**NTNU – Trondheim**
Norwegian University of
Science and Technology

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

**Sindre Schei**

Submission date: April 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 Symbols

$A_B^C$ A dummy symbol.

# List of Acronyms

**6LoWPAN** IPv6 over Low Power Wireless Personal Area Networks.

**ACK** Acknowledgement.

**BLE** Bluetooth Low Energy.

**CoAP** Constrained Application Protocol.

**DNS** Domaine Name System.

**GUI** Graphical User Interface.

**HCI** Host Controller Interface.

**I2C** Inter-Integrated Circuit.

**ICMP** Internet Control Message Protocol.

**ICMPv6** Internet Control Message Protocol version 6.

**IDE** Integrated Development Environment.

**IoT** Internet of Things.

**IPv4** Internet Protocol version 4.

**IPv6** Internet Protocol version 6.

**LAN** Local Area Network.

**M2M** Machine-to-machine.

**MQTT** Message Queueing Telemetry Transport.

**NDP** Neighbor Discovery Protocol.

**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.

# Chapter 1
# Introduction

## 1.1 Motivation

This is a test in the introduction chapter.

## 1.2 Methodology

## 1.3 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 and technologies used, and why these where chosen over other alternatives.

## 2.1 Bluetooth Low Energy

Read this: http://chapters.comsoc.org/vancouver/BTLER3.pdf

BLE is a wireless technology for short range communication developed by the Bluetooth Special Interest Group [4]. 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 [5].

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 Host Controller Interface (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

## 2.2 6LoWPAN

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 [7]. 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 [5].

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 [3]. 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.

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.

[2]

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 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.

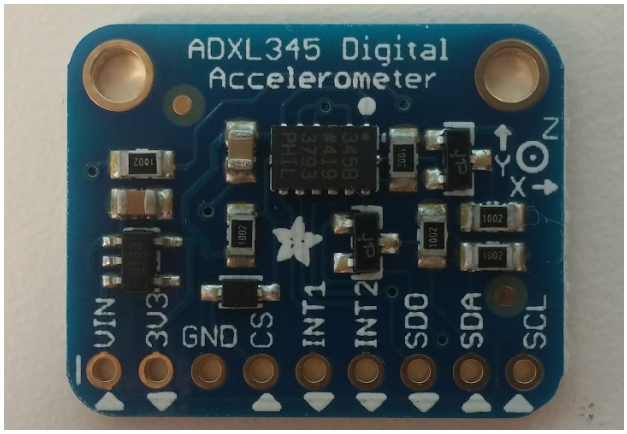


**Figure 2.1:** ADXL345 Accelerometer

## 2.6 Transport protocols

### 2.6.1 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 [8].

### 2.6.2 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

[6]

