NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Carl Richard Steen Fosse

Power Consumption modeling of TCP and UDP over low power cellular networks for a constrained device

Master's thesis in Electronic System Design and Innovation
Supervisor: Snorre Aunet, Sigve Tjora
June 2020

Carl Richard Steen Fosse

Power Consumption modeling of TCP and UDP over low power cellular networks for a constrained device

Master's thesis in Electronic System Design and Innovation
Supervisor: Snorre Aunet, Sigve Tjora
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Abstract

The expanding number of constrained Internet of Things devices is challenging researches and industry to improve and develop new communication technology, protocols and devices. The emergence of new cellular LPWAN technologies like NB-IoT and LTE-M as precursors for 5G has garnered lots of attention with promises of decade-long battery life and cheap devices. Concurrently new and improved communication protocols are developed, though most are still based on the well known UDP and TCP standards. There is a lack of research on how well these protocols will perform on the new cellular technologies, which this thesis aims to assess.

We have devised a model that can be used for predicting the total power consumption of a cellular IoT device that uses the UDP based CoAP protocol or the TCP based MQTT protocol over LTE-M or NB-IoT. A developer can use the model without extensive knowledge about cellular or protocol behavior to estimate the energy budget of their application.

We based the model on the results from linear regression analysis on measurements of energy spent on transmissions with increasing payload sizes. The experiments found a linear relationship between payload size and energy for NB-IoT, while for LTE-M no clear relationship was found. However, it was also discovered that LTE-M is more energy efficient when the message payload exceeds a certain size, given that the modem is released early from the network after transmission.

An nRF9160 Development Kit from Nordic Semiconductor was used to gather data for the experiments and to test the model. Estimations show that 10-year battery life is achievable for an application using either CoAP or MQTT over NB-IoT with transmissions of up to 1280 bytes every 2 hours, under the assumption of 4μA sleep current. We estimated that the same battery life is achievable over LTE-M when transmitting the same amount of data every 4 hours. CoAP performed overall better than MQTT from an energy consumption standpoint.

Sammendrag

Det økende antall ressursbegrensede Internet of Things-enheter utfordrer forskere og industri til å forbedre og utvikle ny kommunikasjonsteknologi, protokoller og enheter. Fremveksten av nye mobilnett LPWAN-teknologier som NB-IoT og LTE-M i rollen som forløpere for 5G har fått mye oppmerksomhet med løfter om ti års lang batterilevetid og billige enheter. Samtidig utvikles nye og forbedrede kommunikasjonsprotokoller, selv om de fleste fortsatt er basert på de velkjente UDP- og TCP-standardene. Det mangler forskning på hvor godt disse protokollene vil fungere for de nye mobilnettteknologiene, og denne oppgaven tar sikte på å vurdere dette.

Vi har utviklet en modell som kan brukes til å forutsi det totale strømforbruket til en mobil IoT-enhet som bruker den UDP-baserte CoAP-protokollen eller den TCP-baserte MQTT-protokollen over LTE-M eller NB-IoT. En utvikler kan bruke modellen uten omfattende kunnskap om mobil- eller protokollatferd for å estimere energibudsjettet for applikasjonen deres.

Vi baserte modellen på resultatene fra lineær regresjonsanalyse på målinger av energi brukt på overføringer med økende størrelse. Eksperimentene fant en lineær sammenheng mellom pakkestørrelse og energi for NB-IoT, mens det for LTE-M ikke ble funnet noen klar sammenheng. Imidlertid ble det også oppdaget at LTE-M er mer energieffektiv når meldingsstørrelsen overstiger en viss størrelse, gitt at modemet frigjøres tidlig fra nettverket etter overføring.

Et nRF9160 utviklingssett fra Nordic Semiconductor ble brukt til å samle data for eksperimentene og for å teste modellen. Estimater viser at 10-års batterilevetid er oppnåelig for en applikasjon som bruker CoAP eller MQTT over NB-IoT med overføringer på opptil 1280 byte hver 2. time, under antagelse av 4 $s\ mu\ ampere$ sovestrøm. Vi estimerte at den samme batterilevetiden er oppnåelig over LTE-M når du overfører samme datamengde hver fjerde time. CoAP presterte generelt bedre enn MQTT fra et energiforbrukssynspunkt.

Preface

This Master's thesis is my final required submission before finishing the 5-year MSc program Electronic System Design and Innovation (elsys) at The Department of Electronic Systems (IES), Norwegian University of Science and Technology (NTNU). The project is an engagement from the company Disruptive Technologies AS on the feasibility of using low power cellular technology in sensor solutions. Preliminary research was conducted during the autumn of 2019, surveying cellular standards and protocols for use in Internet of Things solutions. The master project continued this work through the spring of 2020 by modeling the power consumption of selected communication protocols over cellular networks. Snorre Aunet of the IES has supervised the project together with Sigve Tjora from Disruptive technologies. I, Carl Richard Steen Fosse conducted the research.

The spring of 2020 brought with it some surprises. And while home office and social distancing may be the ideal conditions for working hard, it tested both the will-power and motivation of the author. In the end I did prevail, resulting in this thesis, but that was not only to my own merit.

I want to express my gratitude towards my supervisors, Sigve and Snorre, for closely following my progress suggesting alterative approaches when I was stuck and providing good feedback to my work and research.

I would like to thank the Pål Sturla Sæther and NTNU Internet of Things lab for borrowing out the OTII-ARC, a measurement instrument that revealed itself to be crucial for finishing the research.

Long days working alone eventually led to the formation of a "digital study room" with some of my fellow master candidates. This has been of much help and motivation, so I would like to thank Erik, Embla, Kaja and all the others[1] for sticking together and helping each other towards the finishing line.

Lastly I would like to express my deepest gratitude towards Silje Aagaard. She has been there for me throughout this whole ordeal, even though her own work as a teacher has been challenging.

[1]In general I am very grateful for my study programme, elsys, and what it stands for.

Contents

Abbreviations

3GPP	Third Generation Partnership Project.
CoAP	Constrained Application Protocol.
CSV	Comma Separated Variable.
DL	downlink.
DRX	Discontious Receptions.
DTLS	Datagram Transport Layer Security.
eDRX	Extended Discontious Receptions.
FoTA	Firmware over The Air.
GPIO	General purpose input output.
HTTP	Hyper Text Transport Protocol.
IETF	Internet Engineering Task Force.
IoT	Internet of Things.
IP	Internet Protocol.
LPWAN	Low Power Wide Area Network.
LTE	Long Term Evolution.
MCU	Microcontroller unit.
MQTT	Message Queue Telemetry Transport.
MSS	Maximum Segment Size.
MTU	maximum transmission unit.

NB-IoT	Narrowband-Internet of Things.
NCS	nRF Connect sofware development kit.
PDCCH	Physical Downlink Control Channel.
PSM	Power Saving Mode.
pTAU	Periodic Tracking Area Update.
QoS	Quality of Service.
REST	Representational state transfer.
RRC	Radio Resource Control.
RTOS	Real Time Operating System.
SDK	software development kit.
SiP	System in Package.
TAU	Tracking Area Update.
TCP	Transmission Control Protocol.
TLS	Transport Layer Security.
UART	Universal Asynchronous Receiver/Transmitter.
UDP	User Datagram Protocol.
UE	User Equipment.
UL	uplink.
URI	Unique Resource Identifier.

List of Tables

List of Figures

Chapter 1

Introduction

The vast technological advancements within the electronics industry have opened new fields for the Internet of Things. With devices smaller and more efficient, it is now possible to monitor figuratively everything, relieving their human counterparts of tedious data collection tasks. These devices are connected to the internet using a wide array of protocols and standards ranging from well-known implementations like HTTP to highly optimized proprietary solutions.

The company Disruptive Technologies AS (DT) has developed a very power efficient sensor/-gateway solution that enables a single sensor to have a battery life of up to 15 years, making the solution ideal for long term monitoring of buildings, factories and inaccessible areas. However, the system is inefficient in low-density scenarios where only a small number of sensors are present as the gateway should be connected to at least five sensors to remain sustainable. Possible solutions to this challenge may use a cheap device that achieves a wide area network (WAN) connectivity with low power consumption. The novel NB-IoT[1] and LTE-M[2] cellular standards introduced in 2016 targets this kind of Internet of Things (IoT) device operation.

1.1 Motivation

The increasing constraints on devices do not only pose new requirements for the physical layer connectivity, it also demands more from the high-level communication protocols requiring smaller overhead and new methods for ensuring reliable transmissions. This have resulted in the emergence of protocols like the CoAP[3] using the User Datagram Protocol (UDP)[4], though we still see more demanding protocols like MQTT[5] built on the Transmission Control Protocol (TCP)[6] still being in wide usage [7]. This thesis aims to model the power consumption of LTE-M and NB-IoT in relation to the used communication protocol. Such a model will simplify the development of low power cellular applications and in turn shorten the time to market. Such a model can also be used to assess the feasibility of using a cellular device in DT's solution.

1.2 Goals

We set out to answer the following questions with the research done in this thesis:

1. How do TCP and UDP perform on cellular networks?
2. Is there a relationship between the payload of a message and the consumed time and energy for transmission over NB-IoT and LTE-M?
3. Can the total energy consumption when using either TCP or UDP over cellular networks be correctly modeled assuming the relationship above?

1.3 Structure

The thesis is structured as follows: In chapter 2, we will present relevant background theory and related work. The NB-IoT and LTE-M standards will be thoroughly covered along with similar Low Power Wide Area Network (LPWAN) standards. Additionally, we will describe the power saving mechanisms PSM and eDRX along with the communication standards MQTT (TCP) and CoAP (UDP). The implementation of test applications and data processing are documented in chapter 3. In this chapter, we will also cover the equipment and software that were used. The experiments, together with the results are presented in chapter 4, followed by a discussion of the findings in chapter 5. The thesis is concluded in chapter 6.

Chapter 2

Background

This chapter will provide an overview of the key aspects behind developing a low power cellular sensor device. A set of other LPWAN types will be introduced and compared to their cellular counterparts NB-IoT and LTE-M in order to get a perspective on the competition and advancements within the field. We will then do a more thorough description of the cellular technologies and their respective power-saving mechanisms derived from the author's earlier project [8]. Followed by an explanation of the communication protocols MQTT and CoAPalong with their respective transport layer protocols TCP and UDP. These will also be discussed in the context of usage on cellular networks. We will conclude the chapter by presenting other research on modeling of power consumption on cellular embedded devices and provide a description of this project's approach to the matter.

2.1 Other Low Power Wide Area Networks

The application spectrum of the Internet of Things is growing, with new fields of use within smart city technology, agriculture, asset tracking and other industries. These application requirements for connectivity extends beyond the scope of short-range protocols like Bluetooth and WiFI. On the other end, conventional cellular networks like GSM and 4G provide adequate coverage, but at the expense of high energy use. The lack of any technology targeting constrained devices has opened up a market for a novel type of wide-area networks that targets devices that require low energy consumption and long-range operation, with the accepted cost of lower data rate. These are gathered under the collective name of LPWANs.

The cellular standards, NB-IoT and LTE-M, that are in focus for this report are classified as LPWANs. However, there are other prominent LPWAN alternatives that one could consider. Most prominently are the LoRaWAN and Sigfox technologies, which both are deployed in the unlicensed band as opposed to the cellular standards. Though most LPWANs are recent technologies that all have emerged within the last decade, SigFox(released in 2010) and LoRaWan(released in 2015) have had more time to mature. Research is done in [9] comparing NB-IoT with LoRaWan and Sigfox shows that the two latter ones are targeting heavily constrained devices with slower data rates, optimizations for infrequent transmission and superior battery life. However, NB-IoT has a comparatively superior data rate as well as better support for massive networks.

2.2 Cellular low power wide area network standards

The 3GPP defined the LTE-M and NB-IoT standards in 2016[10] as a part of their commitment to support cellular IoT throughout 4G and eventually 5G. Looking at table 2.1, that contains some key specifications, it is clear that the two protocols target different use cases. The details will be discussed in the following paragraphs.

Table 2.1: Some specifications of LTE-M and NB-IoT[10].

	LTE-M	**NB-IoT**
Deployment	In-Band LTE	In-band & Guard-Band LTE, standalone
Coverage	155.6 dB	164 dB
Bandwidth	1.08 MHz	180 kHz
Peak data rate	1 Mbps	~50 kbps
Latency	50-100 ms	1.5-10s
Target battery life*	~10 years	>10 years
Maximum single packet size**	8188 octets	1600 octets

* Using a 5 watt hour battery (depending on traffic and coverage needs)

** Limited by the Protocol Data Unit of the Packet Data Convergence Protocol defined in 3GPP TS36.323 [11, p.-12]. Larger sizes leads to fragmentation between the layers.

2.2.1 LTE-M

LTE-M extends the LTE standard with the goal of lowering cost and power consumption while maintaining relatively high performance with regards to speed and latency. We can see this in table 2.1 comparing to NB-IoT, as LTE-M excels in both bandwidth and data rate at the expense of battery life and coverage. Furthermore, the standard supports both mobile (i.e. moving) User Equipments (UEs) and voice transmission, making it a good candidate for asset tracking or emergency equipment. The data rate is variable as well, so it can be adjusted according to the application.

Another perk of LTE-M is that a cellular network operator can effortlessly deploy the standard to existing Long Term Evolution (LTE) stations through software updates. This reduces both the complexity and deployment costs of devices. In general, LTE-M is well suited for IoT applications. However, the higher energy usage may invalidate the protocol for use in very constrained devices where power consumption, not latency, is the restriction.

2.2.2 NB-IoT

The NB-IoT standard targets the lower end of the market and is specially designed to support devices that transmit small amounts of data, at long intervals. As stated in table 2.1, it has a bandwidth of only 180 kHz, thereof the "narrowband" name. This results in cheaper units as a 180 kHz frontend requires less sophisticated hardware compared to LTE-M [12]. Besides, the modulation scheme of NB-IoT is restricted so that only one antenna is used for both uplink and downlink transmission, further lowering the device cost[13]. However, the deployment cost of NB-IoT for cellular network operators will be higher as new hardware must be installed on every base station. This cost does not seem to be a big concern as the standard is being adopted by several operators around the world, according to the Global mobile Suppliers Association[14].

The narrow bandwidth enables the NB-IoT to exist in the three different ways listed in the specifications; inside the LTE band, in the guard bands of LTE and on independently licensed bands. This enables the network to support over 50.000 devices per cell, allowing massive sensor networks, like smart cities, to be established [10]. This kind of application is further encouraged as NB-IoT lacks good mobility support as opposed to LTE-M. Hence, devices should be stationary. The higher coverage of NB-IoT also allows for deeper signal penetration, enabling devices to be situated within buildings and underground. This is investigated by Lauridsen, Kovacs, Mogensen, *et al.*[15]. The research discovered that NB-IoT devices achieve a 95% coverage of "deep indoor" users, though at the cost of 2-6 times higher power consumption and support for a tenfold fewer devices per base station. Comparably LTE-M only achieved 80% coverage.

It is clear that NB-IoT is less versatile than LTE-M, but the optimizations and features implemented make it very attractive for certain, specialized applications that require very low power consumption.

2.2.3 Power saving mechanisms for NB-IoT and LTE-M

Both of the aforementioned LPWAN technologies are developed to be used with low power devices, but as described in the previous paragraphs, they target different applications. This aside LTE-M and NB-IoT supports the two same mechanisms for saving power; eDRX and PSM. Which are both part of the minimum specifications for LTE-M and NB-IoT networks[1], [2]. A more thorough description of these follows in the next subsections.

2.2.3.1 Power Saving Mode (PSM)

PSM enables a device to stay attached to the network even after it enters deep sleep, where most of it is circuitry, including the radio receiver, is turned off. The mechanism also saves the device the energy cost of reattaching to the network when waking up, but it will be unreachable during the hibernation. PSM is initiated by the device by suggesting two values to the network: the period of which it will be notifying the network that its still registered, and how long it will stay awake after said notification. The two timers are defined as T_{3412} and T_{3324} respectively. The network will respond with the accepted timer values. Note that these may differ from the requested ones. Figure 2.1 show the interactions between the timers.

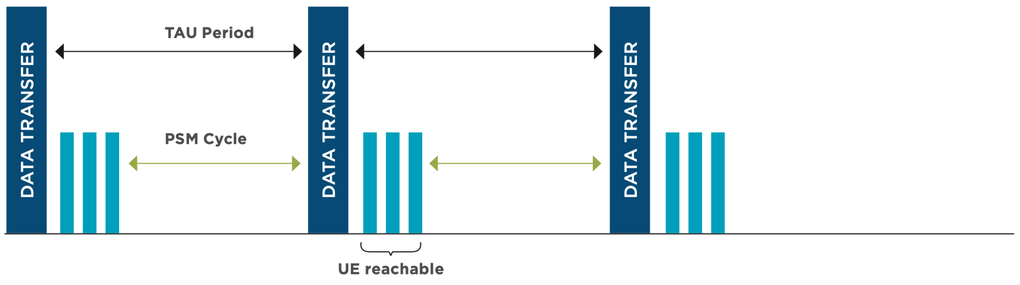


Figure 2.1: Diagram of PSM timing. TAU(Tracking Area Update) period is the time between the device notifying the network. [2].

The encoding of T_{3412} is shown in table 2.2 and the encoding of T_{3324} in table 2.3. Both encoding schemes reserves bit 8 to 6 for value increment and bit 5 to 1 for binary-coded timer value. The maximum TAU period is 413 days, while the reachable time should be a minimum of 16 seconds, with a recommended ratio of 90% between the two [1], [2].

Table 2.2: T_{3412} encoding

bit 8 7 6	meaning
0 0 0	value is incremented in multiples of 10 minutes
0 0 1	value is incremented in multiples of 1 hour
0 1 0	value is incremented in multiples of 10 hours
0 1 1	value is incremented in multiples of 2 seconds
1 0 0	value is incremented in multiples of 30 seconds
1 0 1	value is incremented in multiples of 1 minute
1 1 0	value is incremented in multiples of 320 hours
1 1 1	deactivated

Table 2.3: T_{3324} encoding

bit 8 7 6	meaning
0 0 0	value is incremented in multiples of 2 seconds
0 0 1	value is incremented in multiples of 1 minute
0 1 0	value is incremented in multiples of decihours
1 1 1	deactivated

2.2.3.2 Extended Discontinuous Reception (eDRX)

Discontinuous Reception (DRX) is an already existing LTE power saving mechanism that turns of the receiving part of a device modem for brief periods of time, saving power while it is off. The periods of receiving are called *paging*. DRX is already in extensive use for mobile devices like cellphones and has been expanded in the LTE-M and NB-IoT specifications to extend the time where the modem is not receiving significantly. Hence, the "extended" addition to DRX. Applications using eDRX must tolerate downlink delay, but this is usually acceptable for IoT devices [2].

As with PSM, the device and the network negotiate the period of time in which the receiver will be off. The resulting timing behavior is shown in figure 2.2.

eDRX does not have the same power-saving capabilities as PSM. However, it is more versatile when it comes to the network availability as the device stays awake with only short durations of unreachability as opposed to the long sleep period of PSM.

