NTNU – Trondheim
Norwegian University of
Science and Technology

Understanding Data Analysis in an End-to-End IoT System

Sindre Schei

Submission date: May 2016
Responsible professor: Frank Alexander Kraemer, ITEM
Supervisor: David Palma, ITEM

Norwegian University of Science and Technology
Department of Telematics

Abstract

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of

the original language. There is no need for special content, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Sammendrag

Sikkerheten til nesten all offentlig nøkkel-kryptografi er basert på et vanskelig beregnbarhetsproblem. Mest velkjent er problemene med å faktorisere heltall i sine primtallsfaktorer, og å beregne diskrete logaritmer i endelige sykliske grupper. I de to siste tiårene, har det imidlertid dukket opp en rekke andre offentlig nøkkel-systemer, som baserer sin sikkerhet på helt andre type problemer. Et lovende forslag, er å basere sikkerheten på vanskeligheten av å løse store likningsett av flervariable polynomlikninger. En stor utfordring ved å designe slike offentlig nøkkel-systemer, er å integrere en effektiv "falluke" (trapdoor) inn i likningssettet. En ny tilnærming til dette problemet ble nylig foreslått av Gligoroski m.f., hvor de benytter konseptet om kvasigruppe-strengtransformasjoner (quasigroup string transformations). I denne masteroppgaven beskriver vi en metodikk for å identifisere sterke og svake nøkler i det nylig foreslåtte multivariable offentlig nøkkel-signatursystemet MQQ-SIG, som er basert på denne idéen.

Vi har gjennomført et stort antall eksperimenter, basert på Gröbner basis angrep, for å klassifisere de ulike parametrene som bestemmer nøklene i MQQ-SIG. Våre funn viser at det er store forskjeller i viktigheten av disse parametrene. Metodikken består i en klassifisering av de forskjellige parametrene i systemet, i tillegg til en innføring av konkrete kriterier for hvilke nøkler som bør velges. Videre, har vi identifisert et unødvendig krav i den originale spesifikasjonen, som krevde at kvasigruppene måtte oppfylle et bestemt kriterie. Ved å fjerne denne betingelsen, kan nøkkelgenererings-algoritmen potensielt øke ytelsen med en stor faktor. Basert på alt dette, foreslår vi en ny og forbedret nøkkel-genereringsalgoritme for MQQ-SIG, som vil generere sterkere nøkler og være mer effektiv enn den originale nøkkel-genereringsalgoritmen.

Preface

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

6LoWPAN IPv6 over Low Power Wireless Personal Area Networks.

ACK Acknowledgement.

ACL Asynchronous Connection-Less.

BLE Bluetooth Low Energy.

CoAP Constrained Application Protocol.

CON Confirmable CoAP message.

GUI Graphical User Interface.

HCI Host Controller Interface.

I2C Inter-Integrated Circuit.

ICMP Internet Control Message Protocol.

IDE Integrated Development Environment.

IoT Internet of Things.

IPv4 Internet Protocol version 4.

IPv6 Internet Protocol version 6.

M2M Machine-to-machine.

MQTT Message Queueing Telemetry Transport.

NON Non-Confirmable CoAP message.

NTNU Norwegian University of Science and Technology.

OS Operating System.

PAN Personal Area Network.

radvd Router Advertisement Daemon.

RTT Round Trip Time.

SPI Serial Peripheral Interface.

TCP Transmission Control Protocol.

TFTP Trivial File Transfer Protocol.

UDP User Datagram Protocol.

USB Universal Serial Bus.

Chapter 1
Introduction

1.1 Motivation

This is a test in the introduction chapter.

1.2 Methodology

1.3 Scope and Objectives

1.3.1 Scope

1.3.2 Objectives

O.1: Build a star network of microcontrollers

O.2: Connect sensors to the end-nodes to collect data

O.3: Gather information of the data sent through the network

O.4: Analyse and discuss the gathered information

1.4 Structure

Chapter 2
Background

This thesis describes the setup and usage of an End-to-End Internet of Things (IoT) system. In order for this to be set up by others later to perform reproducible tests, a detailed description of components, sensors and protocols used is needed. This chapter will go through the background information of the devices, technologies and protocols used, and why these where chosen over other alternatives.

2.1 Bluetooth Low Energy (Bluetooth Smart)

Bluetooth Low Energy (BLE) is a wireless technology for short range communication developed by the Bluetooth Special Interest Group [3]. The idea was to create a low energy single-hop network solution for Personal Area Networks (PANs). A major advantage of this is that Bluetooth 4.0 is already a well established technology in cell phones, laptops and several other devices, and BLE can use several similarities with this. The 6LoWPAN Working Group has recognized the importance if BLE in IoT [4].

The protocol stack of BLE has two main parts, the controller and the host. In the system used in this thesis this represents the Raspberry Pi as the controller (master) and Nordic nRF52 as the host (slave). The communication between these is done through the standard HCI. All slaves are in sleep mode by default, and are woken up by the master when communication is needed. Links are being identified by a randomly generated 32-bit code.

In the case of the network used in this thesis, when the BLE slave has been connected to a master, it stops searching for other masters, and it is not possible to connect to several masters. This means that we are only able to create a *star network*, not a *mesh network*. This could possibly be an idea for improvement later. Other than this, BLE seems like a very good alternative in this project.

Check out: Mikhail Galeev - BLE

2.1.1 HCI

2.1.2 ACL

2.2 6LoWPAN

IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN) is a defined protocol for using Internet Protocol version 6 (IPv6) in low enerdy networks, to identify sensors and devices, as defined in the IEEE 802.15.4.

To use the Internet Protocol was proposed by Geoff Mulligan and the 6LoWPAN Working Group in [6]. In this paper the advantage of 6LoWPAN is explained as the easiest way to use standard protocols such as User Datagram Protocol (UDP), Transmission Control Protocol (TCP), Internet Control Message Protocol (ICMP), glsdns and Trivial File Transfer Protocol (TFTP), since they can be used directly without the requirement of a translation mechanism. They are already adapted to run over IPv6.

In addition to this, the paper argues that 6LoWPAN was developed to be used in small sensor networks. Implementations can fit into 32Kb flash memory parts, wich is smaller than Zigbee and Zensys (which is the two main comparisons made in the paper). It also uses an impressive header comparison mechanism that allows the transmission of IPv6 packets in 4 bytes, much less than the standard 40 bytes. This is done by using stacked headers, same as in the IPv6 model, rather than defining a specific header as for Internet Protocol version 4 (IPv4). This means that a device can send only the required part of the stack header, and does not need to include header fields for networking and fragmentation [4].

6LoWPAN was therefore a natural tool to use in this project, in the IPv6 and BLE based network.

2.3 Raspberry Pi

Developed by Element 14, the Raspberry Pi has become a central tool for many people wanting to get started using microcomputers. This was therefore a natural starting point for us as well.

The Raspberry Pi 2 model B+ is the second generation of its kind, and its *ARMv7 processor* has approximately six times the performance of the predecessor [2]. With a USB Bluetooth dongle connected, it is quite simple to enable both 6LoWPAN and BLE, given that the right Unix kernel has been used in the Operating System (OS) of the Pi.

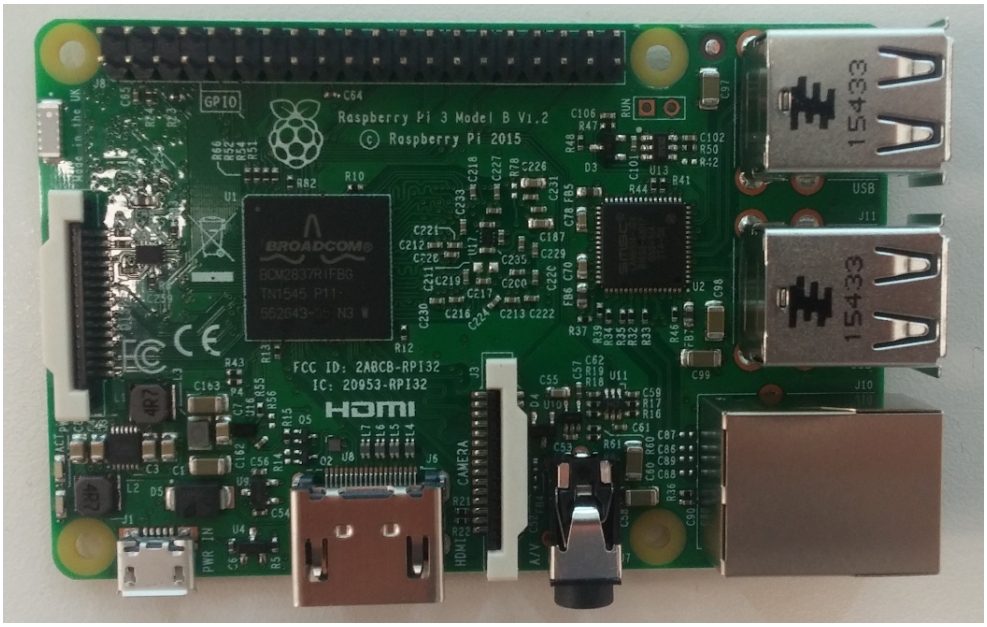

Figure 2.1: Raspberry Pi 3

Later in this thesis, the processing power of the Pi compared to performing calculations in the end-points will be a central topic for discussion.

