



MALMÖ UNIVERSITY

Faculty of Technology and Society
Computer Engineering

Bachelor Thesis
180 credits

Designing Applications for use of NB-IoT

Applikationsdesign för användning av NB-IoT

Dennis Wildmark
John Tengvall

Exam: Bachelor of Science in Engineering
Subject Area: Computer Engineering
Date of final seminar: 2017-05-31

Examiner: Magnus Krampell
Supervisor: Dr. Ulrik Eklund

Abstract

The Internet of Things (IoT) is a market that has grown very fast in the last few years, creating an industry of its own. The core of IoT is the Internet connectivity and many times, the best solution for an IoT device is to use some form of mobile connection to solve this. The problem is that there is no obvious choice of mobile communication standard for use in an IoT device. The mobile communications industry has reacted to this newly emerged need of a mobile communications standard designed for the IoT domain and in 2016 the 3rd Generation Partnership Project (3GPP) released a Low-Power Wide-Area Network (LPWAN) type of standard named Narrowband IoT (NB-IoT). Several companies are working on implementing this standard, and there is a need to investigate how applications can utilize the standard effectively. This thesis presents a comparison between two applications using different Application Layer Protocol (ALP)s, Hyper-Text Transfer Protocol (HTTP) and Constrained Application Protocol (CoAP), in an LPWAN context. The results of this comparison shows that there is a lot to gain by choosing CoAP over HTTP, especially in an IoT environment such as the applications presented in this thesis. The thesis also presents a collection of properties that applications should have to use an LPWAN effectively.

Sammanfattning

IoT är en marknad som har växt fort under de senaste åren och skapat sig en egen industri. Kärnan i IoT är internetanslutningen och i många fall är mobil kommunikation den bästa lösningen för en IoT-produkt. Problemet är att det inte finns något självklart val av mobil kommunikation för användning i en IoT-produkt. Den mobila kommunikationsbranschen har reagerat på det nya behovet av mobil kommunikationsstandard för IoT och 2016 släppte 3GPP en ny standard av typen LPWAN kallad NB-IoT. Flera företag verkar för att implementera denna standard, och det finns ett behov av att undersöka hur applikationer kan utnyttja standarden på ett effektivt sätt. Denna uppsats presenterar en jämförelse mellan två applikationer som använder olika ALP, HTTP och CoAP, i en LPWAN-kontext. Resultaten av denna jämförelse visar att det finns mycket att vinna på att välja CoAP istället för HTTP, speciellt i en IoT-miljö som applikationerna presenterade i denna uppsats. Uppsatsen presenterar även en samling egenskaper som en applikation bör ha för att utnyttja en LPWAN-kommunikationsstandard effektivt.

Keywords: LPWAN, Narrowband IoT, NB-IoT, Application Design, Software Development, HTTP, CoAP, Comparison

Acknowledgement

We would like to express our gratitude to Magnus Midholt and Andreas Elvstam on ARM holdings for the support and giving us the opportunity to work with them.

We would like to thank Dr. Ulrik Eklund for the valuable feedback and support we have been given throughout the thesis.

Table of Contents

	Page
1 Introduction	1
1.1 Research Questions	2
1.2 Limitations and Thesis Scope	2
1.3 Applications Description	2
2 Theoretical Background	3
2.1 Cellular networks and Narrowband Internet of Things	3
2.1.1 Early versions of Cellular Networks	3
2.1.1.1 Second Generation (2G)	3
2.1.1.2 Third Generation (3G)	3
2.1.2 Long Term Evolution (Fourth Generation (4G))	3
2.1.3 Narrowband Internet of Things	4
2.2 The OSI Model and the TCP/IP Model	4
2.3 Transport Layer Protocols	5
2.3.1 Transmission Control Protocol (TCP)	5
2.3.2 User Datagram Protocol (UDP)	6
2.4 Application Layer Protocols	6
2.4.1 HTTP	6
2.4.2 CoAP	7
2.5 Wireshark	7
3 Related work	8
3.1 Comparing the cost-efficiency of CoAP and HTTP in Web of Things applications	8
3.2 Challenges in the migration to 4G mobile systems	8
3.3 Comparing application layer protocols for the Internet of Things via experimentation	9
3.4 Transport protocols behaviour study in evolving mobile networks	9
4 Methodology	10
4.1 Constructing a Conceptual Framework	11
4.1.1 Interviews	11
4.1.2 Literature Study	11
4.2 Develop a system architecture	11
4.3 Analyzing and Designing the System	11
4.4 Prototypes Construction	12
4.4.1 Development Tools	12
4.5 Validation of the Prototypes	13

4.5.1	Measurements and Testing Procedures	13
5	Results	15
5.1	Pre-requisites	15
5.2	Requirements	15
5.2.1	Applications Requirements	15
5.2.2	NB-IoT Requirements	17
5.3	System Architecture and Design	17
5.3.1	HTTP based application	17
5.3.2	CoAP based application	18
5.4	Prototype Construction	19
5.4.1	HTTP design implementation	20
5.4.2	CoAP design implementation	20
5.5	Testing and Validation	20
5.5.1	Testing the Functionality of the Applications	21
5.5.2	Testing the HTTP implementation	21
5.5.3	Testing the CoAP implementation	23
5.5.4	Test and Validation Summary	24
6	Discussion	26
6.1	General Discussion	26
6.2	Method Discussion	26
6.3	Validation of the Requirements	26
6.4	Test Results	26
6.5	Comparison to Related Work	27
7	Conclusions	28
7.1	Answering the Research Questions	28
7.2	Future Work	28
7.3	Contributions	29
A	Test cases	34
A.1	ID 1	34
A.2	ID 2	34
A.3	ID 3	35
A.4	ID 4	35
A.5	ID 5	36
A.6	ID 6	36
A.7	ID 7	37
A.8	ID 8	37

A.9 ID 9	38
A.10 ID 10	38
A.11 ID 11	39
A.12 ID 12	39
A.13 ID 13	40
A.14 ID 14	40
B Measuring the Packet Build Time	41
C Search criteria	42
D System figure	43

1 Introduction

Mobile communications is a constantly evolving technology area with many different application domains, where IoT is one of the most rapidly growing application domains within mobile communications [1]. Enabling Internet for small embedded devices containing sensors and other devices is the essence of IoT. While it is popular to use local area connections like Wi-Fi, it is not always ideal for the application. Imagine a water meter installed at a pipeline far out in the wild. Using Wi-Fi there is not an option. It is more convenient to use some sort of Wide Area Network (WAN) connection. This is what mobile communications offer; an Internet connection wherever you need it.

Mobile communications, however, is just a general concept. In reality, mobile communications consists of a number of different network types and standards. New standards are constantly being developed to support new services and meet customer needs. The 3GPP [2] is a partnership between many large mobile communication operators and other companies in the mobile industry and is formed to develop and set new standards within the mobile communications scope. In 2016 3GPP released a new standard aimed at the IoT domain called NB-IoT. NB-IoT is part of 3GPP Long Term Evolution (LTE) standard, which is a standard of high speed wireless communication for mobile phones and connected devices. The NB-IoT is a LPWAN type of standard, and there are other LPWAN standards made by commercial companies, e.g LoRa Alliance and Sigfox [3, 4]. The reason to why NB-IoT is particularly interesting is that it is a standard released by the 3GPP, making it a global standard which is attractive to deploy for a Mobile Network Operator (MNO). Companies like Telia and Vodafone agrees with this, implying that the NB-IoT standard is more promising. Matt Beal, Vodafone’s director of innovation and architecture, says in an interview [5] that “NB-IoT will crush Sigfox and LoRa because it means there will be no need for them”. Hans Dahlberg, head of Telia’s IoT business, argues that “If you take NB-IoT, which is part of LTE, then you don’t need to build extra networks and you can use the same BSS [business support systems]” [6].

Using this new standard of mobile communication NB-IoT, IoT applications and devices can be less power consuming, less complex and therefore cheaper [7], which makes it interesting to implement for device manufacturers. Using NB-IoT can, however, also put special constraints on the applications in terms of data usage. There is therefore a need to minimize this in such applications.

An IoT application can be designed in many different ways, but the foundation of IoT is that it is connected to the Internet, and thus, sends or receives some form of data via Internet. In order to communicate with another application, most commonly some form of server, effectively over the Internet, an IoT application needs to use an ALP [8]. The purpose of an ALP is to format the data in a way that it can be interpreted by the receiving end. There are a lot of different ALPs to choose from, and they are often designed for specific purposes. One of the most common protocols for transferring web content is HTTP, which has been in use since 1996 by the World Wide Web [9]. However, with the emerging IoT industry, new ALPs are being developed, adapted to constrained applications such as IoT. The CoAP, released in 2014, is a such protocol, and it is designed to be used in IoT applications [10]. This thesis makes a comparison between HTTP and CoAP to investigate the differences between the two protocols in terms of performance and data usage. The theory is that data usage could be minimized by choosing the correct ALP, thus facilitating for the use of NB-IoT and other constrained types of mobile communication standards. Furthermore, this thesis investigates what other properties that an IoT application should have in order to facilitate for use of NB-IoT and other similar standards.

1.1 Research Questions

New mobile communication standards such as NB-IoT are being introduced, presenting new possibilities to the IoT applications domain. In order to use these types of standards effectively, this thesis aims to answer these questions:

- RQ 1** How does CoAP perform compared to HTTP when used in an IoT application?
- RQ 2** What properties should an embedded IoT system application have to be used with an LPWAN type mobile communication standard?

This thesis is based on a collaboration with ARM Wireless business unit. ARM is a company in the field of microprocessors and chips. The company licenses their chip designs to chip manufacturers and their products are present in over 95 % of the mobiles in the world [11]. ARM has recently been investing in mobile communications and have an interest in seeing their upcoming NB-IoT solution in use.

To answer the research questions, two applications were developed, one which uses HTTP and the other using CoAP. These applications were then compared to each other to give an answer to RQ 1. The choice of ALPs was based on a request from ARM to implement these two ALPs in two applications. The applications should demonstrate the use case of a weather station containing a couple of sensors which measures weather data. That data is posted to a cloud service, and in this way the prototypes demonstrates an example of how NB-IoT can be used when it is deployed. By developing the prototypes, measurement could be performed which enabled the authors to compare the two ALPs performance to each other and finding answers to the research questions.

1.2 Limitations and Thesis Scope

This research will be limited to investigating the NB-IoT standard and not any other types of LPWAN standards. NB-IoT is a 3GPP standardized form of LPWAN, as opposed to for example LoRa and Sigfox.

This thesis is limited to the software part of NB-IoT and does not look into link level communication or how the signal is modulated and structured in the NB-IoT technology.

Since 3GPP specifications are not yet finalized for NB-IoT, the thesis is based on the specification of 3GPP release 13 which was released in 2016-03-11.

An IoT system can consist of several components e.g. a client application, a server application, and hardware. This thesis will only look into the client application part of the IoT system.

Because of NB-IoT being in a early stage of development when this thesis were written, it is not possible to implement the NB-IoT technology. Therefore, the applications that are created are using Ethernet to connect to Internet and to demonstrate how NB-IoT can be used. This means that for future work, these applications can be tested with the actual implementation of NB-IoT to further validate them.

The ALPs investigated in this thesis are HTTP and CoAP. These protocols are representative for two types of protocols. HTTP represents the most commonly used protocols for web content, which have been used for a long time. CoAP represents the newer types of protocols developed specifically for IoT and constrained applications and devices. NB-IoT theoretically supports any ALP so in order to limit the scope of the thesis, and as part of the request from ARM, the comparison was narrowed down to the aforementioned ALPs.