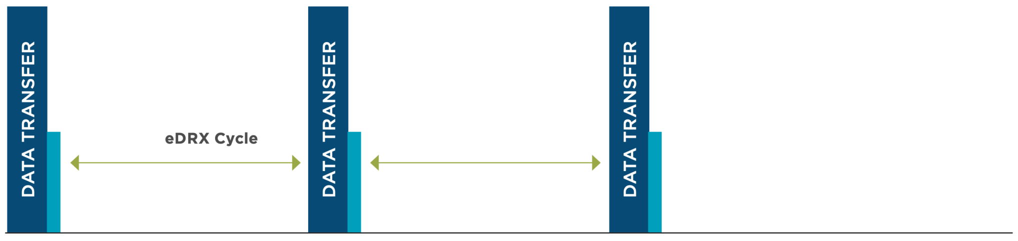
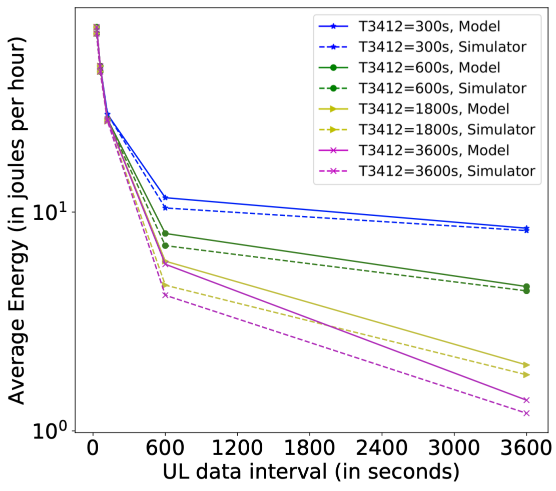


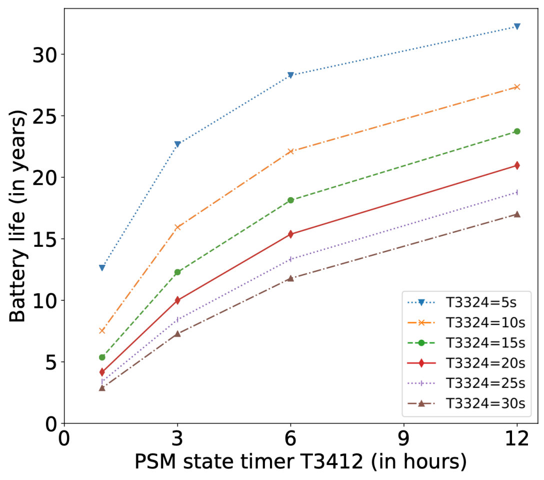
Figure 2.2: Diagram of eDRX timing.[2].

2.2.3.3 Power saving performance of eDRX and PSM

How the power consumption of a device using eDRX and/or PSM will be affected depends on the configuration of the two timers. This has been researched by Sultania, Zand, Blondia, *et al.*[16] for a NB-IoT UE. A mathematical model for power consumption during possible device connection cycles was derived and compared to simulated behavior. Figure 2.3 displays a selection of the results. As expected, a longer T_{3412}(e.g sleep time) value yields better power-saving performance, notice as well in figure 2.3b that estimations show battery life well over 10 years, especially given short idle time (T_{3324}). Furthermore, it is worth noting in figure 2.3a that the combination of a short PSM timer (T_{3412}) yields insignificant power consumption improvements if the device uploads data rarely.



(a) Energy consumption for different PSM timer, T_{3412}, values with an idle timer T_{3324} of 30s.



(b) Estimated battery life of a 1388 mAh battery for different values of T_{3412} and T_{3324}, with an uplink data interval of 24 hours.

Figure 2.3: Results from research done on PSM and eDRX performance[16].

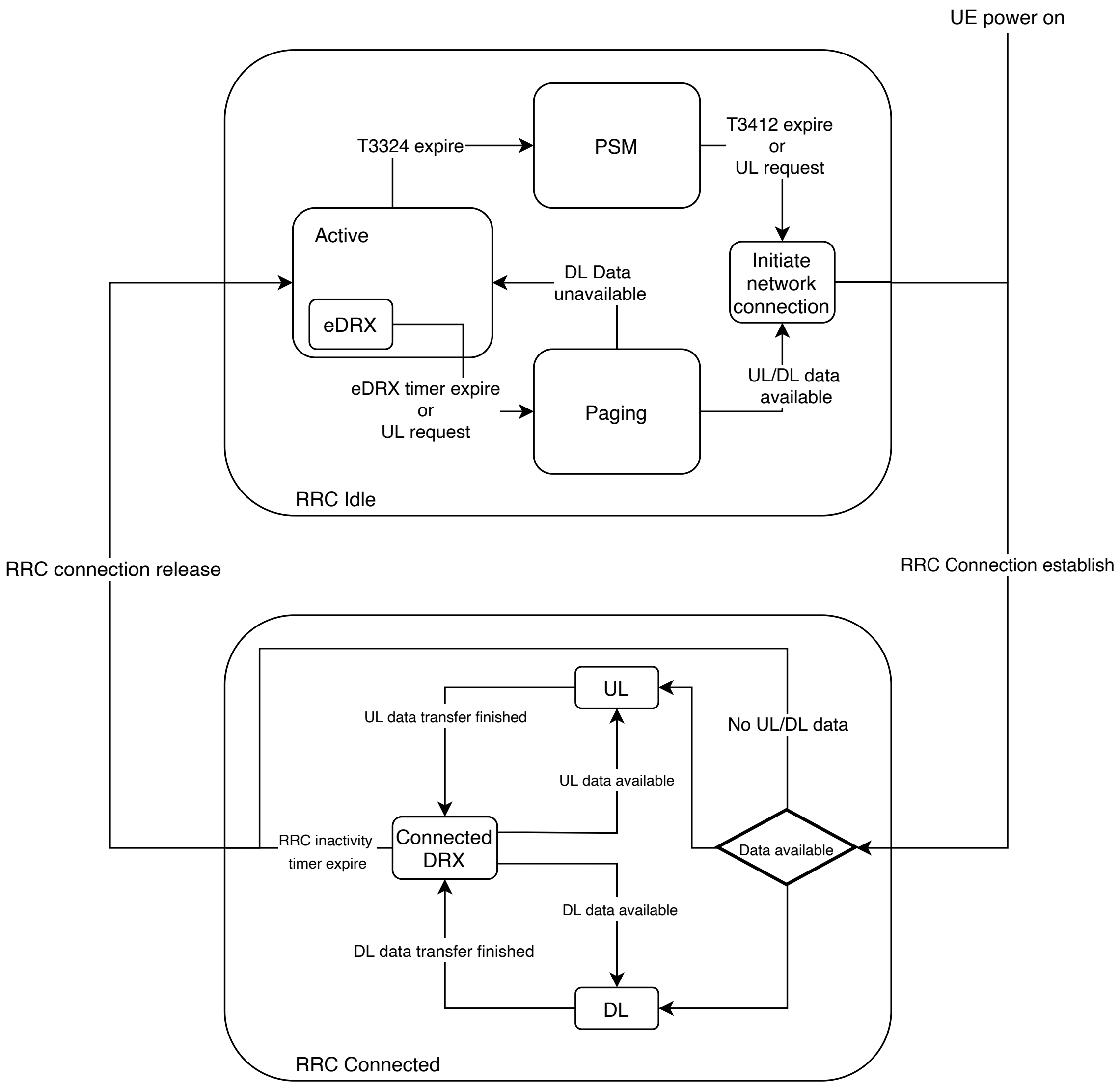
2.3 Radio resource control states

Radio Resource Control (RRC) is the over-the-air interface of the LTE standard [17]. Cellular UE operates in two RRC states *RRC idle* and *RRC connected*. For 5G a third state; *RRC inactive* is introduced as well, but as the 5G network is yet to be extensively deployed it is out of scope for this thesis.

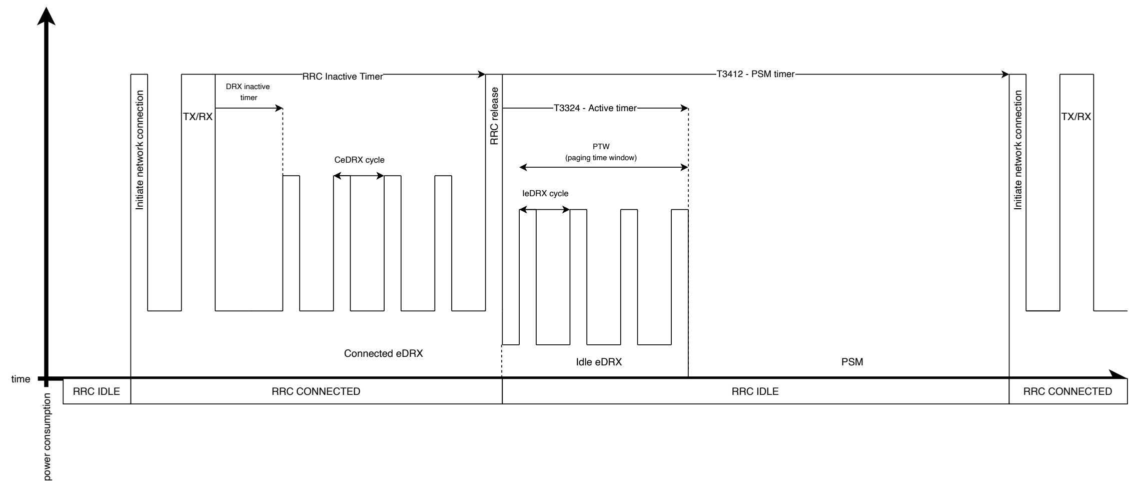
In *idle* mode the modem is inactive and the UE is disconnected from the network. Figure 2.4a displays a transition diagram for the two states, including transitions within the respective states. Figure 2.4b complements the state diagram, showing the typical chain of events and their respective energy levels.

As shown in figure 2.4a the UE can transition from *RRC idle* to *RRC connected* in three manners: if the UE initiates a data upload, if the PSM timer T3412 expires or if downlink data is available during paging. The latter may only occur if the eDRX timer T3324 is larger than zero, as the modem will go straight to PSM sleep otherwise.

When entering *RRC connected* the UE will issue a Tracking Area Update (TAU) to the network. If data is available for either upload or download it will perform the transactions before entering the earlier described connected eDRX mode, staying connected and paging until the *RRC inactive timer* expires. This timer will be restarted if a transaction is initiated during this period. If no data is available after entering the *RRC connected* state, the modem will go straight to *RRC idle* after issuing the TAU. Note that the UE always can initiate a upload, regardless of RRC state. Downlink transactions can only be initiated while the UE is paging.



(a) Flowchart depicting the state transitions of a modem between *RRC idle* and *RRC connected*, with internal transitions included. Derived from the diagram in [16].



(b) Time diagram depicting the different power levels of a modem within the *RRC idle* and *RRC connected* states. Derived from [18].

2.3.1 Connected DRX configuration

The 3GPP deployment guide for LTE-M[2] and NB-IoT[1] recommends eDRX is supported both in idle and connected mode, and that the timers controlling these mechanisms are configurable. The configuration of the idle eDRX is already explained section 2.2.3.2. The duration of connected mode eDRX is restricted to the *RRC inactive timer*, defined on the base station equipment. Two important parameters are mentioned for connected eDRX; the Discontious Receptions (DRX) cycle, describing the duration between paging actions and *onDurationTime*, describing how long the UE will monitor the channel. In between monitoring the UE remains on, but in a low power mode. Possible values the network can use for these timers are shown in table 2.4.

Table 2.4: Possible connected DRX timer values from 3GPP TS 36.331 [17].

Timer name	Timer values
onDurationTimer	psf1, psf2, psf3, psf4, psf5, psf6, psf8, psf10, psf20, psf30, psf40, psf50, psf60, psf80, psf100, psf200
DRX cycle	sf10, sf20, sf32, sf40, sf60, sf64, sf70, sf80, sf128, sf160, sf256, sf320, sf512, sf640, sf1024, sf1280, sf2048, sf2560

These values are given as a number of sub-frames(sf) and Physical Downlink Control Channel (PDCCH) sub-frames(psf). These are a part of the synchronization scheme of LTE, that divides operations into frames and sub-frames. One radio frame is 10 ms long and consists of equally divided sub-frames, each of 1ms length [19]. A PDCCH sub-frame describes the sub-frames while monitoring the network. Based on this, the timer values in table 2.4 can be interpreted in milliseconds.

The RRC Inactivity Timer is referred to as the *dataInactivityTimer* in the 3GPP specifications [20, p.-212]. Note that the same specifications state that the timer is optional, enabling the device to move to idle mode immediately if configured. Table 2.5 show possible values for the RRC inactivity timer. This timer is defined on the network side, making it unavailable for the UE-side configuration. However, release 14 from 3GPP introduces the release assistance indication (RAI) feature for LTE-M and NB-IoT enabling the UE to notify to the network when no more uplink (UL) or downlink (DL) is expected [21, p.-370]. The use of this mechanism among other optimizations for an IoT device using LTE has been researched in [22]. They conclude that the RAI functionality can have a significant effect on power consumption for applications with moderate transmission intervals.

Table 2.5: Possible RRC inactive timer configurations from 3GPP TS 36.331 [17].

Timer name	Timer values
RRC Inactivity timer	s1, s2, s3, s5, s7, s10, s15, s20, s40, s50, s60, s80, s100, s120, s150, s180

2.4 Communication protocols

The topics we have covered so far in this chapter can be mapped to the lower layers of the OSI-model [23]. While it is important to have an understanding of the behavior in these lower layers, most developers will commonly only interact with the application-level communication protocols. The following section will cover two of the most commonly used protocols in IoT; MQTT and CoAP, as well as their respective transport layer protocols; TCP and UDP.

The Transmission Control Protocol is defined in RFC793 from the Internet Engineering Task Force (IETF) with the intent of being a "highly reliable host-to-host protocol" [6]. This reliability comes in the form of ensuring confirmed delivery of packets over a persistent connection between hosts. On the other hand, the User Datagram Protocol strives to minimize the protocol mechanism at the expense of reliability. It was defined in RFC768 in 1980 [4].

2.4.1 TCP

The increased reliability of TCP comes with increased overhead, which in turn can affect power consumption and latency of communications. This could make the protocol unsuitable for the latency-prone operation of constrained IoT devices. Research in [24] has assessed TCP's performance on constrained devices, concluding that even though TCP underperforms UDP trends still suggests that the protocol will be in extensive use for IoT devices in the forseeable future. This is also clear from a survey done by Eclipse in 2018, showing that the TCP based standard MQTT is the most used protocol for IoT applications today [7]. The IETF has provided guidelines for using TCP on constrained devices, suggesting that many of the claims on the unsuitability of the protocol are invalid or solvable [25].

2.4.1.1 MQTT

The Message Queue Telemetry Transport protocol is a an ISO standard currently being maintained by OASIS [5]. It uses a publish/subscribe message pattern allowing individual clients in the network to be decoupled, communicating only with a central server (also known as broker). An example of how this behavior is shown in figure 2.5. The publish/subscribe scheme also enables one-to-many message distribution so that subscribers can read data published by a single client.

The publish/subscribe scheme of MQTT is implemented so that clients can subscribe and publish to defined *topics*. This feature can be useful in many instances, for example, by applications with many sensors providing different kinds of data.

With lightweight headers and a user controllable Quality of Service scheme the MQTT standard targets machine-to-machine and IoT applications. The packet structure of MQTT can be seen in

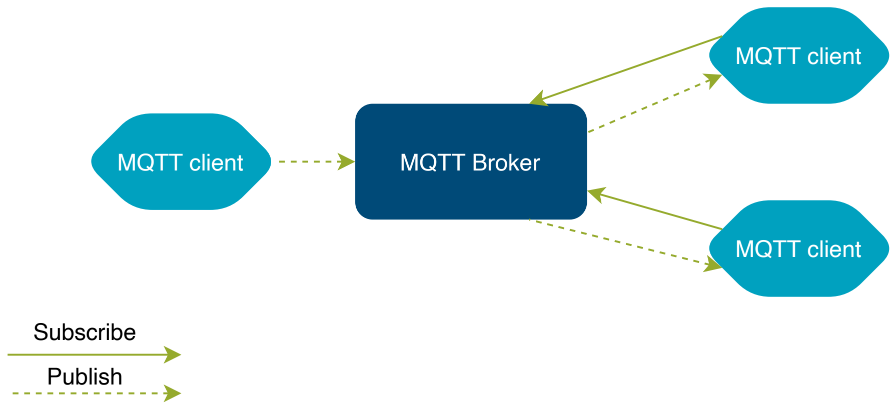


Figure 2.5: MQTT operation example [8].

figure 2.6

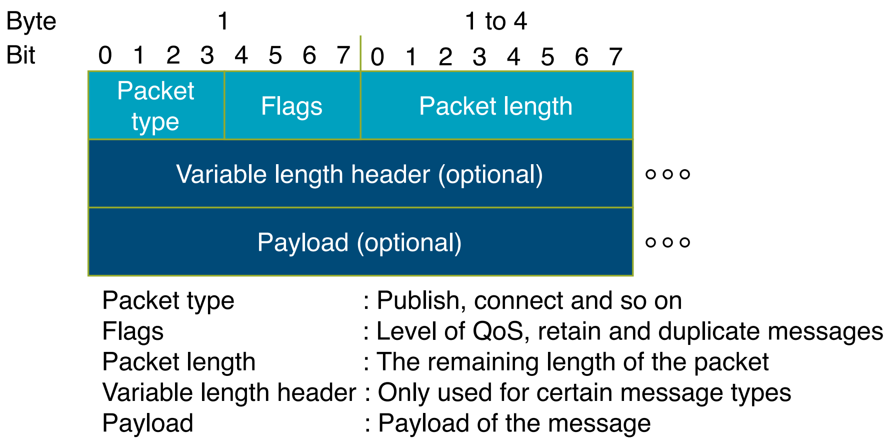

Byte
1
1 to 4
Bit
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
Packet type
Flags
Packet length
Variable length header (optional)
Payload (optional)
Packet type : Publish, connect and so on
Flags : Level of QoS, retain and duplicate messages
Packet length : The remaining length of the packet
Variable length header : Only used for certain message types
Payload : Payload of the message

Figure 2.6: MQTT packet structure.

As we can see from figure 2.6 the minimum packet size is only 2 bytes. Though as we have established TCP requires extensive overhead, which is reflected in the total overhead for MQTT. The headers TCP and Internet Protocol (IP) will add up to the total data actually being transmitted. From the IETF definitions of TCP[6] and IP[26] we can see that both have 20 byte headers, hence the physical layer would have to handle a minimum of 42 bytes for a MQTT transmission. Efforts are made towards lighter operation through the introduction of MQTT Sensor Network (MQTT-SN), which is a UDP based implementation of MQTT [27]. This has not seen much development since its introduction in 2013, but is expected to change with higher demand for constrained protocols in IoT.

Every MQTT packet has a QoS level defined by a flag in the header. The three supported levels are shown in table 2.6.

The QoS mechanism could seem redundant as TCP ensures delivery, but that is only between client and server. The decoupled operation of the publish/subscribe messaging pattern means that the protocol needs to facilitate for guaranteed delivery all the way from publisher to subscriber,

Table 2.6: The QoS levels of MQTT.

QoS 0	at most once (i.e "fire and forget")
QoS 1	at least once
QoS 2	exactly once

something TCP does not account for. For example, would packets lost by a server crash after receiving, but before delivering, only be detected if the packets use the QoS 1 or 2 flags.

2.4.2 User Datagram Protocol

UDP is a *connectionless* protocol, with datagrams being sent from host to receiver with no required setup, or guarantee of delivery or duplicate messages. The simple nature of the protocol makes it ideal for constrained devices though the lack of reliability has required the development of supporting message protocols using UDP as transport. One of these is the Constrained Application Protocol (CoAP).

