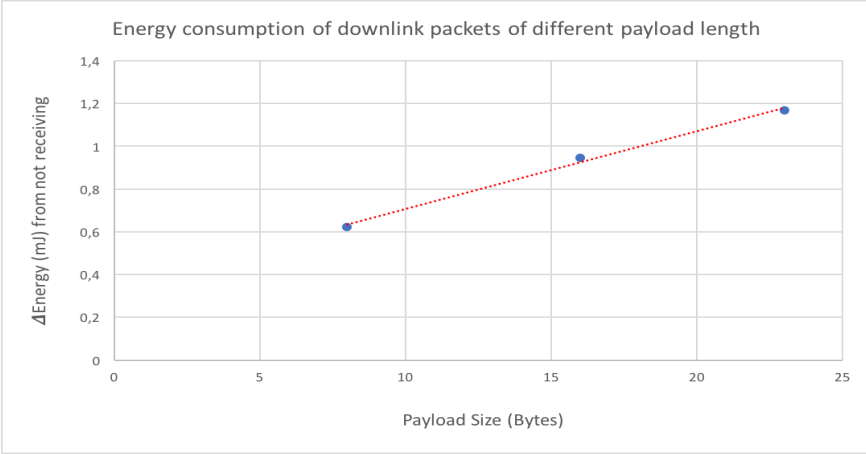

Figure 4.8: Average energy consumption for different down-link payload sizes

Attempts were made to refine the experiment process as done in section 4.4. Another pin was added to the state machine configuration handling if the packet were received or not. The MQTT application responsible for publishing packets of random bytes were refined to push out packets at a fixed interval. However, the end-device only received a small fraction of the packets. Neither increasing the publishing rate (sending out packets more often) or decreasing the rate helped in raising the reception rate. The attempts to refine the experiment was therefore unsuccessful. In an attempt to try to receive more packets, we could have configured the end-device as a Class-C device. The device then would have the radio on at all times, making receiving packets possibly much easier. However, having the radio on at all times would affect the overall energy consumption of the device. Therefore, experiments with this configuration should refine the state machine configuration to only look at the reception state. It is unclear to what extent such a set-up would work, but future experiments should look into this.

4.6 Generalizing Casals et al.'s Energy Consumption Model

Research question: Is Casals et al.'s energy consumption model [CMVG17], described in section 2.6.1, general enough to be used to model the energy consumption on the Fipy?

Method: In this experiment we measure the aggregated energy consumption over a period and compare it with the analytical solution provided by Casals et al. in [CMVG17]. The experiment uses a simplified version of the state machine experiment design, using only two pins: one for handling each aggregated measurement and one

for handling the entire measurement process. We also measured time, and average currents during the process.

Table 4.3: Time and current consumption for a LoRaWAN duty-cycle, based on parameters used in *Modeling the Energy Performance of LoRaWAN* [CMVG17].

State	**Duration (ms)**	**Current consumption (mA)**
Wake up	168.2	22.1
Radio Preparation	83.8	13.3
Transmission	399.6	83.0
Wait 1st window	983.3	27.0
1st receive window	12.29	38.1
Wait 2nd window	987.71	27.1
2nd receive window	33.0	35.0
Radio off	147.4	13.2
Post-processing	268.0	21.0
Turn-off sequence	38.6	13.3
Sleep	*based on* T_{notif}	45 * 10^{-3}

Results: When we calculated the Energy cost using the formulas from [CMVG17], with the parameters defined in table 4.3, we get an Energy consumption of *0.510 mJ*. The calculation was made using a T_{notif} duty cycle time of 50.160 seconds[2], a deep sleep current of 45 μA, and the maximum payload of 242 bytes. The analytic solution for the average current for this calculation is 2.03 mA. When measuring the energy consumed by the Fipy for the same interval, with the same maximum payload size we get *3.357 mJ*, with an average current of 13.4 mA. Hence, the Fipy consumes more than *13 times* the energy of the analytical solution. From experiments with the Fipy, the average deep sleep current lies around 12 mA. When changing the deep sleep current in the formula for the analytic solution to 12 mA, the average current is 13.25 mA. The result using the increased deep sleep current is an energy consumption of *3.323 mJ*, which is close to what we measured on the Fipy. This difference, however small, can be because the FiPy has a bug where the current rises to about 200 mA before the device goes into deep sleep (discussed in section 4.2). Additionally, the research article only conducted tests with one device, so there might be differences between devices. From experiments with the Fipy, we have also seen that the average current when sending is closer to 100 mA. The formula uses 83.0 mA for the average transmission current. If we change the formula to use a transmission current of 100