1.3 Applications Description

As mentioned in section 1.1 the two application prototypes developed in this research demonstrates a way to use NB-IoT. This usage example is developed by ARM, and it is the only usage demonstrated in this thesis. The usage example is described as a weather station, containing a few different weather related sensors, connected to Internet. This weather station could be placed in a greenhouse allowing the owner of the greenhouse to keep track of temperature, air pressure, humidity, and light through a website.

2 Theoretical Background

This part explains the theoretical background related to this thesis. Its purpose is for the reader to get a basic understanding of the upcoming chapters in this thesis. This section covers some of the cellular networks that are operational. It also covers the Open Systems Interconnection (OSI) model and the TCP/IP model to give an understanding of how an ALP works.

2.1 Cellular networks and Narrowband Internet of Things

This section shortly describes the history of cellular networks and gives background information for NB-IoT. It explains the purpose of using NB-IoT and the advantages of using it in an IoT environment compared to other cellular networks.

2.1.1 Early versions of Cellular Networks

The first generation cellular network where based on several different systems. North America, Australia, China and South America used Advanced Mobile Phone Service (AMPS) and northern Europe and Russia used the standard Nordic Mobile Telephone (NMT). When the first generation cellular network came out in the 1980s it became highly popular to such extent that its capacity was not enough [12].

2.1.1.1 2G

The second generation cellular communication was developed to meet this problem, to give users a quality service and to make a union standard. The key differences between first generation and second generation are:

- The first generation was analog and operated with frequency modulation. The second generation use digital transmission.
- Since the second generation went digital it was easier to encrypt the data to prevent eavesdropping.
- The digital system made it possible to use error detection and correction techniques. This enabled the voice to be much clearer.
- With the second generation it enabled channels to have multiple users as opposed to first generation.

2.1.1.2 3G

The main reason to move from 2G to 3G was the high demand on high speed networks to enable support for not only voice but multimedia, video and data [12]. Some of the improvements of 3G are:

- Higher quality on voice compared to Public Switched Telephone Network (PSTN).
- Support for circuit switched and packet switched data service.
- Wider support for a range of types of mobile equipment.
- Higher bit rates available for fixed and moving users.

2.1.2 Long Term Evolution (4G)

The Long-Term Evolution (LTE) was introduced by 3GPP release 8. It later evolved into LTE-advanced and was an answer to the demands for a mobile broadband service with high data rate, speed and Quality Of Service (QoS)[13][14]. The key improvements of LTE are listed below:

- The speed of LTE, also called 4G, has significantly increased with a provided peak data rate of 300 Mb/s [15].
- With LTE, the signal strength is higher and covers a larger area than previous technologies.
- The architecture has changed from a circuit and packet switching to an all-IP network that increases spectrum flexibility, support larger cell sizes and support co-existence of all ready available mobile standards.

2.1.3 Narrowband Internet of Things

Narrowband Internet of Things (NB-IoT) is a new cellular technology introduced in 3GPP Release 13 for providing wide-area coverage for the IoT [7]. It reuses the LTE design extensively, reducing the time and effort required to develop NB-IoT products. NB-IoT can also be deployed in several ways, listed below.

- As a stand-alone carrier using any available span exceeding 180 kHz in the spectrum
- Inside the LTE carrier spectrum
- In the guard-band of an LTE carrier

The way of deployment makes no difference for the User Equipment (UE) but it has great advantages for the MNO that wants to deploy NB-IoT, thus making it more interesting from a business perspective compared to other versions of mobile communications intended for IoT.

NB-IoT has a couple of key differences compared to regular LTE which makes it more suitable for IoT applications.

1. Coverage is improved through increasing the number of repetitions and thus, trading off data transfer speed. The maximum coupling loss is 20 dB higher than that of LTE (3GPP release 12) [7].
2. Reduced complexity for UE. This is achieved with e.g reduced transport block sizes and single antenna requirement.
3. Battery time is extended, much because of the reduced UE complexity. Examples in 3GPPs technical report[16] shows a possible battery life of more than 10 years for a simple device transmitting 200 bytes of data per day.

These key differences listed above are all of great importance in the IoT domain and it makes NB-IoT a viable option for IoT devices. There are, however, a few trade-offs to consider. Data rate is limited to about 250 kbps at most, which means that the application's data usage must be limited to an amount that NB-IoT is capable of handling.

2.2 The OSI Model and the TCP/IP Model

The OSI model is a standardized concept of digital communication that partitions the communication system into abstraction layers. The OSI model consists of seven layers, each with a specific function. The OSI model is an old standard produced in the late 70's but it's still relevant to this day because it has a capacity to expand to the evolving needs. Most of the work that later became the OSI model was made by a group at the Honeywell information systems because they found there were standardization problems. At the same time the British Standard Institute addressed that there was a need for a unified standard communication standard architecture for distributed processing systems [17].

The OSI model is, as stated, built upon seven layers where the routers and network devices only use the bottom three layers and the host acts in all the seven layers. There are several advantages to have the model divided in multiple layers. If there is a change in one of the layers, the vision is that there should be no effect in the remaining six layers. Each layer describes what function that occur and encourage the industry standardization [17].

The TCP/IP model is another model separating the stack into four layers instead of seven. It was created for ARPANET in 1974 but has remained in use by the Internet until this day. Unlike the OSI

model, this model is based on the protocols that were already in use at that time [17]. The OSI model and the TCP/IP model are similar to each other, and a comparison can be seen in table 1.

OSI Model			TCP/IP Model
Layer	Protocol data unit	Function	Layer
7. Application	Data	Responsible for defining the services and presenting data to the user-end.	Application
6. Presentation		Responsible for translating and presenting the data to the application.	
5. Session		Handle communications for multiple sessions at a time.	
4. Transport	Segment / Datagram	Connection-oriented; reliable transmission, end-to-end error detection and recovery.	Transport
3. Network	Packet	Routing and handling network connections from one network to another	Internet
2. Data link	Frame	Error detection, interconnection control, identification and relaying	Link
1. Physical	Bit	Transporting raw bits over a physical medium	

Table 1: The OSI model and the TCP/IP model as described by Alani in [17] .

2.3 Transport Layer Protocols

The transport layer protocol is the layer handles a connection-oriented or connection-less transmission of packets. There are two main Transport Layer Protocol (TLP)s which are mainly used: TCP and UDP, where TCP is connection-oriented and UDP is connection-less. The main differences of these TLPs are further explained in this section.

This section is based on the Network Protocols Handbook [18].

2.3.1 TCP

The TCP protocol is commonly used for transporting data in applications where reliable transfer and the order of information is important. TCP offers complete handling of packet-loss. This involves error detection, segmentation, and re-transmission of packets when needed. The packet-loss handling works by sending a stream of segments and each of these segments has to be acknowledged by the receiver. If a segment is not acknowledged, it will be re-transmitted.

Providing these reliability features requires a lot of overhead data with each transmission, which can be undesirable in some applications. The format of TCP packet can be viewed in table 2.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Source Port																Destination Port															
Sequence number																															
Acknowledgment number																															
Offset				Reserved				Control Bits				Window																			
Checksum																Urgent pointer															
Option + Padding																															
Data																															

Table 2: The structure of a TCP packet.

2.3.2 UDP

The main difference between TCP and UDP is the reliability functionality. The UDP protocol is most commonly used for transporting data in applications where there is no need for reliable transmission of packets. UDP is basically just an interface between the IP and the upper protocols. The UDP protocol uses very little overhead since it does not support any reliability or error-recovery functions. An overview of the structure of a UDP packet can be seen in table 3.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Source Port																Destination Port															
Length																Checksum															
Data																															

Table 3: The structure of a UDP packet.

2.4 Application Layer Protocols

The ALP of the TCP/IP model corresponds to the top three layers in the OSI model, which is illustrated in table 1, meaning that anything above the transport layer (layer 4) is considered an application protocol in the TCP/IP model. There are a number of different ALPs to choose from. This section presents two of them, HTTP and CoAP, as they are a central part of the research.

2.4.1 HTTP

The HTTP protocol has been used in web context since 1990 and is an established standard for data transfer across the Internet. It operates according to a request/response model, where a client requests a resource from the server and the server responds with the requested resource.

The request contains a request method, a Uniform Resource Identifier (URI) and protocol version which decides what method should be used, on what resource the method should be executed, and what version of the HTTP protocol is used. The response contains a success or error code, and depending on what method was used, it can also contain a message body with information.

HTTP does not handle any loss of messages and if it is to be used in a reliable way, TCP is required.

This section is based on the Network Protocols Handbook [18].

2.4.2 CoAP

CoAP is an application layer protocol defined by the Internet Engineering Task Force (IETF) for use in constrained devices and networks. It provides a request/response interaction model similar to the HTTP model, where a client sends a request, and the server answers with a response, but CoAP requires less overhead information for each request/response. The main difference between HTTP and CoAP is that CoAP features optional reliability by using UDP as the TLP and taking care of the reliability in the ALP instead. CoAP can be viewed as two layers, a messaging layer that deals with UDP and the asynchronous interactions with it, and a request/response layer that handles the Method and Response Codes.

CoAP message types		
Name	Symbol	Function
Confirmable	CON	Confirmable message, providing reliability by requiring an ACK for each CON message.
Non-confirmable	NON	Non-confirmable message, unreliable and does not require an ACK.
Acknowledgement	ACK	Acknowledgement message, used for confirming received CON messages.
Reset	RST	Reset message, used if a received CON message cannot be processed.

Table 4: The different message types in CoAP

CoAP uses four different message types, listed in table 4. By using the CON message type to send data, CoAP can guarantee a reliable messaging service. For each CON message sent, an ACK message must be received with the same ID. If an ACK message is not received within a specified time, the original CON message is re-transmitted. The time it waits for an ACK before re-transmitting increases exponentially until the defined number of maximal re-transmits has been performed. The CoAP protocol is designed so that it can be translated easily to HTTP server-side, making HTTP resources available through CoAP.

This section is based on the CoAP reference document by Shellby et al. [10].

2.5 Wireshark

To view and analyze the communication between two endpoints in a computer network, a packet analyzer is used. The Wireshark tool lets users dive into a computer network and display all the network communication accessible from the computer it is run on. The tool lists all the packets sent in the network in chronological order and features a lot of filtering capabilities to help analyze the data. Packets can be dissected and viewed from the context of the layer that is of interest which is useful when examining both TLPs and ALPs. It even has the ability to install plugins to support new or unusual protocols. Wireshark was created in 1998 and is a commonly used packet analyzer, much because it is released under the GNU GPL license, making it free to use.

3 Related work

This chapter presents articles that are relevant to the research of this thesis by either providing background to the NB-IoT domain, or conducting a similar research. The articles are summarized with emphasis on the parts that are most relevant. In order to find relevant research papers to read and to find related work, the first step was to find some key words to search with on databases such as Institute of Electrical and Electronics Engineers (IEEE) and Science Direct. When relevant articles were found in the databases they had to be categorized so that the most relevant articles were presented in this chapter. The search criteria can be found in appendix C.

3.1 Comparing the cost-efficiency of CoAP and HTTP in Web of Things applications

Levä et al. [19] compared CoAP and HTTP in Web Of Things (WoT) applications in regard to cost-efficiency. The IETF created CoAP to deal with the limitations with HTTP in constrained devices and at the time of the authors research, CoAP was under construction and nearly finished. Therefore, the authors wanted to see when the CoAP is justified to be used economically.

