Hardware and software platform for Internet of Things

Master of Science Thesis in Embedded Electronic System Design

Johan Bregell

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, September 2015

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet. The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Hardware and software platform for Internet of Things

Johan Bregell

© Johan Bregell, September 1, 2015.
Supervisor: Lars Svensson
Examiner: Per Larsson-Edefors

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden, September 2015

Abstract

Internet of Things is a relatively new and undiscovered field under constant change. Today several similar technologies exist, all competing to be accepted as the new standard. As the concept of Internet of Things spans from hardware, to operating system, all the way up to application level communication protocol, this project investigated the most promising technologies in each of these areas. The findings of this investigation were used to build a prototype; this prototype was then used to assess the range, response time, connection speed, and power consumption, of the selected communication protocol.

The assessment chapter compares the results to the theoretical values and highlights the differences, while the discussion chapter explores why these differences occur. The obtained results give insight in what kind of applications Internet of Things might bring, and also what issues that currently exist. Amongst the discoveries is the poor response time of an Internet of Things system while using power saving measures. These measures are needed to achieve a battery life of 1 year, resulting in the inability for real-time applications to utilize the power of Internet of Things.

Acknowledgements

I would like to offer special thanks to Peter Olsson and Håkan Rolin, my supervisors at i3tex AB. Håkan Rolin gave me the opportunity to write a thesis in a brand new and interesting field and Peter Olsson for his knowledge and technical support in embedded programming. I would also express my sincere appreciation to my academic supervisor Lars Svensson, for the help with the current readings, constructive critique, and extensive proofreading. Finally, I would like to express my gratitude to Per Larsson-Edefors for being my examiner.

Johan Bregell
Chalmers University of Technology
Gotheburg, May, 2015

Contents

Acronyms

ACK Acknowledge packet.

BLE Bluetooth Low Energy.

CAN Controller Area Network.

CBOR Concise Binary Object Representation.

CCA Channel Clear Assessment.

CoAP Constrained Application Protocol.

FM Fade margin.

FSPL Free-space path loss.

GPIO General-purpose input/output.

I^2C Inter-Integrated Circuit.

IDE Integrated Development Environment.

IETF Internet Engineering Task Force.

IoT Internet of Things.

JSON JavaScript Object Notation.

JTAG Joint Test Action Group.

LPM Low Power Mode.

MAC Medium Access Control.

MCU Microcontroller.

nesC Network Embedded Systems C.

O-QPSK Offset quadrature phase-shift keying.

OS Operating System.

OSI model Open Systems Interconnection model.

PCB Printed Circuit Board.

PHY Physical.

PM Power management.

RAM Random-access memory.

RDC Radio Duty Cycle.

ROM Read-only memory.

RPL IPv6 Routing Protocol for Low-Power and Lossy Networks.

SoC System on Chip.

SPI Serial Peripheral Interface.

UART Universal asynchronous receiver/transmitter.

WLAN Wireless LAN.

WSN Wireless Sensor Network.

Chapter 1

Introduction

1.1 Background

Internet of Things (IoT) is a concept aiming at connecting all things to the Internet [1]. The different kinds of devices range from simple sensor devices to complex machines such as industry robots.

Home automation has been available for a few years in the forms of timers and remotely controlled devices, such as lights, garage door, and climate control equipment. Also in the industry and workplace, there are current systems that have some of the functionality of IoT, e.g, sensors in robots and machines which keep track of the system status so that maintenance can be scheduled at the right time. However, these systems or sensors rarely communicate with each other or make decisions based on other sensor values; instead they depend on input from a user.

In the same way cellphones connected people and made them constantly connected to the Internet, IoT will connect devices and make them constantly connected to the Internet [2]. In theory, this could lead to a future with autonomous technology all around us. The benefits could be huge as it would save time and energy for both the individual at home and for the industry [3]. IoT could be used in industry to automate power-heavy tasks to run when the electricity price is low. This principle can also be applied for the home user with laundry machines and charging of e.g. electric cars. This practice would lead to reduced energy consumption and thus a reduced environmental footprint.

i3tex AB wants to investigate potential fields of applicability of this upcoming technology. i3tex AB has customers in the automotive, communication, and pulp industries; those customers have made inquiries on how to integrate IoT and sensor networks into production. As technology evolves, size and energy consumption of the IoT devices will decrease and computation power will increase [4]. This reduction in size and energy consumption, together with the increased computing power, will open up new fields for

IoT. Thus, i3tex AB want to have an IoT platform to present to their current and potential customers.

The interest in IoT is rapidly increasing, and thus, in the near future, the number of devices connected to the Internet is expected to increase rapidly. To support this huge increase in both number of connected devices and the sheer amount of data that will be sent over both wired and wireless networks, the communication technology must be ready [5].

1.2 Purpose

The purpose of this project is to find and examine a communication method for devices that are made to be a part of IoT. This will be done by examining the available technologies and then developing a prototype based on the findings, which will be used for examining the communication method. This project will examine the physical, link, and network layers [6, 7] of the Open Systems Interconnection model (OSI model) [8], in order to find suitable technologies on the market. As IoT is still only defined as a concept, there are several technologies to take into consideration and examine in further detail. The prototype will be delivered to i3tex AB together with appropriate documentation, e.g. technical specification, hardware manual, software manual, and API specification.

1.3 Limitations

To be able to achieve the project goal within the available time, limitations need to be defined in the three main areas of: Operating System (OS), hardware, and communication method.

The OS will not be custom-made, but rather selected amongst those already on the market. Thus, to simplify the hardware selection, only those OSs which already have hardware support that meets the requirements will be taken into consideration. Furthermore, support for either 6LoWPAN, ZigBee, or Bluetooth Low Energy (BLE) as communication method is required, since development to make those standards available is outside the scope of the project.

On the hardware side, the limitations will be to only use existing devices and parts as there will be no time for developing hardware or Printed Circuit Boards (PCBs). However, the hardware does not need to have an integrated radio transceiver, but needs to support at least one transceiver supporting IEEE802.15.4 [6]. Thus, communication methods will be primarily selected from specifications building on the IEEE 802.15.4.

1.4 Method

To ensure that the right technologies were selected and investigated, the first phase of the project was a literature study. The study served as a foundation when developing and performing the evaluation of the communication methods. At the end of the phase, a requirements specification was formulated to serve as a platform for the next phase.

After the literature study, a selection process was performed, where the most promising technologies that met the requirements were examined in further detail and brought into the development phase. This process included the selection of development tools and other decisions bound to the product development.

In the development phase, the chosen set-up was configured and assembled to prepare for testing; it was then tested according to throughput, range, latency, and energy consumption. Throughput was measured in kilobyte per second (KB/s) and tested by transferring data of different sizes in both congested and uncongested network set-ups to simulate real world and lab environments. The same set-up was used to measure the latency of a transmission, which was measured in microseconds (ms). Range was calculated instead of measured, with meters (m) as the unit. The power consumption was measured in watts (W).

Each week a meeting with the company supervisors was performed, to keep the work on the right track. Here, feedback was be given and other issues and questions handled.

Chapter 2

Technical Background

As of now, IoT is still on a concept level, and is roughly defined as a world-wide network of objects, where each object is equipped with sensors and/or actuators, Microcontroller (MCU), transceiver, and power source [9, 10]. The objects or devices are connected to smaller networks called mesh networks; the mesh networks are then connected to the IPv6 based internet with border routers which translate the IPv6 messages into messages more suited for energy efficient communication. Each node is supposed to be directly addressable thanks to the vast amount of IP addresses available in the IPv6 standard.

Each mesh network is built up from several nodes [11], i.e. the consumer devices, all talking to all other nodes within range, as seen in figure 2.1. The nodes may either be passive or active, depending on the role of the node. *Active nodes* serve as the branches of the network and are constantly ready to relay messages in either direction. These nodes usually have an external power source. *Passive nodes*, often used as data collectors, only turn on when it is time to collect and send new data. Usually, they do not relay data and can be seen as the leaves of the network structure; therefore they can operate on battery power. The *border router* is the link between the Internet and the mesh network; it translates IPv6 frames into mesh networks frames. A mesh network frame differs from a IPv6 frame in both size and addressing; the addressing can be either a variation of IPv6 or a completely independent scheme, and the size is usually only 10% of the IPv6 frame size. All nodes can talk directly to every other node in range i.e. their neighbours, but to reach the border router, one of the neighbours are selected as the active route. The active route is supposed to be the most effective way to reach the border router in terms of latency and hops.

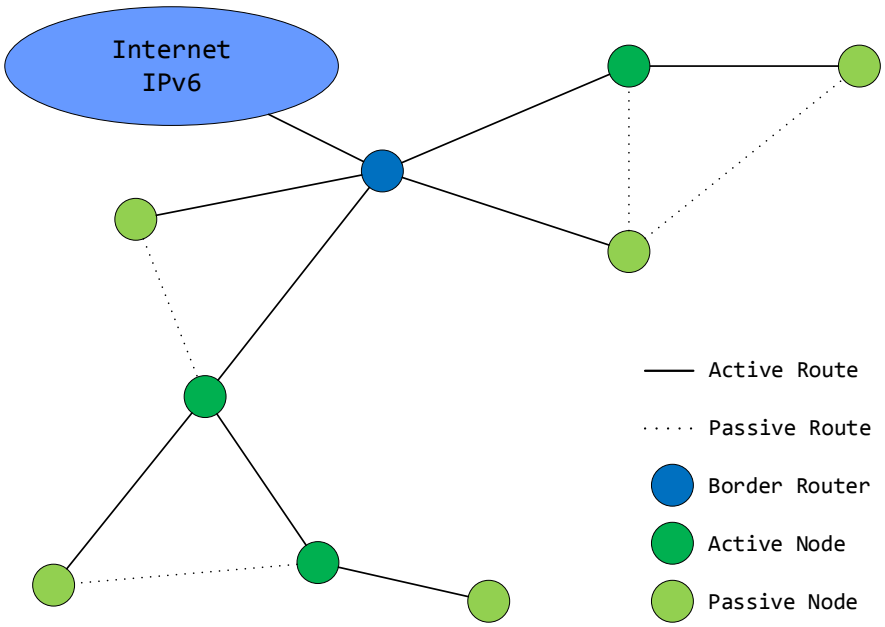


Figure 2.1: IoT mesh network

One of the most common mesh network communication standards is IEEE 802.15.4; it defines the Physical (PHY) layer and Medium Access Control (MAC) layer of the OSI model [8]. This standard has two major implementations: 6LoWPAN and ZigBee. Each implementation can operate in the 2400-2483.5, 902-928, and 868-868.6 MHz frequency spans as defined in the IEEE 802.15.4 standard. One of the main differences is that 6LoWPAN uses native IPv6 addressing; thus the nodes can be accessed directly from any IPv6 enabled device connected to the Internet.

A node is specified to be a low power MCU with a transceiver and some sensors and/or actuators. The assignment of the nodes is to monitor their surroundings or to control particular equipment. A consumer device can have a integrated or external IoT node and in both cases the device as a whole is considered to be a IoT device. Nodes can have different software complexity depending on their role, but in general an embedded OS is needed.

Thus, when designing a IoT device, there are three major components that need to work together: MCU, OS, and communication. In the following sections, the most prominent technologies in each of these fields will be examined and described. This investigation will act as the foundation for the implementation phase of the project.

2.1 Operating system

The operating system is the foundation of the IoT technology as it provides the functions for the connectivity between the nodes. However, different types of nodes need different levels of OS complexity; a passive node generally only needs the communication stack and is not in need of any threading capabilities, as the program can handle all logic in one function. Active nodes and border routers need to have a much more complex OS, as they need to be able to handle several running threads or processes, e.g. routing, data collection and interrupts. To qualify as an OS suitable for the IoT, it needs to meet the basic requirements:

- Low Random-access memory (RAM) footprint
- Low Read-only memory (ROM) footprint
- Multi-tasking
- Power management (PM)
- Soft real-time

These requirements are directly bound to the type of hardware designed for the IoT. As this type of hardware in general needs to have a small form factor and a long battery life, the on-board memory is usually limited to keep down size and energy consumption. Also, because of the limited amount of memory, the implementation of threads is usually a challenging task, as context switching needs to store thread or process variables to memory. The size of the memory also directly affects the energy consumption, as memory in general is very power hungry during accesses. To be able reduce the energy consumption, the OS needs some kind of power management. The power management does not only let the OS turn on and off peripherals such as flash memory, I/O, and sensors, but also puts the MCU itself in different power modes. As the nodes can be used to control and monitor consumer devices, either a hard or soft real-time OS is required. Otherwise, actions requiring a close to instantaneous reaction might be indefinitely delayed. *Hard real-time* means that the OS scheduler can guarantee latency and execution time, whereas *Soft real-time* means that latency and execution time is seen as real-time but can not be guaranteed by the scheduler. Operating systems that meet the above requirements are compared in table 2.1 and 2.2.