[2]50.160 seconds was used for the calculations because this was the measured time for the setup used when testing the Fipy

mA, the average current is the same, 13.4 mA. The increased transmission current also makes the analytic energy consumption become the same, 3.357 mJ. In many ways, this confirms the research done by Casals et al. and makes the model general enough to be used as an energy estimation model for the Fipy.

Chapter 5
Discussion

In the following chapter, we discuss the results of the experiments from Chapter 4 in light of the research presented in Chapter 2. Questions raised in the problem description, and throughout this thesis, will be discussed.

From the research presented earlier we know that a LoRaWAN device with a given duty cycle will behave in a specific manner. The energy consumption for any given settings for the device is more or less constant, with some deviations, e.g., for different manufacturers. Casals et al. wrote that because of this, the only way to reduce the energy consumption of the device, and as a result, increase battery life is to increase the notification period, T_{notif} [CMVG17]. Although true in some cases, this claim seems overly simplistic. We know that a LoRaWAN device will consume different amounts of energy in the different states of a duty cycle. The energy consumption in these different states, and also the time it takes to execute each state is more or less constant for a specific application. For some applications, sending, accounts for a large part of the overall energy cost. Intuitively, removing the state that consumes the most energy would make the energy consumption go down. Although reducing the amount of data provided by a node, in general, may seem like a disadvantage, it may be an advantage if the intended sent packets were insignificant [HEE13]. The goal is to have nodes providing the server with "just enough" information and at the same time consuming the least amount of energy doing so. In other words, send data only when it is needed. Doing this requires knowledge about the environment that is to be measured. For instance, an environment application measuring temperature needs to know something about what it should expect when measuring, to be able to make the right decisions as to whether to send or not.

5.1 Aggregating Data for Transmission

In section 1.2, we asked if it is better to try to send data less often in itself, or if we should focus on aggregating measurement data to send larger packets when first

sending. In the experiment in section 4.4 we saw that there is a linear increase in energy when increasing the payload of the transmitted packets, including a fixed energy cost regardless of the payload size. The formula for the trendline in figure 4.7 was generated using the measurement data and represents the energy of transmitting a packet as:

$$Energy = 0.004 * Payload\,length + 0.6133\,(mJ)$$

This means that regardless of the packet payload size, the end-device will consume 0.6133 mJ of energy. The fixed energy cost is a potentially big incentive for aggregating measurement values, and sending larger packets. So by utilizing in-network processing techniques (see section 2.5.2) such as accumulating and holding sensor values until the next transmission we save the majority of energy consumed when sending. The fixed cost justifies trying to send data less often since we save 0.6133 mJ for every n packet the end-devices does not send. Hence, we save $n * 0.6133$ mJ of energy. There is naturally a trade-off between saving energy by accumulating sensor data, and the value this data has at any given time. E.g., time-sensitive data, like in intrusion detection applications would not benefit from using such a scheme. However, since we are working with environmental data (e.g., temperature) which changes at a slow pace, this type of design could be sufficient. There is an upper bound to the maximum energy consumption for a single transmission, defined by the maximum payload size of 242 bytes. Therefore, the maximum potential energy reduction by using this scheme is $242 * 0.6133 = 148.4$ mJ. Although it may seem unrealistic to aggregate 242 values, an application using a high rate transmission scheme would see huge benefits of such a design. By not aggregating, the maximum total energy consumed when sending 242 individual 1 byte payloads would be $242 * (0.004 * 1 + 0.6133) = 149.4$ mJ. By aggregating all the 242 measurements we instead get $0.004 * 242 + 0.6133 = 1.6$ mJ, requiring only about 1.1 % of the total energy to send the same data. This cut implies a reduction of over 9000 % in energy consumption. If the design of such an application involves doing all these measurements in the same duty-cycle (e.g., to get higher confidence in the measurements) the implications could be big. For most applications, this is not the case, and the reduction in energy cost will be lower. Regardless of the application design used, this shows that aggregating data or simply sending less often has potentially huge implications on energy consumption of end-devices.