The authors used a analytical tool called "total cost of ownership" where the researchers could follow the cost when using HTTP and CoAP from before the acquisition to disposal of the product. They came to the conclusion that CoAP implementation consumes less power in order to function and does not need to change batteries as often as other devices with different protocols. They also found that less complex and cheaper smart devices is sufficient for CoAP and therefore, are more cost-efficient than powerful and expensive HTTP devices.

The most interesting part of their research is that the use of CoAP drastically decrease the data communication if it's volume based, since the protocol and UDP enable a reduction in transferred data volume. In their research they found that HTTP sends seven times more data during the time CoAP does the same.

3.2 Challenges in the migration to 4G mobile systems

The authors of "Challenges in the migration to 4G mobile systems" [20] have researched how the migration from 3G to 4G should progress and they identified key challenges that may occur and proposed solutions. The key features they bring up is more from a layman point of view. According to the authors the key features are "Support for multimedia services at low transmission cost, personalization, high usability: anytime, anywhere, and with any technology and integrated services". They categorize the key features in three different aspects; system, mobile station and services. The authors have come to the conclusion in the mobile system section that it will need software radio devices that can scan for different signals eg. Wireless Local Area Network (WLAN), General Packet Radio Services (GPRS) and Code Division Multiple Access (CDMA).

In the system section, aspect one is fault tolerance and survivability. This is an area that has been tested and a lot of research has been conducted in order to get high tolerance but only on wired networks and high-speed data networks. A cellular network is designed like a tree-topology with several layers eg. devices, switch and cell. That is, according to the authors, a risky topology because if one layer breaks down, the underlying layers are affected. For example, if a base station malfunctions there will be full or partial service loss in that cell and that affects everyone with an mobile phone.

When this article was published the migration to 4G was not considered to be an easy task and there were a lot of things that needed to be researched before it would function properly. The authors believed that multi-mode user terminals, wireless system discovery, terminal mobility, and QoS support is well studied and could be migrated to 4G but there were a lot more areas that needed to be heavily research before 4G could be released.

3.3 Comparing application layer protocols for the Internet of Things via experimentation

Mijovic et al. [8] were investigating three different protocols: CoAP, Message Queue Telemetry Transport (MQTT) and web socket. They did a comparison to see which one works best in a constraint environment and for IoT devices. The three protocols originate from different application fields where CoAP was designed with simple devices in mind, MQTT where intended to be used over satellite and web sockets to be integrated with web browsers and web servers. To do a realistic research and comparison the authors used cheap and low-complexity devices. The test on the protocols Round Trip Time (RTT) was made using a Local Area Network (LAN) to get a reliable low-latency link, and an IoT configuration in two parts, one with commercial Internet Service Provider (ISP) and one with cellular network.

The first stage in the experimentation was to send messages using the different protocols over LAN. The result is presented in two graphs, one where they compare the protocol efficiency as a function of the payload size and one with the RTT. The CoAP protocol is way better than the other protocols, mainly because it uses UDP instead of TCP. UDP do not have acknowledge and has smaller headers than TCP, making it more efficient. The RTT is much higher on MQTT QoS 1 because it has transport and application layer acknowledge. The other protocols compared have a similar RTT when using LAN.

In the IoT case there is a slightly better efficiency for web socket and MQTT with QoS 0 comparing to LAN but CoAP has a higher efficiency also this time. The authors address that cellular network is more volatile than fixed line networks because it depends on random parameters such as network traffic within a cell. The authors came to the conclusion that more experimentation is necessary with cellular networks and that the results are more an indication in how it could be.

The conclusion in the report is that CoAP with UDP is a far better protocol for IoT devices in an constrained environment. The results show that the efficiency does not change if the network setting alter.

3.4 Transport protocols behaviour study in evolving mobile networks

The article written by Li et al. [21] examines and describes the performance of TCP and UDP within LTE based mobile networks. Due to the high demand for data consuming, user scenarios e.g. applications such as media on demand and cloud services, virtual and augmented reality and portable gaming devices drives development towards new mobile communications. The TCP is reported to be the protocol that 95 % of all the internet data traffic is based on but according to the authors TCP do not behave like it should when deployed in real life. TCPs performance may shift due to how the congestion control function acts in a unforeseeable radio link environment. The authors therefor investigates round-trip-time, Congestion Window Sizes, number of packet loss and end-to-end throughput with TCP and UDP traffic.

The authors simulated the interaction with a network simulator called LENA between the transport protocol and the lower layers in LTE in order to test the protocols. The results of the simulation shows that when the hardware that communicate directly with the user equipment moves away from the user equipments active area, the throughput from both TCP and UDP will significantly reduce. The authors came to the conclusion that UDP achieves better throughput compared to TCP.

4 Methodology

In this thesis, a comparison between two ALPs is made in order to answer RQ 1. Thus, two applications were created, one using HTTP and one using CoAP. The answer to RQ 2 was found as part of creating the applications. In order to construct and evaluate these two applications, the method described later in this chapter was used. *Systems Development in Information Systems Research* by Nunamaker et al. [22] was chosen to provide the main research methodology. This is a methodology that aims to develop a system as a research, which suits the research questions in order to construct and evaluate the applications. One of the key advantages to choose this methodology is the possibility to iterate through all the steps. This is useful when there is work that needs to be revised in an early step. The methodology describes a research procedure consisting of five steps that should be iterated throughout the research process which is illustrated in figure 1.

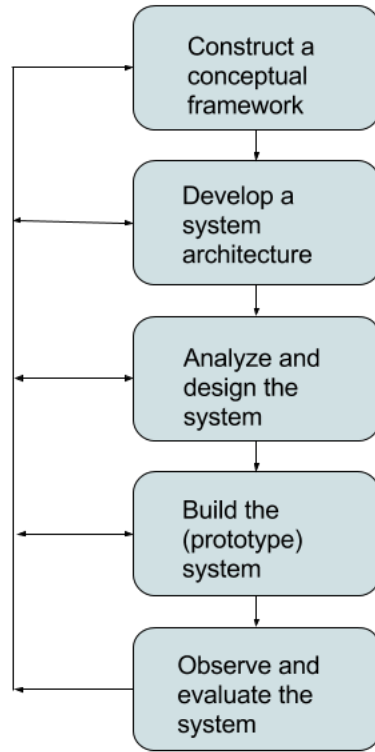


Figure 1: Systems Development Research Methodology, described in [22].

The authors of the methodology [22] has listed some suggestions of what activities should be done in the different steps shown in figure 1. As mentioned in section 1.1, two applications are created for comparison against each other. Therefore, in the second, third and fourth step, two architectures are developed, two designs are made, and two prototypes are built. These prototypes are then evaluated individually and finally compared to each other in the last step of the methodology.

4.1 Constructing a Conceptual Framework

The first step for this research was to construct a conceptual framework. The research questions stated in 1.1 serves as a reference when constructing the framework. The conceptual framework for this research consist of requirements and properties for developing the applications.

4.1.1 Interviews

There were three interviews á 45 minutes with ARM representatives to identify the pre-requisites and requirements for the applications and to get an understanding of how the applications should be shaped. The requirements identified in the meetings were then analyzed through a literature study, to make sure they were relevant to the subject. The pre-requisites and requirements can be found in sections 5.1 and 5.2. The functionality of the application was decided during the interview and laid the foundation for the application, which can be seen in requirement **R 1**. ARM representatives decides that the application should consist of the sensor API described in requirement **R 2**. To be able to store the values from the application we decided with ARM that the application shall connect to a cloud as presented in requirement **R 3**. ARM representatives also gave the advice that the application should use an Real-Time Operating System (RTOS), and because of this, requirements **R 4** and **R 5** were produced.

4.1.2 Literature Study

A literature study was performed in order to gain knowledge about the problem domain. Research about the general area of mobile communication was conducted. Differences between the NB-IoT standard and the older standards were researched. Previous investigations of ALPs [8] were studied in order to determine what requirements should be put on that part of the applications as seen in requirement **R 7**. Different TLPs for the application were also researched [21] to create requirements on that subject and the requirements can be seen in requirement **R 8**. By studying the TCP/IP model and understanding the principles of network communication, the authors understood that there was a need for requirement **R 6**.

4.2 Develop a system architecture

In this stage two system architectures with different connectivity solutions, one using HTTP and one using CoAP, were developed. Based on the requirements for the applications as well as the pre-requisites given for the platform, the basic building blocks of the architectures were identified. Drafts of the system architectures were developed and presented to ARM representatives. After this, ARM representatives contributed with input on how the systems architectures should be designed. The result of this procedure is presented in section 5.3.

4.3 Analyzing and Designing the System

The first step in analyzing the system was to examine the requirements for NB-IoT to get an understanding on how to limit the scope of the applications and to get an overview of NB-IoT. The information needed was found in the Narrowband IoT primer [7] and it gave an insight on the requirements of NB-IoT. Next stage in the analyzing and designing process was to establish the needs for the system. As stated in section 4.1.2 a research about the requirements for the application were conducted. It was decided that two separate versions of the application should be designed, one which uses HTTP and one which uses CoAP. According to the research by Mijovic et al. [8] CoAP is one of the most effective protocols for use in an IoT environment. The HTTP protocol is a widely used protocol in devices and a comparison with CoAP is therefor interesting for the reader.

Two different designs were produced, and can be seen in section 5.3.1 and 5.3.2. The architecture was kept as similar as possible between the two versions to ease development and implementation processes.

4.4 Prototypes Construction

In order to compare and measure the different solutions two prototypes were created. These prototypes were based on the architectures described in section 5.3. The prototypes were constructed with an agile development approach in collaborations with ARM with two week long sprints, short stand-up meetings every day and a backlog to keep track of the tasks involved in constructing the prototype.

4.4.1 Development Tools

The development was performed using git as a version control system. This helped keep track of changes and if needed, revert the changes. Gerrit Code Review was used for getting feedback and reviews of the code continuously. The code was reviewed by ARM representatives to make sure it followed their coding standard and that it did not contain any obvious faults. This was done every time a new commit was pushed to the server. If the code reviewer determined that the commit did not meet the coding standard or if he or she found any faults, the commit had to be updated with the changes pointed out by the reviewer. To further improve the code verification the code was also built by a system called Jenkins in order to verify that it built successfully with the rest of the system. The process of developing a feature can be seen in figure 2.

The complete process of constructing the prototypes is described in section 5.4.