2.4.2.1 Constrained Application Protocol

CoAP was defined by the IETF in 2014 with the intent of creating a web protocol for constrained environments[3]. Similarly to another well known web protocol Hyper Text Transport Protocol (HTTP), CoAP is based on the the Representational state transfer (REST)ful architecture [28]. Though the CoAP's specification clearly states that the protocols main intention is not to compress HTTP, but rather to enable use of the RESTful architecture for constrained machine-to-machine applications.

Entities participating in a CoAP network are named endpoints, and as opposed to MQTT, they follow a request/response message pattern, where transactions are initiated by a sender *requesting* a method on a *resource* hosted on a recipient. The recipient will, in turn, respond with a response code. This transaction is handled asynchronously by CoAP in contrast to HTTP[3].

UDP based protocols must handle reliability schemes for retransmissions and duplicate delivery on the application protocol level. CoAP does this by introducing a lightweight option QoS scheme along with message IDs. All CoAP packages have a message ID so that duplicates can be detected. In addition, a message can be one of four types: confirmable, non-confirmable, acknowledgment or reset. Confirmable messages will require an acknowledgment from the receiver and will start retransmissions with exponential back-off in the case of a time out. Non-confirmable messages are essentially "fire-and-forget", but duplicates will be detected based on the message ID.

The CoAP packet structure is shown in figure 2.7. The minimum packet size is 4 bytes, being an empty packet without payload or options. Though not as small as the MQTT header t is still very light. The token is used to match a request with a response and is always present in a CoAP message. Requests and responses can also carry options, which are information relating to how an endpoint shall handle the message. Options can, for example, consist of Unique Resource Identifier (URI) information, telling a server on what resource to use a method. Hence, most requests/responses will always carry some options.

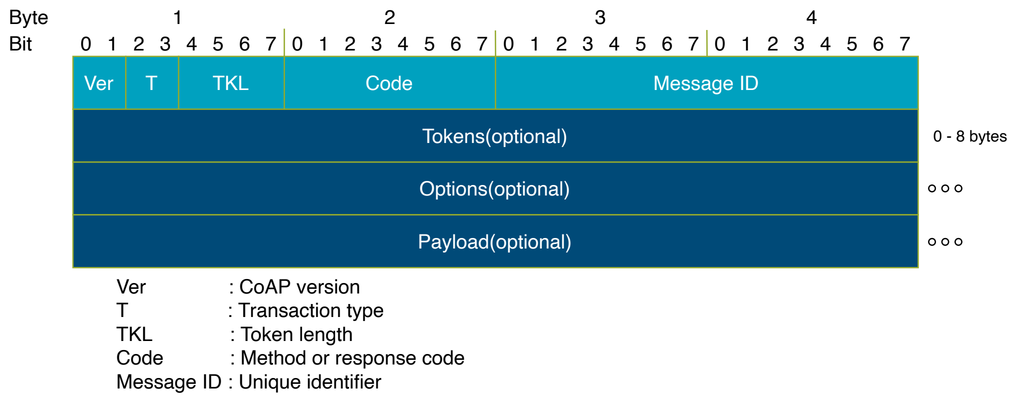


Figure 2.7: CoAP packet structure.

2.4.3 Research on TCP and UDP over cellular connections

The topic of choosing a communication protocol for constrained cellular devices have been assessed in several research papers. CoAP and MQTT over NB-IoT were assessed in [29] which concluded that the UDP based CoAP consistently performed better than the TCP based MQTT over NB-IoT, though for less dense scenarios MQTT still worked well. Similar results were uncovered in [30] when comparing the performance of MQTT and MQTT-SN, the UDP implementation of MQTT. In this scenario, the TCP application had a packet loss of almost 90%, while UDP achieved close to 4% loss. To the author's knowledge, the use of TCP and UDP over LTE-M has not been researched.

2.4.4 Maximum payload calculation

The maximum payload size of a message is under several restrictions. An application using UDP, which boasts a maximum theoretical payload length of 2^{16} bytes, will for example have an actual payload limit far below this, as the other layers of the connection path can be limited by a maximum transmission unit (MTU). Larger messages would either need to be fragmented or dropped. The MTU can be used as a base when calculating the maximum payload size for an application. In the case of CoAP the header plus options and tokens can be expressed as:

$$s_{\text{CoAP overhead}} = s_{\text{header}} + s_{\text{token}} + s_{\text{options}} \tag{2.1}$$

The parameters are as follows:

- s_{token} is the token size of 0 - 8 bytes.
- s_{header} is the header of 4 bytes. The absolute minimum size of a CoAP message.
- s_{options} is the CoAP options, with varying length.

Based on this, we can express the maximum payload of a CoAP message with the following equation.

$$s_{\text{max payload}} = \text{MTU} - (s_{\text{IP}} + s_{\text{UDP}} + s_{\text{overhead}}) \tag{2.2}$$

When connecting to an internet service the MTU commonly follows the connection path, meaning that the link with the smallest MTU decides the MTU for all links in order to avoid fragmentation [31]. To avoid fragmentation CoAP also supports Block-Wise transfers, enabling larger messages to be sent [32].

The issue of payload size is not as pressing for TCP as the protocol supports a sophisticated segmentation scheme, allowing larger payloads to be split into several smaller TCP packets based on the option Maximum Segment Size (MSS). The MSS commonly restricted by the MTU of the IP layer to avoid IP fragmentation. The MSS is assumed to be 536 if not received on connection initiation [33, p.-85].

2.5 modeling energy consumption

A general approach to modeling the power consumption of IoT systems has been proposed in [34]. The research divides the elements contributing to the power consumption into network, acquisition, processing and system before analyzing each of these elements and combining the results into an analytical model. However, the model is not specialized for any particular communication standard, and an application developer will thus require a thorough understanding of the physical behavior of different alternatives. Furthermore, the presented parameters do not easily translate to, for example, payload size. As the this thesis aims to focus on cellular standards in particular, a more specialized model is needed.

The model devised in [16] and discussed briefly in section 2.2.3.3 does model the energy consumption over NB-IoT, but with no comparison to LTE-M. Furthermore, no research on the modeling of high-level communication protocols over cellular networks has been found. Thus, this thesis aims to assess UDP and TCP over both NB-IoT and LTE-M in a general model based on empirical data.

The 3GPP has done an analysis on the power consumption of NB-IoT devices [35, p.-393]. Their estimation takes maximum coupling loss[1] and transmission interval into account. Table 2.7 below shows the resulting prediction. No estimation was present for LTE-M.

Table 2.7: Battery life estimates from 3GPP TS 45.820, section 7.3.6.4 [35, p.-393].

	Battery life [years]		
Packet size, transmission interval	Coupling loss = 144 dB	Coupling loss = 154 dB	Coupling loss = 164 dB
50 bytes, 2 hours	22.4	11	2.5
200 bytes, 2 hours	18.2	5.9	1.5
50 bytes, 1 day	36	31.6	17.5
200 bytes, 1 day	34.9	26.2	12.8

2.5.1 Model

Based on the behavior described in section 2.3 of this chapter the model for the total energy consumption E_{tot} of a cellular connected device is shown in (2.3) below.

$$E_{tot} = \frac{E_{msg}(N) + E_{cDRX} + E_{psm}(N)}{T_{msg}} t + E_{start} \tag{2.3}$$

The parameters are described in the list below:

- N is the payload size in bytes.
- $E_{msg}(N)$ is the energy spent on the data transmission.
- E_{cDRX} is the approximation of energy spent in connected DRX mode after a transmission.
- E_{psm} is the energy spent during RRC idle mode and can be written as $p_{psm}t_{idle}$ where $t_{idle} = T_{msg} - t_{msg}(N)$.

This model specializes for applications that mainly does UL communication and assumes that the device enters PSM-mode straight after transmission and cDRX. Hence, the PSM Active timer, T_{3324}, is zero. An application is expected to upload with fixed intervals T_{msg}. The model does not account for possible alarm notifications and processing done during PSM. An approximation of the power consumed during the RRC Inactive Timer countdown, E_{cDRX}, is shown in equation (2.4). The were parameters introduced in section 2.3 and section 2.3.1.

$$E_{cDRX} = (p_{connected}(T_{cycle} - t_{onDuration} + E_{paging})\frac{t_{cDRX}}{T_{paging}} + E_{release} \tag{2.4}$$

[1]A cellular coverage metric defined by 3GPP.

- $p_{connected}$: power consumption while the device is connected, but inactive.
- E_{paging} : energy spent on a paging action
- $E_{disconnect}$: energy spent on RRC disconnect
- T_{paging} : The period of paging during the inactive countdown
- t_{cDRX} : The RRC Inactive Timer duration
- $t_{onDuration}$: The connected mode DRX onDuration timer, deciding how long the UE will monitor the network for data.

2.6 Linear Regression

The assumption that E_{msg} and t_{msg} are dependent on payload size can be tested using linear regression analysis. This will show if there is a linear relationship between the variables and yield usable models for predictions, should the assumption of linearity be viable. No research was found that used this method to predict power consumption over cellular networks, but it has been used with success in other settings. The method was applied for predicting smart home energy use in [36], and [37] used it for prediction of indoor signal propagation with NB-IoT.

2.6.1 Metrics of linear regression

There are several ways to measure the performance of a linear regression analysis. In [38], two metrics are highlighted as good values for assessing model efficiency and accuracy; the R^2 and the standard error of estimate (SSE). They are explained in detail below:

- R^2, also known as the coefficient of determination is a measure of how much of the variation in the data that can be explained by the model. This is an important score when the model is supposed to be used for prediction, as in the case of this report. The closer to 1 r^2 is the better, a lower score may lead to more error in the resulting model.
- SEE is a similar metric to standard deviation, showing how much the predicted values vary from the observed ones. Lower values of SEE means a more accurate model.

The goal of the experiments done in this project will be to gather enough data to achieve viable linear regression predictors for E_{msg} and t_{msg}.

Chapter 3

Implementation and methodology

This chapter will cover the implementation of test applications and the methodology used for data acquisition and processing. We will provide detailed background for design, equipment and software choices enabling the reader to set up a similar environment for testing and experiments.

3.1 Equipment, tools and software

This section will cover the equipment used for the research. Note that access to a computer is essential as both development and experiments require software running on a computer. This project has been developed and tested on a macOS computer, but both Windows and Linux should support the described procedure.

3.1.1 Cellular hardware: nRF9160 from Nordic Semiconductor

The GSMA lists several available cellular modules with support for LTE-M and NB-IoT [39]. We have chosen to use the nRF9160 System in Package (SiP) from Nordic Semiconductor as it supports both of the standards along with housing a full-fledged Arm Microcontroller unit (MCU) [40].

Nordic Semiconductor provides a versatile development kit (DK) housing the nRF9160 for evaluation and development use, shown in figure 3.1 It has all features necessary for developing a cellular application, including antenna and support for external devices through General purpose input output (GPIO) [41]. This should be adequate for the applications and tests we will design. For more versatile prototyping Nordic provides the Thingy:91 as well, a battery-driven cellular prototyping platform complete with sensors, buzzer, button and LEDs [42]. This product falls outside our scope as it sacrifices some ease of use by being more portable and targeting "Proof-of-concept" designs.

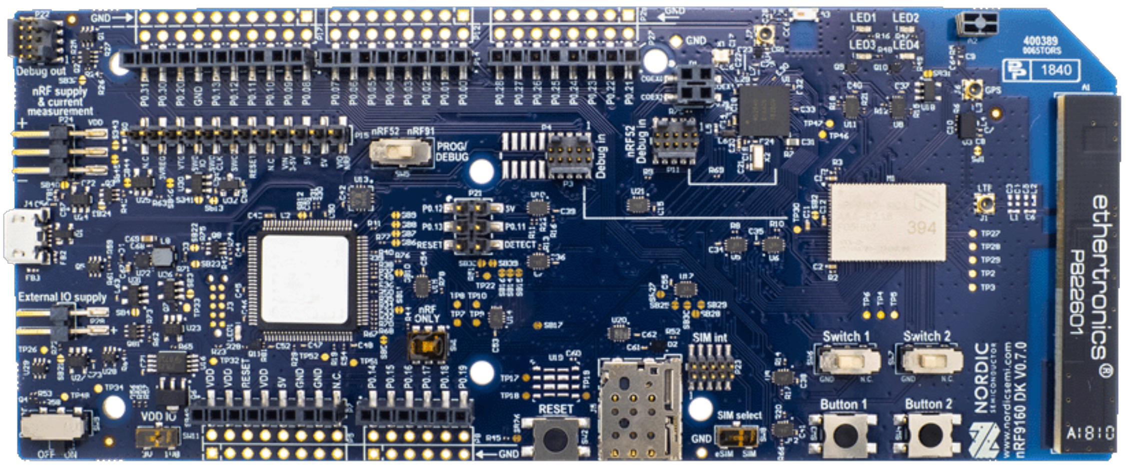


Figure 3.1: nRF9160 Development Kit. Picture credit: Nordic Semiconductor

3.1.2 Cellular provider: Telenor NB-IoT

There are two main providers of LTE-M and NB-IoT in Norway: Telia and Telenor. They have a similar level of deployment for both standards, but we chose Telenor to supply SIM-cards for the project because of availability. Allegedly they cannot deliver one SIM-card that supports both LTE-M and NB-IoT; thus, one for each standard was acquired. However, after testing, it was apparent that the NB-IoT card did indeed support both standards, and it was therefore used for the rest of the project.

3.1.3 Measuring unit: OTII-ARC

An OTII-ARC Qoitech, hereby OTII, was used for data acquisition. It is a specialized measuring and power supply unit targeting the development of power-constrained devices. It measures currents at a μA level with a sample rate of 4 kilo-samples per second, making it ideal for detecting the fluctuations we expect when transmitting as well as the sleep current of a low power cellular application like the ones assessed in this report. Furthermore, the OTII can power the device under test with up to 3.75 V standalone, and 5V with an external power supply. This is sufficient for the specified supply voltage range of 3.3V to 5.5V of the nRF9160 [40].

We control the OTTI from a computer through associated software. The software supports visualization of measurements, along with logging over Universal Asynchronous Receiver/Transmitter (UART) from the device. The latter can be beneficial for linking events on the device under test to power consumption. It is also possible to measure peak, bottom and average current as well as energy consumed during a chosen time window.

3.1.3.1 Considerations when choosing measurement equipment

There is a wide range of equipment and tools for measuring energy consumption, and choosing the correct tools is important for achieving the desired results. The experiments for this report was initially done using an oscilloscope of the type InfiniiVision MSO-X 20002A, but two issues arose that prompted the search for alternative measuring equipment. First of all, most oscilloscopes are not sensitive on a μV level, requiring a large shunt resistor when measuring weak currents. Using large shunt resistors can lead to "brown outs" [1], which can interfere with the operation of the nRF9160 SiP. Secondly, there is the issue of timing. As will be discussed in the result section, transmissions over cellular networks take several seconds and can be expected to enter several different power levels, as seen in figure 2.4b. While the trigger-functionality of oscilloscope helps detect voltage spikes and acquire segments of data, the short time window of sampled data makes it challenging to analyze transactions as a whole.

3.1.4 Power Supply: Gw Instek GPD-3303s

The Gw Instek GPD-3303s is a lab power supply that reliably can deliver the 8 volts needed to drive the OTII externally [43].

3.1.5 MQTT broker and client

There are many free and commercial brokers available for MQTT. The open-source Eclipse Mosquitto MQTT software was chosen for this project[44] as it is easy to use and setup. Self-hosting a broker is also preferable for testing as it enables full control over its operation, contributing to a more reliable and predictable testing environment. Though, it does not necessarily represent open or commercial brokers as these could present more strict requirements with relation to security and client interaction.

The messages published from the nRF9160 were monitored on a computer, using an MQTT client application. The program "MQTT explorer" was chosen for this purpose as it has a simple interface and retains a history of published messages in their respective topics. Simply connect the program to the desired MQTT broker and subscribe to the desired topic. We used this software to verify that the messages arrived correctly.

3.1.6 CoAP server

The CoAP application is connecting to Telenor's own CoAP server, accessible through their online NB-IoT test dashboard [45]. The dashboard allows the user to monitor published messages and to

[1] a drop in voltage across the power supply system, so that the whole device or parts of it are underpowered.

post messages to the device. Access to this server is limited to Telenor networks.

3.1.7 Data processing platform: Python Jupyter Notebook

As the OTII can export raw data as Comma Separated Variable (CSV)-files, virtually any data processing tool is viable. The Jupyter Notebook Python environment was chosen for this purpose as it is a powerful tool for scientific computing [46].

3.2 Development and test environment

This section describes how to set up the hardware and software environments for the tools and software introduced in the previous section.

3.2.1 Hardware setup

Nordic Semiconductor has a guide for using the OTII with the nRF9160 for measuring current [47]. Following these instructions enables the user to flash new applications and log information from the nRF9160 DK while it is still connected to the measuring setup. This is a great advantage for iterative development and testing. Figure 3.2 shows a wiring diagram of the setup. In the guide 3.70V is supplied to the nRF9160 DK. The OTII should, in theory, manage to act as a standalone power supply for that voltage, but an external supply is used in the guide. Thus, we use the same setup with the Gw Instek external power supply.

The nRF9160 and OTII are both connected to the computer with micro-USB cables. The OTII has a barrel connector for external power, so we used an adapter for the wires from the Gw Instek. The power supply was configured to provide 8V with a maximum of 1A through channel 1.

3.2.2 Development environment for the nRF9160

The nRF Connect software development kit (SDK) (also known as NCS) is Nordic's framework for development of cellular IoT and short-range wireless applications [48]. NCS comes with a wide variety of samples and example code to get started with development, along with application protocol stacks, libraries and hardware drivers. Furthermore integrates the Zephyr, a Real Time Operating System (RTOS) governed by the Linux Foundation targeting constrained devices [49]. The SDK is available publicly on GitHub, but is best installed via nRFConnect for Desktop. The desktop app provides a step-by-step Getting Started guide for setting up NCS, and its required components. In addition, the desktop app has applications for updating firmware and serial monitoring [50].

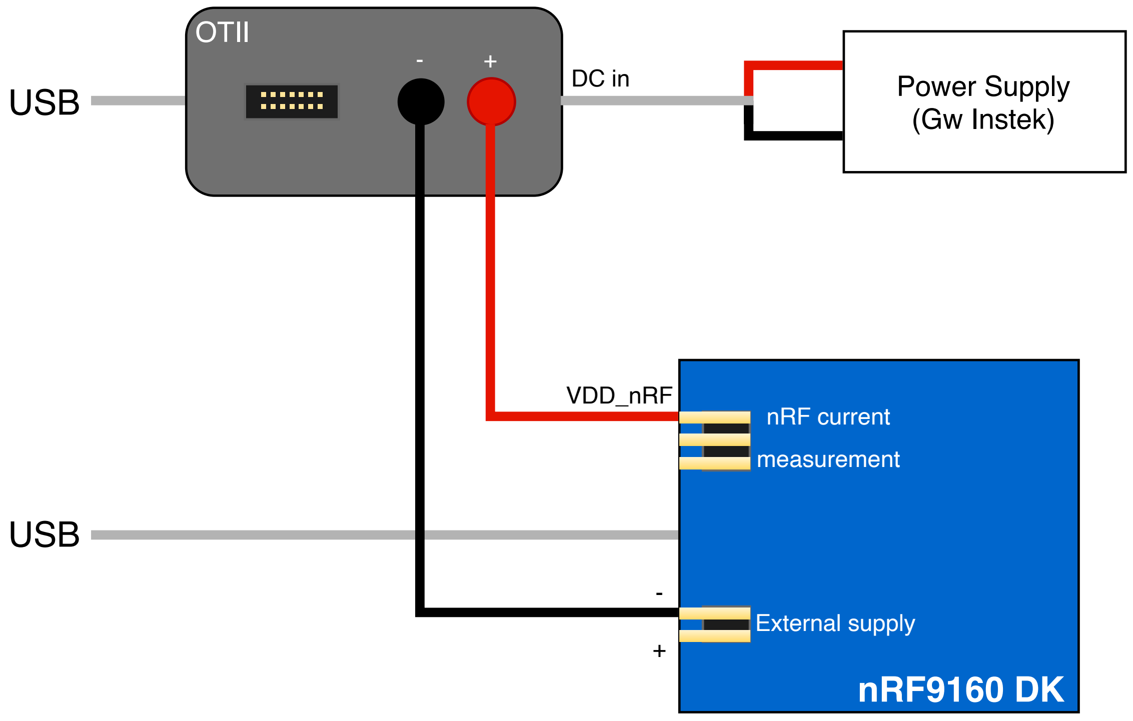


Figure 3.2: Wiring diagram for the hardware setup. Based on the diagram [47].