Table 2.1: OS Requirements Support

	Memory				Real-time	
OS	RAM	ROM	Multi-tasking	PM	Hard	Soft
Contiki	10kB	30kB	X	X	-	X
RIOT	1.5kB	5kB	X	X	X	X
TinyOS	1kB	4kB	X	X	-	X
freeRTOS	1kB	10kB	X	X	X	X

Table 2.2: OS Programming Language Support

OS	C	C++	nesC
Contiki	X	-	-
RIOT	X	X	-
TinyOS	-	-	X
freeRTOS	X	-	-

2.1.1 Contiki

Contiki is a embedded operating system developed for IoT written in C [12]. It supports a broad range of MCUs and has drivers for various transceivers. The OS does not only support TCP/IPv4 and IPv6 with the uIP stack [9], but also has support for the 6LoWPAN stack and its own stack called RIME. It supports threading with a thread system called Protothreads [13]. The threads are stack-less and thus use only two bytes of memory per thread; however, each thread is bound to one function and it only has permission to control its own execution.

Included in Contiki, there is a range of applications such as a HTTP, Constrained Application Protocol (CoAP), FTP, and DHCP servers, as well as other useful programs and tools. These applications can be included in a project and can run simultaneously with the help of Protothreads. The limitations to what applications can be run is the amount of RAM and ROM the target MCU provides. A standard system with IPv6 networking needs about 10 kB RAM and 30 kB ROM but as applications are added the requirements tend to grow.

2.1.2 RIOT

RIOT is a open source embedded operating system supported by Freie Universität Berlin, INIRA, and Hamburg University of Applied Sciences [14]. The kernel is written in C but the upper layers support C++ as well. As the project originates from a project with real-time and reliability requirements, the kernel supports hard real-time multi-tasking scheduling. One of the goals of the project is to make the OS completely POSIX compliant. Overhead for multi-threading is minimal with less than 25 bytes per thread. Both IPv6 and 6LoWPAN is supported together with UDP, TCP, and IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL); and CoAP and Concise Binary Object Representation (CBOR) are available as application level communication protocols.

2.1.3 TinyOS

TinyOS is written in Network Embedded Systems C (nesC) which is a variant of C [15]. nesC does not have any dynamic memory allocation and all program paths are available at compile-time. This is manageable thanks to the structure of the language; it uses modules and interfaces instead of functions [16]. The modules use and provide interfaces and are interconnected with configurations; this procedure makes up the structure of the program. Multitasking is implemented in two ways: trough *tasks* and *events*. Tasks, which focus on computation, are non-preemptive, and run until completion. In contrast, events which focus on external events i.e. interrupts, are preemptive, and have separate start and stop functions. The OS has full support for both 6LoWPAN and RPL, and also have libraries for CoAP.

2.1.4 freeRTOS

One of the more popular and widely known operating systems is freeRTOS [17]. Written in C with only a few source files, it is a simple but powerful OS, easy to overview and extend. It features two modes of scheduling, pre-emptive and co-operative, which may be selected according to the requirements of the application. Two types of multitasking are featured: one is a lightweight Co-routine type, which has a shared stack for lower RAM usage and is thus aimed to be used on very small devices; the other is simply called Task, has its own stack and can therefore be fully pre-empted. Tasks also support priorities which are used together with the pre-emptive scheduler. The communication methods supported out-of-the-box are TCP and UDP.

2.2 Hardware

Even though the hardware is in one sense the tool that the OS uses to make IoT possible, it is still very important to select a platform that is future-proof and extensible. To be regarded as an extensible platform, the hardware needs to have I/O connections that can be used by external peripherals. Amongst the candidate interfaces are Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I^2C), and Controller Area Network (CAN). These interfaces allow developers to attach custom-made PCBs with sensors for monitoring or actuators for controlling the environment. The best practice is to implement an extension socket with a well-known form factor.

A future-proof device is specified as a device that will be as attractive in the future as it is today. For hardware, this is very hard to achieve as there is constant development that follows Moore's Law [4]; however, the most important aspects are: the age of the chip, its expected remaining lifetime, and number of current implementations i.e. its popularity. If a device is widely used by consumers, the lifetime of the product is likely to be extended. One last thing to take into consideration is the product family; if the chip belongs to a family with several members the transition to a newer chip is usually easier.

2.2.1 OpenMote

OpenMote is based on the Ti CC2538 System on Chip (SoC), which combines an ARM Cortex-M3 with a IEEE 802.15.4 transceiver in one chip [18, 19]. The board follows the XBee form factor for easier extensibility, which is used to connect the core board to either the OpenBattery or OpenBase extension boards [20, 21]. It originates from the CC2538DK which was used by Thingsquare to demo their Mist IoT solution [22]. Hence, the board has full support for Contiki, which is the foundation of Thingsquare. It can run both as a battery-powered sensor board and as a border router, depending on what extension board it is attached to, e.g OpenBattery or OpenBase. Furthermore, the board has limited support but ongoing development for RIOT and also full support for freeRTOS.

Table 2.3: OpenMote and extensions specifications

- CC2538 SoC
  - ARM Cortex-M3 MCU
  - RAM 32kB
  - Flash 512kB
  - HW Multiplier/Divider
  - 32 x GPIO Pins
  - I^2C
  - SPI
  - ADC 12-bits
  - AES/RSA Encryption Engine
- Radio Transceiver
  - 2.4 GHz
  - IEEE 802.15.4
  - 7 dBm Power
  - -97 dBm Sensitivity
- Current Consumption
  - 24 mA Active
  - 0.6 mA Standby
  - 0.4 uA Deep Sleep
- OpenBattery
  - 2xAA Battery Slot
  - SHT21 Temp and Humidity Sensor
  - ADXL346 Acceleration Sensor
  - MAX44009 Light Sensor
- OpenBase
  - FTDI FT232R Serial Port
  - ENC28J60 Ethernet Port
- Price
  - OpenMote: 90 euro
  - OpenBase: 60 euro
  - OpenBattery: 30 euro

2.2.2 MSB430-H

The Modular Sensor Board 430-H from Freie Universität Berlin was designed for their ScatterWeb project [23]. As the university also hosts the RIOT project, the decision to support RIOT was natural. The main board has a Ti MSP430F1612 MCU [24], a Ti CC1100 transceiver, and a battery slot for dual AA batteries; it also includes a SHT11 temperature and humidity sensor and a MMA7260Q accelerometer to speed up early development. All GPIO pins and buses are connected to external pins for extensibility. Other modules with new peripherals can then be added by making a PCB that

matches the external pin layout.

Table 2.4: MSB430 Specification

- MSP430F1612 MCU
  - RAM 5kB
  - Flash 55kB+256B
  - 48 GPIO Pins
  - I^2C
  - SPI
  - ADC 12-bits
  - DAC 12-bits
- CC1100 Transciever
  - $<$ 1 GHz
  - IEEE 802.15.4
  - 10 dBm Power
  - -93 dBm Sensitivity
- Sensors
  - SHT11 Temp and Humidity Sensor
  - MMA7260Q Accelerometer
- Current Consumption (MCU+RF)
  - 330 uA + 16.9 mA Active
  - 1.1 uA + 1.6 mA Standby
  - 0.2 uA + 0.4 uA Deep Sleep
- Price
  - 30 - 100 euro depending on manufacturing cost

2.2.3 Zolertia Z1

As many other Wireless Sensor Network (WSN) products, the Zolertia Z1 builds upon the MSP430 MCU [25, 26]. The communication is managed by the Ti CC2420 which operates in the 2.4 GHz band. The platform includes two sensors: the SHT11 temperature and humidity sensor and the MMA7600Q accelerometer. Extensibility is ensured with: two connections designed especially for external sensors, an external connector with USB, Universal asynchronous receiver/transmitter (UART), SPI, and I^2C.

Table 2.5: Zolertia Z1 Specification

- MSP430F2617 MCU
  - RAM 8kB
  - Flash 92kB+256B
  - 48 GPIO Pins
  - I^2C
  - SPI
  - ADC 12-bits
  - DAC 12-bits
- CC2420 Transciever
  - 2.4 GHz
  - IEEE 802.15.4
  - 0 dBm Power
  - -95 dBm Sensitivity
- Sensors
  - ADXL345 Accelerometer
  - TMP102 Temperature Sensor
- Current Consumption (MCU+RF)
  - 365 uA + 18.8 mA Active
  - 0.5 uA + 426 uA Standby
  - 0.1 uA + 0.02 uA Deep Sleep
- Price
  - 95 euro

2.3 Communication protocol

Several different wireless communication protocols, such as Wireless LAN (WLAN), BLE, 6LoWPAN, and ZigBee may be suitable for IoT applications. They all operate in the 2.4GHz frequency band and this, together with the limited output power in this band, means that they all have a similar range. The main differences are located in the MAC, PHY, and network layer. WLAN is much too power hungry as seen in table 2.6 and is only listed as a reference for the comparisons.

Table 2.6: Protocol specification overview

Protocol	Frequency	Range	IEEE	Data Rate	TX Active Power @ 3V
WLAN	2.4GHz	35m	802.11	54 Mbit/s	∼800mW
BLE	2.4GHz	100m	802.15.1	1 Mbit/s	∼25mW
6LoWPAN	2.4GHz	20m	802.15.4	250 kbit/s	<75mW
ZigBee	2.4GHz	20m	802.15.4	250 kbit/s	<75mW

2.3.1 IEEE 802.15.4

The IEEE 802.15.4 standard defines the PHY and MAC layers for wireless communication [6]. It is designed to use as little transmission time as possible but still have a decent payload, while consuming as little power as possible. Each frame starts with a preamble and a start frame delimiter; it then continues with the MAC frame length and the MAC frame itself as seen in figure 2.2. The overhead for each PHY packet is only $\frac{4+1+1}{133} \sim 4.5\%$; when using the maximum transmission speed of 250kbit/s, each frame can be sent in $\frac{133\text{byte}}{250\text{kbit/s}} = 4.265\text{ms}$. Furthermore, it can also operate in the 868MHz and 915MHz bands, maintaining the 250kbit/s transmission rate by using Offset quadrature phase-shift keying (O-QPSK).

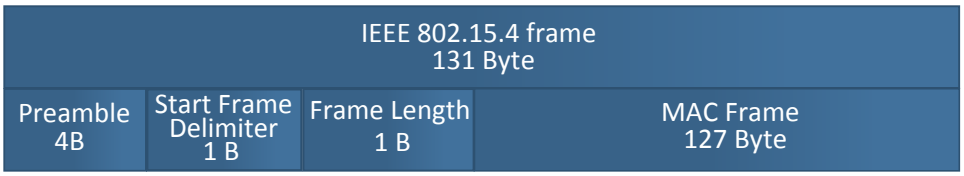


Figure 2.2: IEEE 802.15.4 frame

Several network layer protocols are implemented on top of IEEE 802.15.4. The two that will be examined are 6LoWPAN and ZigBEE.

6LoWPAN

6LoWPAN is a relatively new protocol that is maintained by the Internet Engineering Task Force (IETF) [7, 6]. The purpose of the protocol is to enable IPv6 traffic over a IEEE 802.15.4 network with as low overhead as possible; this is achieved by compressing the IPv6 and UDP header. A full size IPv6 + UDP header is 40+8 bytes which is $\sim 38\%$ of a IEEE 802.15.4 frame, but with the header compression this overhead can be reduced to 7 bytes, thus reducing the overhead to $\sim 5\%$, as seen in figures 2.3 and 2.4.