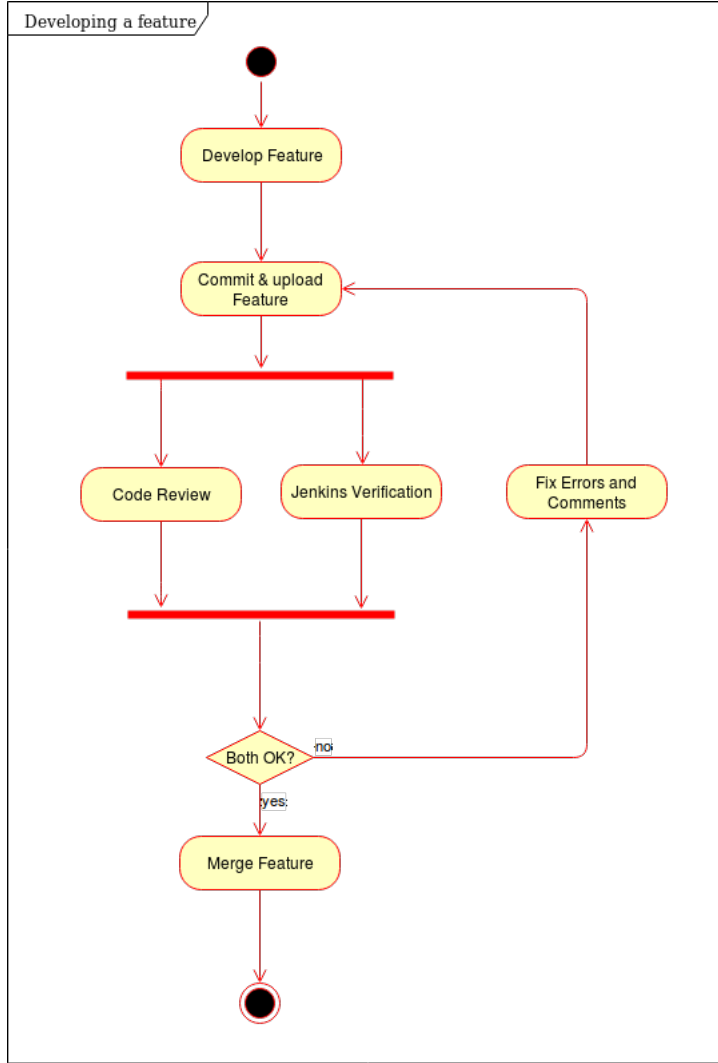


Figure 2: The process of developing a feature.

4.5 Validation of the Prototypes

To ensure the validity of the prototypes they were tested with different test cases that can be viewed in appendix A. The test cases were produced to evaluate the prototypes functionality and to measure their performance. The validation process is to be able to answer the research question and to encounter problems in an early stage of the prototype building.

4.5.1 Measurements and Testing Procedures

To test and compare the HTTP prototype and the CoAP prototype against each other and to find the answer to research question 1 in 1.1, measurements were made with regards to execution time, data usage and packet size.

Measuring the execution time was made by setting an output of the Xilinx board to high level in the beginning of the measurement, and to low level in the end of the measurement. This output was then

connected to a logic analyzer which could measure the time that the output was high level. Execution time was measured for the procedure of formatting a CoAP/HTTP packet based on sensor values. The measurements taken was then graphed using Matlab software.

Measuring the data usage and protocol effectiveness was done by running the two applications and monitoring the network traffic in the network protocol analyzer Wireshark. Calculations were made based on that data and presented in section 5.5.

5 Results

This chapter presents the pre-requisites from ARM, requirements for the applications and NB-IoT, the construction of the prototype applications and it also presents data gathered from tests and measurements. The requirements were produced to answer RQ 2 and to create a conceptual framework for the applications. To answer RQ 1, the two applications were designed, developed and then tested, producing results for drawing a conclusion and thus answering the research question.

5.1 Pre-requisites

To develop the applications ARM provided a hardware platform along with a development environment. The hardware platform provided is a development platform called Xilinx Zynq-7000. The board is multi-functional and suitable for construction of IoT constrained applications. It is a Field-Programmable Gate Array (FPGA) board specialized for prototype development and enables a wide range of applications. The Xilinx Zynq-7000 based development platform uses Dual ARM Cortex-A9 MPCore and enables up to 667 MHz operation.

The development environment is specific to the hardware provided from ARM. The development is conducted in a eclipse-based Xilinx SDK. It is made solely for development to Xilinx boards, with debug functionality and Xilinx specific tools for building a project towards the Xilinx Zynq-7000 platform.

The applications should handle four sensors simultaneously using the Inter-Integrated Circuits (I²C) protocol. The sensors acts like slaves and the Xilinx Zynq-7000 is defined as a master which calls the sensors whenever required. One of the benefits using the I²C protocol is that it needs only two wires from all the slaves to master. As mentioned in requirement **R 1** the sensors need to be able to measure temperature, air pressure, humidity, and light. Based on this, four sensors enabled for I²C were chosen by ARM to use with the application.

The applications consist of four sensors that gather data and sends to the applications. As mentioned in section 1.3, the application are a weather station and therefor the following sensors were provided from ARM:

- **T5403** A barometric sensor that reads temperature and pressure.
- **Pmod TMP2** A temperature sensor.
- **TSL2561** A light sensors that calculates the value to lux.
- **Si7021** A combined humidity and temperature sensor.

Another pre-requisite was the operating system. As is mentioned in requirement **R 4**, a RTOS was required to be used. ARM specifically required for the authors to use the FreeRTOS for the applications because it was already ported to the platform that was used, which enabled faster development of the applications. The application should be built on an IoT device and therefor be programmed using a well known language for constrained devices. The programming language chosen for this thesis was ANSI-C.

5.2 Requirements

Several requirements were produced for the applications in order to answer RQ 2. There are two types of requirements; requirements produced by the authors of this thesis in collaboration with ARM, and requirements set by the NB-IoT standard which has been researched.

5.2.1 Applications Requirements

As a result of the three interviews with ARM mentioned in section 4.1, and the literature study mentioned in section 4.1.2, requirements for the applications was decided. This section presents the requirements.

R 1 Functionality

The application should resemble a weather station, with sensors for measuring temperature, air pressure, humidity, and light. The user should be able to read these measurements from the cloud service via a website. The following requirements were produced to ensure a functionality that satisfies the use case.

- (a) The application shall gather data from the sensors and convert it to the formats specified for four sensors.
- (b) The application should format the values and post them to the cloud.
- (c) Users of the application should be able to choose an interval of posting values of their own choice with predefined commands. This interval should be able to vary between 10 seconds and 60 seconds.

R 2 Sensor drivers Application Programming Interface (API)

The sensor drivers have a common API in order to make the application commands uniform. This would make it easier to dynamically add or remove sensors during run-time, because then you can for example iterate through the sensors and read each one in the same way.

R 3 Cloud Connectivity

The applications should connect to a cloud service available on the internet. The cloud service should be able to store the sensor values sent to it by the applications.

R 4 RTOS

To control the execution rate of the different threads in the system, an RTOS is needed. An RTOS is a type of operating system where the user can very accurately control timings and execution rates of tasks (or threads) in the system.

R 5 Multi-tasking

The application should be able to handle multiple tasks, or virtual threads, simultaneously so that operations do not freeze the application. Multi-tasking is mainly used to get lesser resource consumption and it provides a higher responsiveness.

R 6 TCP/IP Protocol Stack

To enable Internet connectivity for the applications, it needs to build on a protocol stack with drivers for the hardware platform. The Xilinx Zynq-7000, mentioned in pre-requisites 5.1, is delivered with a ported version of the Light-weight IP (LwIP) protocol stack which ARM required for the authors to use in order to facilitate for a faster development of the application.

R 7 Application Layer Protocol

To format the output data that should be sent to the cloud service, an ALP is needed. The application layer protocol formats the data in a specific way which can later be interpreted using the same protocol on the receiving end. IoT applications put more stringent demands on efficiency and therefore, special requirements will apply to this protocol. The criteria that the protocol must fulfil are:

R 7.1 Efficiency

The protocol must be efficient in terms of data usage.

R 7.2 Reliability

The protocol must be able to reliably transfer data.

R 7.3 Execution

The protocol must execute fast, taking up little execution time in the system.

R 8 Transport Layer Protocol

Because of requirement **R 6**, stating that LwIP should be used, the transport layer protocol has to be one of UDP or TCP, since LwIP only supports those two [23]. Furthermore, the TLP needs to be adapted to the choice of ALP, meaning that if the ALP does not handle loss of packages, the TLP has to do it, and vice versa.

5.2.2 NB-IoT Requirements

There are several requirements for NB-IoT listed in the NB-IoT primer [7]. As stated in section 1.2 the specification for NB-IoT are presented in 3GPP release 13. The main requirements for NB-IoT are transmitting data, latency and capacity. The requirements are compared through the application with Ethernet connection and then with the specifications of NB-IoT.

According to the primer [7] the peak data rates for transmitting data are calculated using the highest transfer block size and transmitting it over 3 ms. The transfer block size for narrow band downlink is 680 bits and after calculations it will have a downlink of 226,7 kb/s layer 1 peak data rate. On the uplink the transfer block size is 1000 bits, therefore the uplink peak data rate will be 250 kb/s.

However, in conditions where coupling loss is the theoretical maximum 170 dB, the peak data rates are much lower. Downlink peak data rate is estimated to 35 bps and uplink peak data rate is estimated to 20 bps. In other words, the data rate decreases with the signal strength. This requires the applications to be efficient in its data usage. Using less data will enable an application to be used in places with lower signal strength.

5.3 System Architecture and Design

This section describes the different architectures that was designed for the applications. There are two versions of the architecture. The HTTP version is compared to the CoAP implementation and the combined result of this serves to answer RQ 1. They both build on the same base architecture but differ in some parts, as is described in the following sections. A figure describing the complete system can be found in appendix D

5.3.1 HTTP based application

The first architecture, shown in figure 3 is the HTTP version of the application. It contains a very simple HTTP library used for posting data to the cloud.

Transport Layer Protocol

Based on requirement **R 8**, which states that if the ALP does not handle loss of packages the TLP needs to handle it, TCP was chosen as the TLP for the HTTP implementation. As mentioned in section 2.4, HTTP does not handle loss of packages, but TCP does.

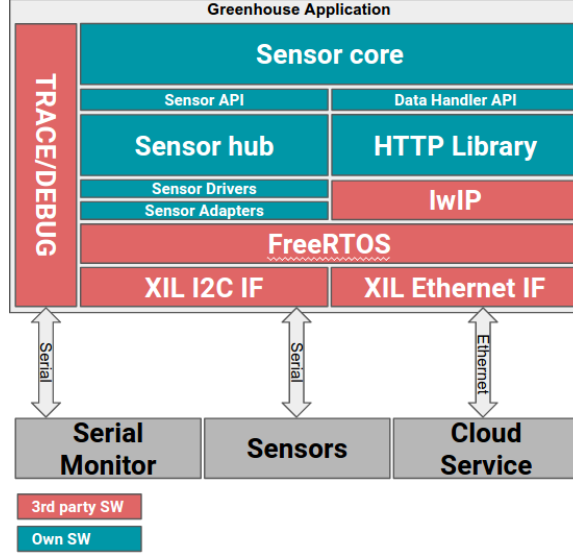


Figure 3: System Architecture using HTTP solution.

Architecture and Design

The gray container labeled *Greenhouse Application* is the part that was designed in this thesis. The red blocks are third party software, not created by the authors, but was implemented in the application. The teal blocks is software that was created by the authors. The functionality of the application is explained block by block in the list below.

- The **sensor hub** is a central block that connects to all the registered sensor **adapters** and **drivers**. The adapters are responsible for a sensor specific implementation of the I²C interface as well as initializing the sensor if required. The drivers handle the way of communicating with the sensor. The drivers all have a common interface towards the rest of the application. The part of the application that wants to read a sensor simply call the *read* function of the selected sensor, and the value will be read and set to the supplied variable. The sensor hub is responsible for connecting the adapters to the sensors and linking the communication between these.
- The **sensor core** is the main task running the application. It contains a list of sensor objects registered in the sensor hub and through that list it is possible to access the sensor interface for each sensor. The sensor core keeps a command queue and takes care of a command when it receives one from the Command-Line User Interface (CLI). Commands can be to post sensor data to the server or to return sensor data from the sensors through the trace module. The core can also be set up to periodically self-invoke a command.
- The **data handler API** is a string builder which concatenates the string in the correct format for it to be passed on to the HTTP library which then makes a HTTP request out of it. The core simply passes on the sensor values to the data handler through the API and then the data handler takes care of the formatting.
- The **HTTP library** uses LwIP API functions to connect to a defined server and send requests. The server address is defined when initializing the HTTP library, which is done by the data handler. The application uses a default server defined in the code, but can be configured to use a different server via a CLI command.