2.3.1 Ubuntu Mate

Along with the Raspberry Pi, we needed a good and stable operating system with a kernel that supported the 6LoWPAN architecture. For this, Ubuntu Mate version 15.10 with kernel version 4.15 was chosen, and used on the Raspberry Pi. As other versions of Ubuntu this is Unix based, and has a complete Graphical User Interface (GUI) of a full OS.

2.4 nRF52

The nRF52 is developed by Nordic Semiconductor, and is being described as a family of highly flexible, multi-protocol system on chip.

[1]

Running at 64MHz it racks up impressive stats: EEMBC Coremark® score of 215 and 58 Coremark®/mA. Built in a cutting edge 55nm process technology the

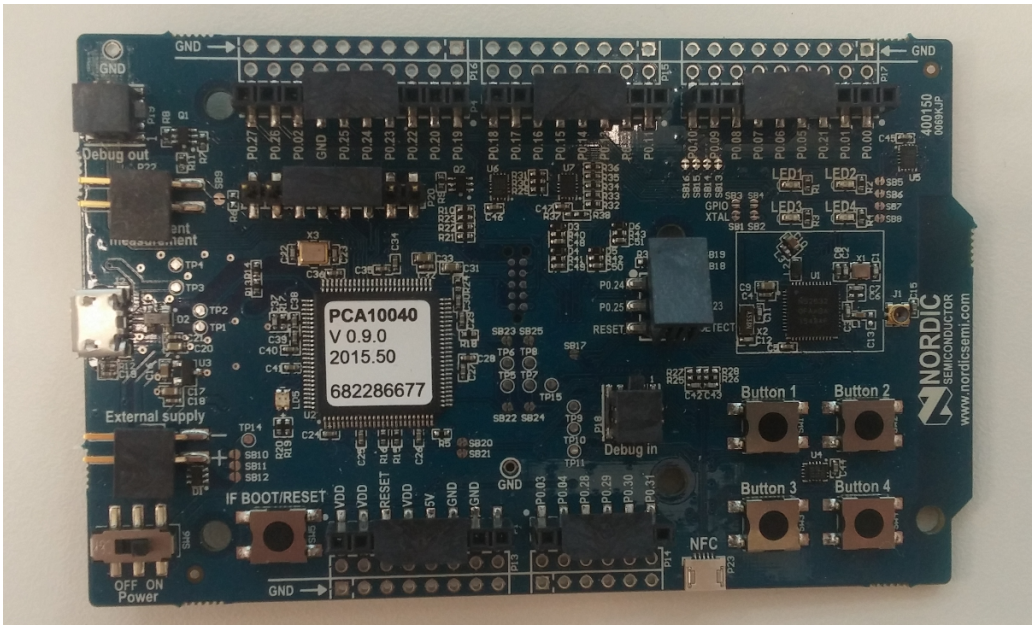

Figure 2.2: Nordic Semiconductor NRF52

nRF52 Series is architected for todays need for speed, speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. Introducing a Cortex-M4F processor at its heart, it is the most capable Bluetooth Smart SoC on the market today. With a brand new multiprotocol radio architecture limits are pushed even further for Bluetooth Smart: A total link budget of 100dBm, -96dBm sensitivity, +4dBm output power and -42dBm selectivity, it is built to exist reliably and effectively in a busy 2.4GHz band. Couple this RF resilience with outstanding power consumption: 5.3mA at 0dBm TX output power and 5.4mA RX it is miserly with power when getting things done on-air.

Designed with wealth of digital and analog interfaces and peripherals, there's something to cover every design requirement. The demands of today's Bluetooth smart single chip designs are increasingly varied and complex, they can range from digital audio processing, to FFT/FIR and security algorithms, to driving device displays and UIs. Whatever the task, nRF52 series has it covered.

We've truly made low power easy. We haven't just just great datasheet numbers, but an automatic and adaptive power management system which combined with extensive EasyDMA and PPI makes the nRF52 Series positively frugal with power.

nRF52 Series brings NFC for the first time to a Bluetooth Smart SoC. The on-chip NFC-A tag allows developers to take advantage of NFC 'touch-to-pair' functionality in their designs.

Maintaining the total flexibility philosophy of the nRF51 Series, it is a flash device. With 512kB flash + 64kB RAM on chip, developers keep all the software stack flexibility and Over-The-Air Device Firmware Upgrade (OTA-DFU) features they enjoyed with the nRF51 Series.

We understand that in the world of wearables, size really is everything, so we built a device that is the smallest Bluetooth Smart SoC to date measuring as small as 3.0 x 3.2mm (CSP). With an on-chip balun and a minimum of external components it has the smallest design footprint out there.

Tomorrow's Bluetooth Smart Solutions are going to ask a lot. The nRF52 Series delivers.

2.4.1 Softdevice

2.5 Adafruit ADXL345 Accelerometer

In order to do collect data, a sensor needed to be connected to the Nordic Semiconductor nRF52. The main thought behind the thesis was to measure vibrations, and a good accelerometer was needed. The Adafruit ADXL345 accelerometer was chosen for several reasons.

- It is the same accelerometer built in on the Zolertia Z1 microcontroller
- It can measure acceleration in all three axes, X, Y and Z.
- It sends digital data right away. This means there is no need to use computational power to calculate digital values as needed if the data was captured by an analog accelerometer.
- It supports both Inter-Integrated Circuit (I2C) and Serial Peripheral Interface (SPI), which makes it easy to connect to the nRF52.

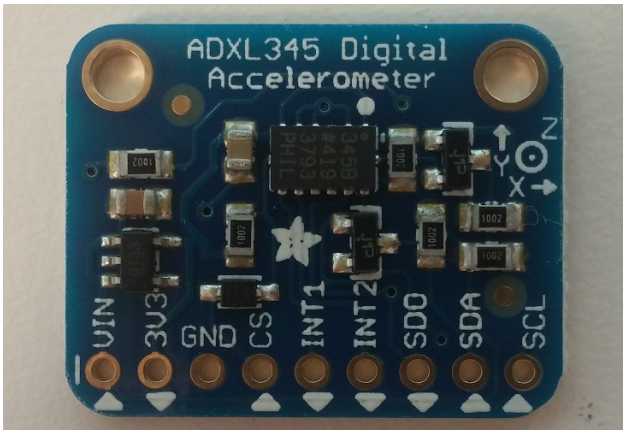

Figure 2.3: ADXL345 Accelerometer

2.6 Transport protocols

2.6.1 Network layers

2.6.2 CoAP

CoAP is a transport protocol designed to be used in constrained networks for Machine-to-machine (M2M) communication. It is UDP based, and works good in low-power and lossy networks. It can be used with microcontrollers, and with IPv6 and 6LoWPAN. Both GET and PUSH functionality can be used, as well as *observable* GET. This means that a server can "subscribe" to end notes in the network, and get updates either after a given timespan or when changes have been made. This therefore looked like a promising protocol to use, and was chosen as the first transport protocol to test the network [7].

2.6.3 MQTT

Another transport protocol that could be used in such a network of microcontrollers is MQTT. This is known as a publish-subscribe based on TCP, using a MQTT broker. Norwegian University of Science and Technology (NTNU) does have a broker that is possible to use, or a broker can be rented from several other places. Here a

[5]

2.6.4 radvd

2.7 Software tools

As Integrated Development Environment (IDE), the *KEIL Vision* was used, as recommended by Nordic Semiconductor (where?), for writing C programming. For other programming languages (for instance Python 3.4, HTML, CSS, JavaScript, Flask and AJAX) *Sublime Text 2* for Windows and Linux was used, as well as *Pluma* for Ubuntu Mate on the Raspberry Pi.

2.7.1 Other

Wireshark, Firefox, GitHub

Chapter 3
System Architecture

The purpose of this thesis is to build an end-to-end system, to be able to transfer data all the way from a sensor connected to a microcontroller to a server. This chapter will describe in detail how the different components of the system has been connected together, and how the different protocols has been configured to read and transfer data efficiently.