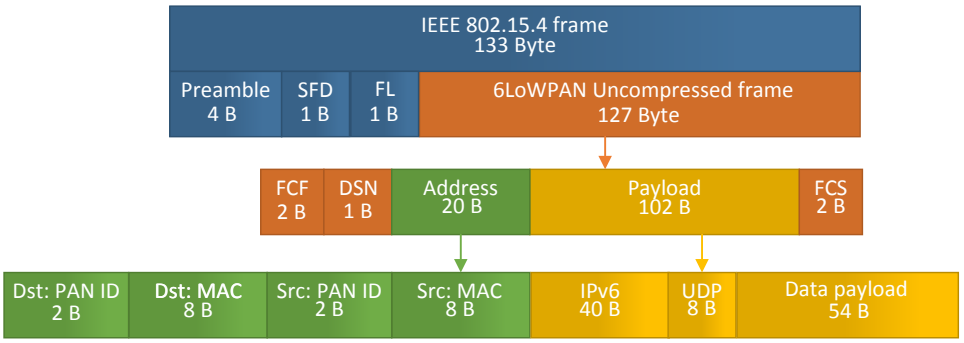


Figure 2.3: 6LoWPAN uncompressed frame

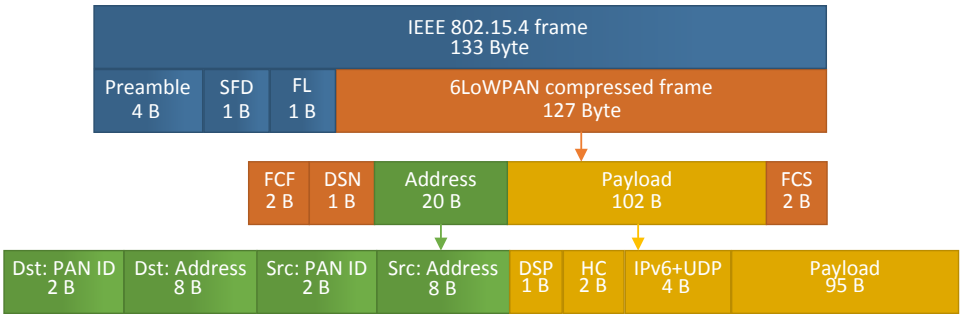


Figure 2.4: 6LoWPAN compressed frame

ZigBee

ZigBee is a communication standard initially developed for home automation networks; it has several different protocols designed for specific areas such as lighting, remote control, or health care [27, 6]. Each of these protocols uses their own addressing with different overhead; however, there is also the possibility of direct IPv6 addressing. Then, the overhead is the same as for uncompressed 6LoWPAN, as seen in figure 2.5.

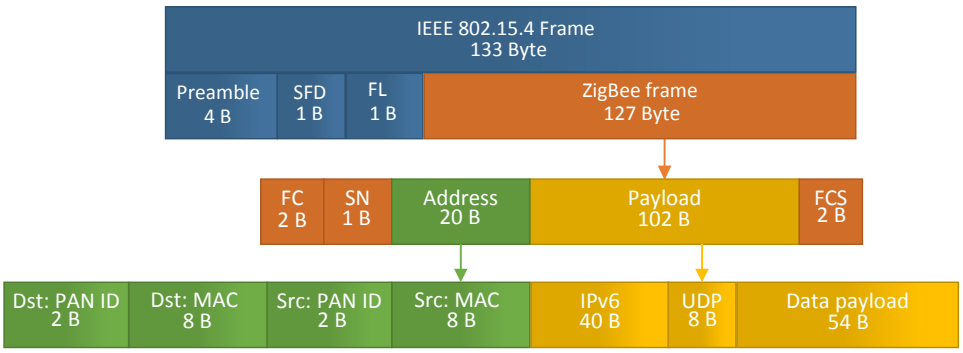


Figure 2.5: ZigBee frame

A new standard called ZigBee 3.0 aims to bring all these standards together under one roof to simplify the integration into IoT. The release date of this standard is set to Q4 2015.

2.3.2 Bluetooth LE

BLE is developed to be backwards compatible with Bluetooth, but with lower data rate and power consumption [28]. Featuring a data rate of 1Mbit/s with a peak current consumption less than 15mA, it is a very efficient protocol for small amounts of data. Each frame can be transmitted in $\frac{47\text{bytes}}{1\text{Mbit/s}} = 376\mu\text{s}$; thanks to the short transmission time, the transceiver

consumes less power as the transceiver can be in receive mode or completely off most of the time. BLE uses its own addressing methods and as the MAC frame size (figure 2.6) is only 39bytes, thus IPv6 addressing is not possible.

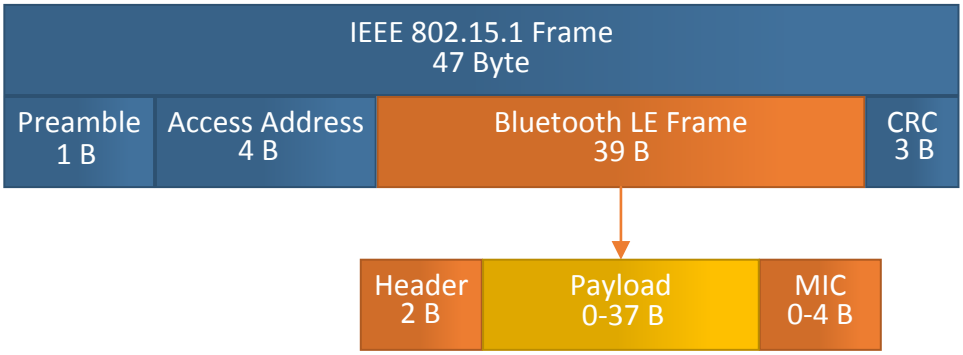


Figure 2.6: BLE frame

Starting from Bluetooth version 4.2, there is support for IPv6 addressing with the Internet Protocol Support Profile; the new version allows the BLE frame to be variable between 2 - 257 bytes. The network set-up is controlled by the standard Bluetooth methods, whereas IPv6 addressing is handled by 6LoWPAN as specified in *IPv6 over Bluetooth Low Energy* [29].

Chapter 3

Implementation

The goal of the implementation phase is to have a working prototype for future assessment. To make the process of implementing the prototype possible, the first part of the implementation process will be to create a set of requirements. When these are set, the process will continue by comparing the data from chapter 2 to find the candidates that fulfil the requirements. After the technologies are selected, the process will continue with setting up the workspace, which includes the platform for development and the required tools to build, debug and test the prototype. Finally, when these three steps have been performed, the next step will be to start with the actual prototype development.

3.1 Selection of technology

3.1.1 Requirements

As the time dedicated for development is limited, the requirements have to make sure that the development process does not run into any major obstacles. All parts of the total prototype need to fit together seamlessly. However, the hardware platform and the operating system are tied most closely together. Therefore, they need requirements that complement each other, so that they can act as a platform for software development. Naturally, both the hardware and the operating system requirements might have to be altered slightly to enable the best match.

Hardware

- Transceiver with IEEE 802.15.4 or IEEE 802.15.1
- Integrated sensor/sensors
- MCU with low power mode under 5μA

- Wakeup from low power mode with timer
- Border router ability
- Joint Test Action Group (JTAG) support

Operating system

- Support for the SoC/MCU and transceiver
- 6LoWPAN, ZigBee or BLE stack
- Soft real-time
- RAM and ROM footprint matching the hardware

Communication protocol

- IPv6 addressing support
- Existing OS support
- Network type is mesh
- UDP

3.1.2 Hardware

Each platform examined in chapter 2 has different strengths and weaknesses. When looking at the MCU, OpenMote has a ARM Cortex-M3 which is more powerful compared to the other two alternatives: it features a 32bit 32MHz core with 32KB RAM and 512KB flash memory compared to the MSP430 16bit 25MHz core with ∼10KB/∼100KB memory configuration. In terms of peripherals all three platforms are comparable, with similar amount of DAC, GPIO pins, and external busses. All of the platforms have a temperature sensor and an accelerometer, but OpenMote also features an light/uv-light sensor and a voltage sensor built into the MCU's ADC. The MSB430 platform has a somewhat lower power consumption in active RF mode thanks to the less power-hungry sub-GHz transceiver. On the other hand, the less powerful MSP430 MCU has a better deep sleep power consumption, but as the radio is not integrated in the chip as it is in the CC2538 SoC, that advantage is offset by the external transceiver.

Comparing the transceivers, there are two 2.4GHz models and one sub-GHz model; the sub-GHz CC1100 has a higher transmit power of 10dBm compared to 0dBm for CC2420 and 7dBm for CC2538. Also the sensitivity is similar for all the alternatives but gives a slight advantage to CC2538 with -97dBm compared to -95dBm and -93dBm for CC2420 and CC1100.

Using these numbers and Friis range equation (equation 4.1) the range of each transceiver with a Fade margin (FM) of 20dB can be seen in figure 3.1. The benefits of working with lower frequencies can clearly be seen as the theoretical range of CC1100 is almost 3 times longer than the 2.4GHz transceivers.

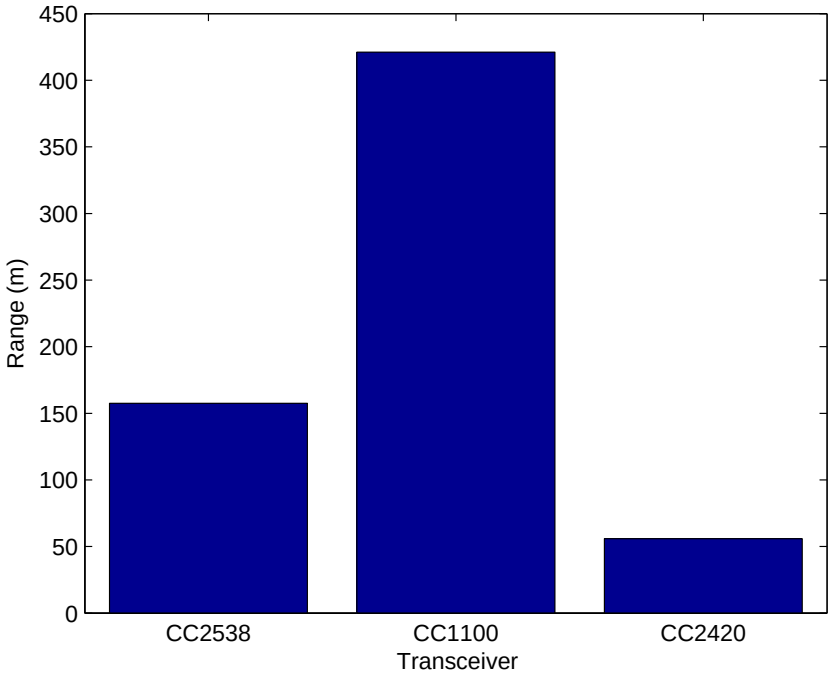


Figure 3.1: Range comparison with a FM of 20dB

Adding all this information together, the choice of platform will land on OpenMote with the CC2538SoC. It both has a MCU with more memory and better performance, and a transceiver with really good characteristics both in terms of energy consumption and range. Also OpenMote is the only option that can act as a border router using OpenBase; it lets the SoC interface with USB, UART, JTAG, and Ethernet, which enables the standalone border router mode without the need to be connected to a computer or other hardware. The OpenBattery extension lets the SoC operate as a node in a mesh network and provides a dual AAA battery slot connected to the PCB.

3.1.3 Operating system

As a modern operating system can be compiled to match almost any hardware, the most important thing to have in mind is the out-of-the-box hardware support. Only RIOT and Contiki have full support for the ARM

Cortex-M3 of the considered operating systems and thus both TinyOS and freeRTOS are directly eliminated as developing the support would take too much time. Compared to RIOT, Contiki also has full driver support for the sensors and transceiver, which should decrease the implementation time significantly. When compiled into binary form, RIOT uses less RAM and ROM and thus probably is a bit faster compared to Contiki, which could be important if the application consumes much resources. The lower memory usage might also give RIOT an advantage in being future-proof.

Contiki also has support for soft real-time scheduling compared to the hard real-time scheduling of RIOT; this is however not crucial, as the software that will be running on the OS does not have any hard real-time constraints.

Both RIOT and Contiki have support for 6LoWPAN but no support for either ZigBee or BLE; this is due to the fact that these are proprietary stacks. Support could be added but would take some time to customize for the given OS. What gives Contiki the largest advantage is that it also have border router software ready for deployment, which in the RIOT case would have to be developed.

All in all, as the project has such a limited time frame, Contiki will be selected as the OS; this, mainly because Contiki comes with most advantages time-wise; this choice means that the focus of the software development will be the creation of a test and evaluation system.

3.1.4 Communication protocol

The OpenMote platform has a IEEE 802.15.4 transceiver and thus supports both ZigBee and 6LoWPAN; this means that BLE is not an option. As ZigBee does not have full IPv6 support yet and is not integrated into Contiki, the natural choice is 6LoWPAN. This choice will not only save some development time but also enables evaluation of the header compression. As seen in figure 3.2, the 6LoWPAN stack in Contiki will replace the IP stack while maintaining the same functionality. As the functionality is the same, TCP and HTTP will work with 6LoWPAN, but including them in the source increases the OS build size considerably. On top of the UDP layer, Contiki also has a working implementation of CoAP that can be used for retrieving data from the nodes in a power efficient manner. CoAP is a stateless protocol that uses the HTTP response headers to achieve a very low overhead in transmissions while using application level reliability methods to ensure packet delivery.