The hardware and software versions for the Nordic environment used in this report are shown in table 3.1 below.

Table 3.1: Development environment.

nRF Connect SDK	version 1.2
nRF9160 DK	version 8.5
modem firmware	version 1.1

When setting up the development environment for the nRF9160 DK, it is recommended to follow Nordic's getting started guide as it is being updated along with software and firmware updates [51]. The project code available on github can be added to the main directory of the NCS installation [52]. After which the desired application can be built and flashed by navigating to the application directory using a terminal of choice and running the following chain of commands:

```
mkdir build
cd build
cmake -GNinja -DBOARD=nrf9160_pca10090ns ..
ninja flash
```

This will result in a working binary file that can be flashed[2] to the nRF9160 DK.

[2]loaded into the persistent memory of the MCU.

3.2.3 Communicating with the nRF9160 modem

The nRF9160 modem is controlled with AT commands. This is a common command set for modems that follows standardized syntax rules defined by the 3GPP [53]. The available commands for the nRF9160 modem are defined in [54]. Though the application backend performs most modem operations, it is possible to directly use these commands to control or request information from the modem. In this report, we used the LTE Link Monitor app from the nRF Connect for Desktop program for serial communication with the modem.

3.3 nRF9160 application design

The initial goal of this report was to make two power-optimized applications, one with CoAP and one with MQTT and compare them with Disruptive Technology's already existing solution. However, we decided that modeling the energy consumption of the two communication protocols over LTE-M and NB-IoT would pose much more usability as a tool for developing future applications. Hence, the applications were designed with the intent of gathering data for the model, focusing on making the parameters configurable and the data acquisition straightforward. The project header, main and configuration files are attached in appendix B and C. The rest of the project code is, as mentioned, available on github [52].

We chose to base the applications on already existing implementations of MQTT and CoAP for the nRF9160, as this yielded more time for testing and we could expect working behavior. The MQTT application was built around Nordic's "Simple MQTT client" sample [55] and for the CoAP application we took basis in an implementation done by Exploratory Engineering, a IoT working group from Telenor [56].

For both instances, the protocol operations were moved to a separate module and the kernel thread features of the Zephyr RTOS to handle events related to the protocol behavior. This cleared up the main-files while also decoupling the protocol behavior, only exposing essential functionality like initiation and messaging functions. All MQTT messages are sent with QoS 1 as specified in 2.6, while all CoAP messages are sent as confirmable.

The function flow of the main file can be seen in table 3.2. This introduces the two test functions `run_size_sweep` and `run_periodic`. The former is used for testing energy consumption in relation to increasing payload size. The latter is used for testing a more realistic sensor scenario where the device transmits a fixed data size at the periodic TAU. We configured the applications to run with PSM behavior for both test functions as applications not using PSM is out of scope for this thesis.

As mentioned the `run_periodic` function will run with fixed intervals based on the requested periodic TAU. This does in fact not comply with the specifications for PSM as described in section

Table 3.2: Application function calls.

Function name	description
`button_leds_init`	Initializes the LEDs and buttons on the board. The LEDs are used for tracking where the current status of the application. The buttons were used initiating a transmission instantly.
`setup_psm`	We provide the defined PSM settings to the modem. It will provision these to the network when running the `modem_configure` function. Providing the values in advance is much faster than negotiating after connecting.
`modem_configure`	Starts the modem and initializes the connection to the chosen network.
Initialize connection	• **MQTT**: initialize TCP connection with the broker and start kernel thread. • **CoAP**: initialize the endpoint and associated variables, like URI-path, as no persistent connection is necessary. Start kernel thread.
Confirm PSM timer values	Reads the PSM timer values finally agreed on by the network in case they differ with the pre-defined ones. If the `run_periodic` test is defined, a timer that triggers on the confirmed periodic TAU value is started.
main loop	Two possible test functions: • **`run_size_sweep`**: send data with increasing payload with a short, fixed delay between each transmission. Exits when reaching maximum payload size for the respective application. • **`run_periodic`**: sends constant sized data at a fixed interval given by the requested PSM value, similar to an actual sensor application. This test end after a specified number of messages has been transmitted.

2.3. According to the definition of the mechanism, the requested periodic TAU determines how long a device will stay in RRC idle mode after releasing from RRC connected. This means that the duration between transmissions is dependent on how long time transmissions take, which in turn leads to non-periodic behavior as transmission time can vary from message to message. We chose to force true periodic transmissions in the application developed for this thesis as this is more predictable for both developer and end-user. In practice, this means that the application transmits data with the interval of the requested periodic TAU, but the time spent in sleep (i.e RRC idle) will be a bit shorter. Commonly the periodic TAU is in the range of hours, days or weeks, meaning the small deviation will be insignificant.

3.3.1 Project Configuration

Every NCS project has a file called "prj.conf" which is used for application- and SDK-wide settings. Table 3.3 shows the key settings that were altered between different test configurations. Other application-specific settings are available in the attached code.

Table 3.3: Project configurations

Configuration name	Value	Description
CONFIG_LTE_NETWORK_MODE_NBIOT	y/n	Used for setting the modem in NB-IoT mode. LTE-M is default.
CONFIG_LTE_PSM_REQ_RPTAU	String of bits	Used for setting the requested periodic TAU. Encoded as mentioned in section 2.2.3.
CONFIG_LTE_PSM_REQ_RAT	String of bits	Used for setting the requested active time. Encoded as mentioned in section 2.2.3.
CONFIG_SERIAL	y/n	Used for logging over UART. This consumes power and are deactivated when logging is not needed.
CONFIG_AT_HOST_LIBRARY	y/n	Used for monitoring modem operation through AT command responses. Must be set to "n" if CONFIG_SERIAL is "n".

3.3.2 Determining maximum payload

During development it was discovered that the CoAP application was restricted to a maximum payload size of 1440 bytes. This is a direct consequence of the MTU of the cellular network, which was determined to be 1500. The AT-command `AT+CGCONTRDP=0` [54, p.-106]can be used to find the network MTU. We can also arrive at this conclusion using the functions defined 2.4.2.1. The CoAP message overhead can be calculated using equation (2.1),

$$\begin{aligned} s_{\text{overhead}} &= 4 + 8 + 20 \\ s_{\text{overhead}} &= 32 \end{aligned}$$

Using this result in (2.2) yields the expected result

$$\begin{aligned} s_{\text{max payload}} &= \text{MTU} - (s_{\text{IPv4}} + s_{\text{UDP}} + s_{\text{overhead}}) \\ s_{\text{max payload}} &= 1500 - (20 - 8 - 32) \\ s_{\text{max payload}} &= 1440 \end{aligned}$$

Hence, the CoAP application behaves as expected. A similar limitation is not seen in the MQTT application, as it manages to transmit packages with a maximum of 4096 bytes of payload. We assume that this is due to the segmentation mechanism of TCP explained in section 2.4.1, where larger transmissions are segmented to avoid IP fragmentation. The restriction of 4096 bytes is assumed due to constant `BSD_IP_MAX_MESSAGE_SIZE`, defined in `bsd_limits.h` of the nRF Connect SDK.

3.3.3 Application challenges

We designed the applications to run as similarly as possible, but there were challenges with seemingly periodic current spikes of 9mA on the MQTT application. Further inspection revealed that the spikes arrived with increasing intervals, restarting at the shortest interval each transmission. This behavior may indicate that it may be related to some kind of exponential backoff scheme[3] in the TCP operation of the application. The issue was also discussed with the support team of Nordic Semiconductor. They agreed that it could be related to the TCP backend and suggested to close the socket used by the MQTT library. Unfortunately, this did not work, and at the time of writing, the issue is not resolved. As the current spikes are small and very short, they are not expected to affect the actual energy consumption much.

In section 2.3.1, the use of Release Assistance Indication was discussed as a possible optimization option for power saving as the device can release from RRC connected mode as soon as the transmission is finished. It was chosen not to make use of this functionality in this report as most measurements were already finished when the possibility of RAI was discovered. However, the developed model can predict the usage of RAI as the cDRX energy can be chosen to be zero. Lastly, the nRF9160 only supports RAI for NB-IoT and not LTE-M [54] at the time of writing, so it would not be possible to test for all configurations.

3.4 Data processing

This section will cover how the bulk of the data acquired with the OTII was processed. The processing code is attached in appendix D, E and F. It is also available on github [57].

A sample of the typical raw data when running running a test with the `run_size_sweep` function is shown in figure 3.3. The experiment is explained in detail in section 4.2. In this case, the measurements displayed are for the startup and first five transmissions of the CoAP over NB-IoT configuration.

In order to determine the energy used during each transmission, the transmissions had to be segmented, so that we could calculate the energy spent individually. This processing was done in

[3]An algorithm for spacing out repeated retransmission with decreasing rate.

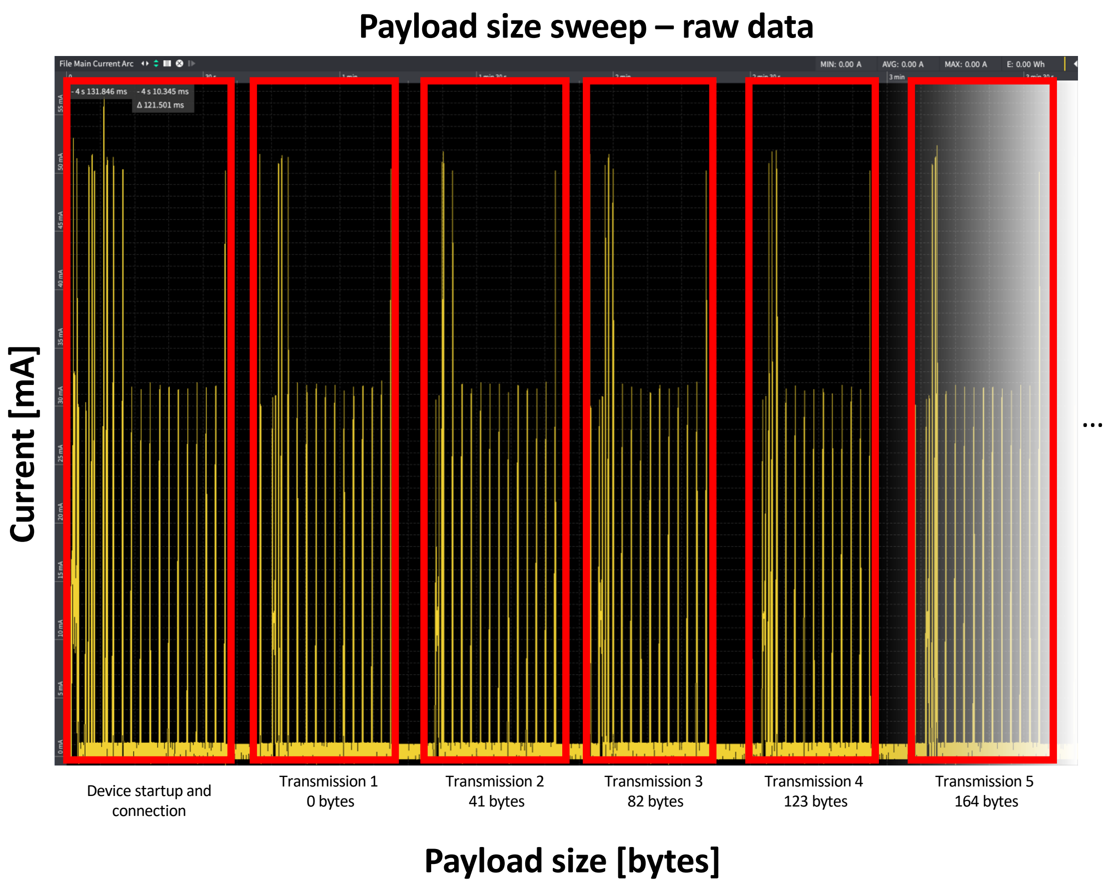


Figure 3.3: Raw data example from a CoAP over NB-IoT payload sweep experiment. Periods of transmission activity are marked with red, with identifying labels underneath.

the Python Jupyter Notebook environment. The segmenting code iterates through the data until it triggers when the current consumption is above a certain threshold that indicates a transmission. The index will then jump past the assumed length of the transmission and iterate backward until the end of the transmission is detected. We then save the segment and duration of it in separate arrays for further processing. The device startup is ignored. Figure 3.4 demonstrates this based on the data in figure 3.3.

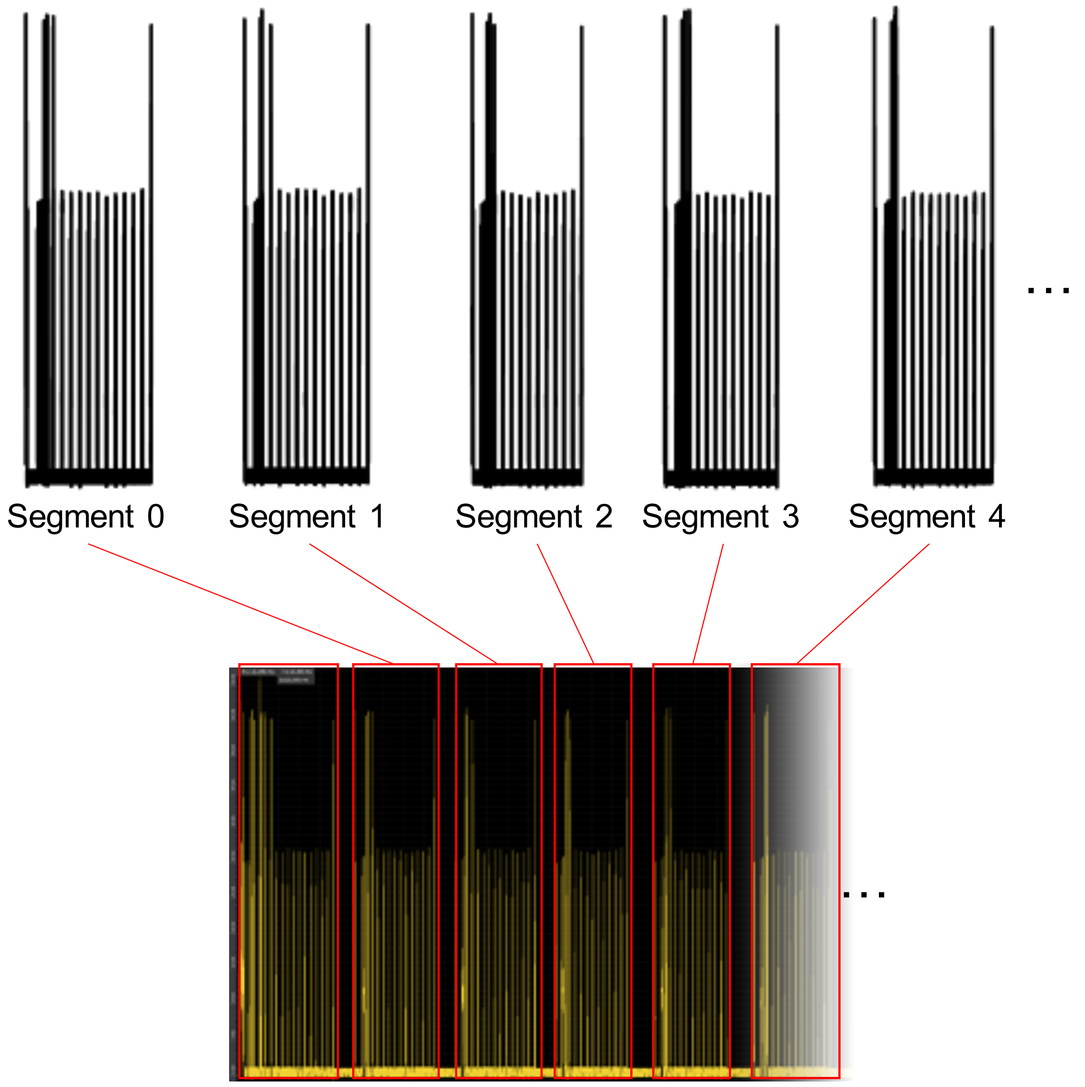


Figure 3.4: Segmentation process demonstrated on the CoAP data from figure 3.3.

Following the segmentation the energy consumed across each segment was calculated using the following equation:

$$E_{segment} = (\Delta t * V \sum_{n=0}^{N} sample(n)) \tag{3.1}$$

Where V is the supply voltage of the system, Δt is the duration between each sample, N is the number of samples in a segment and $sample(n)$ is the current sample. One segment encloses both the actual data transmission and the connected DRX period in which the UE stays connected to the network. As the target of these measurements is to predict the energy consumed during each data transmission regardless of how long the device stays connected afterward, we wish to disregard the cDRX contribution. Hence, the measured cDRX energy for either LTE-M or NB-IoT depending on the dataset is subtracted from each segment energy resulting in the value E_{msg}. We estimate the actual cDRX contribution in the model with equation (2.4). Also, the RRC inactive time for the respective cellular standard is subtracted from the calculated transmission times. This processing was performed on every dataset, eventually storing the resulting segment energies and transmission times in arrays tied to the corresponding payload size. Figure 3.5 shows a scatter visualization example of the data at this point in the process.