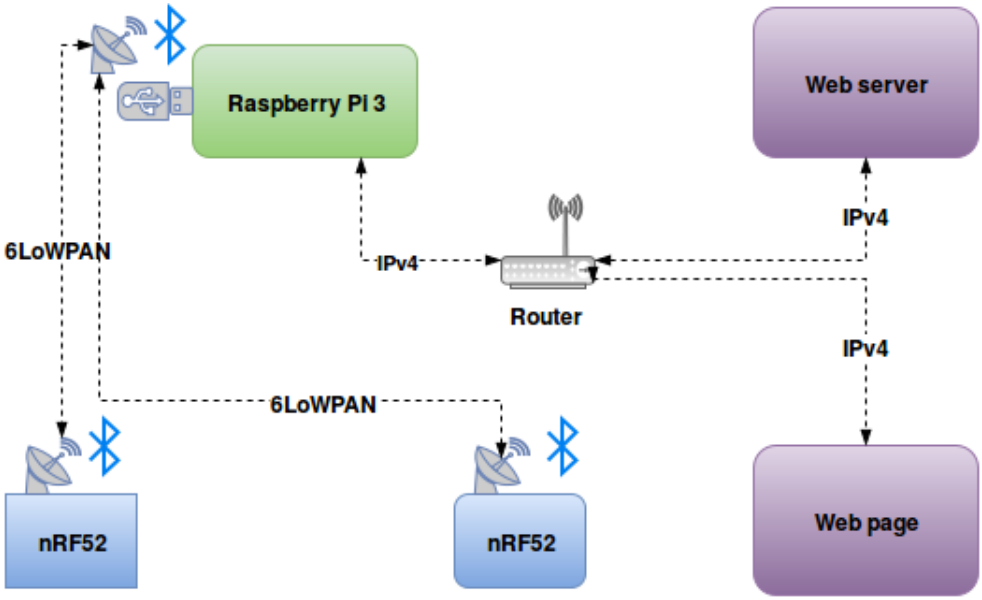

Figure 3.1: System architecture

Figure 3.1 shows how the complete end-to-end system of this thesis is set up. In short terms, the *Nordic Semiconductor nRF52* is connected to a *Raspberry Pi 3* using 6LoWPAN and BLE. This means that they are able to communicate using

IPv6 addresses, even though the nRF52 only has a bluetooth antenna built in. The limitations of BLE means that the nRF52 can only connect to *one* device at the time. Several of these put together are therefore forming a *star network* using the *Raspberry Pi* as a central point of connection. Computations can now either be done in the end nodes at the *nRF52s*, at the *Raspberry Pi*, or forwarded to a web server or another computer with more computational power. Data can also be displayed directly to a web page from the *Raspberry Pi*.

There are in general three main limitations in a network such as this:

* Computational power in the different nodes
* Battery capacity of the end nodes
* Network limitations between the nodes

A central part of the testing in this thesis will be to test the different limitations, and to understand the pros and cons of doing computations in end nodes, compared to transferring information to a server with *much* more computational power. Power usage is very often closely related to computational power, and will also be a central factor. The next section will contain a walk-through of the system, from the smallest to the biggest component, to explain their computational and power capabilities, and how they can communicate efficiently.

Computational power is closely related to power usage. The nRF52 microcontroller is battery powered using a small *3V Lithium CR 2032* battery. Given this limitation the computational power will be limited as well. The optimal solutions therefore seems to handle as little data as possible here.

3.1 Connecting Raspberry Pi and nRF52

To set up the communication between a Raspberry Pi and the nRF52, the two code examples TWI and Observable server from Nordic Semiconductor was used as a starting point for coding on the nRF52. It was however quite tricky to set connect these two together the first time. The following recipe is what worked best.

Install an OS on the Raspberry Pi that has a Linux kernel version later than 3.18. On *Raspbian* version 3.18 is the only stable (Note: Jan. 2016), but *Ubuntu Mate* is stable in version 4.15. *Ubuntu Mate* was therefore chosen as the best and most stable OS. When this is done a resizing of the file system is needed to use all the capacity of the memory card.

To use BLE, install Bluez and radvd using *apt-get* in a *Unix terminal*:

```
apt-get install radvd wireshark bluez
apt-get upgrade
apt-get update
```

To activate IPv6 forwarding, uncomment the following line (remove "#") in the file *etc/sysctl.conf*

```
net.ipv6.conf.all.forwarding=1
```

Add the *bt0 interface* in *radvd.conf*:

```
touch /etc/radvd.conf
pico /etc/radvd.conf
```

Write in the bt0 interface

```
interface bt0
{
    AdvSendAdvert on;
    prefix 2001::/64
    {
        AdvOnLink off;
        AdvAutonomous on;
        AdvRouterAddr on;
    };
};
```

To mount the modules *bluetooth_6lowpan, 6lowpan and radvd*, add the following to */etc/modules.*

```
bluetooth_6lowpan
6lowpan
radvd
```

It is now possible to use the *hcitool* command.

```
hcitool lescan
```

This command will scan for BLE devices nearby, and find the bluetooth address, for instance *0211:22FF:FE33:4455*. To connect, run the following:

```
echo 1 > /sys/kernel/debug/bluetooth/6lowpan_enable
echo "connect 0211:22FF:FE33:4455 1" > /sys/kernel/debug/bluetooth\\/6lowpan_contro
service radvd restart
```

The command *hcitool con* shows the connected BLE devices. If the device is connected, the connection can be tested by typing

```
ping6 2001::0211:22FF:FE33:4455
```

Using the basic example of the observable server it was easy to send CoAP messages without the need of an Acknowledgement (ACK) for every message.

3.1.1 Connection challenges

```
ping6 2001:bt8::0211:22FF:FE33:4455
```

Descibe bt8-problem and dev-zone.nordicsemi.com and infocen-ter.nordicsemi.com

3.2 Nordic Semiconductor example code

Nordic Semiconductor has provided several examples along with the nRF52-DK documentation, that can be used as a starting point when programming applications to the device. These files are written in the programming language *C*.

3.2.1 CoAP

As a first step, the nRF52 needed to communicate with the Raspberry Pi. As explained in section 2.6, a good starting point for this is to use the CoAP trasnport protocol, and the example of the use of this on the device. The *CoAP Server* and *CoAP Observer* was loaded into two different nRF52 devices, and both of them connected to the Raspberry Pi. Now radvd had to be activated on the Pi, to be able to use this as a *forwarding server*.

Remove "Figure 1" from image

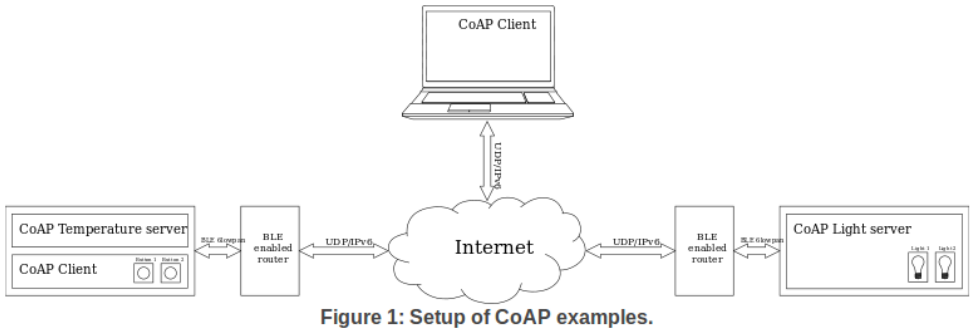

Figure 3.2: CoAP Client-Server communication example

Figure 3.5 shows the initial CoAP example, by using two nRF52 devices, BLE over 6LoWPAN and a BLE enabled router (Raspberry Pi) on both ends in this case. As the figure 3.5 shows, the code example for the CoAP client includes two different cases:

* In the fist case the CoAP client acts as a client requesting services from the CoAP server. Button 1 and 2 will control light 1 and 2 on the nRF52 server, which will send a conformation back if the light was changed or if something went wrong.
* In the second case the CoAP server is excluded, but the CoAP client works as a server that can handle requests from the router. In this case it is possible to use *Copper* in a *Firefox browser* on the Raspberry Pi to act as a client to request values from a simulated temperature sensor on the server. The server will then send the current simulated temperature back, or tell if something went wrong.

When packets are captured using *Wireshark*, we get the following capture:

This gives us the sequence diagram shown in Figure 3.4.

Create new sequence diagram

This works fine, and by writing a Python script running on the *Raspberry Pi* it is possible to request data by *GET messages* as soon as the *nRF52* is ready. However, as shown in figure 4.1 and figure 4.2, the network transfer rate can vary a lot. The

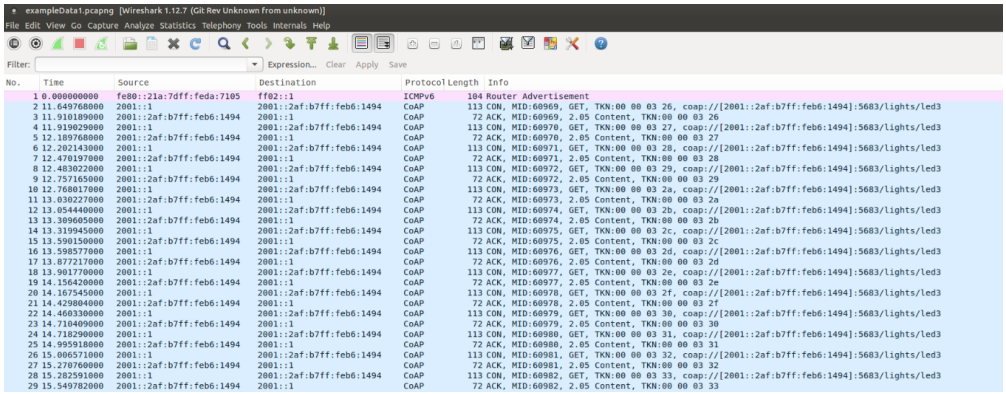

Figure 3.3: CoAP Client-Server example Wireshark capture

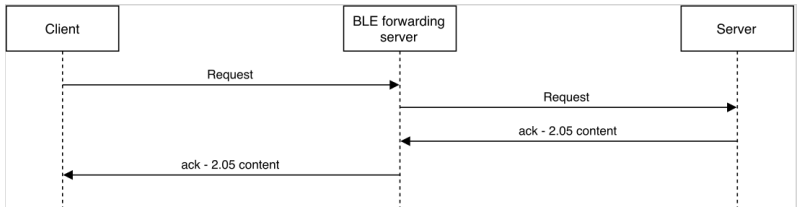

Figure 3.4: Sequence diagram CoAP Client-Server example

limitations in BLE and 6LoWPAN makes this much slower than a normal IPv6 connection, and the *ping* Round Trip Time (RTT) varies from 100 ms all the way up to over 600 ms. As a comparison, to *ping google.com* from the same Raspberry Pi using IPv6 takes on average 15 ms. A comparison of this is shown in figure 4.1. This is therefore a clear limitation of sending rate in the system. Because of this, the maximum transfer rate achieved in the system using the standard CoAP server example is one message every 200 ms.