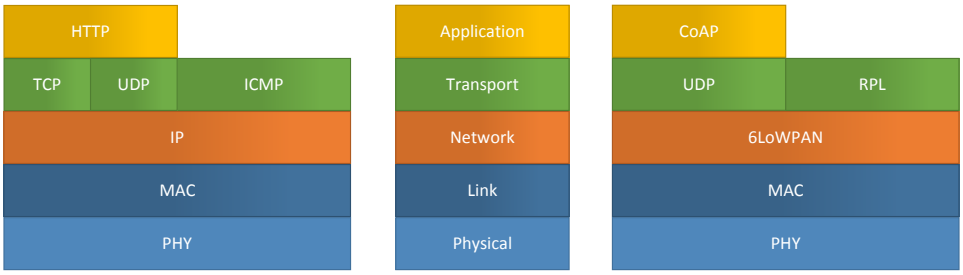


Figure 3.2: TCP/IP v.s. 6LoWPAN stack comparison

3.1.5 Workspace and tools

The ARM Cortex-M3 chip that OpenMote and CC2538 is built upon requires the GCC ARM Embedded compiler. This tool-chain is free and runs on both Linux, OSX, and Windows; however, there is no bundled development application so a secondary application for programming is needed. In Windows, there are several Integrated Development Environments (IDEs) such as IAR Workbench ARM [30], Code Composer Studio [31] and the Eclipse plug-in ARM DS-5 [32, 33]; these IDEs use various proprietary tool-chains and have a price tag ranging from free to several thousand SEK. Most of the IDEs also have a code size restriction for the free versions. To minimize the costs, the development machine development machine used in this project will run Ubuntu 14.04 LTS, the used tool-chain is GCC ARM Embedded, and Geany is used as the code development application. To analyse the network traffic in real-time, the open source tool Wireshark is used together with a IEEE 802.15.4 packet sniffer. Together with a laptop, the packet sniffer will grant the ability to traverse the mesh network and analyse the network in real-time as it is seen by the nodes.

3.2 Prototype Development

The goal of the development process is to have a functional border router and at least two nodes to be able to test how response time and throughput differs with each hop in a mesh network. To be able to measure response time and throughput, each node needs to have a CoAP server which can respond to ping and also receive an arbitrary amount of data for throughput measurement. It is desirable for each node to be able to send information about each sensor so the project can be used as a tech-demo.

The first part of the development was to set-up of the workspace and tools mentioned in section 3.1.5. Ubuntu OS was installed in a VirtualBox Virtual Machine to make it easier to duplicate and backup; this procedure gave a noticeable decrease in performance and it is recommended to have a dedicated native Ubuntu machine for this type of development. Even though

Ubuntu uses an easy-to-use package system, there were some problems in finding a version of GCC ARM Embedded tool-chain that was compatible with Contiki's built-in simulator Cooja [34, 35]; eventually, version 4.82 was used to successfully build Contiki. Cooja is a useful tool for testing and debugging network configurations but does not have support for the CC2538 MCU; instead, nodes called Cooja Motes are simulated with generic hardware. As Cooja is written in Java and runs in a JVM, Oracle Java 1.8 was also installed.

3.2.1 Drivers and firmware

Figure 3.3 shows an overview of the full system. The foundation is the SoC with the MCU, transceiver, and sensors. The Contiki operating system implements the soft real-time kernel together with the firmware for the SoC/MCU and the drivers for the peripherals and sensors. The last part is the communication stack, which provides TCP and UDP connectivity over 6LoWPAN. On top of the TCP and UDP protocols, HTTP and/or CoAP can be implemented.

The firmware required for the OS to work properly on the hardware platform was already implemented. However, the drivers for the I^2C bus and the sensors were not implemented. The I^2C driver is required for the sensor drivers which in turn enables the MCU to communicate with the sensors on the OpenBattery platform.

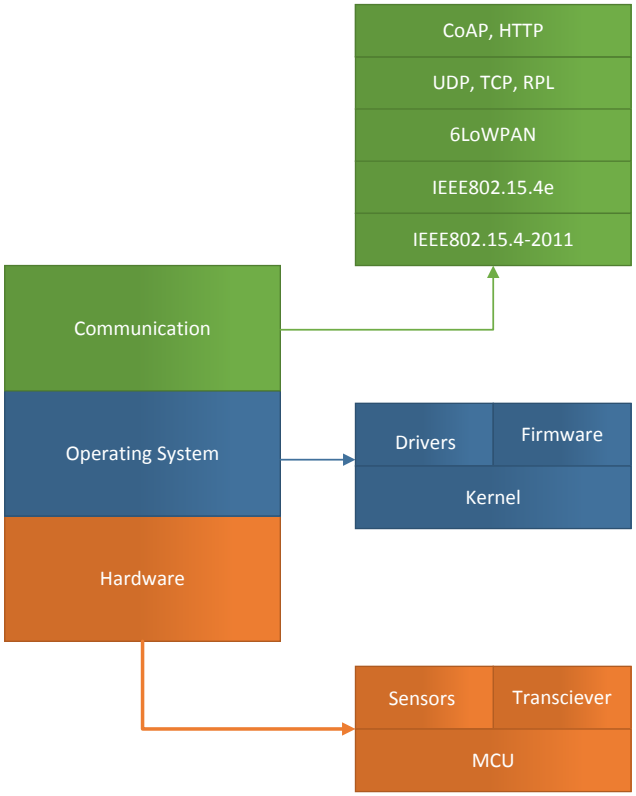


Figure 3.3: Contiki block overview

3.2.2 CoAP server

In order to make each node's sensor data accessible, a CoAP sever was implemented as an application running on top of Contiki. A CoAP server in general can handle any number of resources; in this implementation, one resource was made for each sensor value i.e. temperature, light, humidity, and core voltage. The temperature, light, and humidity sensors all work in a similar fashion. When their value is requested, the I^2C bus is initialized and then a request is sent over the bus. When the response with the value arrives, that data is put into either a plain-text or JavaScript Object Notation (JSON) formatted message depending on the request and then sent back to the requester. As the core voltage sensor is part of the MCU's ADC, that value is retrieved by simply getting data from a register (a somewhat faster operation).

As a buffer for testing throughput speed was also needed, a resource with a circular buffer was implemented. This resource is configured with CoAP's block-wise transfer functionality for arbitrary data size; however, the buffer in itself is only 1024Kb to allow the program variables to fit into the ultra low leakage SRAM. For testing purposes, the data could have been

discarded instead of actually saved into the buffer, but then the transfer can not be verified. Resources are defined by paths as CoAP works in a very similar way as HTTP.

Table 3.1: Resources and paths

Resource name	Resource path	Media type	Content type
Light	sensors/light	application/json	application/json
Temperature	sensors/temp	application/json	application/json
Humidity	sensors/humidity	application/json	application/json
Core Voltage	sensors/vdd	application/json	application/json
Circular Buffer	testing/buffer	text/plain	text/plain

Each resource is registered in the server with its path, media type, and content type. When a package arrives on the CoAP port, the server starts to break down the package to be able to direct it to the right resource. It starts with verifying that the package is actually a CoAP package, and then it checks the path and sends it to the correct resource. The resource then inspects the method field in the package header to direct the incoming data to the right function. CoAP package method can be either GET, PUT, POST or DELETE. This function then inspects the request media type and answer content type so that the function can parse the request and send a correctly formatted answer. If the resource does not implement the received method, the server responds with "405 - Method not allowed" and if the content/media type is not supported the answer is "415 - Unsupported Media Type". The content/media types are text/plain, application/json, application/exi, and application/xml.

3.2.3 Testing

Contiki is shipped with a simulation tool called Cooja which is written in Java; it can simulate an arbitrary number of nodes with different roles and configurations. All simulation data, such as radio packages and node serial output may be viewed through different windows and exported to various formats. Unfortunately Cooja did not have support for ARM Cortex-M3, but the general set-up was still tested by using Cooja Motes, which are nodes without specified hardware, and MSP430 nodes such as Wismote or Skymotes. With this simulator the basic understanding of the communication between nodes was gained; also, before the hardware arrived, early testing was performed to test the OS and application software.

3.3 Final prototype

The final prototype consists of four OpenBattery nodes and one OpenBase border router. Both the nodes and the border router are deployed with Contiki. Each node runs a CoAP server, described in section 3.2.2, on top of the OS in its own thread. The border router runs a router software called 6lbr that acts a as translator between Ethernet and IEEE 802.15.4 [36]. Both types of hardware are configured with a 8Hz Radio Duty Cycle (RDC) driver to keep the power consumption to a minimum. RDC is a OS driver that cycles the listening mode of the transceiver to reduce power consumption. As Contiki puts the MCU into Low Power Mode (LPM) when no function is running and the transceiver is off, the RDC driver indirectly controls when the MCU is in LPM. When using the RDC protocol, the nodes repeatedly send messages until the target node wakes up and sends an Acknowledge packet (ACK); this makes communication seamless, even though most of the time the nodes' transceivers are not active. Also, an always-on RDC driver, where the transceiver is constantly listening, will be used to be able to look at the performance impact of the 8Hz RDC.

Chapter 4

Assessment

In this chapter the results from each type of assessment are presented. The first assessment is range, followed by response time, after that connection speed, and finally the power consumption. The only assessment that is not performed on the prototype is the range assessment.

4.1 Range

Range is very hard to measure without advanced equipment and isolated rooms but can be roughly estimated with equation 4.1 called Friis range equation [37]. P_t is the sender transmit power, P_r the receiver sensitivity, d is the distance between the antennas in meters, f is the signal frequency in hertz, and λ is the wavelength. G_t and G_r is the antenna gain for the transmitter and the receiver. The last term in equation 4.1, when inverted, is the Free-space path loss (FSPL) and can be expanded as shown in equation 4.3.

$$P_r(dB) = P_t + G_t + G_r + 20\log_{10}(\frac{\lambda}{4\pi d}) \tag{4.1}$$

$$\lambda = \frac{c}{f} \tag{4.2}$$

$$\text{FSPL(dB)} = 20\log_{10}(d) + 20\log_{10}(f) - 147.56 \tag{4.3}$$

Unlike Friis range equation, the Link budget equation 4.4 also takes external loss like FM into account [38]. This is needed to make a correct estimation of the actual range as there are several things in the environment that obstructs and distorts the signal.

$$P_r = P_t + G_t + G_r - \text{FM} - \text{FSPL} \tag{4.4}$$

Combining equation 4.1, 4.3 and 4.4 gives us the equation for the estimated distance as seen in equation 4.5.

$$\begin{aligned} d &= 10^x \\ x &= \frac{P_t + G_t + G_r - P_r - FM + 147.56 - 20\log_{10}(f)}{20} \end{aligned} \tag{4.5}$$

With this equation an estimation of the transceiver range can be made for different FMs and transmit powers. When deployed, the transceiver is configured to only accept packages with a signal strength of -70dBm and above to minimize packet loss and corruption. The antenna gain for OpenMote is 0dBi and can thus be omitted. Figure 4.1 shows a comparison between three different levels of FM: 0dB, 10dB, and 20dB. A FM of 0dB means that there is no signal loss except the FSPL and this is very hard to achieve outside of a lab environment. When increasing the FM to 10dB, which corresponds to a normal home environment, the maximum range drops to 22m. However, in these kind of environments the desired range is usually around 10m which would let the device reduce the transmit power to around 0dBm. Finally, the FM is increased to 20dB which is roughly what it would be in a office or industrial environment. The maximum range in this environment is now reduced to only 7m when transmitting at maximum power.