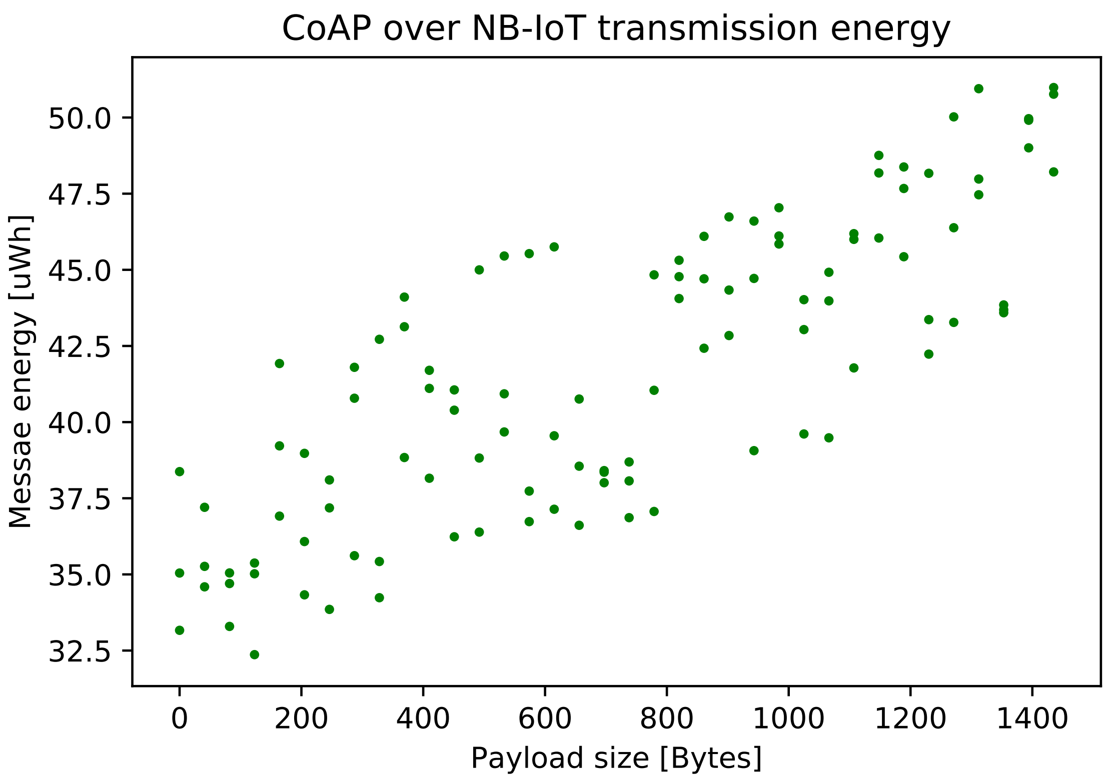


Figure 3.5: Scatter visualization of the transmission energy for CoAP over NB-IoT.

To estimate the relation between payload size, energy consumption and transmission time, we used ordinary least squares linear regression on the processed data. This was done using the Python library "Statmodels", that provides powerful tools for computing linear regression analysis and assessing its quality. The resulting coefficients and intercepts were then used to calculate $E_{msg}(N)$ and $T_{msg}(N)$ for the model.

Chapter 4

Experiments and results

This chapter will cover the experiments done to gather data for the model, as well as the results of these experiments. The first section will present the application performance and list the different values that were measured for use in the model parameters. We will then show how the payload sweep experiment was set up and cover the results from that. This is followed by an assessment of the results and accuracy of the linear regression before showing the results of testing the model against real data. The section concludes with a small case example of how the model can be used. The nRF9160 DK was running at 3.70V throughout all experiments, as per what is used in Nordic's reference tests [40].

4.1 Parameter measurements

The various parameters used in the model were obtained through measurements using the OTII software. This concerns the variables of the cDRX energy estimation in (2.4), as well as the average sleep power, p_{psm} and average power during connected mode, $p_{connected}$. We performed the measurements in the manner described below:

- p_{psm} : the average current over 5 minute window in PSM, between two transmissions. Multiplied with the supply voltage.
- $p_{connected}$: 20 samples of the average current between paging in RRC connected mode. Multiplied with the supply voltage.
- Timers : using the specification values introduced in table 2.4 of section 2.3.1 to identify the settings determined by the network.
- Paging and release energy: 20 samples measured from individual LTE-M and NB-IoT transmissions.

- Total cDRX energy: 20 samples measured over the RRC Inactive timer interval from the end of a transmission. Used to remove cDRX contribution from the transmissions, as well as to verify the cDRX approximation.

An average sleep current of 108 μA was measured for the CoAP application, while 117 μA was measured for the MQTT application. The higher average is due to the current spikes mentioned in the implementation section. We measured the average idle current for both applications to $157.7(+/-0.732)\mu$A. Table 4.1 shows the rest of the resulting values. Full table of measurements is present in appendix A.

Table 4.1: Measurement results

		NB-IoT	LTE-M
t_{cDRX}	[s]	20.48*	10.24*
$t_{onDuration}$	[s]	0.2	0.1
T_{paging}	[s]	2.048	0.320
E_{paging}	[μWh]	$2.58(+/-0.365)$	$3.227(+/-4.098)$
$E_{release}$	[μWh]	$2.171(+/-0.52)$	$0.448(+/-0.193)$
Total cDRX energy	[μWh]	$28.48(+/-1.142)$	$101.95(+/-14.771)$

* Confirmed by Telenor over email.

Important observations to make here is that the results indicate LTE-M being much more spurious than NB-IoT in the matter of energy used during cDRX. The significant standard deviation of E_{paging} during LTE-M is due to the UE monitoring for longer than the designated onDuration resulting in more energy being consumed. The reason for this is unknown. These variations also contribute to the high standard deviation in the total cDRX energy measured for LTE-M.

Given the values in table 4.1 an approximation for E_{cDRX} can be calculated using(2.4). The equation yields 29.874 μWh for the NB-IoT transmissions, differing 4.895% from the measured mean and falling outside the standard deviation. For LTE-M (2.4) yields 102.298 μWh differing 0.34% from the measured mean, being well inside the expected standard deviation. These comparisons only verify the cDRX approximation for one case, Telenor's current network. Testing for other cases would be ideal, but as the values are network specific, it is not possible with the current test environment.

4.2 Payload size sweep

This experiment strives to determine how TCP and UDP perform on cellular networks with relation to message size. We conduct the test by running the two respective applications with increasing payload size while measuring the current consumption using the OTII. These measurements will give insights into how both the protocols and cellular standards handle large data sizes and may

Table 4.2: Settings for payload sweep experiment

		MQTT	CoAP
Maximum test payload	[bytes]	4096	1439
Payload size step	[bytes]	64	41
Transmission interval (NB-IOT / LTE-M)	[s]	40/30	30/30
Requested TAU	[hours]	2	2
Requested Active Time	[hours]	2	2

uncover trends related to the payload size.

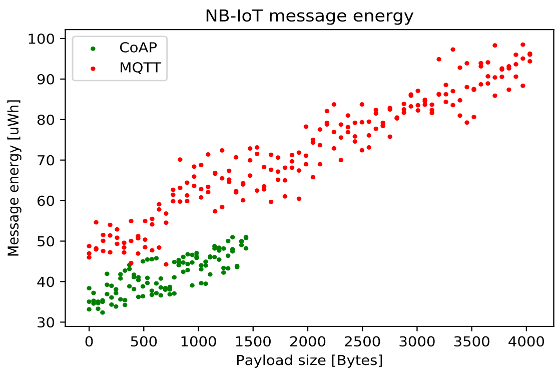
The experiment was performed by transferring messages of increasing size with constant intervals using the already mentioned `run_size_sweep` function. Initially, the incremental value was chosen to 5 bytes per step, starting at a zero byte sent with an interval of 10 seconds. PSM was not used as it was assumed that it would be irrelevant in this test setting. However, the short interval and no PSM resulted in data where the transmissions proved difficult to distinguish. Thus, we redid the tests with different settings shown in table 4.2. Three identical runs were done for every configuration to ensure an adequate amount of data for further processing. The configuration should yield 108 data points for the CoAP implementations and 192 data points for the MQTT applications.

The maximum test payload is based on the determined maximum payload for the respective protocols, calculated in section 3.3.2 of implementation chapter. We chose the step value so that trends related to increased payload would stand out, without requiring unreasonable time to complete the experiment. It was also important that the step value was a factor of a number close to the maximum possible payload so that the test reflects the payload constraints of the respective protocol. The MQTT step size of 64 bytes amounts to a total of 64 samples, with the maximum payload being 4096 bytes. The CoAP step size of 41 bytes amounts to a total of 36 samples, with the maximum payload being 1435 bytes. It could be problematic that a different amount of samples is gathered for the two protocols, but as we expect linear behavior, the general trend should still be apparent.

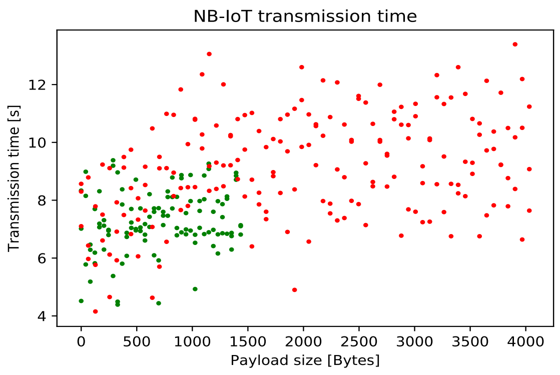
The transmission interval is also varied between the different tests. This is to compensate for the faster transmission time of LTE-M. In retrospect, this value should have been the same for all tests in order to simplify the data processing afterward.

To be able to distinguish the transmissions easily and conform the test more to the scope of low power applications, PSM was used during the test. A periodic TAU of 2 hours was requested to force the device to enter back into PSM after transmissions without worrying about potential Tracking Area Updates. As explained in section 2.2.3.1, the device can freely wake up and transmit at any time during PSM.

There were no failed transmissions to the author’s knowledge, and all messages arrived at the destination. The resulting energy and transmission times for NB-IoT are shown in figure 4.1.



(a) Message energy for CoAP and MQTT over NB-IoT. cDRX contribution of 28.48 μWh subtracted.



(b) Message transmission times for CoAP and MQTT over NB-IoT. RRC Inactive time contribution of 20.48 s subtracted.

Figure 4.1: Results of NB-IoT payload sweep measurements.

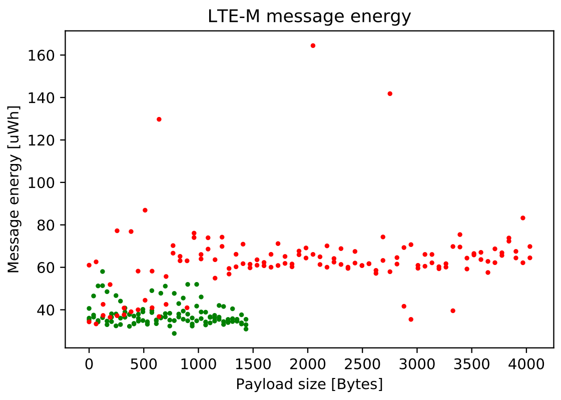
The results of figure 4.1a indicates that the energy consumed by both CoAP and MQTT transmissions over NB-IoT increases linearly with payload size. As expected, MQTT consumes more energy than CoAP, with an average of ~32% more energy consumed per message within the payload range of CoAP. MQTT uses on average 5% more time per message within the same payload range.

The resulting energy and transmission times for LTE-M are shown in figure 4.2. Due to unexpected behavior in one of the MQTT payload sweeps over LTE-M, only the results from two sweeps are presented for that configuration. The processing method presented in 3.4 did not manage to handle the case. In retrospect, we could have avoided this issue by allowing a longer time interval between transmissions when performing the experiment.

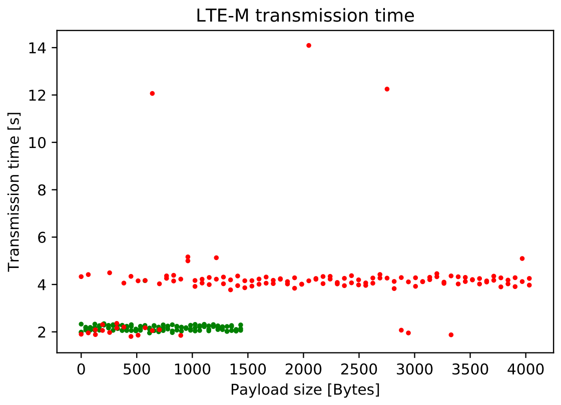
The behavior of LTE-M is vastly different from that of NB-IoT. Both the energy consumed and time spent seems independent of the message payload within the measured range. It appears that the energy and time used for transmitting MQTT messages increase to a new level for payloads of ~500 bytes or more. Notice as well that the MQTT measurements include some major energy peaks.

The average of the measurements was calculated and smoothed using a Savitzky–Golay filter with a window length of 9 and an order 3 polynomial. The results of this calculation, both with and without cDRX and RRC Inactive timer contribution is shown in figure 4.3. The 536 byte MSS of TCP is marked for possible identification of change in energy usage. Note that averaging and smoothing may lead to loss of information. However, this use of the data is only for the sake of comparison and will not be utilized for the model and other further calculations.

These results show that including cDRX energy, which is the actual situation for the measured data, LTE-M performs worse than NB-IoT in terms of energy spent per transaction. However, the



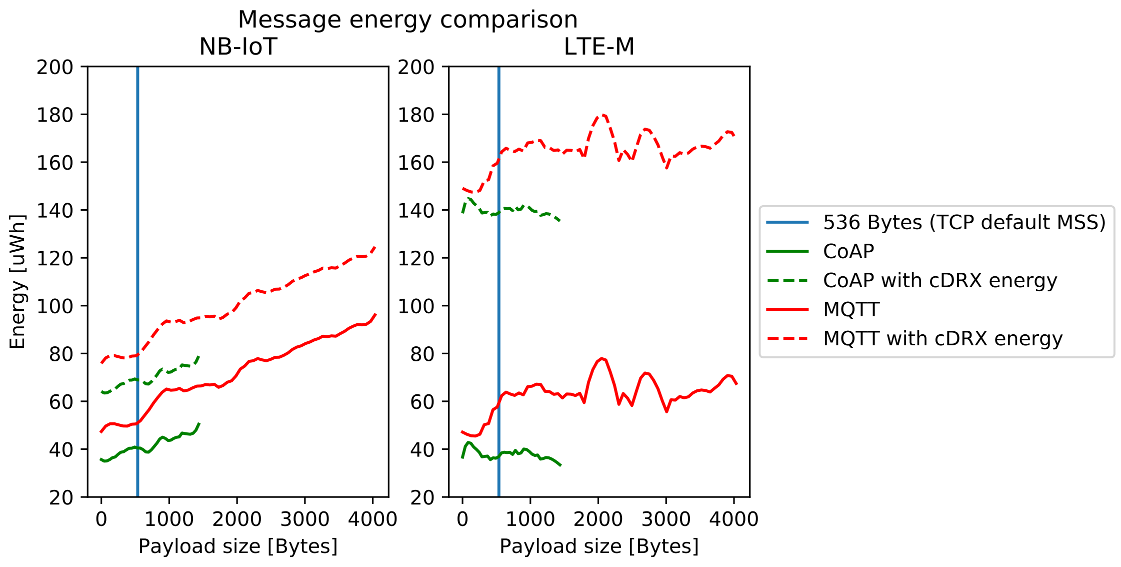
(a) Message energy for CoAP and MQTT over LTE-M. cDRX contribution of 101.95 μWh subtracted.



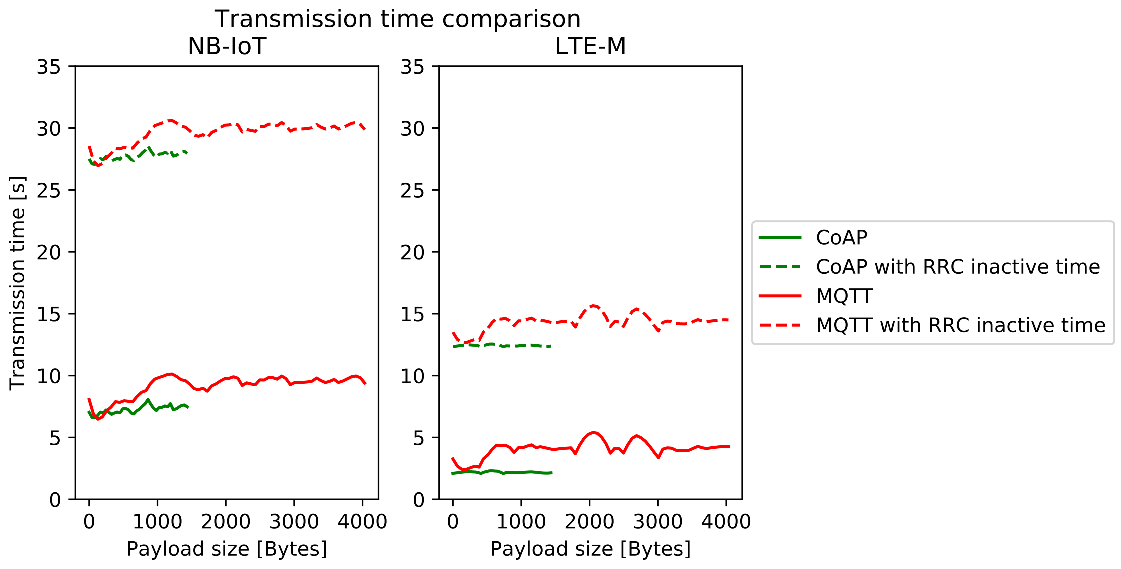
(b) Message transmission times for CoAP and MQTT over LTE-M. RRC Inactive time contribution of 10.24 s subtracted.

Figure 4.2: Results of LTE-M payload sweep measurements.

LTE-M transactions are much faster, which we can expect due to the higher throughput of the standard. It is also worth noticing that the increased energy usage for MQTT with payloads after ~500 bytes and above is seemingly present in both the NB-IoT and LTE-M data.



(a) Comparison of message energy for MQTT and CoAP over both LTE-M and NB-IoT. The results are shown with and without cDRX contribution.



(b) Comparison of message transmission time for MQTT and CoAP over both LTE-M and NB-IoT. The results are shown with and without RRC inactive timer contribution.

Figure 4.3: Comparison of results from payload sweep measurements.

4.3 Regression results

This section will cover the linear regression estimation results used to model the transactions' energy and transmission time. The linear regression is done based on the resulting data after processing, covered in the previous section.

4.3.1 Transmission energy

Table 4.3 shows the resulting coefficients and test scores of the regression analysis performed to predict message energy based on the payload size. We explained the meaning of the scores presented herein section 2.5 of the background chapter.

Table 4.3: Message energy regression results and scores

	Observations	Intersect [μWh]	Slope [μWh/Bytes]	SEE [μWh]	SEE/prediction [%]	R^2
CoAP NB-IoT	108	3.52E+01	9.10E-03	6.947	2.870	0.648
CoAP LTE-M	108	3.98E+01	-2.70E-03	13.834	5.226	0.048
MQTT NB-IoT	192	4.88E+01	1.16E-02	6.041	4.194	0.915
MQTT LTE-M	128	5.68E+01	3.10E-03	26.047	16.363	0.048

The regression scores show that the regression for NB-IoT is promising for predicting message energy. Both CoAP and MQTT achieve high R^2 indicating a good correlation between the message energy and payload size. Furthermore, SEE shows that the predictions, on average, only deviate $\sim$ 6% to $\sim$ 7$. The results are not as favorable for LTE-M, however. Both CoAP and MQTT scores low in the R^2 test, indicating that the payload may not correlate well with message energy. This is strengthened by high variation in the predictions, showing that the model is inaccurate. Besides, the negative slope of the CoAP over LTE-M prediction does seem unlikely and as such the LTE-M models are probably not fit for prediction of message energy.

Note the intersect values here, as this is the base energy required for any transmission. This means that small payloads have a significant energy overhead compared to larger ones.

The predictions of 4.3 is plotted together with test data in figure 4.4. This enables us to assess the regression results graphically. We present the data with a focus on comparing the respective communication protocols within each of the cellular standards. This is to accentuate how LTE-M and NB-IoT perform in relation to each other. Note that cDRX energy is not included in these calculations.

