# Abstract

Nowadays, Internet of things is witnessing a tremendous evolution due to the increasing growth in communication technologies, weather and environmental sensing, health care sensing. Indeed, sensors are being a kind of intelligent mobile agent able to perceive its environment and transmit information to the cloud for processing, This way of perception allow the development of several kinds of applications to enhance human capacity to understand their environment and make appropriate decision. However, developing such advanced applications relies heavily on the quality of the communication between sensors and between sensors and the infrastructure, therefore, such communication can be realized only with the help of a secure data collection and efficient data treatment and analysis.

Data collection in a vehicular network has been always a real challenge due to the specific characteristics of these highly dynamic networks (frequent changing topology, vehicles speed and frequent fragmentation), which lead to opportunistic and non long-lasting communications. Security, remains another weak aspect in these wireless networks since they are by nature vulnerable to various kinds of attacks aiming to falsify collected data and affect their integrity. Furthermore, collected data are not understandable by themselves and could not be interpreted and understood if directly shown to a driver or sent to other nodes in the network. They should be treated and analyzed to extract meaningful features and information to develop reliable applications. In addition, developed applications always have different requirements regarding quality of service (QoS). Several research investigations and projects have been conducted to overcome the aforementioned challenges. However, they still did not meet perfection and suffer from some weaknesses. For this reason, we focus our efforts during this thesis to develop a platform for a secure and efficient data collection and exploitation to provide vehicular network users with efficient applications to ease their travel with protected and available connectivity. Therefore, we first propose a solution to deploy an optimized number of data harvesters to collect data from an urban area. Then, we propose a new secure intersection based routing protocol to relay data to a destination in a secure manner based on a monitoring architecture able to detect and evict malicious vehicles. This protocol is after that enhanced with a new intrusion detection and prevention mechanism to decrease the vulnerability window and detect attackers before they persist their attacks using Kalman filter. In a second part of this thesis, we concentrate on the exploitation of collected data by developing an application able to calculate the most economic itinerary in a refined manner for drivers and fleet management companies. This solution is based on some information that may affect fuel consumption, which are provided by vehicles and other sources in Internet accessible via specific APIs, and targets to economize money and time. Finally, a spatio-temporal mechanism allowing to choose the best available communication medium is developed. This latter is based on fuzzy logic to assess a smooth and seamless handover, and considers collected information from the network, users and applications to preserve high quality of service.

# Résumé

# ᓱᕋᕕᕕᓄᐱ

ᓱᕋᕕᕕᓄᐱ ᐅ ᒪᑕᓄᒐ ᕋᐅᘐᘐᖏᓄᐅᑎ ᑕᓄᐱ ᔪᑦᐠᕋ ᙅᙆᘓᓄᐅᐠᕋᕕ ᕋᐅᙅᒪᓄᑎ ᓱ ᔪᕋᙅᙆ

# ملخص

الأفكار الخضراء عديمة اللون تنام بغضب

# Acknowledgements

# Dedication

I dedicate this work to those who are denied to study by lack of money, health and means.

# Contents

# List of Tables

# List of Figures

# List of Abbreviations

# List of Nomenclatures

# List of Publications

*"There's no absolutely reliable way to achieve a great citation. However, hardworking could be fruitful" - Eraldo Banovac*

## International Conferences

## National Conferences

## Journals

## Survey

# 1 | Testbed

## Contents




### Abstract


## 1 Introduction [1]

### 1.1 Problem Statement

### 1.2 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].

### 1.3 Purpose (Goal)

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.

### 1.4 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.

### 1.5 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.

## 2 Related work

# 3 Background

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.

### 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.

## 3.1 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 MCUs 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 subGHz 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.

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.

### Resume

Transceiver with IEEE 802.15.4 or IEEE 802.15.1 Integrated sensor/sensors MCU with low power mode under 5MicroA Wakeup from low power mode with timer Border router ability Joint Test Action Group (JTAG) support

## 3.2 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.

### Resume

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

## 3.3 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.

### Resume

IPv6 addressing support Existing OS support Network type is mesh UDP

## 3.4 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 toolchains 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.

# 4 Proposed ...

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 Contikis 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.

## 4.1 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 2 C bus and the sensors were not implemented. The I 2 C driver is required for the sensor drivers which in turn enables the MCU to communicate with the sensors on the OpenBattery platform.

## 4.2 CoAP server

In order to make each nodes 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 2 C 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 MCUs 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 CoAPs 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.

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.

### 4.2.1 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.

### 4.2.2 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.

# 5 Experimentation

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.

## 5.1 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  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 (  ) 4d c f FSPL(dB) = 20 log 10 (d) + 20 log 10 (f )  147.56  = (4.1) (4.2) (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  FM  FSPL (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. d = 10 x P t + G t + G r  P r  F M + 147.56  20 log 10 (f )

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.

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

## 5.2 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. Radio duty cycle: Transfer time: 1s = 0.125s = 125ms 8 (4.6) 133B + 4B  4ms 31.25KB/s (4.7) Worst case delivery: 125ms + 4ms + 1ms (4.8) Best case delivery: 4ms + 1ms (4.9) 130ms + 5ms = 67.5ms (4.10) 2 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. Avg. delivery: Avg. response time: (2 ů 67.5ms) ů hops (4.11) Best case response time: (2 ů 5ms) ů hops (4.12) Worst case response time: (2 ů 130ms) ů hops (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.

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.

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

## 5.3 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 developers 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. Data rate: 250kb/s = 31.25KB/s 133B 54B = 0.59 133B Theoretical goodput: 31.25KB/s ů (1 0.59) = 12.81KB/s Overhead: (4.15) (4.16) (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 71as 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. Packet size: 133B 54B + 32B = 111B Actual overhead: 79B = 0.71 111B (4.18) (4.19)

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.

## 5.4 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.

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 112.81W 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.

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 257W as calculated in

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.

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.

### 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 measurement, 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 2 C 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.

# 6 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.

## 6.1 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 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.

## 6.2 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 nodes 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.

## 6.3 Connection speed

The connection speed, when using CoAP or any other protocol with perpacket 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 datarate 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.

## 6.4 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 272s, 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.

## 6.5 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.

# 7 Discussion

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 30the 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.

# Bibliography

[1] J. Bregell, “ Hardware and Software Platform for Internet of Things ”, *Master of Science Thesis in Embedded Electronic System Design*, 2015, 00002.