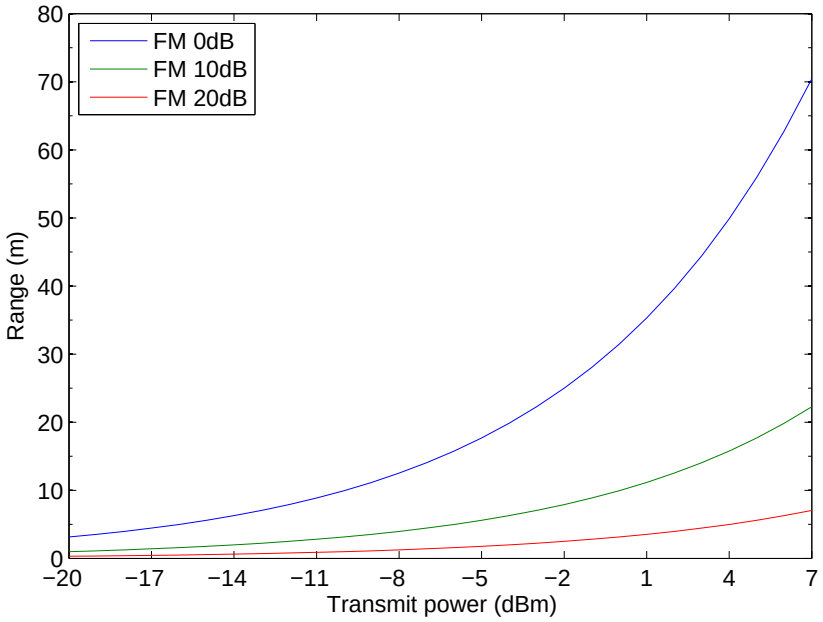


Figure 4.1: Comparison between theoretical ranges at different FMs with a receiver sensitivity of -70dBm

4.2 Response time

Before measuring the response time, some theoretical estimations are needed to be able to evaluate the real values. The theoretical values are based upon the radio duty cycle (RDC) and the average response time to reach a node can thus be derived from equation 4.6, 4.8, 4.9 and 4.10. As each node only

checks the radio every 125ms, this duration combined with the data packet send time of ∼4ms (equation 4.7.) and ACK send time corresponds to the worst case delivery, as the node needs to wait a whole cycle before being able to send the package the desired node. When the target node is already listening, the best case delivery time is 5ms. Thus, the average theoretical delivery time to reach any adjacent node is 67.5ms.

$$\text{Radio duty cycle: } \frac{1\text{s}}{8} = 0.125\text{s} = 125\text{ms} \tag{4.6}$$

$$\text{Transfer time: } \frac{133\text{B} + 4\text{B}}{31.25\text{KB/s}} \sim 4\text{ms} \tag{4.7}$$

$$\text{Worst case delivery: } 125\text{ms} + 4\text{ms} + 1\text{ms} \tag{4.8}$$

$$\text{Best case delivery: } 4\text{ms} + 1\text{ms} \tag{4.9}$$

$$\text{Avg. delivery: } \frac{130\text{ms} + 5\text{ms}}{2} = 67.5\text{ms} \tag{4.10}$$

The delivery time is only calculating the time to send a packet over a link, but when calculating the response time, the acknowledge (ACK) response has to be included in the calculation. Each ACK also needs to wait for the target node to be awake, adding one more instance of average delivery time, resulting in 125ms in average response time. This time will multiply with each hop, resulting in equation 4.11, 4.12 and 4.13.

$$\text{Avg. response time: } (2 \cdot 67.5\text{ms}) \cdot \mathit{hops} \tag{4.11}$$

$$\text{Best case response time: } (2 \cdot 5\text{ms}) \cdot \mathit{hops} \tag{4.12}$$

$$\text{Worst case response time: } (2 \cdot 130\text{ms}) \cdot \mathit{hops} \tag{4.13}$$

After doing a test with real nodes set-up with a 8Hz RDC with three hops, as seen in figure 4.2, the values in table 4.1 were obtained. Each node was pinged 200 times at a one minute interval to simulate some traffic on the network. What can clearly be seen in the average field of the table is that the average of 765ms is much higher than the expected average of 135ms; the difference is mainly due to the worst-case pings that in some cases had response times up to 30 seconds. However, when looking at the geometrical mean which is better at smoothing out big spikes seen in figure 4.3, the observed response time is still 265ms which is a bit longer than the expected worst case response time for one hop. Also, for two and three hops the observed average is high, but the geometrical mean shows that this is due to the spikes. The estimated response time for two hops is 270ms which as seen in the geometrical mean table 4.1 is way off by ∼500ms. The same observation goes for three hops where the observed geometrical mean response time is 1181ms which compared to the estimated response time of 405ms is significantly higher.

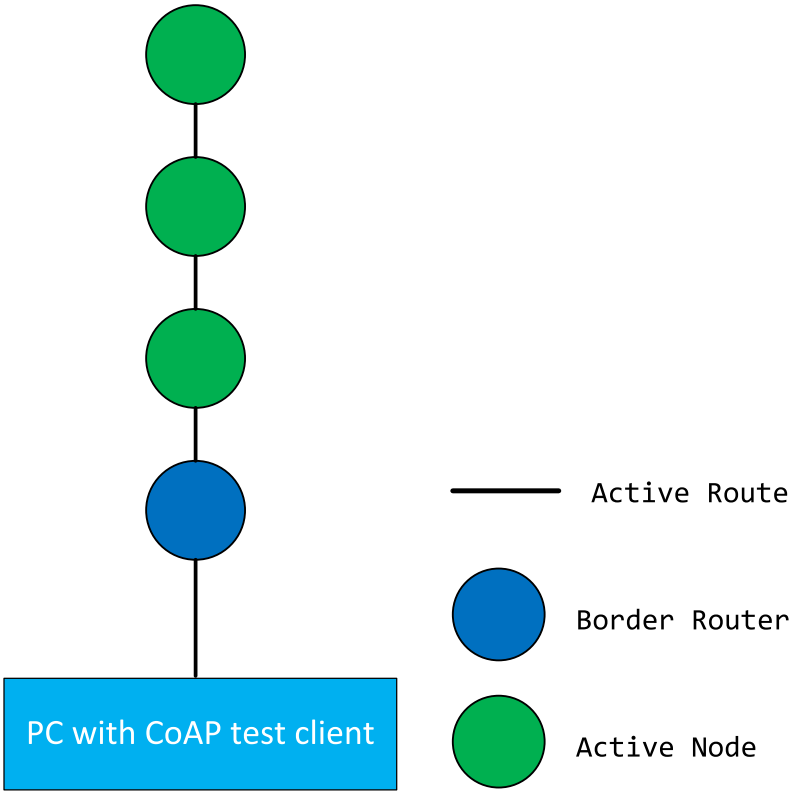


Figure 4.2: Assessment system set-up

Table 4.1: Response time for different hop levels using 8Hz RDC

Hops	Average	Mean	Geometrical mean	Worst	Best
1	765ms	199ms	265ms	34s	31ms
2	1762ms	577ms	763ms	37s	205ms
3	2339ms	932ms	1181ms	60s	313ms

With these observations in mind, the estimation could be described much better with equation 4.14. which would result in an average response time of 266, 532 and 1064 ms for one, two and tree hops. However, this would mean that the response time is doubled for one hop and then doubled for each consequent hop making the response time exponential which should not be the case.

$$\text{Alternative avg. response time: } (2 \cdot 67.5\text{ms}) \cdot 2^{hops} \tag{4.14}$$

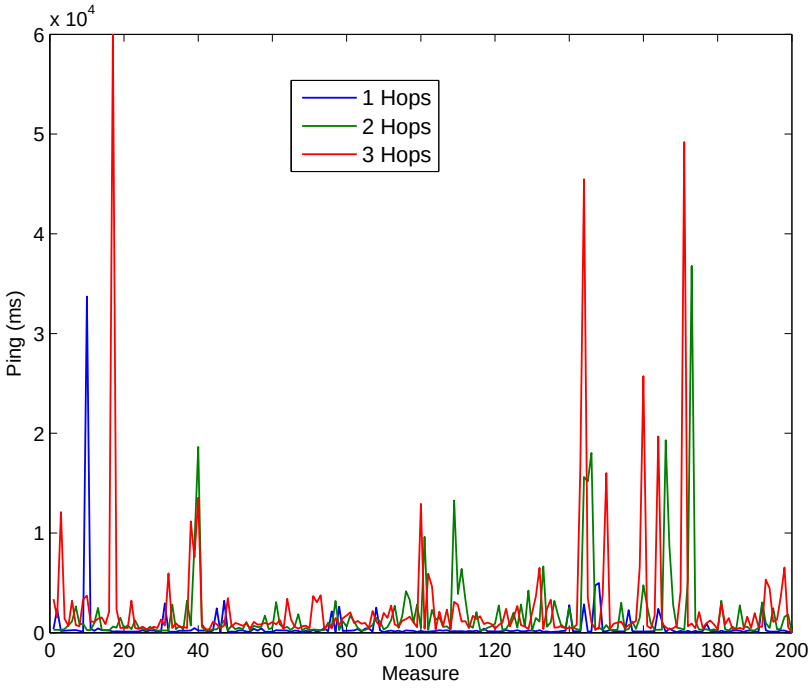


Figure 4.3: Plot of response times to nodes with different hop levels

With the RDC disabled, i.e. the transceiver is always listening and the MCU does not go into sleep mode, the response time is completely different. As seen in table 4.2 the average response time is around 12ms per hop and the spikes seen in the response time for 8Hz RDC is gone. Furthermore, the estimated best case response time of 10ms is very close to the observed average response time. The response time also scales to the number of hops as expected and is roughly 12ms per hop. It appears there might be a problem in the RDC driver as the response time there seems to be exponential.

Table 4.2: Response time for different hop levels when always listening

Hops	Average	Mean	Best
1	12ms	12ms	11ms
2	20ms	20ms	19ms
3	28ms	31ms	27ms

4.3 Connection speed

Connection speeds can be measured in several ways each with their own different pros and cons. One of the most popular ways is *throughput*, i.e. the amount of data over the link is divided by the time it took to reach the target. However, this gives a false picture of how fast the connection actually is from the developer's point of view, as the measured data does not only contain application data but also headers and checksums. IEEE 802.15.4 has a theoretical data rate of 250kb/s as seen in equation 4.15. but this is only a measure of how many bits per second the transceiver is able to output. The application data part, when using no header compression, is only 41% of the total transfer. Thus, resulting in a theoretical application data rate, also called *goodput*, of only 12.81KB/s.

$$\text{Data rate: } 250\text{kb/s} = 31.25\text{KB/s} \tag{4.15}$$

$$\text{Overhead: } \frac{133\text{B} - 54\text{B}}{133\text{B}} = 0.59 \tag{4.16}$$

$$\text{Theoretical goodput: } 31.25\text{KB/s} \cdot (1 - 0.59) = 12.81\text{KB/s} \tag{4.17}$$

When using CoAP as the application level protocol, each package can carry either 32 or 64 bytes of application data. In practice, the 64B mode is only applicable when sending packages between nodes on the same mesh network, as the addressing fields then can be fully compressed. When using applications outside the mesh network, each package can only carry 32B of data, resulting in a packet size of 111B as shown in equation 4.18; this does not affect the theoretical data rate but has a noticeable impact on the goodput due to the large overhead of 71%, as shown in equation4.19. To be able to use the full data rate, the application needs to use a protocol without handshakes, i.e. UDP, as the transceiver then can send the packets as fast as physically possible. CoAP is implemented on top of UDP and thus has a low transport layer overhead, but uses its own mechanism for handshaking, delivery and ordering. The theoretical CoAP application throughput can be estimated by looking at the average response time of the node and then add that time to the the data delivery time. Each package needs to be acknowledged before the next package is sent, and thus the node response time needs to be taken into consideration. When doing so, the throughput as calculated in equation 4.20. is only 1.64KB/s and thus the theoretical goodput is reduced from 12.81KB/s down to 0.48KB/s as shown in equation 4.21.

$$\text{Packet size: } 133\text{B} - 54\text{B} + 32\text{B} = 111\text{B} \tag{4.18}$$

$$\text{Actual overhead: } \frac{79\text{B}}{111\text{B}} = 0.71 \tag{4.19}$$

$$\tag{4.20}$$

$$\text{Theoretical goodput: } 1.64\text{KB/s} \cdot (1 - 0.71) = 0.48\text{KB/s} \tag{4.21}$$

Using the packet size of 111B together with the theoretical response time from equation 4.11 would give the results shown in figure 4.4. To verify these calculations, the same test set-up as shown in figure 4.2, which also was used in section 4.2, was used to test throughput and goodput at different number of hops. Each node was sent 1KB data each minute for 200 minutes; the time from the first package sent to the final acknowledge packet received was measured for each 1KB transmission.

The first test was performed with an RDC of 8Hz and resulted in the values shown in figure 4.5. As the chart shows, the theoretical throughput and goodput is much higher than the observed values, but this is due to the fact that the actual average response time is higher than the theoretical one. With some calculations made the observed throughput and goodput are within range of what is expected, given the observed response times in table 4.1. Equation 4.22 uses the observed values to calculate the average response time, given the values in figure 4.5.

Response time from throughput:

$$\frac{111\text{B} \cdot \text{Ceil}\left(\frac{1000\text{B}}{32\text{B}/pkg}\right)}{0.27\text{KB/s}} = 13.16\text{s} \Rightarrow \frac{13.16\text{s}}{\text{Ceil}\left(\frac{1000\text{B}}{32\text{B/pkg}}\right)} \approx 411\text{ms} \tag{4.22}$$

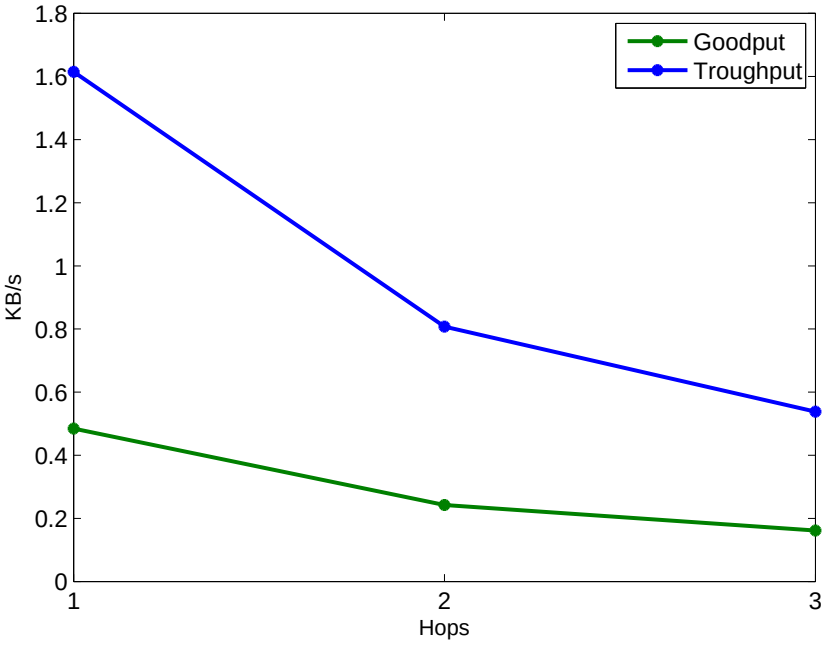


Figure 4.4: Theoretical throughput and goodput vs number of hops with an RDC of 8Hz.