### 2.6.3 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.

# 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.

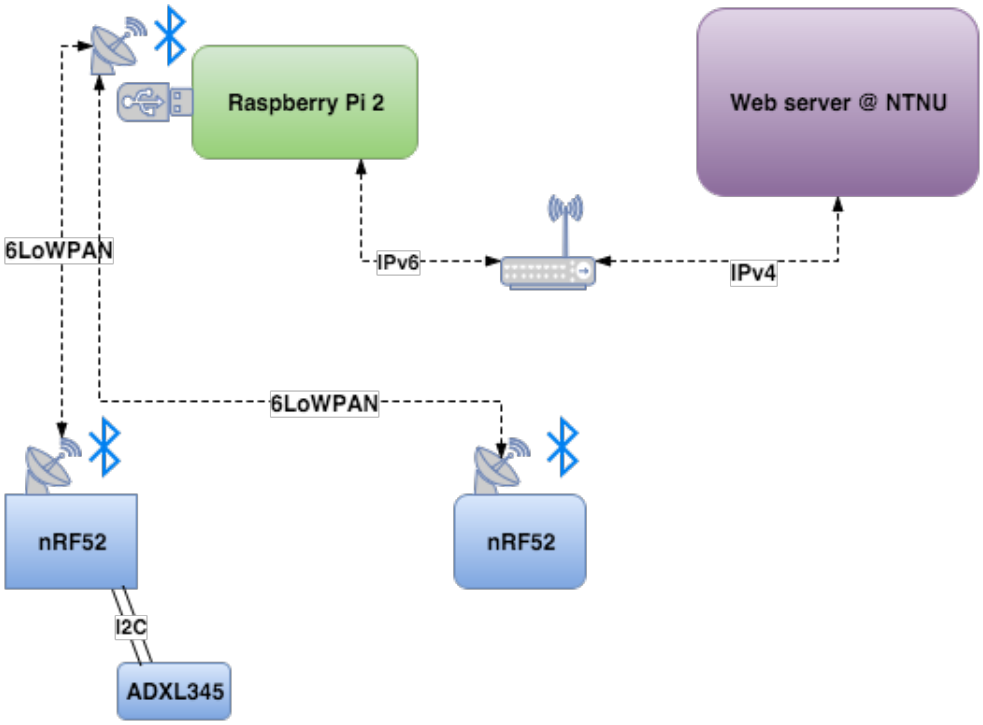


**Figure 3.1:** System architecture

There are two main limitations in a network such as this:

- Computational power in the different 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.

## 3.1 Connecting nRF52 and ADXL345

The ADXL345 Accelerometer used was connected using I2C, which is quite straight forward using the nRF52. Connection scheme is as follows (nRF52 -> ADXL345):

- 5V – VIN *Power source, green cable in Figure3.2*
- GND – GND *Ground, red cable in Figure 3.2*
- P0.27 – SDA *I2C Serial Data Line, orange cable in Figure 3.2*
- P0.26 – SCL *I2C Serial Clock Line, brown cable in Figure 3.2*

After this is done it is possible to write to and read from the registers of the accelerometer over the two SDA and SCL cables.

The communication between the nRF52 in Figure 3.3 and the Raspberry Pi in Figure 3.4, is done using 6LoWPAN over BLE.

## 3.2 6LoWPAN and BLE

## 3.3 Connecting Raspberry Pi and nRF52

As a microcontroller the nRF52 works great, both as a low-power and powerful device.

To set up the communication between these two, the two code examples TWI and Observable server from Nordic Semiconductor was used as a starting point. It was

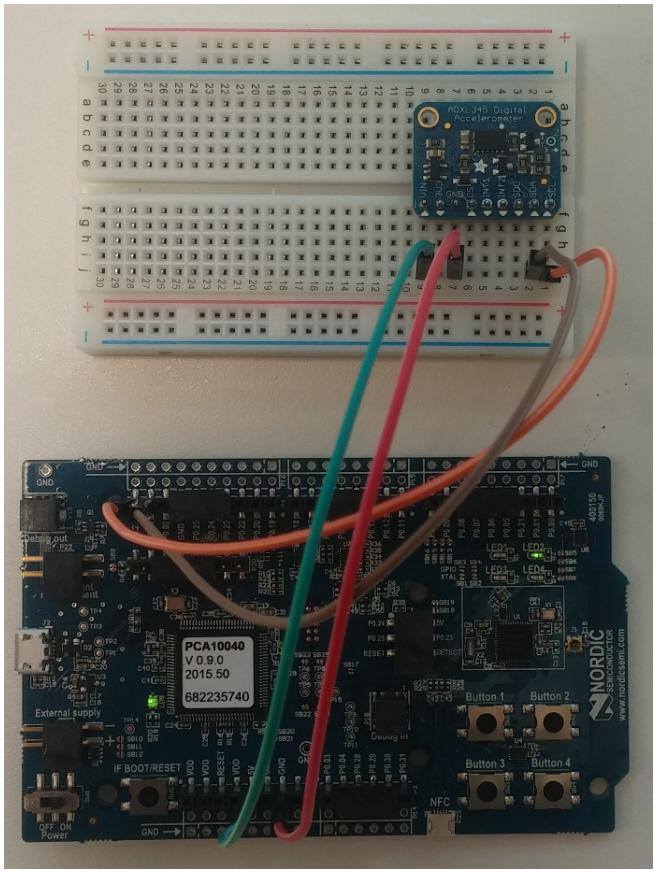


**Figure 3.2:** Connected nRF52 – ADXL345

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: Feb. 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*:

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

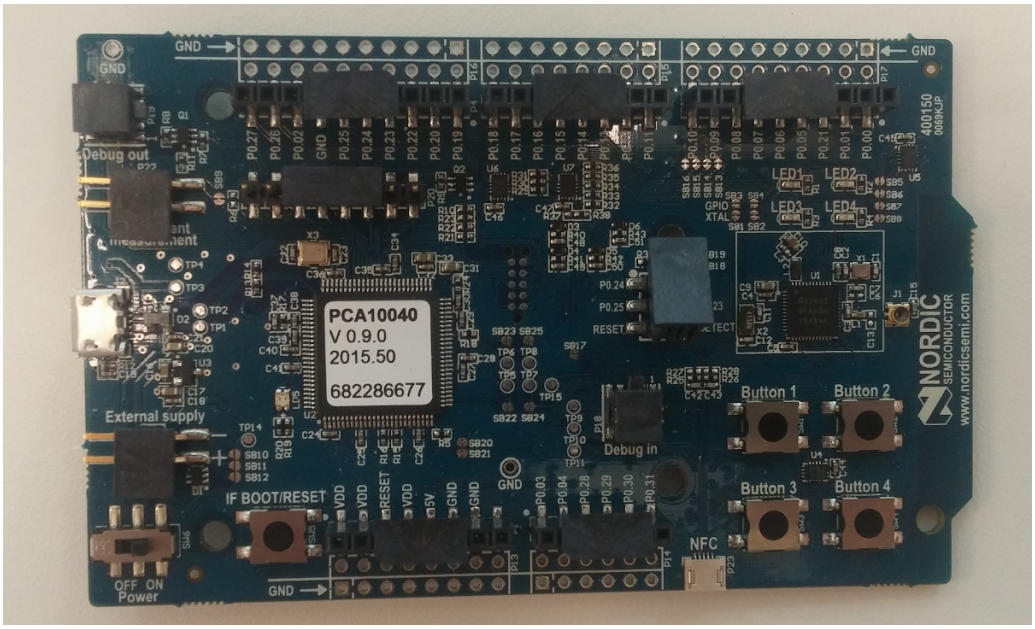


**Figure 3.3:** Nordic Semiconductor NRF52

To activate IPv6 forwarding, uncomment the following line in *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;
```

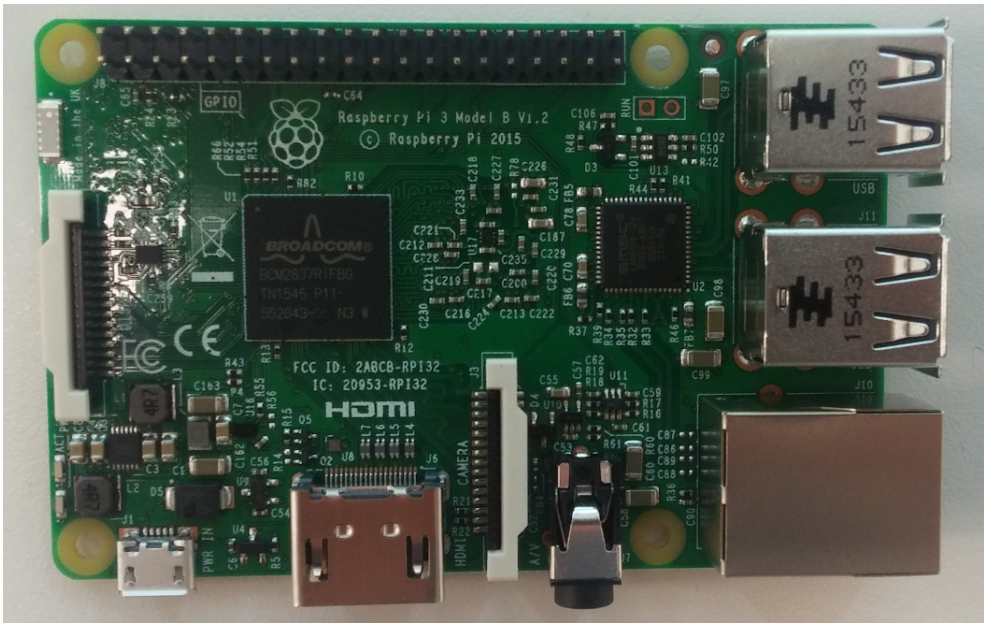


**Figure 3.4:** Raspberry Pi 3

```
        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_control
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.