5.3.2 CoAP based application

The other version of the applications is based on a CoAP client. The architecture is similar, but there are a few differences. Since the CoAP client is more advanced than the HTTP client, and since it, unlike the

HTTP solution, is not just a passive library but depends on its own threads, which are used to handle the reliability part of CoAP, some architectural changes had to be made which are described in the following sections.

Transport Layer Protocol

UDP was chosen as the TLP in the CoAP application. It was chosen primarily based on requirement **R 8** which states that if the ALP handles loss of packets, then the TLP should not do it, and since CoAP does handle loss of packets the TLP should not do it in this case. The choice was also based on the fact that CoAP was built for use of UDP [10], and based on that according to the research done by Li et al. [21], UDP has higher throughput in constrained environments with low signal strength.

Architecture and Design

The CoAP solution architecture is shown in figure 4. The application is using the same sensor handling but most of the connectivity part has been exchanged. One big difference is that in this implementation it is not the sensor core that is the main thread, as opposed to in the HTTP implementation. The CoAP application runner, as seen in figure 4, is the entry point and main task, and it is controlling the data flow in the application. It accomplishes this by sending commands to the sensor core requesting sensor values, and receiving the values asynchronously through a callback function. The asynchronous part is critical for this architecture to work because there are two tasks running, the sensor core and the CoAP client, that are sharing information and by doing the information sharing asynchronously both tasks can continue executing instead of the first task waiting for the other to finish before being able to continue. When the application runner receives the values it requested, it forwards it to the CoAP client to post it to the cloud.

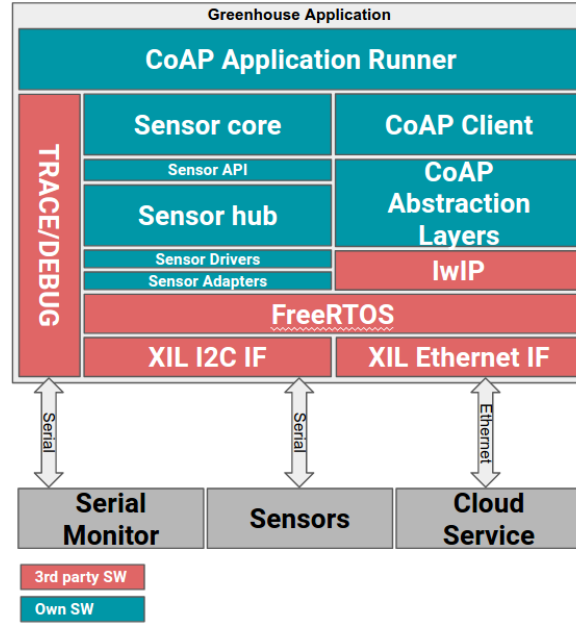


Figure 4: System Architecture using CoAP solution.

5.4 Prototype Construction

This section describes the construction of the prototype and the work behind the Greenhouse applications. As stated in the section 5.3 there are two different design implementations using CoAP and HTTP. The first step were to get a functioning HTTP solution. The implementation was easier and the client less complex than with CoAP.

5.4.1 HTTP design implementation

The first step was to get the sensors to work one by one and test them thoroughly with different test cases as seen in appendix A. Every sensor have a corresponding data sheet specifying the characteristics, performance and how they shall be implemented in hardware and in software. To print the values from the sensors and to see that it displays reasonable values a CLI command was implemented and used. When all the sensor adapters worked separately, the next step was to implement a CLI command that read the values of all connected sensors. This concluded the sensor part of the application.

The next challenge was to use the collected sensor values, by putting them in HTTP packets and post them to the HTTP server. This was solved by implementing the data handler, which is essentially a string builder. It takes an array of sensor value objects, and puts them in a HTTP request string next to their corresponding identifier.

The last step comprised to implement the HTTP client to connect to a cloud service and post the request string to it. This was done in an asynchronous way so that LwIP handles the connection and the HTTP client receives callbacks when events happen, for example when a request is sent.

The advantages to implement HTTP design presented in section 5.3.1 on the application is the simplicity to apply it. It has very few dependencies other than LwIP which makes it easy to implement in a system.

5.4.2 CoAP design implementation

The CoAP implementation is similar to the HTTP in the underlying sensor core and hub. Internet connectivity is done in a different fashion, where the application is connected with a CoAP client. The client is a library which enables communication with a CoAP enabled cloud service.

Implementing the CoAP library was a lot more complicated than implementing the HTTP library. The CoAP library has many dependencies which had to be imported in the project and linked correctly in the build settings. Furthermore, the CoAP client was not adapted to this specific platform, which meant that some configurations had to be changed for it to work. A problem that occurred was that in FreeRTOS, the timer task overflowed its stack and this had to be configured to allow a larger stack size. However, the option for configuring the timer task stack size was not working correctly. The problem was solved by correcting a bug in the development environment libraries, making it possible to change the stack size as expected.

While implementing the CoAP client, spoofed sensor data was used to push to the cloud in order to see that the communication with the cloud service was working. When that was working, the sensor core needed to be tied to the CoAP client. Since they are both tasks running independently from each other, it was necessary to allow the communication between these to be asynchronous. This was solved by letting the sensor core receive a callback function which it could execute when it had gathered the sensor data requested by the CoAP application runner. The callback proceeded to post the values through the CoAP client. In that way, none of the tasks were waiting for the other task, but could continue executing until the callback was called.

5.5 Testing and Validation

To validate that the applications fulfill the requirements listed in section 5.2, testing had to be performed.

Each of the applications were tested according to the functionality requirements. They were also tested in their differences in order to answer the research question. By measuring the execution time of building a HTTP packet and CoAP packet respectively, the effectiveness could be compared. By measuring packet sizes in a protocol analyzer, the data usage could be compared. This chapter presents a summary of the tests performed and the result of them. It also presents a summary comparing the test results to each other.

5.5.1 Testing the Functionality of the Applications

As stated in the requirement **R 1** in the section about requirements there are three different requirements for the functionality of the applications. Testing of these properties was performed on each of the two applications to validate that they were fulfilling the functionality requirements. The test protocol can be viewed in appendix A.

The first test on the sensor drivers is to see if they connect to the applications as they should, with an acknowledge from the sensor. The test description can be viewed in figure 9. Similar tests are conducted with two, three and four sensors together.

The next test is the data gathering from the sensors which also involves testing that the sensor drivers calculates the correct measurement based on the raw data received from the sensors. The description of the tests can be viewed in figure 13 to figure 16. The procedure is as follows: the application starts up and send signals to the sensors to get a response that they are connected. The sensors responds and the printout can be viewed by the user in the terminal. The user writes a CLI command in the terminal and send it to the application to get the sensor values. The sensor responds with a value and presents it in the terminal.

In the third test, the application should format the values and post it to the cloud. The values should then be accessible on the cloud service. The test description can be viewed in figure 18 and figure 19.

This test differed somewhat depending on which application that was tested. When testing the HTTP application, the application gathered the values from the sensors and then it formats the value and posts it via HTTP. The sensors values are then presented on a website accessible from a web browser. A similar test is conducted using the CoAP application where it sends the values to a different website, which presents the values in a graph.

The last test is a test where the user should be able to choose the interval of posting values to the cloud. The user writes a CLI-command in the terminal stating which interval the application should post the values to the cloud. The application starts posting values as soon as the user types in another command.

All tests succeeded according to the test criteria viewed in the corresponding test ID. The first and second test followed throughout the construction of the application as a quality check.

5.5.2 Testing the HTTP implementation

The HTTP protocol is a widely used protocol for transferring web content between endpoints. HTTP, however, does have the disadvantage of requiring a lot of overhead information in each request. Another disadvantage is that it requires a TCP connection between the endpoints, which can be problematic when a mobile connection is used. It does, however, guarantee reception of the messages sent, thus meeting requirement **R 7.2**.

The other requirements were measured as mentioned in section 5.5 and the results for each of them are listed below.

Packet Build Time

Build time for HTTP packets was measured during 50 cycles. In figure 5, the measurements has been plotted. Using this data, the following performance figures were calculated:

1. The average build time was calculated to $198\mu s$.
2. The build time standard deviation was calculated to $1.2\mu s$.
3. The percental build time standard deviation was calculated to 0.6% of the average build time.

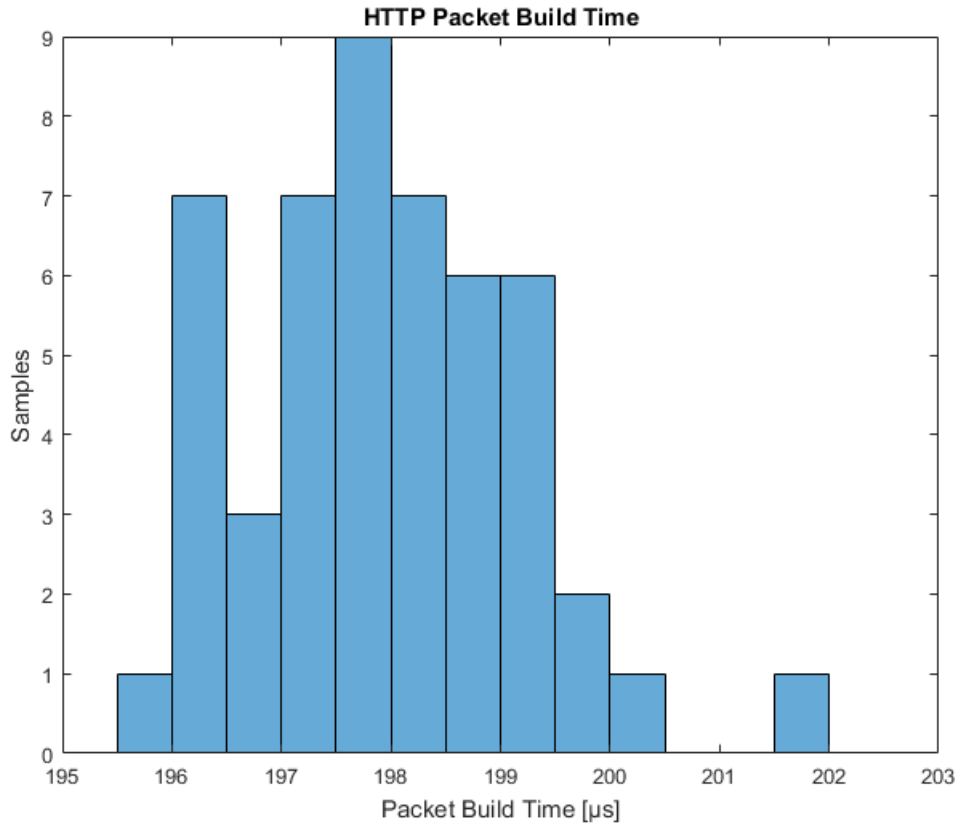


Figure 5: Plot of the time it takes to build a HTTP packet.

Packet Size and Data Usage

The packet size of the HTTP request varies with the data sent, but typical packet size is 160-170 bytes. A single conversation of sending one sensor value involves the following steps:

1. Application connects to the server
2. Server acknowledges the connection
3. Application sends the HTTP request
4. Server sends the response
5. Application closes the connection
6. Server acknowledges the closing
7. Application closes the socket

The whole process uses a total of 970-980 bytes. Since HTTP uses TCP the process needs to perform a lot of steps before actually sending the request.