The plots do show that the NB-IoT models achieve a more satisfactory fit with the measured data. The LTE-M fits for both CoAP and MQTT seem to be affected by considerable variation, but it is not randomly distributed throughout the data set, affecting the fit's quality. However, we can

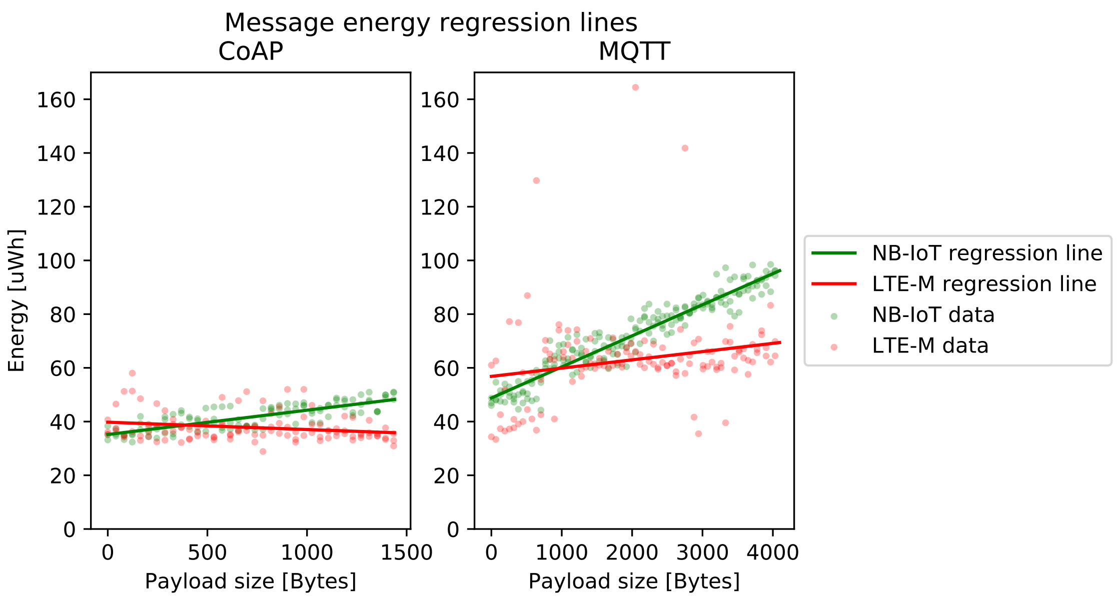


Figure 4.4: Regression lines and calculated values for transmission energy of messages.

make one important observation. Both for CoAP and MQTT, it will be more beneficial from an energy perspective to use LTE-M when the payload reaches a specific size, given that the device enters PSM straight after transmission. The intersections between NB-IoT for the regression lines in figure 4.4 is 390 bytes for CoAP and 948 bytes for MQTT. Though, as the models for LTE-M, in this case, are deemed inaccurate, these intersections are probably likewise.

4.3.2 Transmission time

The results for the simple regression analysis performed to predict message transmission time based on payload size is presented in table 4.4. We use the same metrics for assessing regression quality, as in the previous section.

Table 4.4: Message transmission time regression results and scores

	Observations	Intersect [s]	Slope [s/Bytes]	SEE [s]	SEE/prediction [%]	R^2
CoAP NB-IoT	108	2.74E+01	5.00E-04	1.062	3.830	0.045
CoAP LTE-M	108	1.24E+01	-2.79E-05	0.217	1.749	0.003
MQTT NB-IoT	192	2.87E+01	5.00E-04	1.693	5.713	0.107
MQTT LTE-M	128	1.38E+01	2.00E-04	1.563	10.925	0.032

The results are represented graphically as well in figure 4.5. Unfortunately, these results are not promising for transmission time prediction. LTE-M performs badly in this instance as well, with very low values for R^2. For CoAP over LTE-M transmission time is practically unchanged with increased payload within the measured range, indicating no correlation. The resulting prediction could still prove useful, as the SEE is low. However, a constant value would likely suffice as well. MQTT over LTE-M scores better in terms of R^2, but as discussed in 4.2 above, MQTT has similar behavior to CoAP over LTE-M apart from increased energy and time spent per messages over ~500 bytes. This can also be seen in 4.5. The scores of the NB-IoT predictions are lower than in the case of message energy, especially in the case of CoAP over NB-IoT. These results could overall indicate that the correlation between transmission time and payload size, in general, is not very strong and that other factors affect the transmission time to a higher degree.

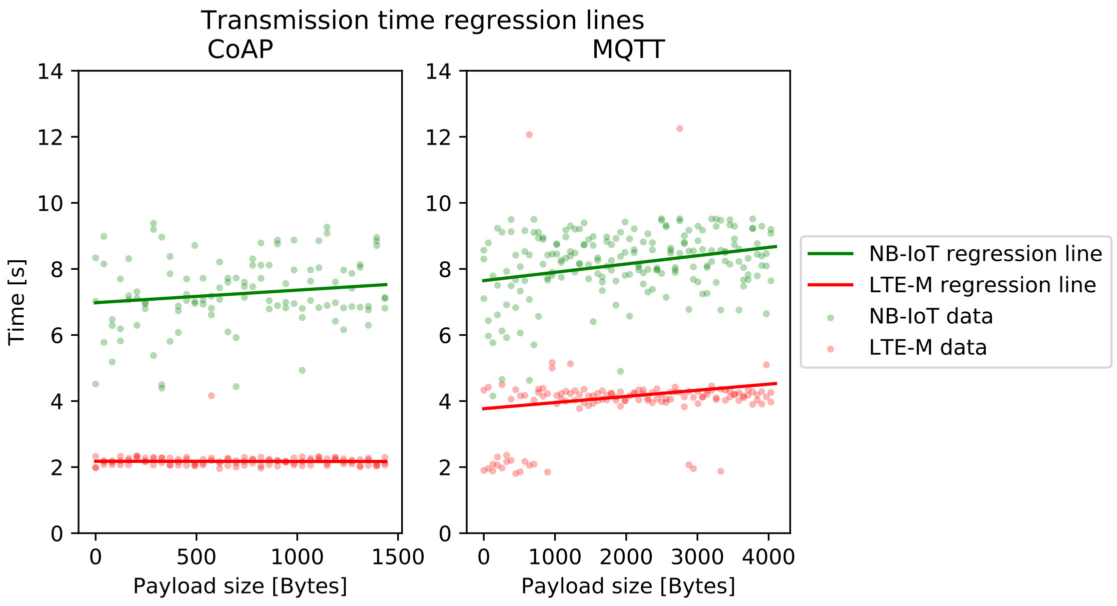


Figure 4.5: Regression lines and measured values for transmission time of messages.

4.3.3 Regression remarks

From the regression results presented so far, it may seem that simple linear regression is inadequate for predicting anything other than the message energy consumption for CoAP and MQTT over LTE-M. However, the results of the regression analysis will still be used for the final models, and any error will be assessed with the results of this section in mind.

4.4 Model performance

In this section the testing of the final model will be described followed by the results and errors. The four configurations, CoAP and MQTT over NB-IoT and LTE-M, was tested for two scenarios `run_periodic` function and the current consumption measured using the OTII. Five messages were transmitted, yielding five samples. The scenarios are described below.

- Small: A $T_{msg} = 300$s transmission interval with $N = 256$ byte payload upload.

- Large: A $T_{msg} = 300$s transmission interval with $N = 1280$ byte payload for CoAP and $N = 4096$ byte payload upload.

The same configuration was put into the model (2.3) yielding the resulting equations shown in table 4.5 for the small test and table 4.6. The values discussed in section 4.1 of this chapter are used for E_{cDRX} and p_{psm}. As no approximation for E_{start} has been made in this thesis, we chose to use the starting energy of the measured data. This will also make the comparison with the model fairer as it starts with the same origin as the measured data.

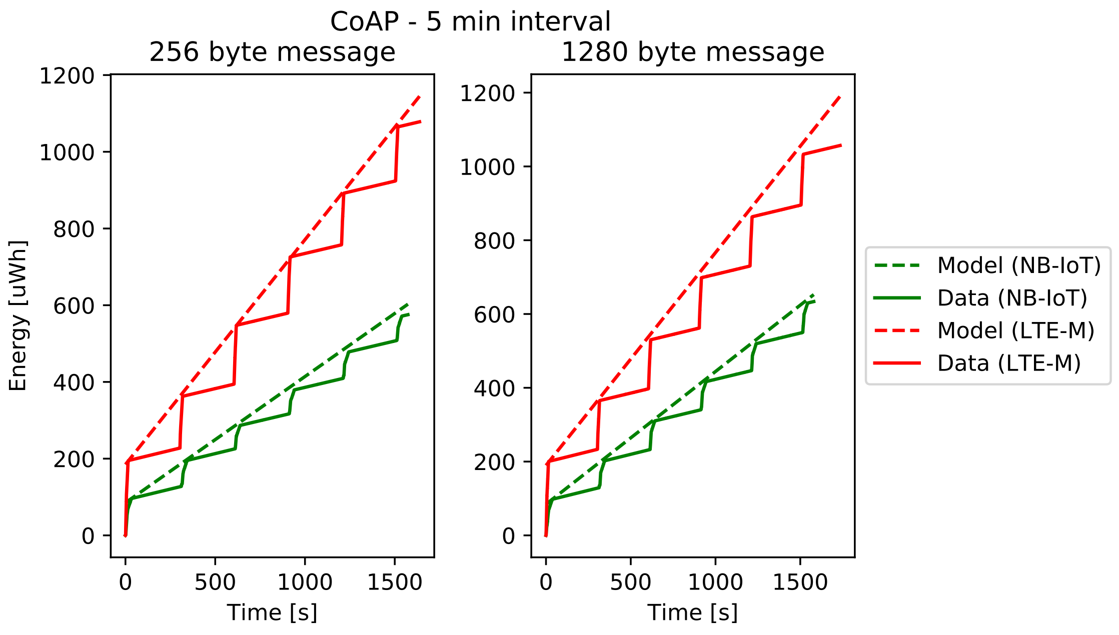
Table 4.5: Prediction of total energy for small test. ‹

CoAP over NB-IoT	$\frac{37.490\mu\text{W h}+29.874\mu\text{W h}+108\text{A}*3.7\text{V}(300\text{s}-7.011\text{s})}{300\text{s}}t + E_{start}$	$0.329t + 85.031$
CoAP over LTE-M	$\frac{39.072\mu\text{W h}+102.298\mu\text{W h}+108\text{A}*3.7\text{V}(300\text{s}-2.197\text{s})}{300\text{s}}t + E_{start}$	$0.586t + 184.766$
MQTT over NB-IoT	$\frac{51.730\mu\text{W h}+29.874\mu\text{W h}+117\text{A}*3.7\text{V}(300\text{s}-8.297\text{s})}{300\text{s}}t + E_{start}$	$0.384t + 121.592$
MQTT over LTE-M	$\frac{57.616\mu\text{W h}+102.298\mu\text{W h}+117\text{A}*3.7\text{V}(300\text{s}-3.658\text{s})}{300\text{s}}t + E_{start}$	$0.656t + 245.139$

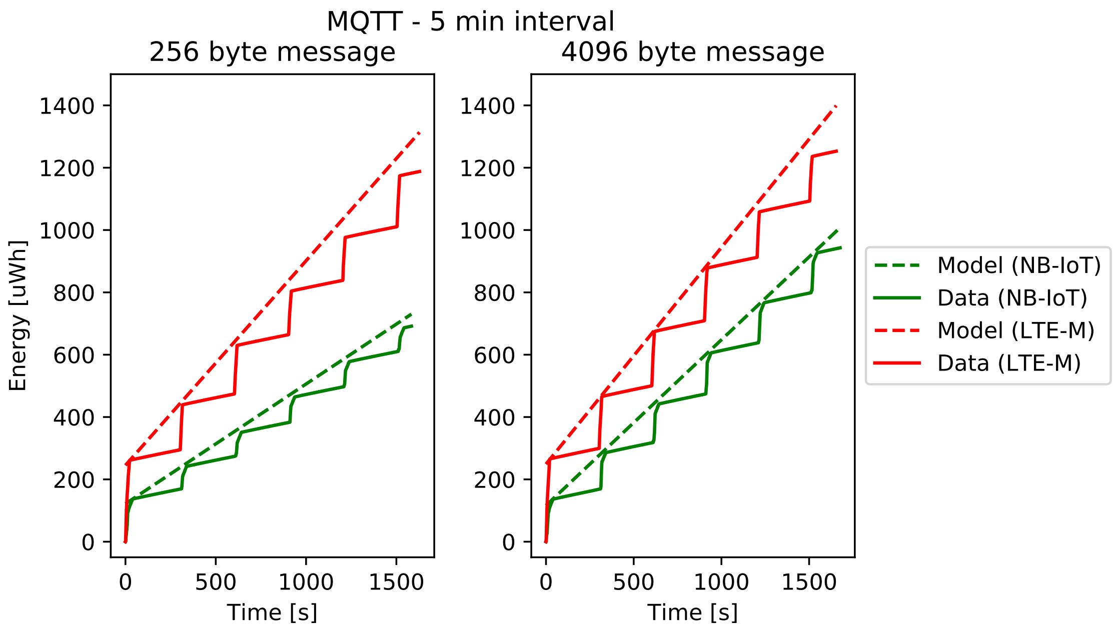
Table 4.6: Prediction of total energy for large test. [μW h]

CoAP over NB-IoT	$\frac{46.773\mu\text{W h}+29.874\mu\text{W h}+108\text{A}*3.7\text{V}(300\text{s}-7.559\text{s})}{300\text{s}}t + E_{start}$	$0.360t + 84.378$
CoAP over LTE-M	$\frac{36.281\mu\text{W h}+102.298\mu\text{W h}+108\text{A}*3.7\text{V}(300\text{s}-2.168\text{s})}{300\text{s}}t + E_{start}$	$0.577t + 189.604$
MQTT over NB-IoT	$\frac{96.180\mu\text{W h}+29.874\mu\text{W h}+117\text{A}*3.7\text{V}(300\text{s}-10.191\text{s})}{300\text{s}}t + E_{start}$	$0.532t + 116.007$
MQTT over LTE-M	$\frac{69.426\mu\text{W h}+102.298\mu\text{W h}+117\text{A}*3.7\text{V}(300\text{s}-4.568\text{s})}{300\text{s}}t + E_{start}$	$0.695t + 248.850$

The predicted energy consumption along with the measured total energy is plotted in figure 4.6.As expected the linear model is not be able to predict the staircase-like behavior of the real data, though the plots show that the model manages, with variable success, to predict the current total energy after a transmission have occurred.



(a) Plot depicting predicted energy versus actual energy for the CoAP application.



(b) Plot depicting predicted energy versus actual energy for the MQTT application.

Figure 4.6: Results of model testing.

The error of the model has been calculated based on the difference between the predicted energy and the actual energy after transmissions. See figure 4.7 for a graphical representation. As the slope of the predictions is constant, any error will most likely increase with time, which can be seen when inspecting the results in figure 4.7. This behavior is especially present for MQTT over LTE-M, where the overestimation shows an increasing difference between the predicted and the real value. Similar behavior is seen for neither MQTT over NB-IoT or CoAP over LTE-M. As a matter of fact, the other predictions seem to stabilize over time, though more data is needed to determine the actual accuracy.

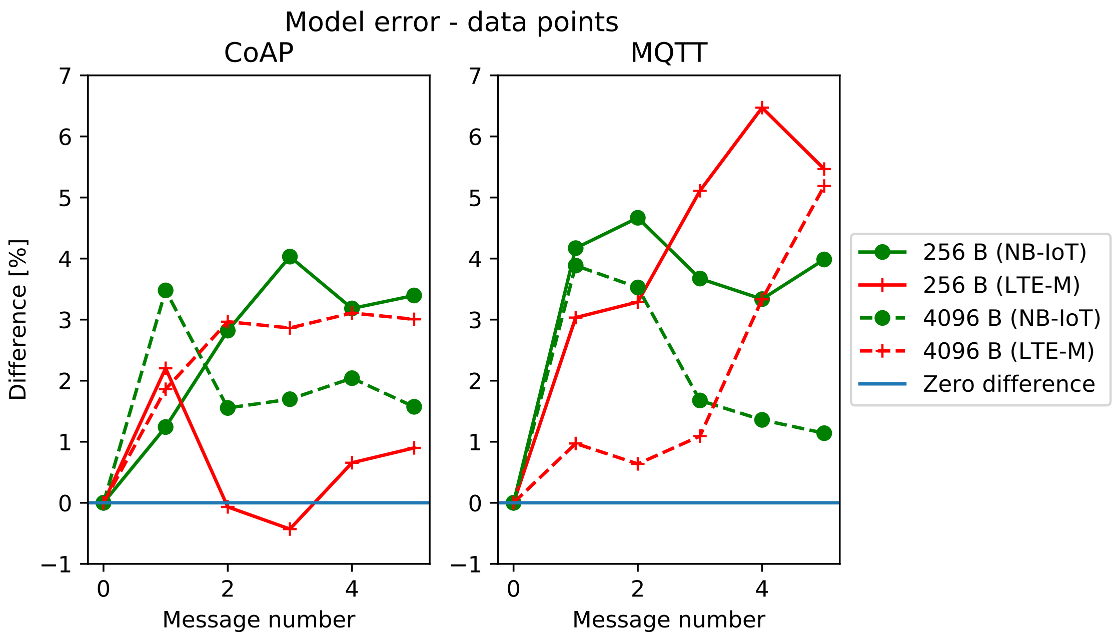


Figure 4.7: Results of model testing.

We can also assess the relative change or slope between each data point. This is depicted in figure 4.8 by plotting the difference in change between each point for the model and the real data. These results confirm that the slope of the prediction in most cases is steeper than that of the measured data.

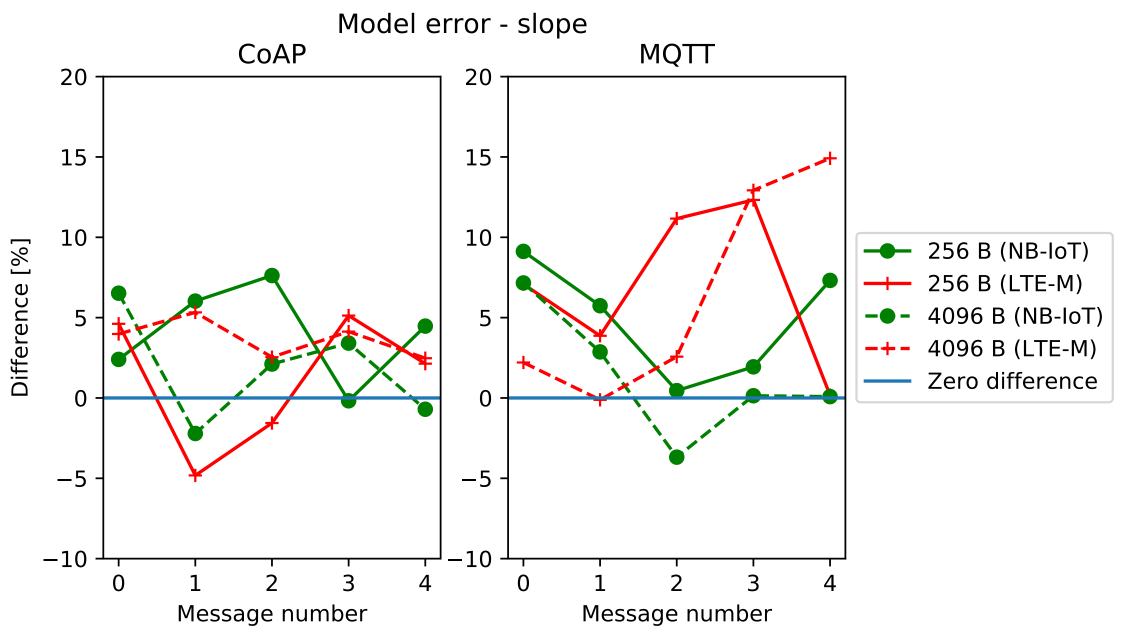
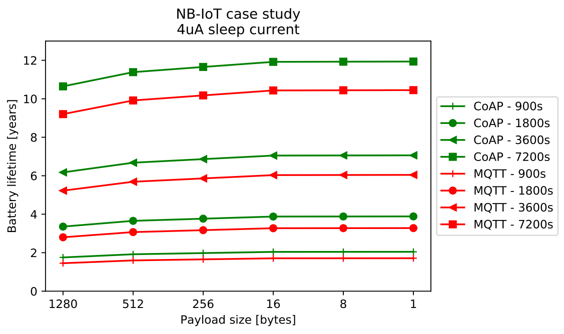


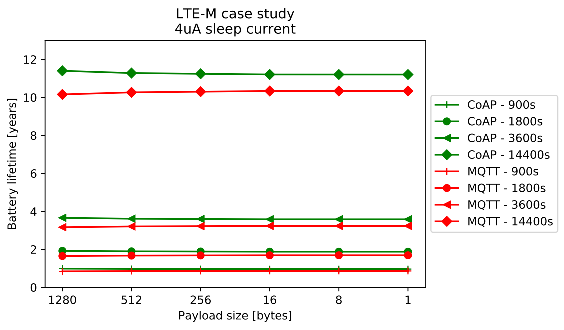
Figure 4.8: Plot depicting the difference between the total energy increase between transmissions for real and predicted total energy.