Using the standard CoAP, described as *CoAP CON* in this thesis, messages are sent with a "TCP mindset". This means that every message sent by the client must receive an ACK. This is very stable, and ensures that every message gets to its destination, but it requires a lot of unnecessary computational power in the leaf node in systems where the importance of each message is not that high. The system is meant to be used with tools for analysing, and it is not essential that data needs to get to its destination right away. In these cases there is another alternative, known in this thesis as *CoAP NON*. This is more like a "*glsudp mindset*", where each packet does *not* need an ACK. This halves the number of packages sent, saves a lot of the network as well as actions needed to be done by the nRF52 where power consumption

is a huge issue. Therefore the *CoAP Observable* function looked very interesting, and will therefore be directly compared to *standard glscoap* later in this thesis.

3.2.2 Observable CoAP

In the given Nordic example of a CoAP observable server, the problem of unnecessary packets being sent can be resolved. Using this it is easy to set up a server that can be observed by either a nRF52 client or a BLE forwarding server. The observable end point sends a *CON-ACK request* in a given time interval to assure that the observing client is still there. As long as this message is being responded the server will continue to send data without any requirements of an ACK.

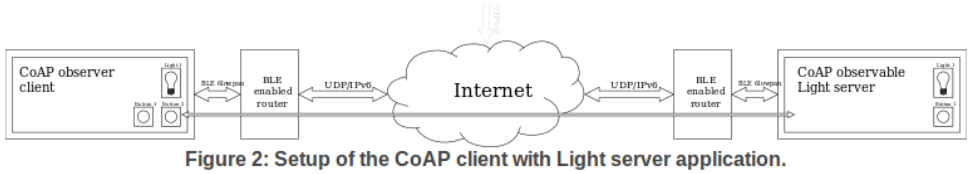

Figure 3.5: CoAP Observable Server example

Remove "Figure 1" from image

Create new sequence diagram for Observable as well

3.3 Raspberry Pi to Network Computer or Server

When running the Unix based OS Ubuntu Mate, the raspberry Pi can be used more or less as a regular computer. This has a pre-installed version of the most basic programs needed, for instance *Mozilla Firefox Browser*, *Pluma text editor* and *Unix terminal*. This means that the user has several options on how to process data on this device:

* No computation: All data arrives as useful data, and can be posted directly to a web page or a server for storage
* No computation: Forward all data directly to a computer with more computational power
* Some computation: Analyse the data to find data that is not relevant to filter out

* Full computation: Do a full analyse of the data. The results can then be posted directly to a server or displayed on a web page.

Which of these three that is most relevant depends on the data, and how computational power is needed.

Chapter 4
Network Measurements

This chapter will display the experiments conducted on data sending rate from the nRF52 to the Raspberry Pi in the form of and graphs, tables and figures to try to determine the most efficient combination of amount of data to send, sending frequency as well as protocols to use.

4.1 Initial limitations in the Network

4.1.1 Stable transfer rate

As soon as the end nodes of the network can communicate with the Raspberry Pi, the next step is to test the transfer rate of the connection. To measure the network transfer rate, *ping6* was used. This is a software tool used to test networks using IPv6 in a network. The results is measured in ms used for every RTT.

```
ping6 2001::2e6:6aff:fe64:54dd
ping6 google.com
```

When tested, the connection turns out to be very slow and in some cases very unstable, especially at maximum transfer rate. In the test of ping shown in figure 4.1 the connection between the Raspberry Pi and the nRF52 had 14 % packet loss, 1 out of 7 packets. Other test showed similar results, but the CoAP Confirmable CoAP message (CON) has in general got a higher stability at maximum sending rate than CoAP Non-Confirmable CoAP message (NON). After a test period it was therefore decided that the best solution for NON would be to gather data from sensors at a higher rate, and store them in the nRF52 temporarily. Every second all the measured values are being transferred to the Raspberry Pi, the temporarily values are deleted and the measurement continues. This has proven to be a very stable solution, with successful tests over several days. CON can handle more frequent transportations

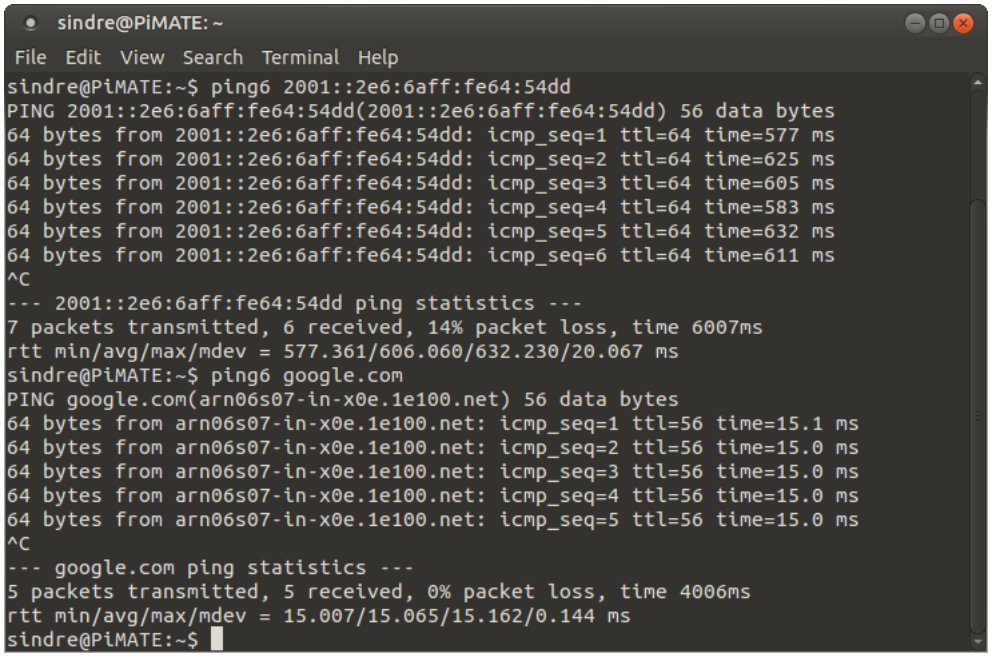

Figure 4.1: Comparison: ping nRF52 and ping google.com

than this in this system, on average 4 times per second. See the test shown in chapter 4.6. .

Make figure

$$\overline{x}_n = \frac{\sum_{i=1}^{n} x_i}{n} \tag{4.1}$$

Equation 4.1 shows the standard way used to calculate an average of the measured values. Here a is any real number and n is any integer.

Skriv om problemer med bt8 i Architecture?

After a test period, the problems explained in was fixed. A more reliable test could then be performed, as shown in figure 4.2. Using equation 4.1 to calculate average gives an approximate average of 250 ms RTT. This is very slow, and way beneath the transfer limitations of both BLE, 6LoWPAN and CoAP, as explained in chapter 2 . Another factor could be limited power supply and computational power, but it is not clear what is the main cause at this point. This will regardless be a huge limitation in the network.