## 3.4 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.

### 3.4.1 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*.

Figure 3.8 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 3.8 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

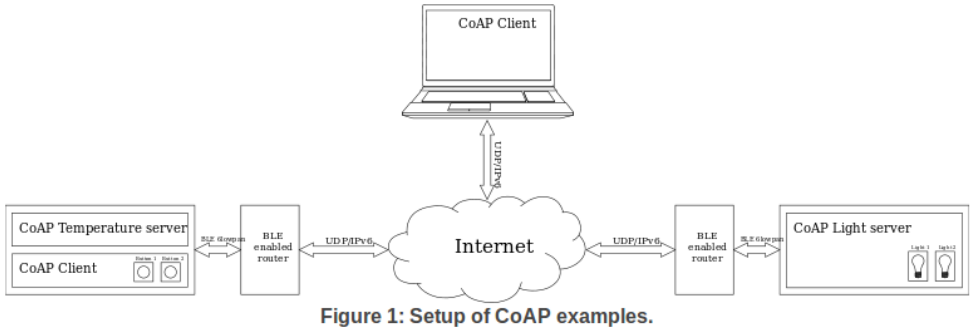


**Figure 3.5:** CoAP Client-Server communication example

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:

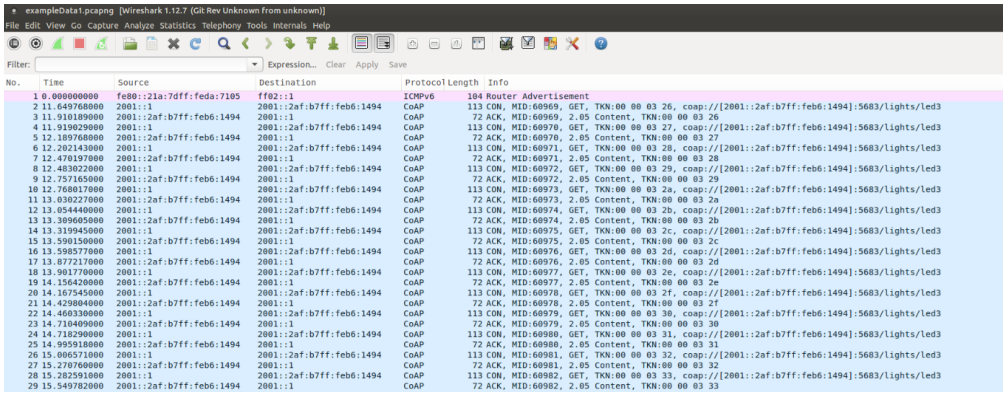


**Figure 3.6:** CoAP Client-Server example Wireshark capture

This gives us the sequence diagram shown in Figure 3.7.

This works fine, but in the case of the system built in this thesis there is no need to receive a confirmation for every single message. The system is meant to be used with tools for analysing, and it is not essential that data needs to get to its

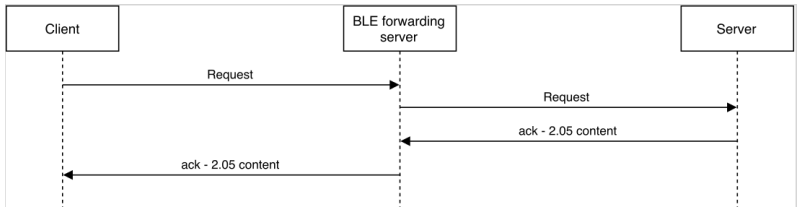


**Figure 3.7:** Sequence diagram CoAP Client-Server example

destination right away. It therefore makes sense to send messages in an *UDP fashion*, where packages are being sent right away without acknowledgement. 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 like an even better solution.

### 3.4.2 Observable CoAP

In the given Nordic example of a CoAP observable server, this problem can be resolved. It is here 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.

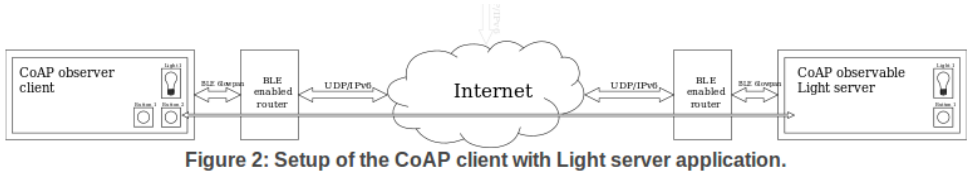


**Figure 3.8:** CoAP Observable Server example

### 3.4.3 Stable transfer rate

When the observable CoAP is working fine, the next step is to determine the optimal transfer rate of the connection. Since the number of messages sent has been halved, it should be possible use this as an advantage. However, the link in itself is quite slow. The 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 3.9. This is therefore a clear limitation of sending rate in the system.