In an example where the application transmits all four of the sensor values every 10 seconds, the absolute minimum needed link layer data transfer speed is calculated to 3126 bits per second.

A typical request can be seen in figure 6. The sensor values are placed in the URI next to their respective key name, making the request very compact.

```
GET /input/{public_key}?private_key={private_key}&temp=2372
HTTP/1.1
Host: data.sparkfun.com
```

Figure 6: A typical HTTP request.

In figure 7 a typical HTTP response can be observed. It contains information about the server, what HTTP methods are allowed, and much more, but the only thing this application really needs is the first line, which indicates if the transmission went well or not.

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET,POST,DELETE
Access-Control-Allow-Headers: X-Requested-With, Phant-Private-Key
Content-Type: text/plain
X-Rate-Limit-Limit: 300
X-Rate-Limit-Remaining: 299
X-Rate-Limit-Reset: 1493728626.26
Date: Tue, 02 May 2017 12:22:06 GMT
Transfer-Encoding: chunked
Set-Cookie: SERVERID=; Expires=Thu, 01-Jan-1970 00:00:01 GMT; path=/
Cache-control: private
```

Figure 7: A typical HTTP response.

5.5.3 Testing the CoAP implementation

CoAP is a lightweight version of HTTP focused on IoT platforms. It has reduced overhead information and can be used with UDP. Because the protocol is based on the RESTful protocol, it is also easy to translate a CoAP message to a HTTP-request server-side.

Another important feature of CoAP is the way it handles messaging. Since CoAP is built for use with UDP, which in itself does not provide a guarantee that each message is received, the CoAP protocol has solved this on application protocol level instead. Each message sent with the CON-flag set requires the recipient to reply with a message acknowledging the reception of the message containing the CON-flag. If the sender does not receive the acknowledgment, it will resend the message. In this way, CoAP can guarantee that the recipient has received the message even though the transport is done by UDP, and therefore meets requirement **R 7.2**.

Packet Build Time

Build time for CoAP packets was measured during 50 cycles. In figure 8, the measurements has been plotted in a histogram. Using this data, the following performance figures were calculated:

1. The average build time was calculated to $92.1\mu s$.
2. The build time standard deviation was calculated to $0.5\mu s$.
3. The percental build time standard deviation was calculated to 0.5% of the average build time.

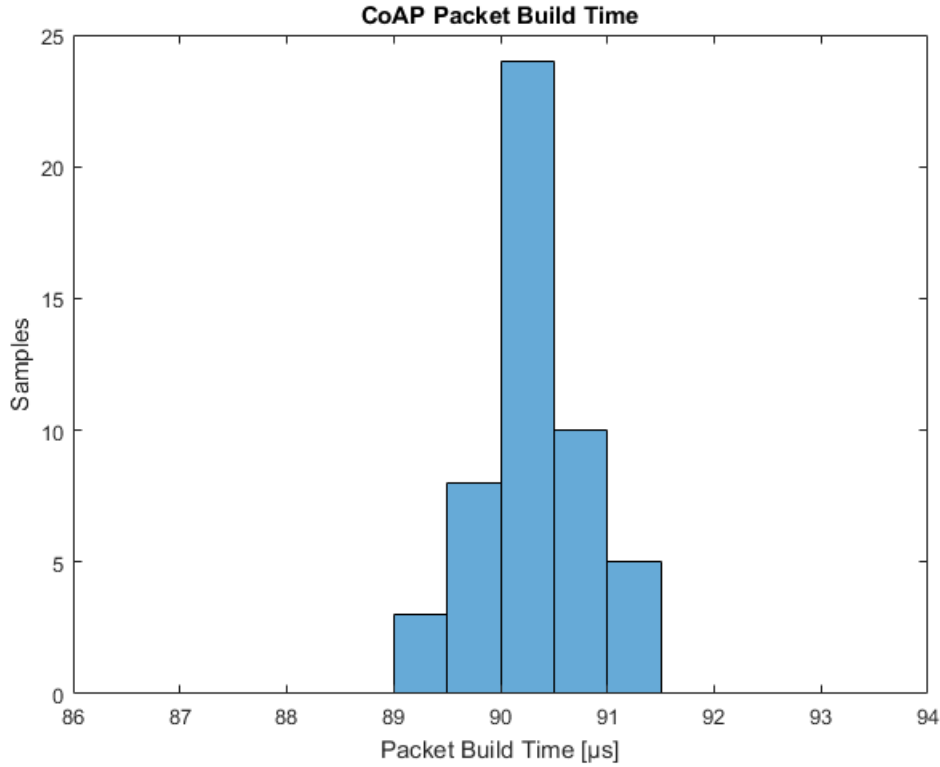


Figure 8: Plot of the time it takes to build a CoAP packet.

Packet Size and Data Usage

As with the HTTP implementation, the CoAP packet sizes also varies with the data sent, but typical packet size for a CoAP packet containing sensor data is 96-110 bytes. The procedure of sending the data is different from the HTTP implementation in that since the CoAP implementation uses UDP instead of TCP, there is no need to establish a connection to the server before sending the packet.

A single conversation of sending one sensor value involves the following steps:

1. Application sends CoAP packet
2. Server acknowledges CoAP packet

The whole conversation typically consists of 180-190 bytes of data, where the acknowledging messages takes up 83 of those bytes. Because of the way CoAP handles packet loss, this conversation is very simple, and does not require any handshaking before transmitting the actual data.

In an example where the application transmits all four of the sensor values every 10 seconds, the absolute minimum needed link layer data transfer speed is calculated to 588 bits per second.

5.5.4 Test and Validation Summary

In order to draw conclusions from the test results, the performance of the two applications must be compared with respect to the requirements. In table 5, the results from the testing procedure are listed. As can be seen, CoAP did perform better in all of the test aspects.

Measurement Description	Applications	
	HTTP	CoAP
Packet Building Time - Average	199 μs	92 μs
Packet Building Time - Standard Deviation	1.2 μs	0.5 μs
Packet Building Time - Standard Deviation % of average	0.6%	0.5%
Data Usage - Single value	980 bytes	190 bytes
Data Usage - All 4 values	3920 bytes	760 bytes
Minimum Link Speed - Posting every 10 seconds	3126 bps	608 bps

Table 5: Comparison of the two applications

In requirement **R 7.1** it is stated that the protocol must be efficient in terms of data usage. CoAP uses on average 790 bytes less than HTTP, proving it to be more efficient. Requirement **R 7.3** states that the protocol must execute fast. Also in this case the CoAP protocol proves to be the winner, executing in 184 μs less than HTTP.

In section 5.2.2 it is stated that maximum link speed is 227 kb/s for the downlink and 250 kb/s for the uplink. This requirement is clearly met by both of the applications. However, as the signal strength decreases, link speed is decreased as low as 35 bps for the downlink and 20 bps for the uplink.

It is important to note that none of the applications are capable of operating at the lowest speed of NB-IoT. But it is clear that by using less data the CoAP application is capable of operating at much lower speeds, and thus, much lower signal strength, than the HTTP application.

6 Discussion

Mobile communication is a constant evolving area where connected devices will be more or less obsolete as soon as a new standard reaches the market. With that in mind, its hard to say how long the results in this thesis is valid. The organization 3GPP is developing new standards constantly in order to meet the needs for upcoming connected Machine To Machine (M2M) devices and cellphone users. This thesis is based on a certain 3GPP specification and not all the available standards.

6.1 General Discussion

At the time this research was conducted, NB-IoT was still in an early stage of development. Actual commercial implementations of NB-IoT were non-existent, which means that the layers 1 to 3 in table 1 was not available for the NB-IoT technology yet. This have made it impossible to verify the applications by implementing the NB-IoT technology and those layers were replaced by Ethernet versions of the layers. The conclusions regarding NB-IoT in this thesis were therefore drawn based on the theoretical knowledge about NB-IoT that is available. Had the research been conducted at a later time, when NB-IoT was in a later stage of development, verification by implementing the NB-IoT technology could have been performed. This is mentioned in section 7.2 and suggested as future work.

6.2 Method Discussion

The chosen method by Nunamaker et al. [22] proved to be a useful asset when constructing the prototypes. The construction of the framework in collaboration with ARM was a helpful step to identify the requirements for the applications. This was a necessary step in a construction process in order to not go head first into the problem.

Designing and constructing the prototypes enabled the authors to perform tests based on real world data, instead of the tests being based purely on theoretical conclusions and simulations. By constructing two versions of the prototype with different ALPs the opportunity to compare them against each other was enabled. This was important in order to give an answer to RQ 1.

6.3 Validation of the Requirements

The requirements composed a conceptual framework for the designing process of the applications. They were developed by interviewing ARM representatives as well as conducting a literature study. By interviewing ARM representatives, the requirements were narrowed down to a scope in which ARM have an interest, but which also is relevant to other researchers and readers. Arguably, these requirements could be used by any company developing software for use of NB-IoT. The requirements are also suitable for developing applications towards other mobile communications standard, with focus on constrained IoT environments and LPWAN type mobile communication standards. There will be more requirements needed for the implementation of an LPWAN technology, which will be specific to the communication standard used, but the requirements presented in this thesis serves as a reference for constructing an application with respect to low data usage.

6.4 Test Results

As mentioned in section 5.5.4, CoAP does perform better than HTTP with respect to the requirements set for this kind of resource constrained application. The protocols were tested in packet build time, which measures the efficiency of the protocol implementations. This is important in IoT applications to minimize execution time to save clock cycles and, consequently, reduce energy usage. To evaluate the stability of the protocol, the standard deviation was also calculated and compared to the average build time. The protocols were also tested regarding the data usage aspect, and CoAP proved to use less data in this test.

This is an important aspect specifically when using NB-IoT because, as mentioned in section 5.2.2, data rates are limited with the NB-IoT technology and this research makes it possible to minimize data usage by choosing the optimal ALP.

6.5 Comparison to Related Work

In section 3 we presented four previous works related to our thesis. Two of the papers are looking into 4G and IoT, while two are looking into the protocols for mobile networks. The common denominator for the articles is the mobile communication.

The authors of the article [8] "Comparing application layer protocols for the Internet of Things via experimentation" compares three different protocols, CoAP, MQTT and WebSocket. They come to the conclusion that CoAP is the most effective protocol out of these three. Our results of this thesis confirm that CoAP is in fact more effective than HTTP as well, which is in line with what the article concludes.

The article by Levä et al. [19] also compares HTTP to CoAP, but they have an economic perspective of the comparison. In their calculations, however, they include an estimation of data usage for each of the protocols. They calculate that HTTP uses about 7 times more data than CoAP, which differs a bit from the results in this thesis which show that HTTP uses about 5 times more data than CoAP. This difference of results could be a result of CoAP not being fully completed when they conducted their research and so their research was based on theoretical values. Our results are produced by measuring an actual implementation of the two protocols, providing a practical aspect of the comparison. The results are, however, quite similar and so it is very much in line with what that research concluded.