4.5 Case Study

One of the main goals of producing this model is for a developer to utilize it to predict the energy consumption of an application. This section will cover a small case study where the model is used to estimate an application's battery life based on payload size and transmission interval. Nordic Semiconductor boasts a minimum sleep current during PSM of 4μA [47], and we use this value in the following case. In a real application, other factors would most likely lead to higher base current. A battery with a capacity of 5Wh is used for the calculations, similar to the specifications from Third Generation Partnership Project (3GPP) in 2.1. Figure 4.9 shows the resulting predictions.



(a) NB-IoT predictions.



(b) LTE-M predictions.

Figure 4.9: Estimated life of a 5Wh battery for an application transmitting at various intervals using MQTT and CoAP over NB-IoT and LTE-M. Note that the maximum interval is different for LTE-M and NB-IoT.

Table 4.7: Battery life difference between 1280 byte and 1 byte transmissions

	NB-IoT		LTE-M	
Transmission interval[s]	MQTT [%]	CoAP [%]	MQTT [%]	CoAP [%]
900	17.77	16.62	2.38	-2.35
1800	17.01	15.78	2.33	-2.29
3600	15.68	14.34	2.23	-2.19
NB-IoT: 7200 | LTE-M: 14400	13.56	12.12	1.78	-1.71

These results show that 10-year battery life can be ensured for NB-IoT for a transmission interval of 2 hours. Similar battery life for LTE-M requires a four-hour transmission interval. Furthermore, it is worth noting payload size has an impact on the battery life for applications using NB-IoT. Though, as can be seen in table 4.7, the difference decreases with transmission interval. The results from the same table show that the payload size has an insignificant effect on the batty life of LTE-M applications.

Chapter 5

Discussion

This report's main goal is to deduce a model for predicting the total energy consumption of the communication protocols TCP and UDP over the cellular standards NB-IoT and LTE-M. Furthermore, we wish to assess the usefulness of the communication protocols and cellular standards with respect to each other. The CoAP and MQTT protocols were chosen as they use UDP and TCP respectively. It was hypothesized that the energy and time required for transmitting a single message over LTE-M and NB-IoT would increase with payload size. The previous chapter covered the different experiments performed and the results of these. This chapter will assess the results.

5.1 UDP and TCP over cellular networks

In the section 1.2 of the introduction, we stated three research questions. The first of which asked, "How UDP and TCP perform on cellular networks?". TCP performed noticeably worse than UDP throughout all experiments both in terms of energy and transmission time, which one could expect comparing to related research on the matter. Two papers were discussed in section 2.4.3 with [29] comparing MQTT and CoAP, showing the same behavior as the results presented here. However, in [30] the TCP implementation (using MQTT) experienced a very high rate of packet loss. We had no means of measuring the packet loss rate, but any instability relating to such an issue was absent.

The MQTT application did surge in energy consumption after the payload size exceeded ~500 bytes, as seen in figure 4.3, suggesting that the segmentation scheme of TCP is used due to the default maximum segment size being 536 bytes. This behavior is not seen for the CoAP measurements, further strengthening the theory that it is TCP-segmentation. An important point that can we can draw from this behavior is that knowing what restrictions relating to payload size are in place for a particular protocol is imperative when developing a application for constrained devices.

Even though TCP did perform worse than UDP over the cellular standards, it should not be rejected for use in constrained environments. However, further innovation with the protocol is necessary as current research shows that UDP based communication excels. MQTT has shown many qualities that are very beneficial within the IoT industry and with introduction of the UDP based MQTT-SN [27] research effort should be made in comparing that with CoAP.

5.2 Payload size effects

The second research question asked, "Is there a relationship between the payload of a message and the consumed time and energy for transmission over NB-IoT and LTE-M?". The results of linear regression show a strong correlation between payload size and message energy for NB-IoT in the range of 0 to 4096 bytes. A similar, linear relationship was found in [58] though, the resulting slope of their energy prediction was steeper than the slope of our predictions for both CoAP and MQTT over NB-IoT. Still, it is clear that NB-IoT energy consumption can be modeled linearly, which is an essential premise for the third research question. We got more ambiguous results from the transmission time measurements. The linear regression scores are much lower, as seen in table 4.4, suggesting that the transmission time of NB-IoT might not be dependent on the payload size. This could be tied to NB-IoT's expected maximum latency of 10 seconds as specified in table 2.1, as that will induce a more asymptotic curve with increased payload. There are tendencies to this behavior in our results, though, we can not determine the actual behavior without doing more measurements. With the way the model (2.3) is defined, inaccuracy in the prediction of transmission time will affect the expected PSM period. This error will be negligible with long transmission intervals, which is commonplace for many constrained devices. Should the transmission intervals be shorter, it may lead to errors.

The less fortunate results for LTE-M clearly show that within the measured range of 0 to 4096 bytes, there is no linear relationship between payload size and neither transmission energy nor time. This is likely related to LTE-M being a standard that favors low latency and high data rate. As shown in table 2.1, the maximum size of a single LTE-M packet is 8188 octets (bytes). That is over double the maximum payload transmitted during our experiments suggesting that the standard uses more or less the same energy for messages with sizes smaller than the maximum packet size. However, we do not see this behavior when the payload size exceeds the 1600 octet maximum packet size of NB-IoT, indicating that there may be other causes of this behavior. The knowledge needed to determine why LTE-M behaves as it does may require an understanding of how the physical layer of the standard works, something which has not been covered in this thesis. In conclusion, LTE-M does not fit for modeling the total energy consumption, and as the results in figure 4.6b show, it leads to overestimation.

The behavior of LTE-M is still interesting from a developer's point of view, as the more or less constant energy required to transmit messages means that the energy per message at some point will be lower than that of NB-IoT. This does require the device to avoid the energy spent during cDRX through the use of Release Assist Indication (RAI). RAI is supported for both NB-IoT and LTE-M by Telenor, but only for NB-IoT on the nRF9160 making this difficult to confirm at this point in time.

When processing the data from the MQTT over LTE-M payload sweep measurements, one of the data sets was not possible to handle. This was due to unexpected behavior in the application, as it did not enter PSM mode between two successive transmissions. The RRC inactive timer was restarted later than expected due to a TX/RX initiation. As this only happened for the TCP enabled application, it is reasonable to believe that some traffic related to this protocol is to blame. We could not pinpoint the exact transaction, and more sophisticated monitoring of individual data transmission would be required to investigate this issue further. However, it could have been avoided with a longer duration between test transmissions.

5.3 modeling

We defined the third and final research question as "Can the total energy consumption when using either TCP or UDP over LTE-M and NB-IoT be correctly modeled assuming the relationship above?". Based on the results from the test described in section 4.4, predicting the total power consumption based on the empirical data gathered is possible. When basing a model on several parameters, including empirical data, there are many possible points of failure. Firstly the results are gathered on one type of device. In this case, the nRF9160 and does not necessarily apply to other devices. Secondly, the parameters measured in section 4.1 shows that LTE-M, in particular, is suspect to high variation in energy use during cDRX, something which is difficult to model without a more thorough understanding of the protocol. Thirdly, the LTE-M message energy prediction's inaccuracy likely contributed to overcompensation seen for MQTT in figure 4.6b. The predictions for CoAP (i.e. UDP) over LTE-M are less erroneous, though this might be deceiving given that the slope of the prediction for this case is negative, which in turn compensates eventual errors in cDRX approximation. The overall more accurate predictions for NB-IoT confirms that both the parameter measurements and linear regression models for NB-IoT are more viable for estimation.

The research in [16] and [34] does indeed show that modeling the power consumption of IoT devices is possible. However, both papers devise complicated, analytical models, while the research done in our project uses simple empirical data while still achieving good accuracy. However, the model is tied to specific configurations of MQTT and CoAP; thus, it would not necessarily be viable for applications using other QoS levels, for example. This could be mitigated by modeling on a per-byte level, ignoring the overlying protocol. One then would be able to predict power consumption based on the exact number of bytes transmitted. This is what has been done in the papers above. However, this would require even more complicated models when applied to applications using, for example, MQTT as handshaking and so on must be taken into consideration. Hence, modeling based on empirical data using the desired protocol can be much more efficient.

The estimations in figure 4.9 show that 10-year battery life is achievable for both NB-IoT and LTE-M for payloads smaller than 1280 bytes transmitted with 2 and 4-hour intervals respectively. Our model is based on empirical data, with measurements done on a specific device, so other estimations are likely to differ. However, our NB-IoT results are comparable to the 3GPP estimation results in table 2.7, though they estimate a drastically shorter lifetime with increased payload size. This difference could be related to assumptions and estimation setup. They do assume the same PSM current as us, 4μA, but the actual calculation is not presented, so it was challenging to compare the results correctly. Furthermore, we have not taken coupling loss into account in this report, so it is unclear which estimations from 3GPP's estimation that actually apply. Accounting for coupling loss in the model is reserved for further research.

5.4 Working with a cellular device

This section will make some remarks on the experience gained from working with novel cellular technology. When working with a cellular-connected device, the developer is at the mercy of the network provider. Ultimately the possibilities for using mechanisms like eDRX and PSM rely on what is possible with the respective provider. During the development and testing of the application discussed in this thesis, it has been discovered that there are many possible configurations and limitations, especially concerning connection configuration.

Chapter 6

Conclusion

In this paper, we have evaluated and compared the energy consumption spent to transmit messages using UDP and TCP over the cellular networks NB-IoT and LTE-M. Notably, we have used empirical data to estimate the total power consumption of a cellular IoT device, the nRF9160, using the TCP-based MQTT and UDP-based CoAP communication protocols. The model parameters are payload size and desired transmission interval, as well as network parameters for connected DRX.

CoAP was shown to perform better than MQTT from the standpoint of energy usage, though, both protocols were deemed functional. We used linear regression to analyze the measured energy usage for a range of payload sizes, under the assumption that there is a relationship between payload size and energy spent for transmissions over LTE-M and NB-IoT. Our findings show a clear linear dependency between payload size and energy for NB-IoT, which resulted in promising accuracy for both CoAP and MQTT over NB-IoT in the model. No clear relation was found for LTE-M, and energy usage per byte was more or less constant within the tested range of payload sizes. Though this finding resulted in less accurate estimations using the model, it also suggests that LTE-M is more energy efficient when payload size exceeds ~400 bytes over CoAP and ~950 bytes over MQTT given that Release Assist Indication is used. In an example of utilization, the model was applied to show that a cellular device can achieve 10 years of battery life under some constraints.

The high abstraction level of this model allows a developer without extensive knowledge of cellular and communication protocol behavior to estimate the power consumption of a device easily, but we suggest that modeling of LTE-M should be further investigated as the method used in our research was insufficient, and no other research has been found on the matter. Furthermore, the model only takes uplink application into account, though many devices today require regular updates with firmware and configuration. Extending the model with the predictions of downlink data will expand on their usefulness.

Appendices

Appendix A

Parameter measurements

Table A.1: NB-IoT cDRX parameter measurements

Measurement number	cDRX energy [μ Wh]	Paging energy [μ Wh]	Release energy [μ Wh]
1	28	2.66	2.38
2	29.2	2.69	2.36
3	29.2	2.36	2.32
4	27.7	2.32	1.99
5	28.7	2.68	2.1
6	28.8	2.69	2.12
7	27.9	2.81	1.99
8	29	2.61	1.83
9	28.8	2.69	2.69
10	28.9	2.68	2.3
11	27.9	2.38	2.45
12	27.9	2.39	1.9
13	28	2.68	2.26
14	27.5	2.67	2.3
15	28.8	2.16	1.9
16	27.9	2.35	1.94
17	29.1	2.71	2.53
18	28.9	2.68	1.75
19	28.3	2.8	2.38
20	29.1	2.59	1.92

Table A.2: LTE-M cDRX parameter measurements

Measurement number	cDRX energy [μ Wh]	Paging energy [μ Wh]	Release energy [μ Wh]
1	98.5	2.3	0.409
2	109	2.31	0.172
3	101	2.28	0.584
4	95.4	2.28	0.405
5	97	2.26	0.509
6	96.7	2.29	0.368
7	101	2.3	0.378
8	97.8	2.29	0.505
9	97.4	5.73	0.536
10	112	2.29	0.546
11	95.7	6.04	0.407
12	95.5	6.35	0.505
13	113	2.34	0.387
14	110	9.7	0.392
15	117	2.3	0.535
16	97.2	2.29	0.384
17	96.5	2.28	0.512
18	114	2.29	0.521
19	98	2.32	0.38
20	96.3	2.29	0.526