Considering a temperature application where sensor nodes measure temperature and send data every hour, aggregation of packets can also justify making more measurements. If we use two decimal point precision in the temperature measurements, the resulting payload size would be 5 bytes (e.g., 25.51 degrees Celcius). Measuring the temperature every hour could be good enough, but increasing the rate at which each node measures the temperature would lead to higher data granularity as well as

higher confidence in the measurements. Instead of measuring the temperature once and sending the 5-byte measurement value, we measure every 2 minutes, leading to 30 measurements of a total of 150 bytes. Sending the 150-byte packet only doubles the energy consumption compared to sending a single 5-byte packet (1.2 mJ compared to 0.6 mJ). The calculation used does, however, not consider the added cost of the other states introduced when doing so.

When aggregating packets, we could potentially save more energy, besides the reduced cost of not sending, since the device will not open receiving windows RX1 and RX2 unless it has completed an uplink packet transmission. The research of the energy performance of LoRaWAN conducted by Casals et al. presents a summary of measurement data for the different states of the duty cycle (see table 4.3). We can use these results to get an indication of the added energy reduction when not sending, based on the removal of the receive window states. From the table we could remove all the states indicated in *italic*, leading to a reduction of power consumption of 626.5 mW.

The relevance of these findings for an end-device, or a sensor network as a whole, depends on end-devices either being able to accumulate values without lowering the data granularity of the system, or the server producing high-quality sensing models for the end-devices. How to determine the quality of a sensing model is not an easy task, as this will also be application-dependent. Future research should test different methods and evaluate the results with regards to the energy consumption of end-devices.

5.2 Battery Lifetime as a Function of Reducing Transmissions

From the experiment in section 4.6 we concluded that the analytical solution of Casals et al. was general enough to be used to model the energy consumption of the Fipy, when changes to average deep sleep current and transmission current were made. In section 2.6.1 we presented the formula for the system lifetime of an end-device based on the battery capacity and the average current consumption as:

$$T_{lifetime} = \frac{C_{battery}}{I_{avg}}$$

In the following we implement this formula in Python, showing the effects of reducing the percentage of packets transmissions on system lifetime. We introduce an algorithm for the Python implementation in Algorithm 5.1 (removing all states in italic from table 4.3 when not sending) and show the effects of the reduced packet transmissions as a graph in figure 5.1.

Algorithm 5.1 Python implementation for calculating the effects on battery life of end-devices when reducing the percentage of transmitted packets

```
def avgCurrent(minutes, sending):
    tnotif = minutes*60.0
    totalTime = 168.2 + 268 + 38.6 #time used regardless of sending
    totalCurr = 22.1 + 21 + 13.3 #current used regardless of sending
    if(sending):
        totalTime += 2646.7 #added time for sending
        totalCurr += 236.7 #added current for sending

    totalTime += (tnotif-totalTime) #add deep sleep time
    sleepCurrent = 0.045 * (tnotif-totalTime) #45 microAmpere
    totalCurrent += sleepCurrent #add deep sleep current
    totalTime = totalTime * 0.001 #normalize time to seconds
    totalpower = totalTime*totalCurrent
    Iavg = totalpower/tnotif #calculate the average current

    return Iavg


def getDeviceLifeTimeWhenReducingPercentageOfSentPackets():
    Results = [0.0]*101 #list of length 100

    for i in range(0, 101):
        pSent = (100-i)*0.01 #percentage of sent packets
        pNotSent = i*0.01 #percentage of sent packets
        notificationPeriod = 10.0
        Results[i] += avgCurrent(notificationPeriod,True)*pSent
        Results[i] += avgCurrent(notificationPeriod, False)*pNotSent

    BC = 2400 # battery capacity in mAh

    #calculate the battery life for each value based on formula
    lifetime = np.array([(BC/avgcurr)/24 for avgcurr in Results])
    return lifetime


percentageNotSent = np.array(range(0,101))
lifetime = getDeviceLifeTimeWhenReducingPercentageOfSentPackets()

plt.plot(percentageNotSent,lifetime)
plt.title('Battery lifetime when reducing amount
of sent packets for 10 min duty cycle\n')
plt.xlabel('Percentage of packets not sent\n')
plt.ylabel('Battery lifetime (days)\n')
plt.show()
```