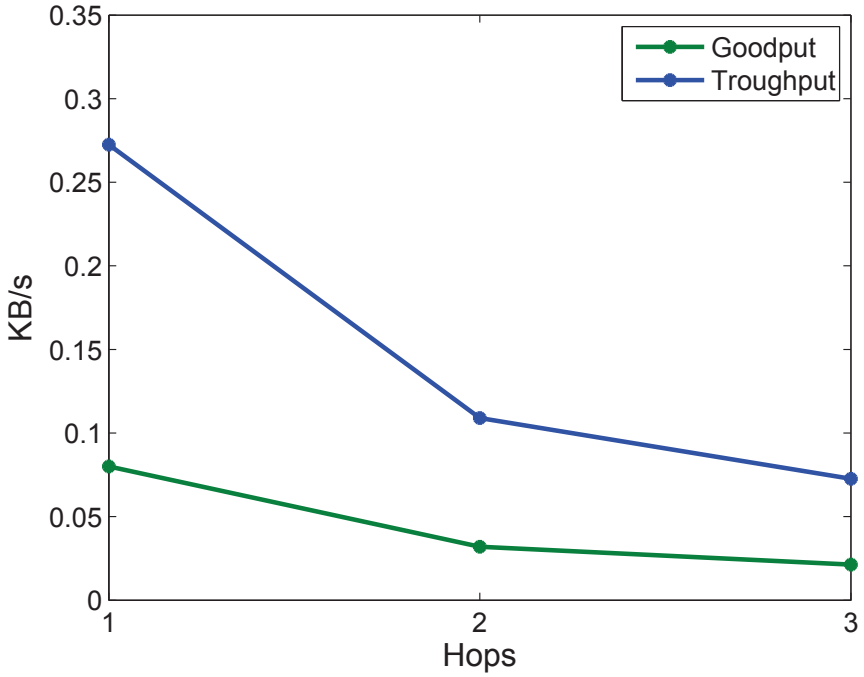


Figure 4.5: Practical throughput and goodput vs number of hops with an RDC of 8Hz.

Using the same test set-up as in the previous test but with RDC disabled, i.e. letting the transceiver always listen and disabling the sleep modes for the MCU, give the results seen in figure 4.6. This can be verified with the same method as before and when compared to the actual values from table 4.2, as seen in equation 4.23, the values are consistent.

$$\begin{aligned} 111\text{B} \cdot \text{Ceil}\left(\frac{1000\text{B}}{32\text{B}/pkg}\right) &= 3552\text{B} \\ 12\text{ms} \approx \frac{\frac{3552\text{B}}{9\text{KB/s}}}{32\text{B}/pkg} &= 12.3\text{ms} \\ 20\text{ms} \approx \frac{\frac{3552\text{B}}{5.5\text{KB/s}}}{32\text{B}/pkg} &= 20.2\text{ms} \\ 28\text{ms} \approx \frac{\frac{3552\text{B}}{3.9\text{KB/s}}}{32\text{B}/pkg} &= 28.5\text{ms} \end{aligned} \tag{4.23}$$

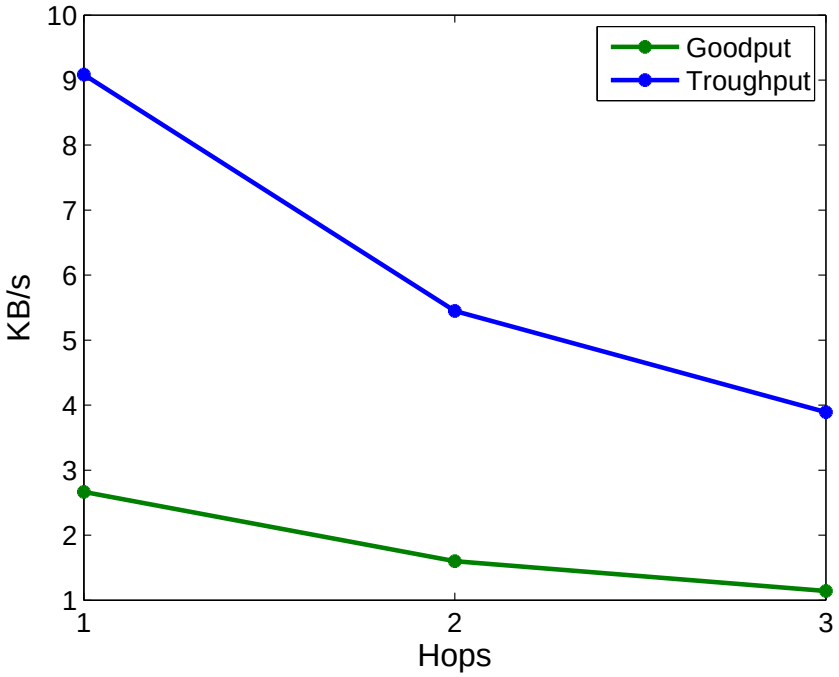


Figure 4.6: Practical throughput and goodput vs number of hops with an always-on RDC.

4.4 Power consumption

To measure power on devices that use very low power and also changes the power consumption very rapidly and frequently is not an easy task. According to the currency specification from the CC2538, the different power modes have the consumption seen in table 4.3 using the built in voltage regulator TSP6750 that switches the input voltage down from the 3V to 2.2V. The components on the OpenBattery supplied directly by the 3V batteries have the current and power consumption specifications as seen in table 4.4.

Table 4.3: Current and power consumption for OpenMote @ 2.1V

Mode/Component	Current	Power
MCU	13mA	27.3mW
TX @ 0dBm	24mA	50.4mW
TX @ 7dBm	34mA	71.4mW
RX	20mA	42mW
PM1	0.6mA	1.26mW
PM2	1.3μA	2.73μW
Sleep Timer (ST)	0.9μA	1.98μW
TSP6750 @ 3V	34μA	102μW

Table 4.4: Current and power consumption for OpenBattery @ 3V

Component	Current	Power
MAX44009	1.6μA	4.8μW
SHT21	0.5μA	1.5μW
ADXL346	0.2μA	0.6μW

This gives a base power consumption of $102 + 4.8 + 1.5 + 0.6 = 107.4\mu$W by adding the constant component consumption. Combined this adds up to the power profiles in table 4.5.

Table 4.5: Power profiles for the different operating modes of OpenBattery and OpenMote

Mode	Active parts	Power consumption
Measuring	MCU+Base	27.3mW + 107.4μW = 27.41mW
RX	RX+Base	42mW + 107.4μW = 44.11mW
TX @ 0dBm	TX+Base	50.4mW + 107.4μW = 50.51mW
TX @ 7dBm	TX+Base	71.4mW + 107.4μW = 71.51mW
Sleep	PM2+ST+Base	2.73μW + 1.98μW + 107.4μW = 112.81μW

Given these power profiles, combined with the time it takes to receive and transmit packages, and retrieve a measurement, the theoretical power for one RDC cycle results in the chart seen in figure 4.7. The node starts in sleep mode using 112.81μW and after 109ms wakes up and goes into RX mode where a request for a sensor value is received. The node then switches off the radio and fetches the sensor value. After the value is retrieved from the sensor, the radio is once again put in to RX mode for a Channel Clear Assessment (CCA) before entering TX mode and sending the payload. The transmission is successful and the node goes into RX mode to listen for the ACK, when it is received the node enters sleep mode again. For this cycle the average power consumption is 4.8mW which would drain the 2250mWh batteries in 19 days.

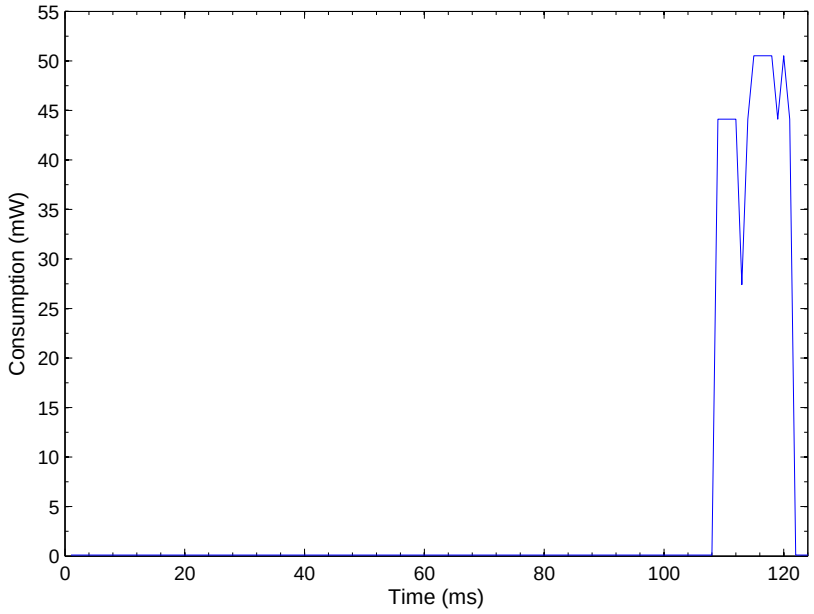


Figure 4.7: OpenBattery send and measure power cycle

However, as the nodes have a RDC running at 8Mz, most of the time there will be no package for the node to receive and thus no measuring and transmitting, as seen in figure 4.8. This cycling reduces the average power consumption to 0.47mW, which would make the batteries last for ca 200 days. The goal is to have a node that can run for one year without having to change the batteries and to be able to do this on 2xAAA batteries with 750mAh the average consumption has to be under 257μW as calculated in

equation 4.24.

$$\begin{aligned} 750\text{mAh} \cdot 3\text{V} &= 2250\text{mWh} \\ 2250\text{mWh}/(365\text{d} \cdot 24\text{h}) &= 257\mu\text{W} \end{aligned} \tag{4.24}$$

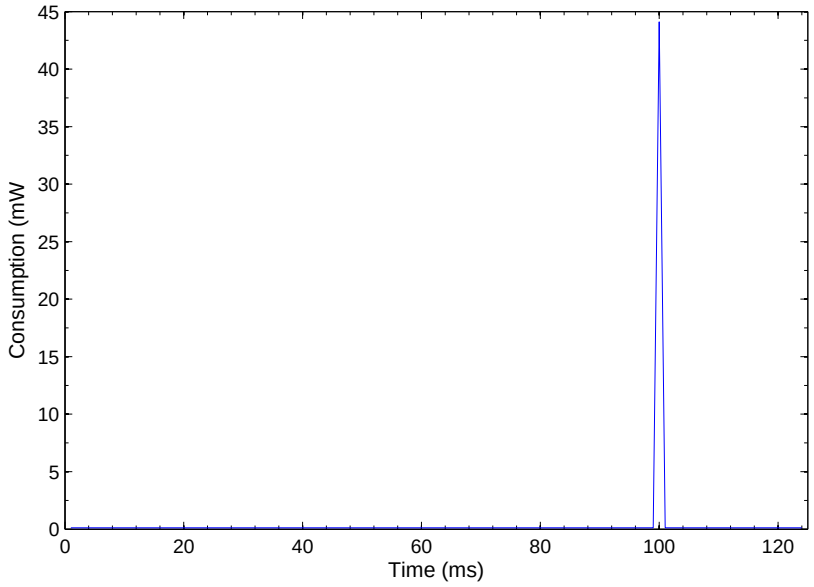


Figure 4.8: Theoretical OpenBattery power cycle

To verify these assumptions, we used a Keithley 2280S power supply [39] to measure the total current draw of the prototype. The node was connected to the power supply, which was set up to make 277 measures each second with a supply voltage of 3V. Several measurements were performed. One of the most interesting ones can be seen in figure 4.9. In this picture, we can clearly see the different operating modes, as the node performs 3 transmissions during the interval. In the first transmission at the 1.6s mark, the strobing feature of the RDC protocol is seen as the package is sent 5 times before the receiving node is awake and can receive the package. In the two following transmissions, the package is delivered on the first try. As our measurement is limited to 277Hz, the current peaks when only waking up to listen for traffic are sometimes missed, and the peak value is hard to extract; but the 8Hz RDC cycle is still visible. The average power consumption for these cycles is 8mW, which would make the batteries only last for 11 days. However, when taking the average of a measuring series without any transmissions, the average goes down to 4mW, which increases the battery time to 23 days. The theoretical sleep power of 0.11mW compared to the measured of 3mW is what makes the average power consumption that high.