7 Conclusions

This thesis describes two ways in which an application can be designed for demonstrating a specific use case of NB-IoT. These two applications have been measured and compared against each other to give the reader some guidelines for best practices when designing applications for the IoT domain, and also to give an idea of what ALP to use when doing so. The thesis also presents a list of requirements applicable to designing other applications in the IoT domain.

7.1 Answering the Research Questions

This thesis presents two research questions in section 1.1. To answer these research questions, the following actions were taken:

1. Requirements for the applications were researched and produced.
2. Two applications were designed and developed, one using HTTP and one using CoAP.
3. Testing of the functionality was performed on each of the applications to verify them with respect to the requirements.
4. Measurements were made regarding data usage and execution efficiency on both of the applications. These measurements were used to compare the applications performance.

By measuring the performance of the two applications, and comparing the results, RQ 1 could be answered. The answer is that the CoAP protocol proved to be more effective in terms of execution time, packet size and data usage. By studying the result of the test and validation process, the conclusion can be drawn that by choosing CoAP over HTTP, an IoT application can be more effective in terms of data usage and execution. As can be seen in section 5.5.4, by choosing an ALP that is more effective and uses less data, an application can be used with lower data speeds, and thus in the case of NB-IoT, lower signal strength than with a less effective ALP. This has been proven by researching requirements of NB-IoT, and then comparing the HTTP protocol against the CoAP protocol by implementing each of them in similar applications.

The list of requirements presented in section 5.2 serves as a reference for constructing a similar application in the IoT domain. Some requirements are more specific to the application created in this thesis, while others are more general for IoT and LPWAN type of applications. **R 1 - R 3** presents requirements applicable to those looking to create a similar application, functionality wise, that is described in this thesis. **R 4 - R 8** presents requirements that are important to almost any IoT application that uses a constrained mobile communication standard such as NB-IoT. These requirements give an answer to RQ 2.

7.2 Future Work

As NB-IoT is yet to be put into use in the real world, the applications in this thesis are using Ethernet for connecting to the Internet. When the NB-IoT standard gets implemented and deployed by MNOs in the future, the conclusions of this thesis could be tested by designing an implementation that uses the actual NB-IoT network instead of Ethernet.

This thesis investigates HTTP and CoAP. New protocols customized for IoT and NB-IoT in particular might be available in the future, as M2M communication is a growing market, it is likely to be explored and optimized further. The research of this thesis could then be conducted on newer protocols to answer the same research questions.

There are also research possibilities in the use of NB-IoT's Short Message Service (SMS) service if it gets deployed. As Rohde & Schwartz states in their paper [24], the MNOs will choose whether or not they want to deploy the SMS service. If they choose to do so, research needs to be done using this as a replacement for IP based networking. A comparison between IP based and non-IP based networking could

then be researched. As stated, the application is connected via Ethernet to the internet when it post the sensor values. To simulate the scenario when an application send information over NB-IoT, the application could be integrated with a Global System for Mobile communication (GSM) module.

7.3 Contributions

Answering RQ 1 gives the reader a guideline to what ALP to choose for an application in the IoT domain, and also how much of a difference it makes in such an application by choosing CoAP instead of HTTP. By comparing the test results to the requirements for NB-IoT, this research provides an indication for what it will take to adapt an application to using NB-IoT. Furthermore, the thesis presents a list of requirements which could be used as a reference when constructing other IoT systems focused on constrained types of mobile communication standards.

References

- [1] Nurse JRC, Erola A, Agraftiotis I, Goldsmith M, Creese S. Smart Insiders: Exploring the Threat from Insiders Using the Internet-of-Things. 2015 Sept;p. 5–14.
- [2] 3GPP. About 3GPP; 2017. Available from: <http://www.3gpp.org/about-3gpp/about-3gpp>.
- [3] Sigfox. Sigfox LPWAN; 2017. Available from: <https://www.sigfox.com/en>.
- [4] Alliance L. LoRa Alliance Wide Area Networks for IoT; 2017. Available from: <https://www.lora-alliance.org>.
- [5] Morris I. Vodafone to 'Crush' LoRa, Sigfox With NB-IoT. 2016 April;.
- [6] Morris I. Telia 'Betting' on NB-IoT Over LoRa, Sigfox. 2016 Nov;.
- [7] Wang YPE, Lin X, Adhikary A, Grövlén A, Sui Y, Blankenship YW, et al. A Primer on 3GPP Narrowband Internet of Things (NB-IoT). CoRR. 2016;.
- [8] Mijovic S, Shehu E, Buratti C. Comparing application layer protocols for the Internet of Things via experimentation. 2016 Sept;p. 1–5.
- [9] Berners-Lee T, Fielding R, , Frystyk H. Hypertext Transfer Protocol – HTTP/1.0. 1996 May;.
- [10] Shelby Z, ARM, Hartke K, Bormann C. The Constrained Application Protocol (CoAP). 2014;.
- [11] ARM. ARM Marketshare; 2017. Available from: <https://www.arm.com/markets/mobile>.
- [12] Stalling W. Wireless communication and networks. Pearson; 2004.
- [13] Astely D, Dahlman E, Furuskär A, Jading Y, Lindström M, Parkvall S. LTE: the evolution of mobile broadband. IEEE Communications Magazine. 2009 April;47(4):44–51.
- [14] Kanchi S, Sandilya S, Bhosale D, Pitkar A, Gondhalekar M. Overview of LTE-A technology. 2013 Nov;p. 195–200.
- [15] Shen Z, Papasakellariou A, Montojo J, Gerstenberger D, Xu F. Overview of 3GPP LTE-advanced carrier aggregation for 4G wireless communications. IEEE Communications Magazine. 2012 February;50(2):122–130.
- [16] 3GPP. Cellular system support for ultra low complexity and low throughput internet of things. 2015;.
- [17] Alani, M M. Guide to OSI and TCP/IP Models. 2014;p. 5–16.
- [18] Dong J. Network Protocols Dictionary: From A to Z and 0 to 9.; 2007.
- [19] Levä T, Mazhelis O, Suomi H. Comparing the cost-efficiency of CoAP and {HTTP} in Web of Things applications. Decision Support Systems. 2014;63:23 – 38. 1. Business Applications of Web of Things 2. Social Media Use in Decision Making. Available from: <http://www.sciencedirect.com/science/article/pii/S0167923613002376>.
- [20] Hui SY, Yeung KH. Challenges in the migration to 4G mobile systems. IEEE Communications Magazine Volume: 41, Issue: 12. 2003 Dec;.
- [21] Li R, Shariat M, Nekovee M. Transport protocols behaviour study in evolving mobile networks. 2016 Sept;p. 456–460.
- [22] Nunamaker JF, Chen M. Systems development in information systems research. 1990 Jan;p. 631–640.
- [23] Foundation FS. lwIP Wiki - Protocols; 2017. Available from: <http://lwip.wikia.com/wiki/Protocols>.
- [24] KG RSGC. Narrowband Internet of Things Whitepaper. 2016 Aug;.

Glossary

2G

Second generation mobile communication. iii, 3

3G

Third generation mobile communication. iii, 3

3GPP

3rd Generation Partnership Project (3GPP) is a collaboration between different organizations to standardize and define technologies. i, 1–4, 17, 26

4G

Fourth generation mobile communication. iii, 3, 27

ALP

Application Layer Protocol. i, 1–3, 6, 7, 10, 11, 16, 17, 19, 26–29

AMPS

Advanced Mobile Phone Service is the first generation mobile network in America. 3

API

Application programming interface specifies in what way the software components can interact with each other. 16

CDMA

Code Division Multiple Access is a method where several transmitters can send information at the same time over one channel. 8

CLI

Command-Line User Interface, the user or client can communicate with the computer program using text. 18, 20, 21

CoAP

Constrained Application Protocol is a protocol for web transfers with constrained devices. i, iii, iv, 1, 2, 6–11, 13, 14, 17–21, 23–29, 41

FPGA

Field-Programmable Gate Array is an integrated circuit where the user can program it in the field after it being manufactured. 15

GPRS

General Packet Radio Service, is a service where the device can send data to other devices with a data speed of 30 and 100 kbit/s and be connected to internet.. 8

GSM

Global System for Mobile communication. 29

HTTP

Hyper-Text Transfer Protocol. i, iii, iv, 1, 2, 6–8, 10, 11, 13, 14, 17–29, 41

I²C

Inter-Integrated Circuits is a multi-master bus which means that several chips can be connected to one bus and each chip can act as a master. 15, 18

IEEE

Institute of Electrical and Electronics Engineers. 8

IETF

Internet Engineering Task Force. 7, 8

IoT

Internet of Things (IoT) is the generic term for everyday objects embedded with sensors, connection and computers. i, 1–4, 9, 11, 15, 16, 23, 26–29, 32

ISP

Internet Service Provider. 9

LAN
Local Area Network. 9

LPWAN
Low-Power Wide-Area Network is a network designed for M2M and IoT devices. i, 1, 2, 26, 28

LTE
The Long Term Evolution is a mobile communication standard between 3G and 4G. 1, 3, 4, 9

LwIP
Light-weight IP is a light version of the TCP/IP protocol for embedded systems. 16, 18, 20

M2M
Machine to Machine is a definition of machines and devices that interact and communicate without the involvement of humans. 26, 28, 32

MNO
Mobile Network Operator. 1, 4, 28

MQTT
Message Queue Telemetry Transport is a lightweight messaging protocol on top of TCP/IP protocol. 9, 27

NB-IoT
Narrowband IoT (NB-IoT) is a new cellular standard for providing wide area coverage for IoT. i, 1–4, 8, 11, 15, 17, 25–29

NMT
Nordic Mobile Telephone is the first generation mobile network in Europe and part of Russia. 3

OSI
The Open Systems Interconnection model is a standardized concept of digital communication that partitions the communication system into abstraction layers. 3–6

PSTN
Public Switched Telephone Network is the public network with telephone lines and fiber optic cables. 3