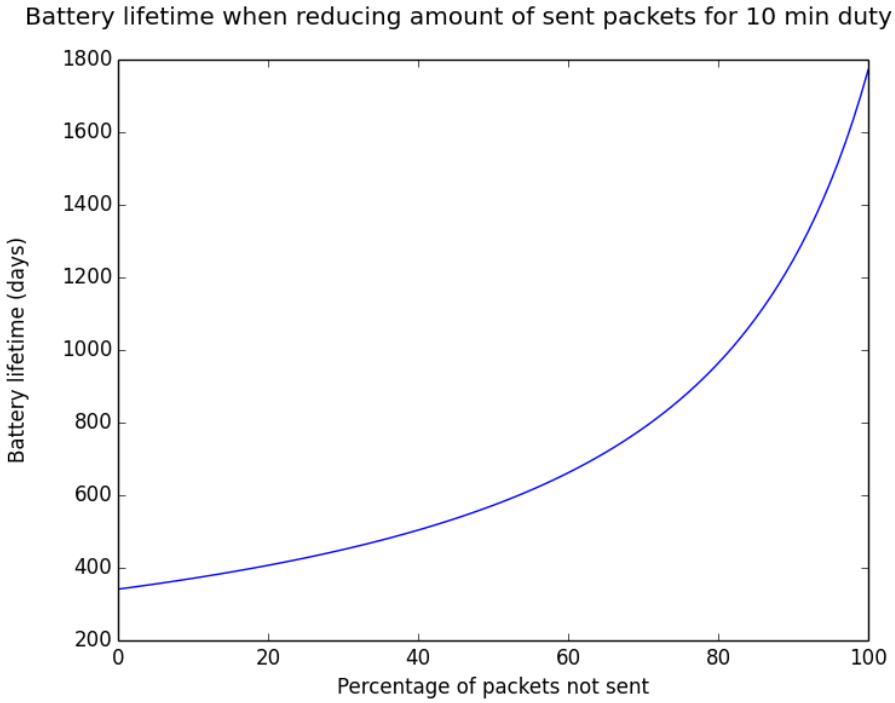

Figure 5.1: Effect on battery life when reducing percentage of (242 byte payload) packets sent for 10 minute duty cycle and a 2400 mAh battery

From Figure 5.1 we see that when sending every packet for each duty cycle (i.e. *Percentage of packets not sent* equals 0) the battery lifetime is 341 days. If we achieve sending half the amount of packets, by accumulating data or using an adaptive sensing model, we increase the battery life to 564 days. So, by sending only fifty percent of the packets, we get an increase of 223 days, which is around seven months of added battery life. This increase results in an extended battery life of 65 %.

5.3 Piggybacking Prediction Models on Existing Packets

For energy-constrained devices, the radio should be used as little as possible to reduce energy consumption. However, some radio transmissions may be initiated by the server to set or update network parameters. It is therefore interesting to investigate if application data can be piggybacked on these packet transmissions, to reduce the need for sending extra packets between the server and end-devices. Sending data from the server to Class A end-devices can only be done after a successful uplink data packet

from the end-device. Acknowledged packet transmissions are possible in LoRaWAN by setting the MType field in the MAC header of the PHY payload (although not best practice due to the duty cycle restrictions). The server will then send an ACK in response to all uplink packets from end-devices. The LoRaWAN protocol does not provide support for piggybacking information on the ACK packets, although this might reduce the need for downlink packets from the server for applications running an acknowledged transmission scheme. The LoRaWAN specification defines a FOpts field in the Frame header (FHDR) of the Medium Access Control (MAC) payload. This field consists of 15 octets that can be used to transport MAC commands to end-devices. The MAC commands can be for instance *rejoin*. The rejoin mac command could be of particular interest because this message can be used to transport parameters to end-devices on top of their regular application traffic. The parameters could be used to initialize a new session context for the end-device, however, the way that this fields are implemented in the current LoRaWAN specification, this would not be possible. The FOpts field is however still interesting because of unused fields. The FOpts can consist of 15 octets, but 0x80 to 0xFF are fields reserved for proprietary network command extensions. This could make it possible to transfer data to end-devices through the use of MAC commands. These MAC commands could contain application parameters for setting device deep-sleep time or model predictions about future sensor measurements. The network periodically sends rejoin MAC of type *RejoinParamSetupReq* to end-devices to reinitialize network parameters. One possible implementation would be that when the server gets a rejoin request for an end-device, it gets the device's configuration parameters and sends them in the FOpts field. The end-device could then set the application parameters based on this data. The same approach could be used when the end-device sends the initial join MAC message to the network. The server could then piggyback application initialization parameters removing the need to set application parameters by default during development. This would limit the amount of downlink messages needed from the server to the end-devices. According to the LoRa specification, a MAC command request initiated by an end-device will be answered within the RX1/RX2 immediately following the request, if possible. Therefore it should be possible to send a MAC command in all uplink packages (join, rejoin, standard data transmission) and receive a MAC response back from the server. There was not conducted any tests with this in this thesis as this would require access to the firmware of the FiPy device. It is a theoretical possibility that this could work, and future research could investigate this further.