Reducing this power consumption by a tenfold would result in an average consumption of 0.39mW, which is closer to the theoretical average power consumption.

A discovery made when measuring the power was that the nodes consumed less power when supplied with a lower input voltage. Simply by reducing the voltage from 3V to 2.6V reduced the power consumption in LPM by 15%. However, this reduction could affect the range of the nodes.

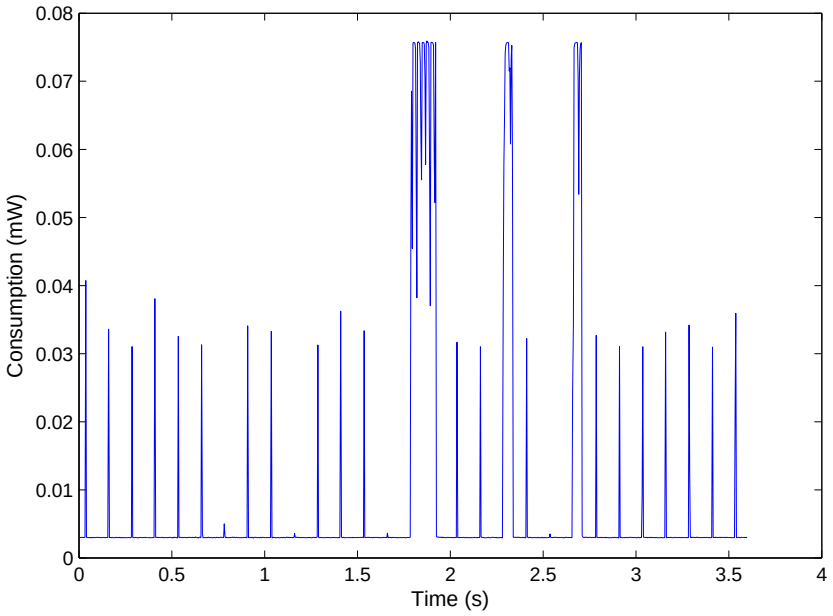


Figure 4.9: Power measurement of OpenBattery node for 3.6 seconds at 277Hz

Chapter 5

Discussion

Internet of Things can be realised in several ways as there are still many viable options on the market, mainly in terms of hardware, operating systems, and communication standards.

Given the recent development in the field, Thingsquare recently released a technology demo using the same practices as used in this thesis; the choices taken are on track with the latest development [40]. Also, both Google and Microsoft have announced that they are developing IoT OSs. When these products are released, it would be very interesting to compare them with Contiki. It would be exciting to see if an open-source project can surpass the commercial offerings in terms of speed, RAM and ROM footprint, and device support. Furthermore, an in-depth comparison between RIOT and Contiki would give much insight into the kind of OS practices that benefit IoT development the most.

Google have also started to develop a substitute for 6LoWPAN and UDP that they have named Thread [41]. As 6LoWPAN and ZigBee, it runs on top of IEEE 802.15.4 and thus might be able to out-compete the existing implementations. Google promises lower latencies and power consumption compared to the existing technologies.

5.1 The prototype

The prototype development took more time than initially planned; mostly because of the complexity of the OS, but also due to bugs in the untested drivers. The prototype combines the technology from each field, i.e. hardware, OS, and communication protocol, and fulfils the requirements set in section 3.1.1. Even though the OS is relatively simple, compared to Linux, Windows, and OS X, understanding the mechanics of the RDC driver and the LPM driver was difficult, but necessary to be able to interpret the test results. The prototype worked very well during most of the testing, with only a few unforeseen deviations. One occurred during the power measure-

ment, where the power consumption in low power mode tripled in one of the test series; this behaviour could not be reproduced and is therefore not included in the results. Also, in the early stages when working with the 8Hz RDC driver, packet losses over 50% were recorded for packets with more than one hop; this problem was solved, when a new version of radio driver was released by the OS development team.

Selecting OpenMote to be the hardware platform together with Contiki as the OS, was a very good choice as companies are starting to build their IoT solutions around Contiki and similar hardware platforms [42, 43]. Already in the beginning of the development, several benefits were noticed; new drivers and bug-fixes were released increasing the stability and functionality of the OS. The active community around the combination of OpenMote and Contiki was really helpful when developing the drivers for the I^2C and sensor drivers. Example projects for other platforms could be used as references, giving much insight to how the programming for this type of OS worked.

It would have been interesting to examine the differences between two operating systems; not only to test which one has the better performance, but also to compare which one that has the more favourable code structure and development procedure.

5.2 Results

Collecting the data went well and were reasonably straight forward; it was easy to transition between the two different test set-ups and thus making several test scenarios. Assessments were made in the areas of range, response time, connection speed, and power consumption. In each area, the theoretical values were first calculated and then compared to the retrieved measurements; except in the range case, as the required equipment for measuring was not economically justifiable to purchase.

Range

The theoretical range for OpenMote when transmitting at full power in an office environment is only 7m. As measuring the range was not a viable option due to the cost of measuring equipment, only distance estimations from the placement of the nodes when maintaining a stable connection can be used as a reference. Using a map of the office and the position of the nodes the range seems to be around 10m, which would mean that the effective FM of the office is around 16dB using the always-on RDC. The FM changed a bit when using the 8Hz RDC as more packages congested the air and the range dropped to somewhere around 5m; resulting in an effective FM of $\sim$23dB.

To increase the range of the transceiver, a switch to the 860MHz frequency band would be the most effective solution; with a FM of 23dB, the

theoretical range would increase to 14m with the same transceiver properties, and with a FM of 16dB the range would be 31m. Usually, transceivers with a lower frequency output also have a lower power consumption while transmitting. Working in sub-GHz also gives the benefit of less interference as fewer other devices uses those frequencies. Changing to a sub-GHz band would thus decrease the power consumption and increase the range, without changing the functionality of the nodes.

Response time

Initially when measuring the response time the always-on RDC was used and the measured response time was very close to the theoretical value. However, when using the 8Hz RDC protocol the values started to drastically differ from the theory. This behaviour is likely to originate from the way the RDC driver predicts the next time when the target node should be awake. The procedure is called *phase optimization*; when enabled, the node saves the time when the node was last seen, it then uses this value to predict the next time the node should be awake based on the RDC cycle. However, this prediction is based on the node's internal clock. As the clock can differ from those of the other nodes, misalignments seem to occur, resulting in misses when trying to reach the target node. Each misalignment increases the time it takes to reach the target node as the node then needs to strobe the package until the target nodes wakes up again. In theory, when sending strobes the target node should wake up and receive the package within one cycle (125ms); however, this is not guaranteed as other transmissions might occupy the air, further increasing the response time. If the phase optimization could be improved to guarantee the alignment between the nodes, the response time should get much closer to the theoretical value; as the time to reach the node would be maximum one cycle and the air would not be as congested by nodes sending strobes.

Connection speed

The connection speed, when using CoAP or any other protocol with per-packet ACK, is directly bound to the response time. IEEE 802.15.4 has a relatively low data-rate, only 250kbps, compared to other solutions, e.g. BLE (1Mbps) and WLAN (>54Mbps). As throughput is based on data-rate over a longer period of time, both the overhead and the response time is needed to make a good estimation. CoAP has a very low header size compared to many other communication protocols, but due to the very small frame size, the overhead is still relatively high. As of now, the results clearly show that when a reliable transfer is desired the connection speed of IEEE 802.15.4 and CoAP is only sufficient for data exchanges around 32 bytes. When the nodes use the always-on RDC, the goodput is less than

3KB/s for one hop and is halved for every hop; however, when the 8Hz RDC is enabled, the goodput is reduced to under 0.1KB/s.

Using messages without per-packet ACK, thus removing the response time from the equation, would let the nodes transfer real-time audio and maybe even highly compressed video. However, using messages without the per-packet ACK disables the reliable transmission guarantee, and thus it can only be used with data streams where packet loss is acceptable.

Power consumption

Making a rough estimation of the power consumption of the platform was straight forward task and so was measuring the actual consumption. When comparing, the two the values differed by a factor of 30, which was not expected. The reason probably originates from the clock interrupt which is triggered every 8ms. Initially, this interrupt was assumed to be disabled when the system entered the lower power modes, but this was not the case. As the interrupt fires at 125Hz and the time to wake up and go back to LPM is only 272μs, the power spikes from these interrupts were not seen on the measuring instruments. As seen in figure 4.9, even the peaks from the listening cycles were hard to record and those lasted for at least 4ms; instead, the power consumption from the clock timer looks like an increased LPM power consumption. At the time this was discovered there was no time to fix it, but doing so should decrease the average power consumption to within the limits, granting the nodes the ability to run on battery power for a year.

As no delays from calculation could be observed, the clock speed on MCU could, in all probability, have been reduced to save power on the nodes. However, this reduction would only have affected the consumption when the node was in active mode, which is only a few percent of the total cycle time.

The OpenMote chip has a step-down DC-DC converter for this purpose which is switched off in LPM mode to reduce quiescent currents; however, as most of the time is spent in LPM, reducing the input voltage to 2.1V by changing battery type and removing the step-down converter would be preferable as it would reduce the power consumption. These changes could affect the range of the device, but this has to be assessed.

5.3 Project execution

Looking at the time plan and the milestones, as seen in Appendix A and B, each milestone matches a task or transition in the time plan. The planning report was not submitted to the examiner until the 6/2-15, which is two weeks behind schedule, exceeding the time planned for milestone M1. The first draft was submitted before deadline, but several revisions were

necessary. In retrospect, the litterature study should probably have been planned in parallel with the planning report, as the information from the study helped with the report.

Milestone M2 marks the switch from the literature study and selection of technology to the development phase. This milestone was met and development could begin in the following week. As seen in Appendix C, the development phase have several risks to consider. The only risk encountered in this phase was R4, as one of the hardware platforms was delivered with a broken sensor. However, this malfunction did not affect the time plan as the development could continue regardless of the malfunction.

The end of the development phase was defined by milestone M3, approval of prototype, which was completed ahead of schedule granting an early transition into the assessment phase. In the assessment phase, it could be argued that risk R9 was encountered when measuring the power consumption, as the results from those measurements did not properly show the wake-ups from the clock timer. This phase contained milestone M4 and M5, of which of only M5 was done in time. The Half-time presentation, milestone M4, was performed on the 8/4-15 in the form of a meeting, where the progress, results and continuation plan were discussed. Also, a half-time version of the report was sent the 17/4-15 and approved by the examiner. Milestone M6, deliver the final prototype, was completed a few days before the set deadline which eliminated risk R11 and gave more time to work on the writing and the presentation.

Both of the oral presentations were attended on the 1/6-15 to grant some experience in how the presentation and opposition are carried out, thus now following the time plan. However, there were not many presentations to watch during the planned weeks, as the presentation schedule follow the academic semesters. The presentation for this thesis was not performed until the 3/6-15, thus being two weeks behind schedule. However, it was scheduled on the first available date suggested by the institution. The final version of the report will be submitted to the examiner before the 19/6-15, thus successfully completing milestone M7.

Chapter 6

Conclusion

The purpose of the project was to find and examine a communication protocol that could be suitable for IoT applications, by investigating the current hardware, OS, and communication protocols and building a prototype from the selected choices. What can be said about the investigation is that it is difficult to examine all candidates in detail; this means that a rough selection has to be made based on initial knowledge potentially discarding good options. The general feeling is, however, that all of the examined candidates in this project were relevant and added valuable insights to the current technology status.

The assessment gave relevant and interesting results that improved the understanding in what IoT can be used for, and what further areas of investigation could be. One of the most interesting areas of further investigation would be the RDC driver, as it directly affects the response time and thus also the connection speed. Even though the power consumption was not in line with the expectations, the reason has been found and can be resolved. Another conclusion is that IoT is not ready for real-time applications as the latency is much higher than expected, for the technologies assessed in this thesis, and also has a high spread. As the latency increases for each subsequent network hop and the minimum observed latency per hop is 11ms, when using the always-on RDC, this type of communication will probably only be used for applications where response time can vary greatly, without affecting the functionality.