Skriv spesifikt om begrensninger for disse tre i background

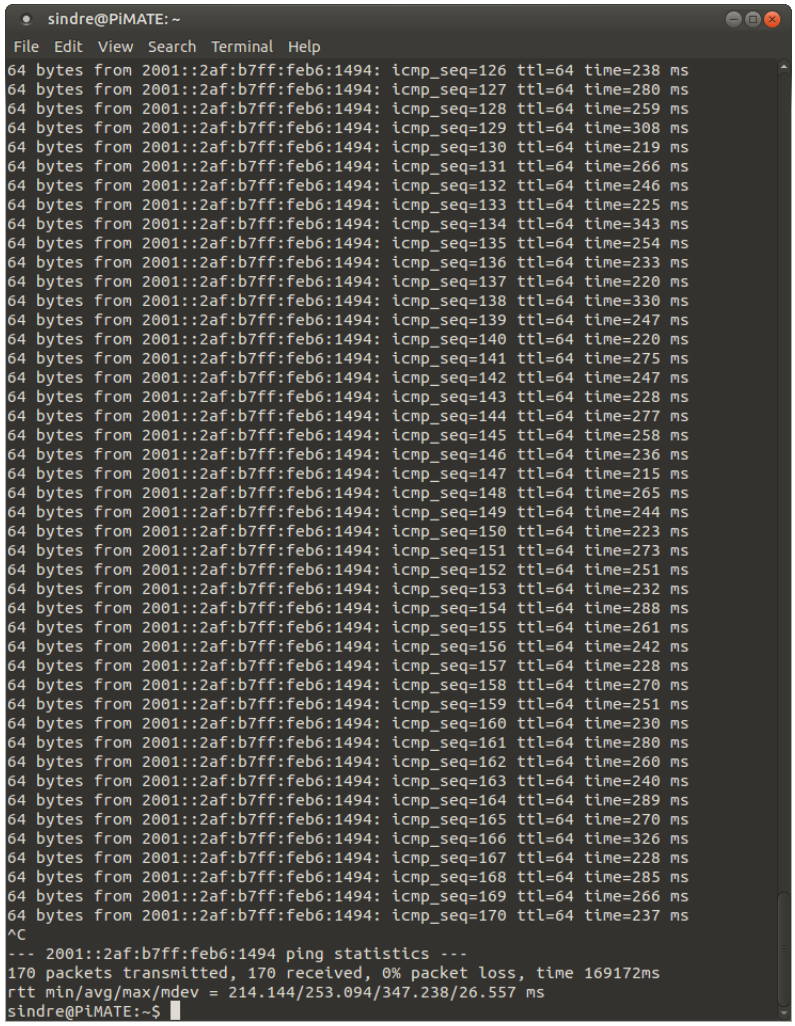

Figure 4.2: Ping nRF52, different time

The conclusion from these initial tests is therefore that CoAP CON is stable at at lower transfer interval than CoAP NON, and can in best case cope with sending every 250th ms, which is very good compared to the high RTT. NON has been so unstable at initial tests that it was decided to only send every second. This makes quite a huge difference when testing the different protocols, as *CoAP CON* can send a lot more *goodput* per second, but *CoAP NON* requires less power to send each message, and a much higher percentage of the packets sent through the network contains useful data.

4.1.2 Packet fragmentation

In Internet Routing, *fragmentation* is known as the action of splitting data into smaller packets, to satisfy the maximal limits of the different technologies or protocols used (e.g. BLE and 6LoWPAN in the network described in this thesis). Each of these packets needs header fields of a certain size, or other requirements. In a network of microcontrollers used in this thesis fragmentation can be a factor that needs to be taken into account to optimize the goodput of the system. This will be a central part of the testing, and will therefore be explained in detail in the following section.

To better understand fragmentation, imagine a train with carriages as shown in Figure 4.3. To be able to operate at all, the train needs a locomotive with an engine driver, a conductor and a cafe carriage. As soon as these things are already there, the company owning the train gets better and better off for every passenger buying a ticket. Lets assume that every carriage can carry 4 employees and 27 passengers, to make it directly comparable to the BLE packets in the network. Eventually all the carriages will be full, and a decision has to be made if it will be profitable to fit another carriage. It will in general be most profitable to use as many carriages as the locomotive can handle, and to fill up every carriage as much as possible. It will however not be a good idea to connect another carriage if there will only be one additional passenger sitting there, since the extra weight of the carriage adds unnecessary additional weight to the train set compared to the income.

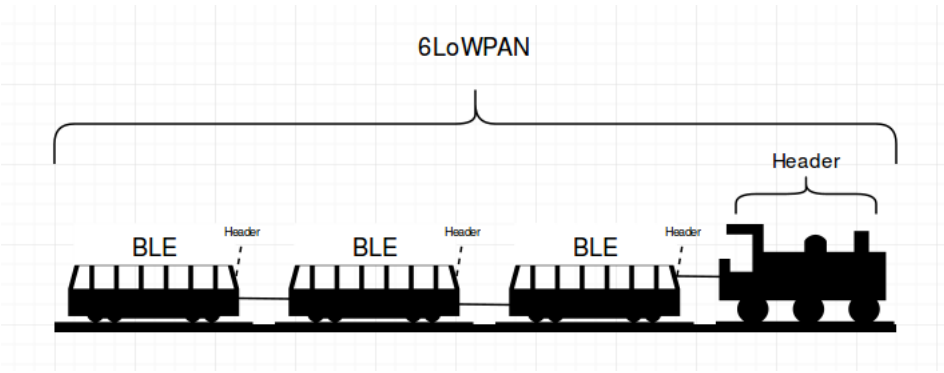

Figure 4.3: Packet fragmentation - train comparison

In this example, the locomotive and employees are the 6LoWPAN packet, that are needed no matter what to get the train working. Each additional carriage is a BLE packet. The goal is therefore to find the maximal number of passengers compared to the cost of adding additional carriages, in other words the maximal number of bytes compared to the number of packets sent. This is known as *fragmentation*, fragmenting data into smaller pieces to satisfy the maximal limitations of packet sizes

in the different protocols. This can be exploited by a system developer to maximize *goodput* vs *throughput* in the network.

4.2 Description of measurements

The following sections will show experiments performed to to test if *fragmentation* is a huge issiue when sending data through low energy networks, and which of the protocols described in chapter 2 is the most efficient to use when the goal is to get as high *googput* as possible. This will be done by sending data of constant length, capture packets using *Wireshark*, and then increasing the packet size to see the changes.

Before these experiements started, the expected results was that sending a small amount of data at a time would not be preferable, because of the needed bytes to set up the connection, header files and so on. It was not known how big the packages needed to be before it would be considered *profitable* to send.

When sending BLE packets over the network, observations from the system shows that the maximum packet size over BLE is 31 bytes. Each of these packages needs a header field of 4 bytes, meaning 27 bytes left for useful data. However, to start the connection at all, 76 byte is needed, meaning three BLE packets. The ratio between *useful* and *needed* (known as *goodput* and *throughput*) data transferred therefore start out very poorly if the payload sent is very small.The best possible percentage of useful data we can hope to acheive will also be limited by this, 27 bytes goodput and 4 bytes header field, shown in the following calculation.

$$\delta = \frac{x_2}{x_1} * 100\% \tag{4.2}$$

Equation 4.2 shows the standard way used to calculate the percentage, δ, that the natural number x_2 is of the total amount x_1.

$$\frac{27b}{31b} * 100\% = 87,094776\% \approx 87,1\% \tag{4.3}$$

In the actual system the number will quite certain be a lot lower than this. Both because header files in 6LoWPAN packages needs to use some of the capacity, and because of other limitations.

Like what?

Table 4.1: Wireshark CoAP CON 0 bytes

Number	Time	Protocol	Length	Info
36	3.7471	CoAP	72	ACK, MID:57083, 2.05 Content
37	3.7759	CoAP	113	CON, MID:57084, GET
38	3.9571	HCI_ACL	31	Rcvd [Reassembled in #40]
39	3.9584	HCI_ACL	31	Rcvd [Continuation to #38]
40	4.0274	L2CAP	5	Rcvd Connection oriented channel
41	4.0367	L2CAP	58	Sent Connection oriented channel
42	4.0368	L2CAP	50	Sent Connection oriented channel
43	4.0975	L2CAP	16	Rcvd LE Flow Control
44	4.0977	HCI_EVT	7	Rcvd Number of Completed Packets
45	4.1678	HCI_EVT	7	Rcvd Number of Completed Packets
46	4.0275	CoAP	72	ACK, MID:57084, 2.05 Content
47	4.0366	CoAP	113	CON, MID:57085, GET

4.3 CoAP CON

As explained in Chapter 4.1.1, CoAP CON messages can be sent quite frequently, but every message needs to get an ACK before the next message can be sent. This means that it is quite fast, but a lot of packets needs to be transported through to get usable data at the other end.

Table 4.1 shows an excerpt of a capture of packets in Wireshark. The full capture can be seen in the Appendix A. In this case an empty char array was sent, meaning an goodput equal to 0 byte. All of these bytes are therefore sent to route packages through the network. The maximum of 31 bytes per BLE packet have been exceeded twice, meaning that three packets needed to be sent. The first packet is labeled *[Reassembled in #40]*, the second *[Continuation to #38]* and the last *Connection oriented channel*. After this, the ACK packages follows, two packages of 58 and 50 bytes, respectively. The last bytes received tells how many packages was completed, as a built in feature in BLE HCI and Asynchronous Connection-Less (ACL). All of these packages can fit into one 6LoWPAN packet, since the total number of bytes are less than 270 bytes.