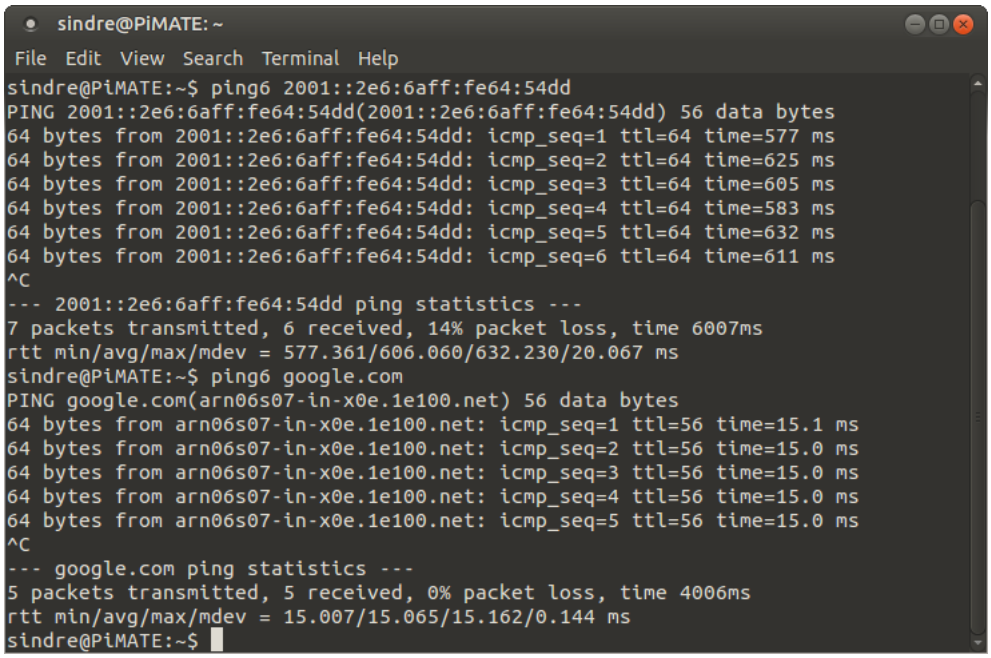


**Figure 3.9:** Comparison: ping nRF52 and ping google.com

In addition, at tests, the connection turns out to be quite unstable at maximal transfer rate. Even in the test of ping shown in figure 3.9 the connection between the Raspberry Pi and the nRF52 had 14 % packet loss, 1 out of 7 packets. Other tests showed similar results, and that any sending rate much faster than once per second was more or less unstable when sending data. After a test period it was therefore decided that the best solution 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.

## 3.5 Getting acceleration values

As soon as the sending of values is being handled in a good way, the next step is to read acceleration values from the ADXL345 accelerometer, which is connected to the NRF52 using the I2C interface.

Acceleration values can only be read from the ADXL345 if this has been correctly initialized at compile time. In order to do this, code to write to and read from the registers had to be added. The initializing process is described in the accelerometer data sheet [1]. In short, registers for *data format control, initial power saving, interrupt enable control* and *the offset of each axis* has to be written to in that order before the acceleration value from the different axes can be read.

In the solution proposed in this thesis the acceleration values are being read as often as possible, limited by the processing power of the nRF52, and then stores in a simple dynamical char array before being sent and reset after one second. This turned out to be very time consuming and hard to solve in a good way, both because of problems with initializing the accelerometer correctly and making the nRF52 read and store the values fast enough to get proper data. The ideal solution would be to read at least 1000 values every second, to get a good starting point before analysing values. Neither the nRF52 or the ADXL345 turns out to be fast enough to do this in the implemented solution. To not loose too much time on hardware problems, it was decided to focus more on analysing the data sent over the network communication with random generated data.

Since the network connection between the nRF52 and Raspberry Pi was already stable, it was easy to generate random values of fixed length to send on the nRF52, and do measurements to calculate the optimal throughput between these two. The next chapter will therefore describe the data analysis of the data sent through this network in detail, and how to optimize the percentage of usable data being transported.