CoAP as a communication protocol shows a lot of promise when combined with 6LoWPAN and IEEE 802.15.4. It performs well given its simplicity but has one disadvantage: the large overhead which comes from the MAC addressing fields in the IEEE 802.15.4 frame. If this overhead could be reduced from the current 71% to only 30%, the goodput would double. A solution would be to use a similar mechanism as BLE where the packet size varies depending on application.

Each node also has computing time left as the MCU is more powerful

than needed for the given application; an improvement would be to use a less powerful MCU, like the ARM Cortex-M0+, to reduce the clock speed as suggested in the discussion. When looking at the future-proof aspect the later suggestion is probably the better, as the clock then could be increased if more computing power is needed. In the future, batteries will hopefully be able to store more energy, thus increasing the time between battery changes or reducing the battery size.

Chapter 7

Bibliography

[1] S. C. Mukhopadhyay, *Internet of things: challenges and opportunities*, vol. 9.; 9, pp. 1–7. Cham: Springer, 2014.

[2] A. Marqués and M. Serrano, *The PECES Project: Ubiquous Transport Information Systems*, pp. 359–366. New York, NY: Springer New York, 2010.

[3] C. Buratti, A. Ferri, and R. Verdone, *An IEEE 802.15.4 Wireless Sensor Network for Energy Efficient Buildings*, pp. 329–338. New York, NY: Springer New York, 2010.

[4] J. Daintith, *Oxford Dictionary of Physics*. Oxford University Press, April 2010.

[5] E. Wilde, C. Pautasso, and R. Alarcón, *REST: advanced research topics and practical applications*, pp. 27–29. New York: Springer, 2014.

[6] O. Hersent, D. Boswarthick, and O. Elloumi, *IEEE 802.15.4*. Chichester, UK: John Wiley & Sons, Ltd, 2011.

[7] Z. Shelby and C. Bormann, *6LoWPAN: the wireless embedded internet*. Chichester, U.K: Wiley, 2009.

[8] *OSI Model-Chapter 5*, pp. 247–315. Elsevier Inc, 2005.

[9] J.-P. Vasseur and A. Dunkels, *Interconnecting smart objects with IP: the next Internet*. Burlington, MA: Morgan Kaufmann/Elsevier, 2010.

[10] G. Fortino and P. Trunfio, *Internet of Things based on smart objects: technology, middleware and applications*. Cham: Springer, 2014.

[11] Y. Li, M. T. Thai, and W. Wu, *Wireless sensor networks and applications*. New York; London: Springer, 2008.

[12] Thingsquare, "Contiki OS homepage." http://www.contiki-os.org/index.html, 2015. [Online], Accessed May, 28, 2015.

[13] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, "Protothreads: simplifying event-driven programming of memory-constrained embedded systems," pp. 29–42, ACM, 2006.

[14] Imprint, "RIOT OS features." http://www.riot-os.org/#features, 2015. [Online], Accessed May, 28, 2015.

[15] P. Levis, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, E. Brewer, and D. Culler, *TinyOS: An Operating System for Sensor Networks*, pp. 115–148. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

[16] S. S. Iyengar, N. Parameshwaran, V. V. Phoha, N. Balakrishnan, and C. D. Okoye, *Tiny Operating System (TinyOS)*, pp. 92–97. Hoboken, NJ, USA: John Wiley and Sons, Inc.

[17] Real Time Engineers ltd., "freeRTOS features." http://www.freertos.org/FreeRTOS_Features.html, 2014. [Online], Accessed May, 28, 2015.

[18] OpenMote Technologies, "OpenMote features." http://www.openmote.com/hardware/openmote-cc2538-en.html, 2015. [Online], Accessed May, 28, 2015.

[19] Texas Instruments Incorporated., "CC2538 Powerful Wireless Microcontroller System-On-Chip for 2.4-GHz IEEE 802.15.4, 6LoWPAN, and ZigBee Applications." http://www.ti.com/lit/ds/swrs096d/swrs096d.pdf, April 2015. [Online], Accessed May, 28, 2015.

[20] OpenMote Technologies, "OpenBase fetures." http://www.openmote.com/hardware/openbase.html, 2015. [Online], Accessed May, 28, 2015.

[21] OpenMote Technologies, "OpenBattery features." http://www.openmote.com/hardware/openbattery.html, 2015. [Online], Accessed May, 28, 2015.

[22] Thingsquare, "Thingsquare homepage." http://www.thingsquare.com/, 2015. [Online], Accessed May, 28, 2015.

[23] Freie Universität Berlin, "MSB430 features and specification." http://www.mi.fu-berlin.de/inf/groups/ag-tech/projects/Z_Finished_Projects/ScatterWeb/modules/mod_MSB-430H.html, 2015. [Online], Accessed May, 28, 2015.

[24] Texas Instruments Incorporated., "MSP430F15x, MSP430F16x, MSP430F161x Mixed Signal Microcontroller." http://www.mi.fu-berlin.de/inf/groups/ag-tech/projects/Z_Finished_Projects/ScatterWeb/moduleComponents/msp430f1612.pdf, May 2009. [Online], Accessed May, 28, 2015.

[25] Zoletia, "Zolertia Z1 Datasheet." http://zolertia.com/sites/default/files/Zolertia-Z1-Datasheet.pdf, March 2010. [Online], Accessed May, 28, 2015.

[26] M.-P. Uwase, N. T. Long, J. Tiberghien, K. Steenhaut, and J.-M. Dricot, "Poster abstract: Outdoors range measurements with zolertia z1 motes and contiki," vol. 281, pp. 79–83, 2014.

[27] D. Gislason, *ZigBee wireless networking*. Oxford: Newnes, 2008.

[28] N. C. Gupta, *Inside Bluetooth Low Energy*. Boston: Artech House, 2013.

[29] J. Nieminen, T. Savolainen, M. Isomaki, Nokia, B. Patil, AT&T, Z. Shelby, Arm, C. Gomez, and U. P. de Catalunya/i2CAT, "IPv6 over BLUETOOTH(R) Low Energy draft-ietf-6lo-btle-13." https://tools.ietf.org/pdf/draft-ietf-6lo-btle-13.pdf, 2015. [Online], Accessed May, 28, 2015.

[30] IAR Systems, "IAR Embedded Workbench for ARM." https://www.iar.com/iar-embedded-workbench/arm/, 2015. [Online], Accessed May, 28, 2015.

[31] Texas Instruments Incorporated., "Code Composer Studio (CCS) Integrated Development Environment (IDE)." http://www.ti.com/tool/ccstudio#descriptionArea, 2015. [Online], Accessed May, 28, 2015.

[32] ARM Ltd., "ARM DS-5 Features." http://ds.arm.com/ds-5/, 2015. [Online], Accessed May, 28, 2015.

[33] "ARM Launches DS-5 Professional Edition and ARM Compiler V5.0," *Technology News Focus*, p. 180, 2011.

[34] F. Österlind, J. Eriksson, and A. Dunkels, "COOJA TimeLine: a power visualizer for sensor network simulation," pp. 385–386, ACM, 2010.

[35] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-Level Sensor Network Simulation with COOJA," pp. 641–648, IEEE, 2006.

[36] Cetic, "6lbr border router software." http://cetic.github.io/6lbr/, 2015. [Online], Accessed May, 29, 2015.

[37] J. A. Shaw, "Radiometry and the Friis transmission equation," *American Journal of Physics*, vol. 81, no. 1, p. 33, 2013.

[38] M. Tolstrup, *The Link Budget*, pp. 329–344. Chichester, UK: John Wiley and Sons, Ltd.

[39] Keithley Instruments, "Series 2280S Precision Measurement, Low Noise, Programmable DC Power Supplies." `http://www.keithley.com/data?asset=58094`, 2015. [Online], Accessed May, 28, 2015.

[40] Thingsquare, "Private Beta Launched." `http://www.thingsquare.com/blog/articles/private-beta/`, 2015. [Online], Accessed June, 8, 2015.

[41] Thread Group, "About Thread." `http://threadgroup.org/About.aspx`, 2015. [Online], Accessed June, 8, 2015.

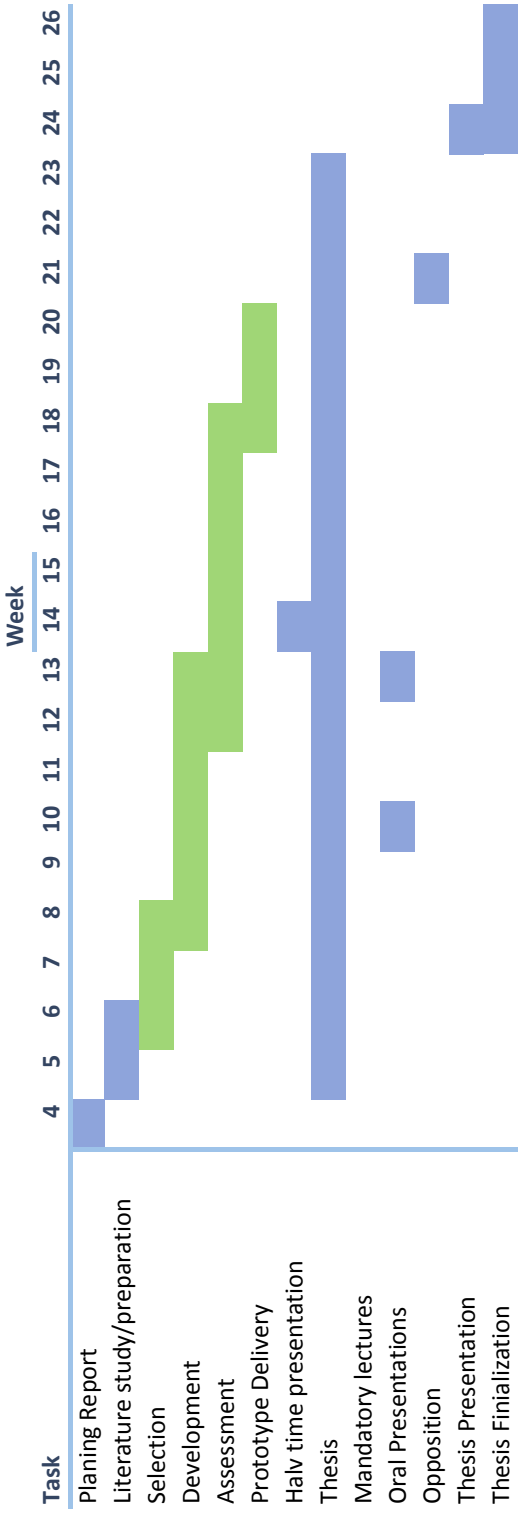
[42] LiFi Labs, Inc., "LIFX." `http://www.lifx.com/`, 2015. [Online], Accessed September, 1, 2015.

[43] Alex Chapman, "Hacking into Internet Connected Light Bulbs." `http://www.contextis.com/resources/blog/hacking-internet-connected-light-bulbs/`, 2014. [Online], Accessed September, 1, 2015.

Appendices

Appendix A

Gantt chart



Appendix B

Milestones

Milestones	Explanation	Responsible	Approved by	Deadline
M1 - Submission of planning report	Planning report submitted for approval	Johan Bregell	Lars Svensson	23/1 - 15
M2 - Approval of technology choices	Select, present and get the selected technology approved	Johan Bregell	Peter Olsson	20/2 - 15
M3 - Approval of prototype	Demonstrate the prototype and get it approved for assessment	Johan Bregell	Peter Olsson	27/3 - 15
M4 - Half-time presentation	Perform a half time presentation	Johan Bregell	Lars Svensson	3/4 - 15
M5 - Approval of assessment results	Present and get the findings and results of the assessment approved	Johan Bregell	Lars Svensson, Peter Olsson	1/5 - 15
M6 - Deliver final prototype	Deliver the final prototype to i3tex	Johan Bregell	Peter Olsson, Håkan Rolin	15/5 - 15
M7 - Final Academic Approval	Oppose on a fellow student's report, Perform the final presentation, Submit the final report to the institution	Johan Bregell	Lars Svensson, Per Larsson-Edefors	19/6 - 15

Appendix C

Risk Analysis

Risk Description	Milestone	Risk Level
R1 - The current level of technology is not sufficient to complete the task	M2	Moderate
R2 - The available technology does not meet the requirements	M2	Low
R3 - The desired technology is not in stock	M2	Considerable
R4 - Hardware malfunction and need to be replaced	M3	Moderate
R5 - Set-up and configuration of the devices are too time consuming	M3	Moderate
R6 - The devices does not meet the manufacturers' specifications	M3	Low
R7 - Operating system does not run on hardware	M3	Low
R8 - Communication cannot be established	M3	Considerable
R9 - Measuring tools are not producing usable data	M5	Moderate
R10 - Test environments are not representing the real world	M5	High
R11 - Delivery of the prototype are not manageable in time	M6	Considerable
R12 - The delivered prototype does not meet the requirements	M6	Low