In table 4.2, 100 bytes of goodput is being sent through. The same basic packages needed are still there, but in addition the 100 bytes of data is added. This means adding more BLE packets, but also that the percentage of useful data sent through is a lot higher, approximatley 55 % in this case. By doing several experiments like this, it was possible to create the graph in Figure 4.4. This shows the correlation between goodput and throughput compared to the number of packets sent, measured

Table 4.2: Wireshark CoAP CON 100 bytes

Number	Time	Protocol	Length	Info
29	2.4514	HCI_ACL	31	Rcvd [Reassembled in #36]
30	2.4516	HCI_ACL	31	Rcvd [Continuation to #29]
31	2.2425	CoAP	173	ACK, MID:16354, 2.05 Content
32	2.2538	CoAP	113	CON, MID:16355, GET
33	2.5217	HCI_ACL	31	Rcvd [Continuation to #29]
34	2.5218	HCI_ACL	31	Rcvd [Continuation to #29]
35	2.5907	HCI_ACL	31	Rcvd [Continuation to #29]
36	2.5921	L2CAP	25	Rcvd Connection oriented channel
37	2.6099	L2CAP	58	Sent Connection oriented channel
38	2.6100	L2CAP	50	Sent Connection oriented channel
39	2.6610	L2CAP	16	Rcvd LE Flow Control
40	2.6621	HCI_EVT	7	Rcvd Number of Completed Packets
41	2.5922	CoAP	173	ACK, MID:16355, 2.05 Content
42	2.3097	CoAP	113	CON, MID:16356, GET
43	2.7311	HCI_EVT	7	Rcvd Number of Completed Packets

every 10th byte from 0 byte to 200 bytes large packets.

In this particular case it makes no sense to send less than 50 bytes of useful data at once, since more than 50 % of the bytes sent will be header files. This is comparable to having a locomotive and full crew at disposal, but only a few or none paying passengers. The best possible result is to have every carriage full, with 27 passengers and 4 employees. Since at least 4 bytes out of every 31 sent needs to be used to header information, the best possible result will be 87,1 %, as shown in equation 4.1. In mathematics, this is described as a *horizontal asymptote* since the distance between the graph and $y = 87{,}1$ will approach zero after an infinite number of bytes has been transferred. The graph will therefore converge to 87,1 %, just as the values climbing in figure 4.4, even before packet size of 200 bytes.

4.3.1 Discussion

Even though this first test was done with a very limited amount of data transferred, it is easy to see that the curve clearly flattens out and forms the shape of a *parabola* with a vertical *directrix* at $x = 0$. The next step would be to transfer a larger amount of data, to verify that the assumptions that the graph will converge to the asymptotic value $y = 87{,}1$ when $lim_{x \to \infty}$. The limitations of transfer rate is not nearly yet met by either BLE or 6LoWPAN. This test will therefore be explained in the next section.

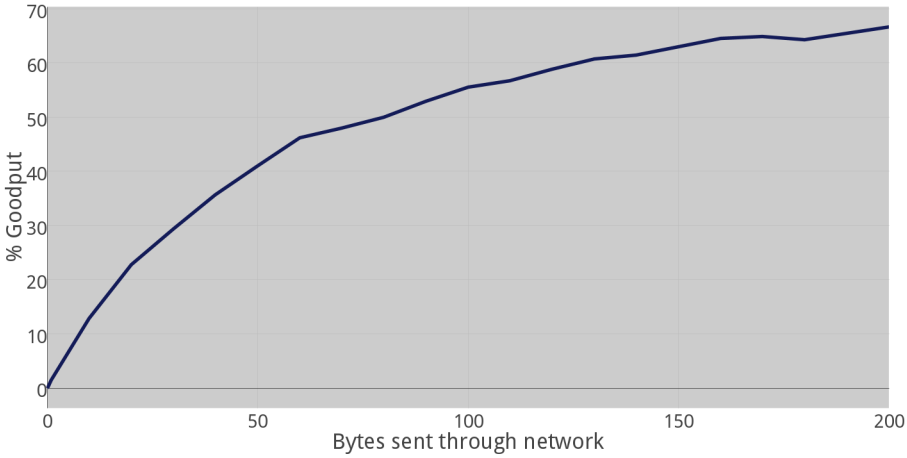

Figure 4.4: CoAP CON, 0-200 bytes sent

Since there is already a clear trend in the form of the graph in figure 4.4 even before the transfer rate reaches 200 bytes, it makes sense to also test the other version included in CoAP, CON. If it is possible to find a way of transportation that could give a higher mathematical maximum, and also get to this limit faster, it could be very profitable to the system. It also makes sense to test CoAP NON because it doesn't need to send an ACK for every packet, as CON does. Maybe this means that the header files can be split up in another way, and that the useful amount of data sent through therefore will be higher.

4.3.2 CoAP CON with more data

To see what happened when the limit of a 6LoWPAN packet was breached, tests was being set up to send larger amounts of data than 200 bytes. This would also be a good way to test if the percentage of goodput will converge to 87,1 %, only considering BLE packets. The following test and examples will therefore send a fixed number of bytes at once from 0 to 1000 bytes (1 kB), with a 50 byte interval. As before, the messages are being sent as soon as possible, meaning approximately four messages per second on average using CoAP CON.

Table 4.3 shows one of these cases, where 700 bytes of data are being sent at once. In total 889 bytes are being sent, with a CON packet size of 774 bytes. This gives a percentage of goodput at 78.74 %. The maximum packet length of a 6LoWPAN packet in this system of 270 bytes is therefore exceeded, as we can see in the figure. After 8 BLE packages of 31 bytes each has been sent, there is only room for 22 bytes

Table 4.3: Wireshark CoAP CON 700 bytes

Number	Time	Protocol	Length	Info
53	3.2570	CoAP	774	ACK, MID:35081, 2.05 Content
54	3.2727	CoAP	113	CON, MID:35082, GET
55	3.4671	HCI_ACL	31	Rcvd [Reassembled in #63]
56	3.4747	HCI_ACL	31	Rcvd [Continuation to #55]
57	3.5374	HCI_ACL	31	Rcvd [Continuation to #55]
58	3.5375	HCI_ACL	31	Rcvd [Continuation to #55]
59	3.6077	HCI_ACL	31	Rcvd [Continuation to #55]
60	3.6078	HCI_ACL	31	Rcvd [Continuation to #55]
61	3.6767	HCI_ACL	31	Rcvd [Continuation to #55]
62	3.6782	HCI_ACL	31	Rcvd [Continuation to #55]
63	3.7533	L2CAP	22	Rcvd Connection oriented channel
64	3.7534	L2CAP	16	Sent LE Flow Control
65	3.8172	HCI_EVT	7	Rcvd Number of Completed Packets
66	3.8172	HCI_ACL	31	Rcvd [Reassembled in #74]
67	3.8185	HCI_ACL	31	Rcvd [Continuation to #66]
68	3.9575	HCI_ACL	31	Rcvd [Continuation to #66]
69	3.9577	HCI_ACL	31	Rcvd [Continuation to #66]
70	4.0266	HCI_ACL	31	Rcvd [Continuation to #66]
71	4.0342	HCI_ACL	31	Rcvd [Continuation to #66]
72	4.0979	HCI_ACL	31	Rcvd [Continuation to #66]
73	4.0982	HCI_ACL	31	Rcvd [Continuation to #66]
74	4.1671	L2CAP	22	Rcvd Connection oriented channel
75	4.2372	HCI_ACL	31	Rcvd [Reassembled in #83]
76	4.2373	HCI_ACL	31	Rcvd [Continuation to #75]
77	4.3075	HCI_ACL	31	Rcvd [Continuation to #75]
78	4.3076	HCI_ACL	31	Rcvd [Continuation to #75]
79	4.3777	HCI_ACL	31	Rcvd [Continuation to #75]
80	4.4466	HCI_ACL	31	Rcvd [Continuation to #75]
81	4.4480	HCI_ACL	31	Rcvd [Continuation to #75]
82	4.4481	HCI_ACL	31	Rcvd [Continuation to #75]
83	4.5170	L2CAP	22	Rcvd Connection oriented channel
84	4.5871	HCI_ACK	31	Rcvd [Reassembled in #86]
85	4.5875	HCI_ACL	31	Rcvd [Continuation to #84]
86	4.6574	L2CAP	17	Rcvd Connection oriented channel
87	4.6731	L2CAP	58	Rcvd Connection oriented channel
88	4.6732	L2CAP	50	Rcvd Connection oriented channel
89	4.6577	CoAP	774	ACK, MID:35082, 2.05 Content
90	4.6729	CoAP	113	NON, MID:35083, GET

in the last packet before the 6LoWPAN packet is full. This is repeated several times, once for each 6LoWPAN packet, until the last BLE packets at the size of 17 bytes. After this the standard packages for ACK and *Number of completed packages* follows. This is in general a good example on how fragmentation of packets works in this system.