QoS
Quality of Service, Internet communication with pre-agreed quality. 3, 8, 9

RTOS
Real-Time Operating System, an operating system designed to allow accurately timed scheduling.. 11, 15, 16, 20

RTT
Round Trip Time, the time it takes for a data package or signal to be sent from one computer to another and back. 9

SMS
Short message service is a service to send short messages between mobile phones.. 28

TCP
Transmission Control Protocol is a data transmission protocol used for almost all communication over the internet. iii, 5, 6, 9, 16, 17, 22, 24, 32

TLP
Transport Layer Protocol. 5, 7, 11, 16, 17, 19

UDP
User Datagram Protocol. iii, 5–9, 16, 19, 23, 24

UE
User Equipment is any type of device connected with mobile communication. 4

URI

Uniform Resource Identifier is a string of characters used to identify or name a resource. 6, 22

WAN

Wide Area Network. 1

WLAN

Wireless Local Area Network, in other words Wifi. 8

WoT

Web of Things is a concept where in the future, everyday objects are integrated with the web. 8

A Test cases

This appendix describes the different test cases conducted on the applications. The test cases consist of ID, a title on the test, a description on what the test is going to show, start condition on the test object, a list of events that will occur during the test and at last the expected results.

A.1 ID 1

ID	<i>ID 1</i>
Title	<i>Connect one sensor</i>
Description	<i>The sensor shall connect without errors</i>
Start conditions	<i>Application running</i>
Course of event	<i>1. The sensor initialize 2. Application (master) sends a signal to sensor 3. Sensor (slave) sends either "OK" or "Error" back to master.</i>
Expected result	<i>The sensor sends OK back to application as a clearance it have contact.</i>

Figure 9: Test case "Connect one sensor"

A.2 ID 2

ID	<i>ID 2</i>
Title	<i>Connect two sensors</i>
Description	<i>The sensors shall connect without errors</i>
Start conditions	<i>Application running</i>
Course of event	<i>1. The sensors initialize 2. Application (master) sends a signal to all the sensors 3. The sensors (slave) sends either "OK" or "Error" back to master.</i>
Expected result	<i>The sensors sends OK back to application as a clearance they have contact.</i>

Figure 10: Test case "Connect two sensors"

A.3 ID 3

ID	<i>ID 3</i>
Title	<i>Connect three sensors</i>
Description	<i>The sensors shall connect without errors</i>
Start conditions	<i>Application running</i>
Course of event	<i>1. The sensors initialize 2. Application (master) sends a signal to all the sensors 3. The sensors (slave) sends either "OK" or "Error" back to master.</i>
Expected result	<i>The sensors sends OK back to application as a clearance they have contact.</i>

Figure 11: Test case "Connect three sensors"

A.4 ID 4

ID	<i>ID 4</i>
Title	<i>Connect four sensors</i>
Description	<i>The sensors shall connect without errors</i>
Start conditions	<i>Application running, Terminal running</i>
Course of event	<i>1. The sensors initialize 2. Application (master) sends a signal to all the sensors 3. The sensors (slave) sends either "OK" or "Error" back to master.</i>
Expected result	<i>The sensors sends OK back to application as a clearance they have contact.</i>

Figure 12: Test case "Connect four sensors"

A.5 ID 5

ID	<i>ID 5</i>
Title	<i>Get temperature value.</i>
Description	<i>The temperature sensor post temperature value</i>
Start conditions	<i>Application running, temperature sensor connected, Terminal running</i>
Course of event	<i>1. User sends CLI command in terminal 2. Application posts the temperature value to the Terminal 3. Application posts the temperature value to the cloud application</i>
Expected result	<i>The application posts the same value in the Terminal as in the cloud application.</i>

Figure 13: Test case "Get temperature value"

A.6 ID 6

ID	<i>ID 6</i>
Title	<i>Get humidity value.</i>
Description	<i>The humidity sensor post humidity value</i>
Start conditions	<i>Application running, humidity sensor connected, Terminal running</i>
Course of event	<i>1. User sends CLI command in terminal 2. Application posts the humidity value to the Terminal 3. Application posts the humidity value to the cloud application</i>
Expected result	<i>The application posts the same value in the Terminal as in the cloud application.</i>

Figure 14: Test case "Get humidity value"

A.7 ID 7

ID	<i>ID 7</i>
Title	<i>Get lux value.</i>
Description	<i>The light sensor post lux value</i>
Start conditions	<i>Application running, light sensor connected, Terminal running</i>
Course of event	<i>1. User sends CLI command in terminal 2. Application posts the lux value to the Terminal 3. Application posts the lux value to the cloud application</i>
Expected result	<i>The application posts the same value in the Terminal as in the cloud application.</i>

Figure 15: Test case "Get lux value"

A.8 ID 8

ID	<i>ID 8</i>
Title	<i>Get barometrical value.</i>
Description	<i>The barometrical sensor post pressure value</i>
Start conditions	<i>Application running, barometrical sensor connected, Terminal running</i>
Course of event	<i>1. User sends CLI command in terminal 2. Application posts the barometrical value to the Terminal 3. Application posts the barometrical value to the cloud application</i>
Expected result	<i>The application posts the same value in the Terminal as in the cloud application.</i>

Figure 16: Test case "Get barometric value"

A.9 ID 9


ID	ID 9
Title	<i>Get all sensors value.</i>
Description	<i>All sensors post values on one command.</i>
Start conditions	<i>Application running, All sensors connected, Terminal running</i>
Course of event	<i>1. User sends CLI command in terminal</i> <i>2. Application posts all sensor values to the Terminal</i> <i>3. Application posts all sensor values to the cloud application</i>
Expected result	<i>The application posts the same values in the Terminal as in the cloud application.</i> 

Figure 17: Test case "Get all sensors value"

A.10 ID 10

ID	ID 10
Title	<i>The application post data via HTTP</i>
Description	<i>The application shall post the values to a website via HTTP</i>
Start conditions	<i>Application running, All sensors connected, Terminal running, connected to internet via HTTP</i>
Course of event	<i>1. User sends CLI command in terminal</i> <i>2. All sensor responds with values</i> <i>3. The application post the values to a website via HTTP</i>
Expected result	<i>The values posted by the application shall be visible on a website.</i>

Figure 18: Test case "The application post data via HTTP"

A.11 ID 11

ID	<i>ID 11</i>
Title	<i>The application post data continuously via HTTP in an interval of 10 sec</i>
Description	<i>The application shall post the values to a website via HTTP continuously in 10 seconds intervals.</i>
Start conditions	<i>Application running, All sensors connected, Terminal running, connected to internet via HTTP</i>
Course of event	<i>1. User sends CLI command in terminal 2. All sensor responds with values 3. The application post the values to a website via HTTP continuously</i>
Expected result	<i>The values posted by the application shall be visible on a website.</i>

Figure 19: Test case "The application post data continuously via HTTP in an interval of 10 sec"

A.12 ID 12

ID	<i>ID 12</i>
Title	<i>The application post data via CoAP</i>
Description	<i>The application shall post the values to a website via CoAP</i>
Start conditions	<i>Application running, All sensors connected, Terminal running, connected to internet via CoAP</i>
Course of event	<i>1. User sends CLI command in terminal 2. All sensor responds with values 3. The application post the values to a website via CoAP</i>
Expected result	<i>The values posted by the application shall be visible on a website.</i>

Figure 20: Test case "The application post data via CoAP"

A.13 ID 13

ID	<i>ID 13</i>
Title	<i>The application post data continuously via CoAP in an interval of 10 sec</i>
Description	<i>The application shall post the values to a website via CoAP continuously in 10 seconds intervals.</i>
Start conditions	<i>Application running, All sensors connected, Terminal running, connected to internet via CoAP</i>
Course of event	<i>1. User sends CLI command in terminal 2. All sensor responds with values 3. The application post the values to a website via CoAP continuously</i>
Expected result	<i>The values posted by the application shall be visible on a website.</i>

Figure 21: Test case "The application post data continuously via CoAP in an interval of 10 sec"

A.14 ID 14

ID	<i>ID 14</i>
Title	<i>The application shall not crash without the ethernet cable</i>
Description	<i>The application shall not crash with the ethernet cable unplugged.</i>
Start conditions	<i>Application running, All sensors connected, Terminal running, connected to internet via HTTP</i>
Course of event	<i>1. The ethernet cable is unplugged 2. User sends CLI command in terminal 3. Error messages in terminal 4. The ethernet cable connects into the application 5. User sends CLI command in terminal 6. The application post the values to a website</i>
Expected result	<i>The values shall be visible on a website.</i>

Figure 22: Test case "The application shall not crash without the Ethernet cable"

B Measuring the Packet Build Time

Screenshots of the measurement of packet build time for both HTTP and CoAP are presented in this appendix.

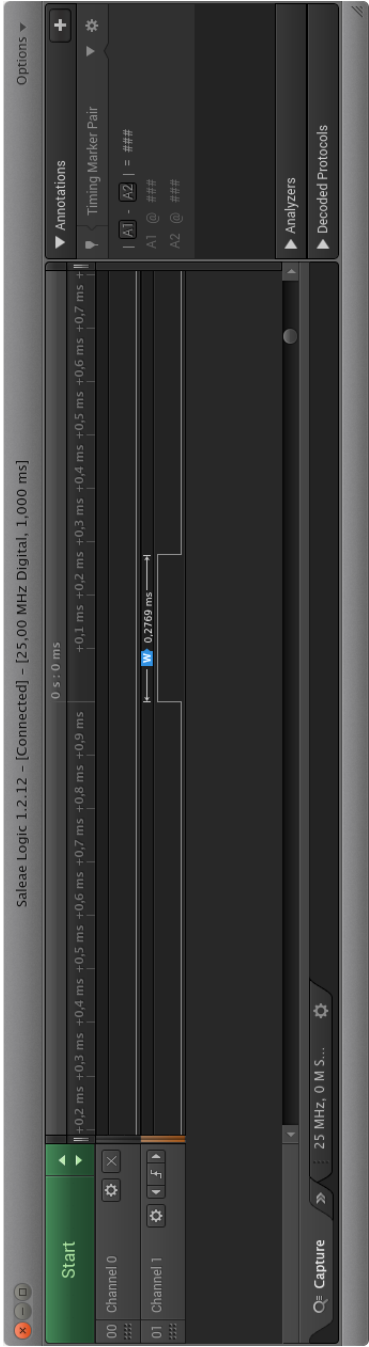


Figure 23: HTTP measurement

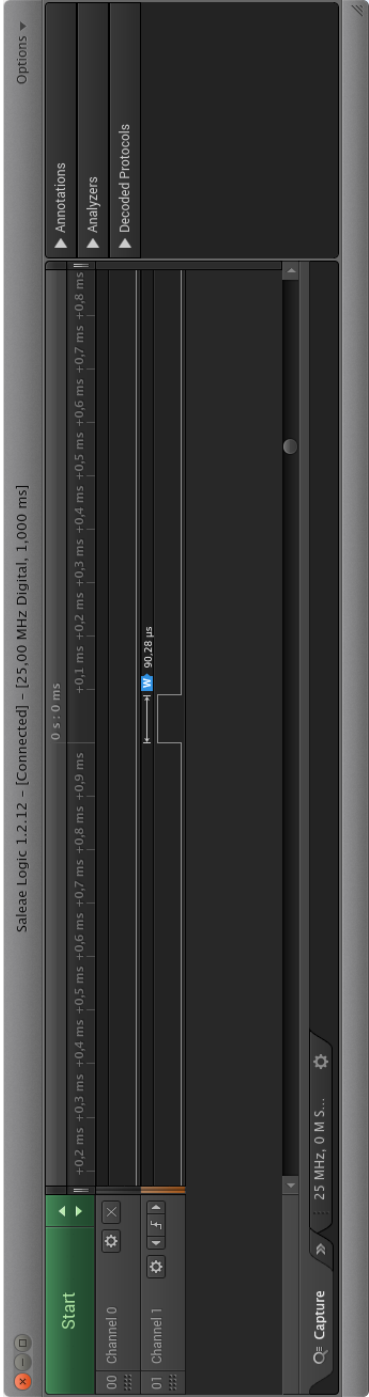


Figure 24: CoAP measurment

C Search criteria

The referenced articles and books presented in this thesis are based on the keywords presented below. The databases that the authors searched in were among others IEEE, Science Direct, Google Scholar and Springer.

- Mobile communication
- Mobile communication and transition
- Mobile communication and transition and adaption
- Nb-IoT
- 3G Mobile Communication and 4G Mobile communication and adaption
- GSM AND transition
- Mobile communication and LTE and internet of things
- Mobile communication and Nb-IoT
- Software Engineering and Research methodology
- Mobile communication and transition and 4G
- Mobile communication and migration and 4G and GSM
- Research and development and computer science
- Resource constrained and software design
- HTTP
- CoAP
- HTTP and CoAP
- HTTP and CoAP and comparison
- Application layer protocol
- Wireless communication
- Transport protocol

D System figure

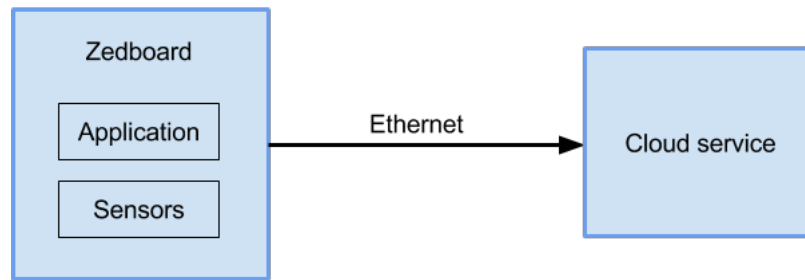


Figure 25: End to end system figure