5.4 What about NB-IoT?

Since we have not been able to conduct any experiments with NB-IoT in this thesis, the focus has been on understanding LoRaWAN and how to use the protocol in the

best way. Not much research exists on the energy consumption characteristics of NB-IoT and how to reduce them. However, we know from the background material presented in Chapter 2 that NB-IoT enables the use of different energy modes, based on the needs of different applications. To limit the energy consumption, it seems that we should use Power Saving Mode (PSM) as much as possible since it reduces signaling with eNodeB base stations to a minimum. We do not, however, know what the effects of using this scheme have on the battery life of end-devices. We expect it to be much lower than the other modes (DRX and eDRX), but we do not know what the cost of entering and exiting the mode is. The end-devices intended to use this mode, are nodes that sleep for extended periods of time, waking up rarely to send or receive messages. The cost of the negotiation with the server of the Tracking Area Update (TAU) messages needs research before using PSM, especially if the application design requires nodes more frequently sending or receiving messages. Although it is not possible to come to any conclusions about NB-IoT, these devices will also behave in more or less the same way as LoRaWAN end-devices. When removing the signaling with base stations, the duty cycle will probably look more or less the same. An investigation into what separates NB-IoT from LoRaWAN regarding end-device energy consumption, especially with concerns to the signaling messages would be an exciting future research topic. Future research should also investigate what effects changing the medium access protocols of NB-IoT (from FDMA to OFDMA and SC-FDMA) has on energy consumption of end-devices, compared to results in table 2.3.

Chapter 6
Conclusion

In this thesis we have presented Low Power Wide Area Networks, focusing on two of the most prominent technologies, LoRaWAN and NB-IoT. The goal has been to understand the factors impacting energy consumption of LPWAN devices and how to reduce them. We presented adaptive sensing techniques from a general perspective and discussed how they could be used to improve end-device lifetime in Low Power Wide Area Networks. Experiments have been conducted to understand the energy consumption characteristics of LoRaWAN devices, using a private network in a lab setting. Through the research we wanted to answer the following research question:

What are the main factors driving energy consumption of LPWAN devices, and what is the effect of adding adaptive sensing as a means of reducing overall energy consumption?

From the research conducted we better understand the key factors comprising energy consumption in LPWAN technologies. In most cases, the cost of transmission is the primary factor leading to high energy consumption and should be the focus when trying to extend the lifetime of end-devices. We know some, but not all of the effects of the LPWAN energy modes. When designing applications for energy-constrained devices, we should always use functionality class A for LoRaWAN. More investigation must be made into the energy modes in NB-IoT to better understand them in practice. From the background material, Power Saving Mode shows the most potential in extending the end-device lifetime. However, regardless of using LoRaWAN or NB-IoT, the network will always impose some energy consumption on end-devices. Much of the factors leading to these energy costs are fixed and is unconfigurable by the end-devices. From an application point of view, end-devices must be configured to use the best possible network parameters to save energy. Additionally, adaptive sensing techniques should be applied to ensure that applications are energy aware, and use as little energy as possible.

The experiments investigating the baseline energy cost for LoRaWAN devices, both

in accumulative tests and tests of the cost of transmission, show a linear increase in energy consumption of end-devices when increasing the packet payload length. We showed that accumulating and sending larger packets implies reducing energy. Reduction of the amount of energy used by end-devices when transmitting packets can be as much as 1.1 % of what it costs to send non-aggregated packets. This energy reduction is a potentially significant incentive for both aggregating packets and using an adaptive sensing model to try to send data less often.