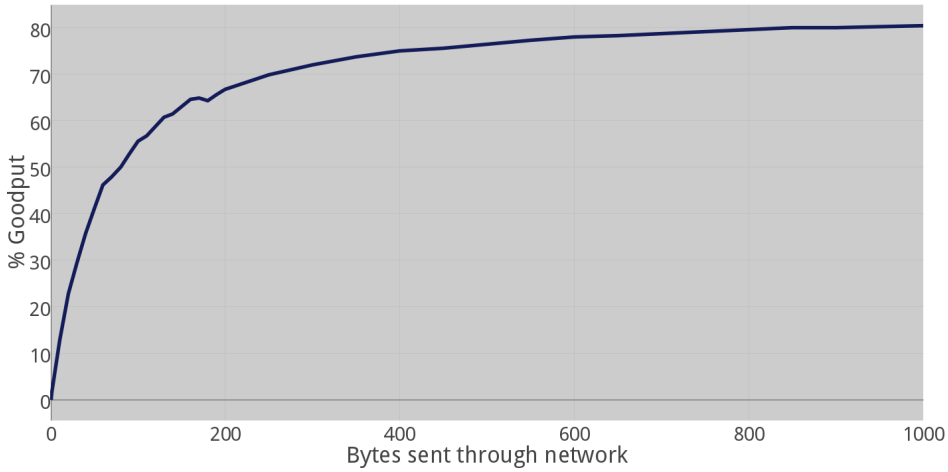

Figure 4.5: CoAP CON plot, 200 bytes to 1 kB

Make graph more clear

Using these results it was possible to plot the graph in figure 4.5 . Here it is easy to see the same trends as in 4.4, but in more detail over a wider span of bytes sent. These results are as expected after the previous tests, and in compliance with the calculations done in equation 4.1.

In this example points in the graph was denoted every 50th byte sent. This gives us

set in graph for 0-1000 before this, to see the flatness?

4.4 CoAP NON

A CoAP NON request does not require a response in form of an ACK. This means that the 108 bytes sent to and handled by the end node can be skipped, which means less computational power in the end node and network capacity to the node is needed. This solution makes sense to use in networks where the demand for all data is low, since packets can be lost without the use of ACKs. As explained in chapter 4.1.1, the transfer frequency is limited using NON in this system, due to unstable results

Table 4.4: Wireshark CoAP NON 0 bytes

Number	Time	Protocol	Length	Info
90	23.0405	HCI_ACL	31	Rcvd [Reassembled in #92]
91	23.0411	HCI_ACL	31	Rcvd [Continuation to #90]
92	23.1107	L2CAP	9	Rcvd Connection oriented channel
93	23.1109	CoAP	76	NON, MID:14, 2.05 Content

in tests at the beginning of the set up. The transfer rate is therefore set to one per second for this protocol.

Make CoAP NON 100 bytes

Bilde 4 LED's?

Using these measurements the following plot could be drawn.

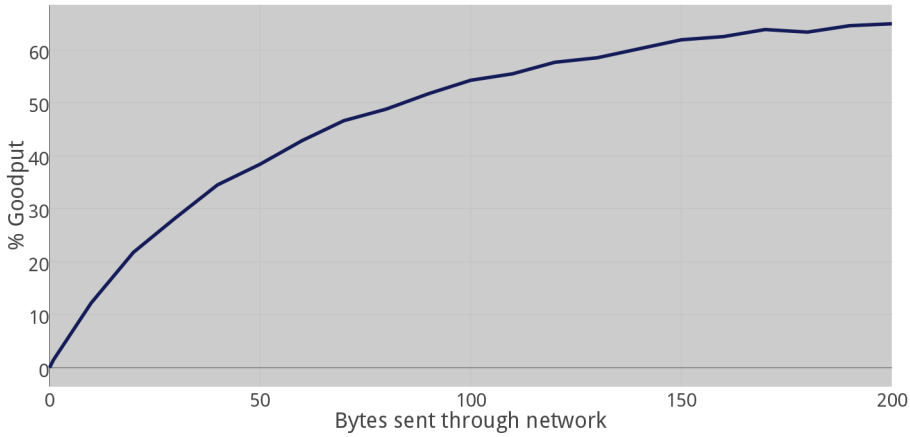

Figure 4.6: NON 0-200 bytes

4.4.1 CoAP NON, with more data

A basic example of this connection is shown in table 4.4. The entire Wireshark capture can be seen in Appendix A. This is directly comparable to figure **??**, that shows a Wireshark capture of CoAP CON packages. It is easy to see that a lot fewer packages needs to be sent using BLE, and no use of ACKs. The total amount of bytes sent is 31+31+9=71 bytes, meaning three BLE packages and one 6LoWPAN packet. This is less than half of what was needed using CoAP CON, where the 108 ACK

packages was needed in addition. The packets are still recognizable the same way as before, the first packet is labeled *[Reassembled in #40]*, the second *[Continuation to #38]* and the last *Connection oriented channel*.

Fewer packages sent means less energy used, less network capacity needed and less computational power in the end node. Hopefully this will lead to a higher percentage of goodput compared to throughput as well.

Table 4.5: Wireshark CoAP NON 700 bytes

Number	Time	Protocol	Length	Info
264	57.1476	CoAP	777	NON, MID:3, 2.05 Content
265	57.2177	HCI_ACL	31	Rcvd [Reassembled in #273]
266	57.2178	HCI_ACL	31	Rcvd [Continuation to #265]
267	57.2879	HCI_ACL	31	Rcvd [Continuation to #265]
268	57.2943	HCI_ACL	31	Rcvd [Continuation to #265]
269	57.3570	HCI_ACL	31	Rcvd [Continuation to #265]
270	57.3583	HCI_ACL	31	Rcvd [Continuation to #265]
271	57.4272	HCI_ACL	31	Rcvd [Continuation to #265]
272	57.4286	HCI_ACL	31	Rcvd [Continuation to #265]
273	57.4975	L2CAP	22	Rcvd Connection oriented channel
274	57.4979	L2CAP	16	Sent LE Flow Control
275	57.5676	HCI_ACL	31	Rcvd [Reassembled in #284]
276	57.5685	HCI_EVT	7	Rcvd Number of Completed Packets
277	57.5753	HCI_ACL	31	Rcvd [Continuation to #275]
278	57.6378	HCI_ACL	31	Rcvd [Continuation to #275]
279	57.6379	HCI_ACL	31	Rcvd [Continuation to #275]
279	57.7080	HCI_ACL	31	Rcvd [Continuation to #275]
280	57.7082	HCI_ACL	31	Rcvd [Continuation to #275]
281	57.7771	HCI_ACL	31	Rcvd [Continuation to #275]
282	57.7785	HCI_ACL	31	Rcvd [Continuation to #275]
283	57.7785	HCI_ACL	31	Rcvd [Continuation to #275]
284	57.8474	L2CAP	22	Rcvd Connection oriented channel
285	57.9176	HCI_ACL	31	Rcvd [Reassembled in #293]
286	57.9176	HCI_ACL	31	Rcvd [Continuation to #285]
287	57.9878	HCI_ACL	31	Rcvd [Continuation to #285]
288	57.9879	HCI_ACL	31	Rcvd [Continuation to #285]
289	58.0580	HCI_ACL	31	Rcvd [Continuation to #285]
290	58.0581	HCI_ACL	31	Rcvd [Continuation to #285]
291	58.1270	HCI_ACL	31	Rcvd [Continuation to #285]
292	58.1284	HCI_ACL	31	Rcvd [Continuation to #285]
293	58.1972	L2CAP	22	Rcvd Connection oriented channel
294	58.2673	HCI_ACK	31	Rcvd [Reassembled in #296]
295	58.2688	HCI_ACL	31	Rcvd [Continuation to #294]
296	58.3378	L2CAP	20	Rcvd Connection oriented channel
297	58.3379	CoAP	777	NON, MID:4, 2.05 Content

Table 4.5 shows the packets captured in Wireshark when 700 bytes of data was sent through the network.