# Chapter 4
# Network Measurements

This chapter will display the measured data from the accelerometer, and graphs, tables and figures to try to determine the most efficient solution.

## 4.1 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 used (e.g. BLE and 6LoWPAN in the network described in this thesis). Each of these net packets needs header fields of a certain size, or other requirements.

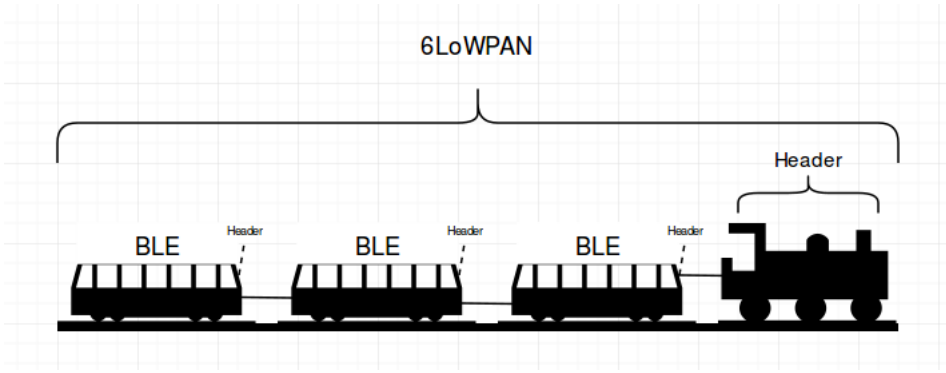


**Figure 4.1:** Packet fragmentation - train comparison

To better understand fragmentation, imagine a train with carriages. 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. 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 requires additional weight to the train set.

In this example, the locomotive and employees are the 6LoWPAN packet, that are needed no matter what. 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 , to maximize *goodput* vs *throughput*.

## 4.2 Goodput vs throughput

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.

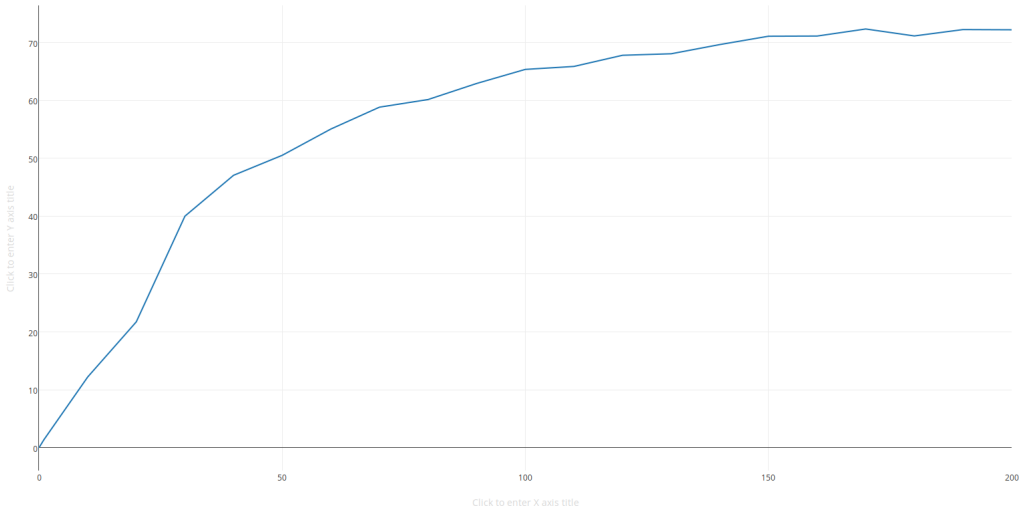


**Figure 4.2:** Goodput compared to throughput in %

Figure 4.2 shows the correlation between goodput and throughput compared to the number of packets sent. In this particular case it makes no sense to send less than 50 bytes at once, since more than 50 % of the data will be header files. But,

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 %. This graph will therefore converge to the *horizontal asymptote* of 87,1 %, just as the graph shows after just a packet size of 200 bytes.

# References


[1] ADXL345:digital accelerometer data sheet. http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf. Accessed: 20-04-2016.

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

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

[4] 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.

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

[6] 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.

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

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