We have only, to a limited extent, been able to investigate establishing an inexpensive way of sending data to sensor nodes. The investigation into the energy consumption of downlink packets was in many ways impaired by the fact that the equipment we used only had one radio. Since LoRaWAN is half duplex, and the gateway handled downlink and uplink packets from and to our end-device, as well as other devices trying to connect to it, the results of the experiment are limited. However, for LoRAWAN we know that class A devices open the receive windows RX1 and RX2 after uplink transmission. The small set of data from the experiments show a relatively low energy consumption when receiving downlink packets. However, due to the methodology used when measuring this, we cannot be sure if the measurement included current consumption for RX1 and RX2 from the model of Casals et al. or not. Therefore, we do not model the downlink data to a satisfactory level in this thesis.

We also raised questions concerning the possibility of implementing adaptive sensing model data within the Medium Access Control (MAC) layer of LoRaWAN. Since there are unused fields in the MAC messages transmitted between server and end-devices, using this scheme could potentially reduce overall energy consumption, since transmission of MAC messages happens periodically. One justification of research into this area is on the function MAC has in the protocol stack with regards to controlling the access to the physical transmission medium. Although Adaptive sensing in many cases is achieved at the application layer, controlling how, and how often IoT nodes gain access to the transmission medium is essential (cf. duty-cycle limitations of the ISM band). If it is possible to use adaptive sensing techniques in MAC messages, and how to best implement them, should, therefore, be a future research question.

Lastly, we showed that a device using a 2400 mAh battery, running an application with a 10-minute duty cycle could increase battery life more than seven months when reducing half of the packets to be sent. So if an adaptive model making predictions on future temperature values is correct half the time, it would have significant implications for battery life of end-devices. If end-devices use larger batteries or applications send smaller packet payloads[1], the implications are even

[1]The energy model is based on 242-byte payloads

more prominent.

Future works within the topics presented in this thesis could be:

- Investigations into real-world applications of adaptive sensing techniques with regards to the energy consumption of LPWAN devices. The research should also conduct experiments using batteries, to better understand how well the methods can perform with regards to the non-linear properties of batteries.
- Combining the methods proposed in this thesis with network topology based adaptive sensing techniques.
- Further research into what separates NB-IoT from LoRaWAN regarding energy consumption, especially on the effects of signaling between end-devices and base stations on battery life.
- Research into what an adaptive sensing model should, and needs to be to reduce energy consumption. What form should the models have to reduce the energy of sending them to nodes?
- Investigate the use of energy-harvesting and adaptive sensing methods within LPWAN systems, in an effort towards reaching self-sustained devices. An interesting research question would be: *How self-sufficient can such a system get?*
- Studies on how to implement adaptive sensing techniques in the MAC layer of LoRaWAN, and what effects this might have on end-device battery life.

References

[3GPa] 3GPP. The 3rd generation partnership project. http://www.3gpp.org/.

[3GPb] 3GPP. Release 13. http://www.3gpp.org/release-13.

[AA04] T. Arici and Y. Altunbasak. Adaptive sensing for environment monitoring using wireless sensor networks. In *2004 IEEE Wireless Communications and Networking Conference (IEEE Cat. No.04TH8733)*, volume 4, pages 2347–2352 Vol.4, March 2004.

[AAFR09] C. Alippi, G. Anastasi, M. Di Francesco, and M. Roveri. Energy management in wireless sensor networks with energy-hungry sensors. *IEEE Instrumentation Measurement Magazine*, 12(2):16–23, April 2009.

[ACDFP09] Giuseppe Anastasi, Marco Conti, Mario Di Francesco, and Andrea Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3):537–568, May 2009.

[AVT+17] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, and T. Watteyne. Understanding the Limits of LoRaWAN. *IEEE Communications Magazine*, 55(9):34–40, 2017.

[CMVG17] Lluís Casals, Bernat Mir, Rafael Vidal, and Carles Gomez. Modeling the Energy Performance of LoRaWAN. *Sensors*, 17(10):2364, October 2017.