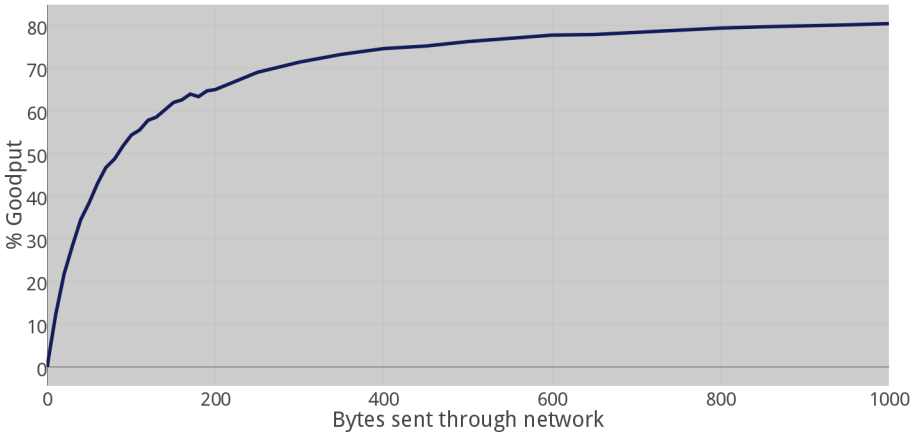

Figure 4.7: NON 0-1000 bytes

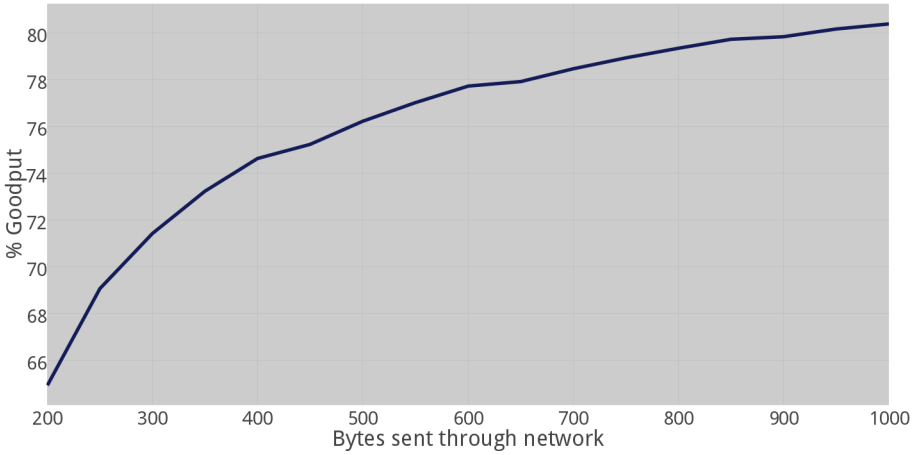

Figure 4.8: NON 200-1000 bytes

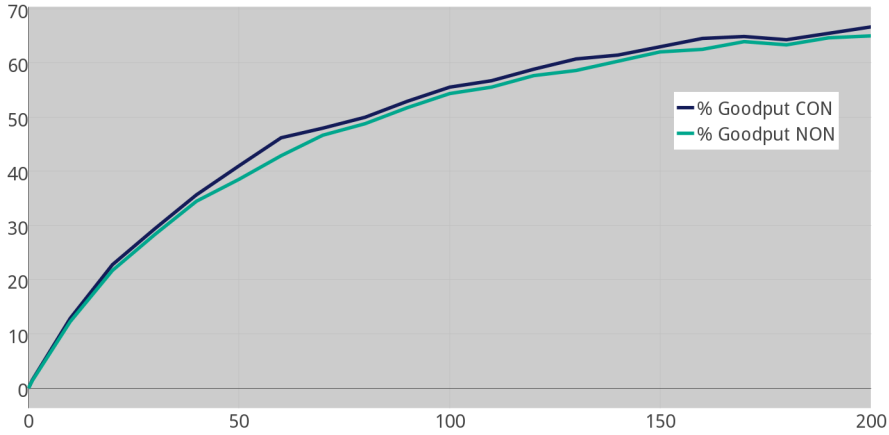

Figure 4.9: CON vs NON 0-200 bytes

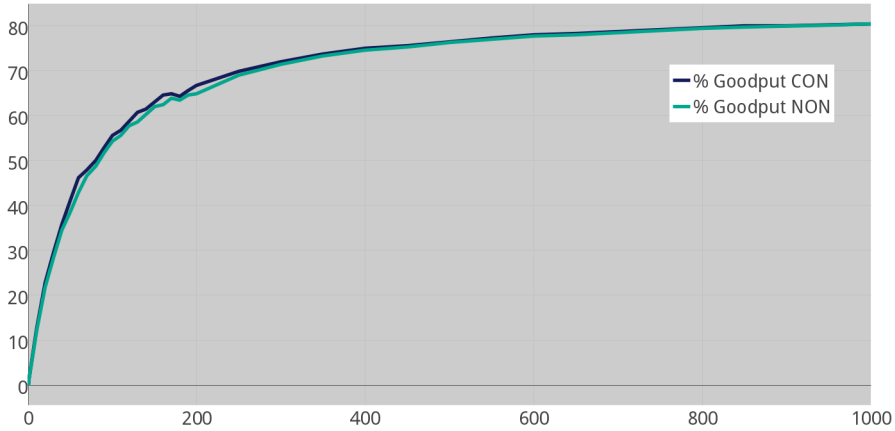

Figure 4.10: CON vs NON 0-1000 bytes

4.4.2 Discussion

4.5 Comparison

4.6 Time spent in network

The previous tests shows

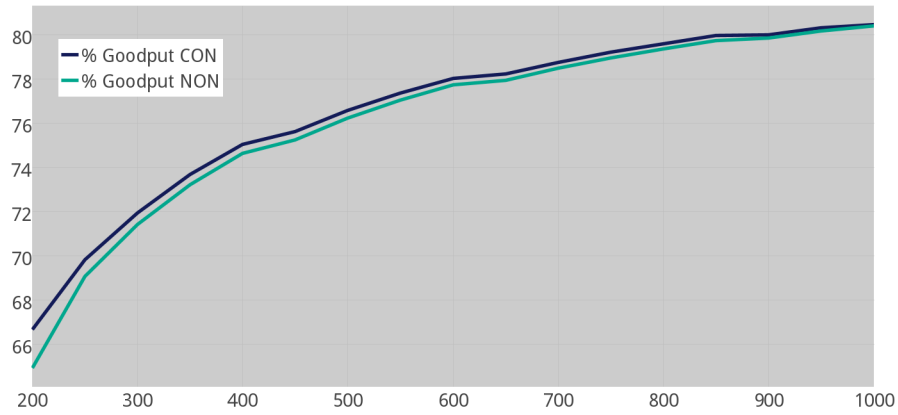

Figure 4.11: CON vs NON 200-1000 bytes

Calcumate time for CoAP NON, 700 bytes sent:

$$((55.9578 - 54.7750) + (57.1476 - 55.9578) + (58.3379 - 57.1476))/3 = 1.1876 \quad (4.4)$$

$0.1876/3 = 0.0625$

Chapter 5
Data Anaysis discussion

IDEA: "Collect" the strings put out in the first chapter, which told us the challenges in a system of microcontrollers. This chapter could tell what we learned from the system, and what is still a challenge. Or should all of this be in the Result chapter?

5.1 Data analysis in microcontrollers

5.2 Computational Power

5.2.1 Power consumption

Windows error message: "The device needs more data than the computer can supply".

5.3 Devices used

5.3.1 Raspberry Pi

5.3.2 nRF52

The nRF52 is perhaps the biggest disappointment in the system, from the authors perspective.

As a comparison, take the successful hybrid car Toyota Prius. Even though the factory tells that the car is able to drive X kilometres per liter of fuel used, this is only possible for a trained driver in a fixed environment. The average driver can still use the car, but is not skilful enough to drive as careful when it comes to power usage. The average driver therefore ends up using up the batteries much faster than the test driver, and uses the petrol powered engine after this. It may still be more economical than a normal car, but it is worth discussing if this is optimal, given that a car with two engines is more heavy and expensive to make in a direct comparison with a fully petrol powered car.

Its the same story with the nRF52. A professional skilled programmer working on this specific device from Nordic Semiconductor will be able to use the low power abilities and still use do efficient computations. When starting from the example code for the device that is not optimized for power consumption and trying to make it do more and more work, the battery consumption simply is too much for this device. An average programmer will not have specific experience enough to program this device to be power efficient.

In the case of the system built in this thesis, the small CR 2033 batteries was drained so fast that an additional power source was needed to keep a stable and reliable system, since tests has shown that the probability of an nRF52 disconnecting from the connected BLE service is higher at low battery. This meant connecting either a power bank with higher capacity or drain power from an outlet using a micro Universal Serial Bus (USB). In this case, it would have been able to use the Raspberry Pi right away, since this needs the same source of power. This means in the end, that the system only had low performance end points that still needed a power source.

References

[1] Nordic Semiconductor: nrf52 series soc. https://www.nordicsemi.com/Products/nRF52-Series-SoC. Accessed: 25-03-2016.

[2] Raspberry Pi: raspberry pi 2 model b. https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. Accessed: 27-03-2016.

[3] Gomez, C., J. Oller, and J. Paradells (2012). Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors 12*(9), 11734–11753.

[4] Hui, J. W. and D. E. Culler (2008). Extending ip to low-power, wireless personal area networks. *Internet Computing, IEEE 12*(4), 37–45.

[5] Hunkeler, U., H. L. Truong, and A. Stanford-Clark (2008). Mqtt-s—a publish/subscribe protocol for wireless sensor networks. In *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on*, pp. 791–798. IEEE.

[6] Mulligan, G. (2007). The 6lowpan architecture. In *Proceedings of the 4th workshop on Embedded networked sensors*, pp. 78–82. ACM.

[7] Shelby, Z., K. Hartke, and C. Bormann (2014). The constrained application protocol (coap).

Chapter A

Appendix A

Appendix A contains samples of programming code used to gather and transfer data in the IoT system described in this thesis.

Chapter B

Appendix B

Appendix B contains screenshots and detailed figures from the measurements done in the network built in this thesis.

Goodput	Throughput	NON packet size	(Goodput/Throughput)*100
0	71	76	0
1	73	78	1,37
10	82	87	12,20
20	92	97	21,74
30	75	107	30,0
40	85	117	47,06
50	99	127	50,51
60	109	137	55,05
70	119	147	58,82
80	133	157	60,15
90	143	167	62,94
100	153	177	65,36
110	167	187	65,87
120	177	197	67,80
130	191	207	68,06
140	207	217	69,65
150	211	227	71,09
160	225	237	71,11
170	235	247	72,34
180	253	257	71,15
190	263	267	72,24
200	277	277	72,20