[ETS] ETSI. European telecommunications standards institute. http://www.etsi.org/.

[Fun17] Software Testing Fundamentals. Black Box Testing, December 2017.

[Got18] Petr Gotthard. Lorawan-server: Compact server for private LoRaWAN networks, May 2018.

[HEE13] Anar A. Hady, Sherine M. Abd El-kader, and Hussein S. Eissa. Intelligent Sleeping Mechanism for wireless sensor networks. *Egyptian Informatics Journal*, 14(2):109–115, 2013.

[Hiv] HiveMQ. HiveMQ - The Enterprise MQTT Broker. https://www.hivemq.com/.

[IEE] IEEE. Institute of Electrical and Electronics Engineers. https://www.ieee.org/index.html.

[IET] IETF. Internet Engineering Task Force. https://www.ietf.org/.

[LBSB07] Yann-Aël Le Borgne, Silvia Santini, and Gianluca Bontempi. Adaptive model selection for time series prediction in wireless sensor networks. *Signal Processing*, 87(12):3010–3020, December 2007.

[LL17] Jinseong Lee and Jaiyong Lee. Prediction-Based Energy Saving Mechanism in 3GPP NB-IoT Networks. *Sensors (Basel, Switzerland)*, 17(9), September 2017.

[MBCM18] Kais Mekki, Eddy Bajic, Frederic Chaxel, and Fernand Meyer. A comparative study of LPWAN technologies for large-scale IoT deployment. *ICT Express*, January 2018.

[Mic] Microchip. PAC1934 4 Channel DC Power Monitor Evaluation Board - ADM00805 | Microchip Technology Inc. https://www.microchip.com/DevelopmentTools/ProductDetails.aspx?PartNO=ADM00805&utm_source=MicroSolutions&utm_medium=Link&utm_term=FY18Q3&utm_content=MSLD&utm_campaign=Article.

[Mic17] Microchip. PAC1934 Evaluation Board (ADM00805) User's guide, 2017.

[Pyca] Pycom. 5.3.6 LoRaWAN Nano-Gateway · Pycom Documentation. https://docs.pycom.io/chapter/tutorials/lora/lorawan-nano-gateway.html.

[Pycb] Pycom. Expansion Board 2.0 Specs.

[Pycc] Pycom. Preface · Pycom Documentation. https://docs.pycom.io/.

[Pycd] Pycom. Specification Sheet for the FiPy five network development board.

[Pyce] Pycom. Specification Sheet for the LoPy LoRa, WiFi and Bluetooth development board.

[Ras] RaspberryPi. Raspberry Pi - Teach, Learn, and Make with Raspberry Pi. https://www.raspberrypi.org/.

[RKS17] U. Raza, P. Kulkarni, and M. Sooriyabandara. Low Power Wide Area Networks: An Overview. *IEEE Communications Surveys Tutorials*, 19(2):855–873, Secondquarter 2017.

[Spa16] Spartan. Agile + Lean Development. https://blog.joinspartan.com/agile-lean-development/, October 2016.

[SWH17] Rashmi Sharan Sinha, Yiqiao Wei, and Seung-Hoon Hwang. A survey on LPWA technology: LoRa and NB-IoT. *ICT Express*, 3(1):14–21, March 2017.

[Tel18] Telia. Telia-iot-workshop: Guide and software for Telia IoT Workshop, May 2018.

[THK17] Nattachart Tamkittikhun, Amen Hussain, and Frank Alexander Kraemer. Energy Consumption Estimation for Energy-Aware, Adaptive Sensing Applications. In *Mobile, Secure, and Programmable Networking*, Lecture Notes in Computer Science, pages 222–235. Springer, Cham, June 2017.

[Wie14] Roel J. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*, volume 1. Springer Heidelberg, London, first edition, 2014.

[WLG+11] Guiyi Wei, Yun Ling, Binfeng Guo, Bin Xiao, and Athanasios V. Vasilakos. Prediction-based data aggregation in wireless sensor networks: Combining grey model and Kalman Filter. *Computer Communications*, 34(6):793–802, May 2011.

[WMN04] R. Willett, A. Martin, and R. Nowak. Backcasting: Adaptive sampling for sensor networks. In *Third International Symposium on Information Processing in Sensor Networks, 2004. IPSN 2004*, pages 124–133, April 2004.