Appendix B

nRF9160 - CoAP application

Listing B.1: Main file of the CoAP application

```c
#include <zephyr.h>
#include <logging/log.h>

#include <dk_buttons_and_leds.h>
#include <lte_lc.h>

#include "coap.h"

LOG_MODULE_REGISTER(app_main, CONFIG_APP_LOG_LEVEL);

/* Application settings */
#define WITH_PSM
#define TEST_PERIODIC
//#define TEST_SWEEP

/* Test settings */
#define SAMPLE_INTERVAL      1 //Seconds
#define SWEEP_INTERVAL      40 //Seconds
#define SIZE_STEP           41 //1435 bytes is max, so we get as close to the
    limit as possible.
#define MAX_SAMPLES          5
#define PERIODIC_MSG_SIZE 1280 //Bytes

// The periodic TAU given by the network
static int actual_tau;

/* Indicates when the application is ready to transmit *
*  according to the periodic TAU                       */
static bool transmit = false;

// Size is restricted my MTU
#define TEST_DATA_SIZE 1439
```

```c

//1440 characters in an array for upload testing
u8_t * testData="yK3vQHgUQ1WBUNPGprMSh0o1ZOTpGzC788DMB0OoQytSTDmKo7zeybWdx1DGh3SX"
        "IpfkYHSkX3hQuEUdWC8jWBq6qRAzv4NB79aECZwwUsReylQcJOzZ4NW1rY3xbyya"
        "ep9DOEWnRsWkjILrSh4crLHlfvmqVLxRjA1dDvHx72JVD4rvhhLbcQ6Gi94lvVF7"
        "0KmnO4Lh7IRGUm37TVXzQXtRnb228WPngoCC5Ge4GZNmBRFhXWtgeuU9Vt2JJbID"
        "jvdEpZVL88RszUn7Ah4pnTC7rkHRft6apfuZCUqo0udcvNbEaUFncwjsU8zkw8j7"
        "mfiD7QxF2A9Kv7XTztxef2Ryj1MbWe0vDAPXUz3yb4AqfgcxPb3TCocDCgAd2F2S"
        "xlAZ69913oxzD0M24Sl0YIztsCnTuUrzrgIOrdXWvjOcEcuEJltiIZMygVx8gxwc"
        "pwY4YNybojiLfuRET4w91tbTgn33IvFcY8J7tu5Y8LZjk5ZfkekJg5zhZs6Bo2Jm"
        "N0mC7eCqYvSBGm4No2TPbLjYD2fB5ERubVuo2rGeZjbnWEx8jcP9jgq049pEjjS9"
        "MRXvJnDtpo8hIZcZpz1HKyXbOXbz60baSbpW5RHOhwg1TBh8wrBTOOORMMCBhl4O"
        "QApYjcf2w4ZlbyfWUjQY6gkGR21599Wb1IjraQL911QeFjiRFGtcDEpxo5GMWL1O"
        "ZKM4Gnkp4LP0A0yK9FlHeopsaCOBxOI0dTaq2gWDD8rRRCbYykck0J5IZfQnrBbv"
              "AH1MSzQuBq5BLjPC6KhWj519pymLg11fSvhgWlOnhfuSNlmqq9pysYmZIPUNKGOP"
        "9gfpEKm8tCuvpUWZvoFsrmxfYQNe9vUznG0PZMhHDSc5C6wDBpFqDBhHEHRdg0KR"
        "Df8CU5RsaaviBtI8yFb0plaRQjzTYg2xZcppX4NANeqB0udVdEdfhIxX6iVXcEb5"
        "lGY0a35dDRjCgL7ePgZn7oQbLuusUurDbprEu2msxDXz94KJPwhnMldretN5bgq7"
        "yK3vQHgUQ1WBUNPGprMSh0o1ZOTpGzC788DMB0OoQytSTDmKo7zeybWdx1DGh3SX"
        "IpfkYHSkX3hQuEUdWC8jWBq6qRAzv4NB79aECZwwUsReylQcJOzZ4NW1rY3xbyya"
        "ep9DOEWnRsWkjILrSh4crLHlfvmqVLxRjA1dDvHx72JVD4rvhhLbcQ6Gi94lvVF7"
        "0KmnO4Lh7IRGUm37TVXzQXtRnb228WPngoCC5Ge4GZNmBRFhXWtgeuU9Vt2JJbID"
        "jvdEpZVL88RszUn7Ah4pnTC7rkHRft6apfuZCUqo0udcvNbEaUFncwjsU8zkw8j7"
        "mfiD7QxF2A9Kv7XTztxef2Ryj1MbWe0vDAPXUz3yb4AqfgcxPb3TCocDCgAd2F2S"
        "xlAZ69913oxk5ZfkekJg5zhZs6Bo2Ja";





/* @brief Message post function derived from Telenor's code */
static int message_post(struct coap_resource *resource, struct coap_packet *request, struct sockaddr *addr, socklen_t addr_len) {
  coap_endpoint *coap = resource->user_data;

  u16_t payload_len;
  const u8_t *payload = coap_packet_get_payload(request, &payload_len);

  u8_t *buf = k_calloc(payload_len + 1, 1);
  memcpy(buf, payload, payload_len);
  LOG_INF("Received CoAP POST: %s", log_strdup(buf));
  k_free(buf);

  int err = coap_endpoint_respond(coap, request, COAP_RESPONSE_CODE_CREATED, NULL, 0, addr, addr_len);
  if (err != 0) {
    LOG_ERR("coap_endpoint_respond: %d", err);
  }

  return 0;
}

/* Structures for CoAP*/
static const char * const message_path[] = { "message", NULL };
static struct coap_resource resources[] = {
  {
    path: message_path,
    post: message_post,
  },
  {},
};
static coap_endpoint *coap;
struct sockaddr_in  remote_addr = {
    .sin_family = AF_INET,
    .sin_port = htons(5683),
};
static const char * const path[] = { "straight", "and", "narrow", NULL };

/* @brief Used to instantiate different variables needed for CoAP operation */
void init_endpoint(void) {
  struct sockaddr_in local_addr = {
    .sin_family = AF_INET,
    .sin_port = htons(5683),
  };
  coap = coap_endpoint_init((struct sockaddr *)&local_addr, sizeof(local_addr),
     resources);
  if (coap == NULL) {
    LOG_ERR("coap_endpoint_init");
    return;
  }

  resources[0].user_data = coap;

  // Telenor's IP address.
  net_addr_pton(AF_INET, "172.16.15.14", &remote_addr.sin_addr);
}


/**@brief Callback for button events from the DK buttons and LEDs library. */
static void button_handler(u32_t button_states, u32_t has_changed)
{
   static size_t test_index = 0;

  if (has_changed & button_states & DK_BTN1_MSK) {
    LOG_INF("DEV_DBG: button 1 pressed\n");
    LOG_INF("Current test_index: %d", test_index);
    if(test_index <= TEST_DATA_SIZE) {
        int ret = coap_endpoint_post(coap, (struct sockaddr *)&remote_addr,
    sizeof(remote_addr), path, testData, test_index);
        if (ret != COAP_RESPONSE_CODE_CREATED) {
          LOG_ERR("coap_endpoint_post: %d", ret);
          return;
        }
      test_index += 10;
    } else {
      test_index = 0;
    }
  }
  else if (has_changed & button_states & DK_BTN2_MSK) {
    //resetting the test
    test_index = 0;
  }
  return;
}

/**@brief Configures modem to provide LTE link. Blocks until link is
 * successfully established.
 */
static void modem_configure(void)
{
#if defined(CONFIG_LTE_LINK_CONTROL)
  if (IS_ENABLED(CONFIG_LTE_AUTO_INIT_AND_CONNECT)) {
    /* Do nothing, modem is already turned on
     * and connected.
     */
  } else {
    int err;

    LOG_INF("LTE Link Connecting ...\n");
    err = lte_lc_init_and_connect();
    __ASSERT(err == 0, "LTE link could not be established.");
    LOG_INF("LTE Link Connected!\n");
  }
#endif /* defined(CONFIG_LTE_LINK_CONTROL) */
}

/**@brief Initializes buttons and LEDs, using the DK buttons and LEDs
 * library.
 */
static void buttons_leds_init(void)
{
  int err;

  LOG_INF("DEV_DBG: Initalizing buttons and leds.\n");

  err = dk_buttons_init(button_handler);
  if (err) {
    LOG_ERR("Could not initialize buttons, err code: %d\n", err);
  }

  err = dk_leds_init();
  if (err) {
    LOG_ERR("Could not initialize leds, err code: %d\n", err);
  }

  err = dk_set_leds_state(DK_ALL_LEDS_MSK, DK_NO_LEDS_MSK);
  if (err) {
    LOG_ERR("Could not set leds state, err code: %d\n", err);
  }
}

void setup_psm(void)
{
  /*
  * GPRS Timer 3 value (octet 3)
  *
  * Bits 5 to 1 represent the binary coded timer value.
  *
  * Bits 6 to 8 defines the timer value unit for the GPRS timer as follows:
  * Bits
  * 8 7 6
  * 0 0 0 value is incremented in multiples of 10 minutes
  * 0 0 1 value is incremented in multiples of 1 hour
  * 0 1 0 value is incremented in multiples of 10 hours
  * 0 1 1 value is incremented in multiples of 2 seconds
  * 1 0 0 value is incremented in multiples of 30 seconds
  * 1 0 1 value is incremented in multiples of 1 minute
  * 1 1 0 value is incremented in multiples of 320 hours (NOTE 1)
  * 1 1 1 value indicates that the timer is deactivated (NOTE 2).
  */
  char psm_settings[] = CONFIG_LTE_PSM_REQ_RPTAU;
  printk("PSM bits: %c%c%c\n", psm_settings[0], psm_settings[1],
         psm_settings[2]);
  printk("PSM Interval: %c%c%c%c%c\n", psm_settings[3], psm_settings[4],
         psm_settings[5], psm_settings[6], psm_settings[7]);
  int err = lte_lc_psm_req(true);
  if (err < 0) {
    printk("Error setting PSM: %d Errno: %d\n", err, errno);
  }
}

/* Timer used during TEST_PERIODIC to initiate transmission at the designated
    interval */
void app_timer_handler(struct k_timer *dummy)
{
  static u32_t minutes;

  minutes++;
  /* This shall match the PSM interval*/
  if (minutes % actual_tau == 0) {
    transmit = true;
    LOG_INF("Ready for transmit");
  }
  LOG_INF("Elapsed time: %d\n", minutes);
}

K_TIMER_DEFINE(app_timer, app_timer_handler, NULL);

/* @brief initializes timer that triggers every minute */
void timer_init(void)
{
  k_timer_start(&app_timer, K_MINUTES(1), K_MINUTES(1));
}


#ifdef TEST_PERIODIC
static int run_periodic(void) {
  static u8_t sample_cnt = 1;
  static bool transmit_finished = false;

    if (transmit) {
      //Lighting LED2 to indicate that transmission is initiated
      dk_set_led(DK_LED2, 0);

      //Data upload
      int ret = coap_endpoint_post(coap, (struct sockaddr *)&remote_addr,
    sizeof(remote_addr), path, testData, PERIODIC_MSG_SIZE);
      if (ret != COAP_RESPONSE_CODE_CREATED) {
        LOG_ERR("coap_endpoint_post: %d", ret);
        return -1;
      }
      transmit_finished = true;
    }

    k_sleep(K_SECONDS(SAMPLE_INTERVAL));

    if(transmit && transmit_finished) {
      //Transmission phase over.
      dk_set_led(DK_LED2, 1);
      transmit = false;
      transmit_finished = false;

      if(sample_cnt >= MAX_SAMPLES) {
        //exit test
        return -1;
      }

      sample_cnt++;
    }

    return 0;
}

#endif

#ifdef TEST_SWEEP
static int run_size_sweep(void) {
  static size_t test_index = 0;

  LOG_INF("Current test_index: %d", test_index);
  int ret = coap_endpoint_post(coap, (struct sockaddr *)&remote_addr, sizeof(
    remote_addr), path, testData, test_index);
  if (ret != COAP_RESPONSE_CODE_CREATED) {
    LOG_ERR("coap_endpoint_post: %d", ret);
    return -1;
  }
  test_index += SIZE_STEP;

  if(test_index > TEST_DATA_SIZE) {
    //ending test
    return -1;
  }

  return 0;
}
#endif

void main() {
  LOG_INF("\nDT CoAP application example started\n");

  int err;

  buttons_leds_init();

  #ifdef WITH_PSM
  setup_psm();
  #else
  /* Force PSM off in case the modem uses old PSM settings */
  err = lte_lc_psm_req(false);
  #endif


  modem_configure();

  init_endpoint();

  #ifdef WITH_PSM

  // The network can provide other PSM values. So we fetch the actual values of
     the network
  int curr_active;
  lte_lc_psm_get(&actual_tau, &curr_active);
  LOG_INF("Reqested: TAU = %s | AT = %s", log_strdup(CONFIG_LTE_PSM_REQ_RPTAU),
     log_strdup(CONFIG_LTE_PSM_REQ_RAT));
  LOG_INF("Got: TAU = %d | AT = %d", actual_tau, curr_active);
  #endif

  #ifdef TEST_PERIODIC
  // Converting TAU to minutes
  actual_tau = actual_tau/60;
  timer_init();
  #endif

  #ifdef TEST_SWEEP
    k_sleep(K_SECONDS(SWEEP_INTERVAL));
  #endif

  //Lighting LED1 to indicate that the application is connected and entering
    main loop.
  dk_set_led(DK_LED1, 0);
  while(1) {
    #ifdef TEST_PERIODIC
      err = run_periodic();
      if (err < 0) {
        dk_set_led(DK_LED3, 0);
        LOG_ERR("Error or finished sweep");
        return;
      }
      k_sleep(K_SECONDS(SAMPLE_INTERVAL));
    #elif defined(TEST_SWEEP)
      dk_set_led(DK_LED2, 0);
      if (run_size_sweep() < 0) {
        dk_set_led(DK_LED3, 0);
        LOG_ERR("Error or finished sweep");
        return;
      }
      dk_set_led(DK_LED2, 1);
      k_sleep(SWEEP_INTERVAL);
    #else
      k_sleep(K_SECONDS(SAMPLE_INTERVAL));
    #endif
  }
}
```

Listing B.2: The CoAP header file from Telenor

```c
#pragma once

#include <net/coap.h>

// A coap_endpoint is a CoAP client and server.
typedef struct _coap_endpoint coap_endpoint;

// coap_endpoint_init creates a CoAP endpoint and starts listening for requests
    .
coap_endpoint *coap_endpoint_init(struct sockaddr *local_addr, socklen_t
    local_addr_len, struct coap_resource *resources);

// A coap_response_handler is called when a response is received or an error
    occurs.
// The response packet is freed after the handler returns, so the handler must
    copy any data it wants to retain.
// It is the responsibility of the response handler to acknowledge or reject
    the response.
// The error code may be:
//   -EAGAIN    if an acknowledgement was never received for a Confirmable
    message
//   -ECANCELED if the remote endpoint sent a Reset message
//   -ENOMEM    if the endpoint could not allocate memory
typedef void (*coap_response_handler)(void *handler_data, int err, struct
    coap_packet *response);

// coap_endpoint_post performs a POST request towards the given address with
    the given path and payload.
// It returns an enum coap_response_code or a negative error code.
int coap_endpoint_post(coap_endpoint *ep, struct sockaddr *addr, socklen_t
    addr_len, const char *const *path, u8_t *payload, int payload_len);

// coap_endpoint_post_async asynchronously performs a POST request towards the
    given address with the given path and payload.
int coap_endpoint_post_async(coap_endpoint *ep, struct sockaddr *addr,
    socklen_t addr_len, const char *const *path, u8_t *payload, int payload_len
    , coap_response_handler response_handler, void *response_handler_data);

// coap_endpoint_respond responds to a CoAP request.  The response is
    piggybacked on a acknowledgement if the request was confirmable.
// This function must only be called from a request handler, not a different
    thread.
int coap_endpoint_respond(coap_endpoint *ep, struct coap_packet *request, enum
    coap_response_code code, u8_t *payload, u16_t payload_len, struct sockaddr
    *addr, socklen_t addr_len);

// coap_endpoint_acknowledge send an acknowledgement for the given packet if it
     is confirmable, otherwise it does nothing.
int coap_endpoint_acknowledge(coap_endpoint *ep, struct coap_packet *packet,
    struct sockaddr *addr, socklen_t addr_len);

// coap_endpoint_reset sends a reset message for the given packet.
int coap_endpoint_reset(coap_endpoint *ep, struct coap_packet *packet, struct
    sockaddr *addr, socklen_t addr_len);
```

Listing B.3: Project configuration file for CoAP

```
# General config
CONFIG_ASSERT=n #Consumes a lot of power.
CONFIG_TEST_RANDOM_GENERATOR=n

# Logging
CONFIG_SERIAL=y #set to "n" for no logging. Saves power
CONFIG_LOG=y
CONFIG_LOG_STRDUP_MAX_STRING=64
CONFIG_APP_LOG_LEVEL_DBG=y
CONFIG_LTE_LINK_CONTROL_LOG_LEVEL_INF=y
CONFIG_COAP_LOG_LEVEL_DBG=y

# AT Host
CONFIG_UART_INTERRUPT_DRIVEN=y
CONFIG_AT_HOST_LIBRARY=y #set to "n" when serial is "n".

# Network
CONFIG_NETWORKING=y
CONFIG_NET_SOCKETS=y
CONFIG_NET_SOCKETS_OFFLOAD=y

# BSD library
CONFIG_BSD_LIBRARY=y

# LTE link control
CONFIG_LTE_LINK_CONTROL=y
CONFIG_LTE_AUTO_INIT_AND_CONNECT=n
CONFIG_LTE_NETWORK_MODE_NBIOT=y
CONFIG_LTE_PDP_CMD=y
CONFIG_LTE_PDP_CONTEXT="0,\"IP\",\"mda.ee\""

#PSM test
CONFIG_LTE_PSM_REQ_RPTAU="10100101" #5 min
CONFIG_LTE_PSM_REQ_RAT="00000000"   #0 min

CONFIG_LTE_EDRX_REQ=n
CONFIG_LTE_EDRX_REQ_VALUE="0000"
CONFIG_LTE_PTW_VALUE="0001"
#CONFIG_MODEM_INFO=n

# CoAP
CONFIG_COAP=y

# Heap and stacks
CONFIG_HEAP_MEM_POOL_SIZE=16384
CONFIG_MAIN_STACK_SIZE=8192
CONFIG_SYSTEM_WORKQUEUE_STACK_SIZE=2048
CONFIG_HW_STACK_PROTECTION=y

```

```
# Library for buttons and LEDs
CONFIG_DK_LIBRARY=y

# Disable native network stack to save some memory
CONFIG_NET_IPV4=n
CONFIG_NET_IPV6=n
CONFIG_NET_UDP=n
CONFIG_NET_TCP=n
CONFIG_NET_RX_STACK_SIZE=1024
CONFIG_NET_TX_STACK_SIZE=1024

# Uncomment the lines below to disable optimizations when debugging
CONFIG_NO_OPTIMIZATIONS=n
CONFIG_DEBUG=n
```

Appendix C

nRF9160 - CoAP application

Listing C.1: Main file of the MQTT application

```c
/*
 * Copyright (c) 2018 Nordic Semiconductor ASA
 *
 * SPDX-License-Identifier: LicenseRef-BSD-5-Clause-Nordic
 */

#include <zephyr.h>
#include <stdio.h>
#include <string.h>
#include <logging/log.h>

#include <net/mqtt.h>
#include <net/socket.h>
#include <lte_lc.h>

#include <dk_buttons_and_leds.h>

#include "mqtt_module.h"

LOG_MODULE_REGISTER(app_mqtt_main, CONFIG_APP_LOG_LEVEL);

/* Application settings */
#define WITH_PSM
#define TEST_PERIODIC
//#define TEST_SWEEP


/* Test settings */
#define SAMPLE_INTERVAL       1 //Seconds
#define SWEEP_INTERVAL       40 //Seconds
#define SIZE_STEP            64 //Bytes
#define MAX_SAMPLES           5
```

```
#define PERIODIC_MSG_SIZE 4096 //Bytes

// The periodic TAU given by the network
static int actual_tau = 10;

/* Indicates when the application is ready to transmit *
*  according to the periodic TAU               */
static bool transmit = false;

// Actual maximum payload size
#define TEST_DATA_SIZE 4096

//4096 characters in an array for upload testing
u8_t * testData = "
    yK3vQHgUQ1WBUNPGprMSh0o1ZOTpGzC788DMB0OoQytSTDmKo7zeybWdx1DGh3SX"
            "IpfkYHSkX3hQuEUdWC8jWBq6qRAzv4NB79aECZwwUsReylQcJOzZ4NW1rY3xbyya"
            "ep9DOEWnRsWkjILrSh4crLHlfvmqVLxRjA1dDvHx72JVD4rvhhLbcQ6Gi94lvVF7"
            "0KmnO4Lh7IRGUm37TVXzQXtRnb228WPngoCC5Ge4GZNmBRFhXWtgeuU9Vt2JJbID"
            "jvdEpZVL88RszUn7Ah4pnTC7rkHRft6apfuZCUqo0udcvNbEaUFncwjsU8zkw8j7"
            "mfiD7QxF2A9Kv7XTztxef2Ryj1MbWe0vDAPXUz3yb4AqfgcxPb3TCocDCgAd2F2S"
            "xlAZ69913oxzD0M24Sl0YIztsCnTuUrzrgIOrdXWvjOcEcuEJltiIZMygVx8gxwc"
            "pwY4YNybojiLfuRET4w91tbTgn33IvFcY8J7tu5Y8LZjk5ZfkekJg5zhZs6Bo2Jm"
            "N0mC7eCqYvSBGm4No2TPbLjYD2fB5ERubVuo2rGeZjbnWEx8jcP9jgq049pEjjS9"
            "MRXvJnDtpo8hIZcZpz1HKyXbOXbz60baSbpW5RHOhwg1TBh8wrBTOOORMMCBhl4O"
            "QApYjcf2w4ZlbyfWUjQY6gkGR21599Wb1IjraQL911QeFjiRFGtcDEpxo5GMWL1O"
            "ZKM4Gnkp4LP0A0yK9FlHeopsaCOBxOI0dTaq2gWDD8rRRCbYykck0J5IZfQnrBbv"
                  "
    AH1MSzQuBq5BLjPC6KhWj519pymLg11fSvhgWlOnhfuSNlmqq9pysYmZIPUNKGOP"
            "9gfpEKm8tCuvpUWZvoFsrmxfYQNe9vUznG0PZMhHDSc5C6wDBpFqDBhHEHRdg0KR"
            "Df8CU5RsaaviBtI8yFb0plaRQjzTYg2xZcppX4NANeqB0udVdEdfhIxX6iVXcEb5"
            "lGY0a35dDRjCgL7ePgZn7oQbLuusUurDbprEu2msxDXz94KJPwhnMldretN5bgq7"
            "yK3vQHgUQ1WBUNPGprMSh0o1ZOTpGzC788DMB0OoQytSTDmKo7zeybWdx1DGh3SX"
            "IpfkYHSkX3hQuEUdWC8jWBq6qRAzv4NB79aECZwwUsReylQcJOzZ4NW1rY3xbyya"
            "ep9DOEWnRsWkjILrSh4crLHlfvmqVLxRjA1dDvHx72JVD4rvhhLbcQ6Gi94lvVF7"
            "0KmnO4Lh7IRGUm37TVXzQXtRnb228WPngoCC5Ge4GZNmBRFhXWtgeuU9Vt2JJbID"
            "jvdEpZVL88RszUn7Ah4pnTC7rkHRft6apfuZCUqo0udcvNbEaUFncwjsU8zkw8j7"
            "mfiD7QxF2A9Kv7XTztxef2Ryj1MbWe0vDAPXUz3yb4AqfgcxPb3TCocDCgAd2F2S"
            "xlAZ69913oxzD0M24Sl0YIztsCnTuUrzrgIOrdXWvjOcEcuEJltiIZMygVx8gxwc"
            "pwY4YNybojiLfuRET4w91tbTgn33IvFcY8J7tu5Y8LZjk5ZfkekJg5zhZs6Bo2Jm"
            "N0mC7eCqYvSBGm4No2TPbLjYD2fB5ERubVuo2rGeZjbnWEx8jcP9jgq049pEjjS9"
            "MRXvJnDtpo8hIZcZpz1HKyXbOXbz60baSbpW5RHOhwg1TBh8wrBTOOORMMCBhl4O"
            "QApYjcf2w4ZlbyfWUjQY6gkGR21599Wb1IjraQL911QeFjiRFGtcDEpxo5GMWL1O"
            "ZKM4Gnkp4LP0A0yK9FlHeopsaCOBxOI0dTaq2gWDD8rRRCbYykck0J5IZfQnrBbv"
                  "
    AH1MSzQuBq5BLjPC6KhWj519pymLg11fSvhgWlOnhfuSNlmqq9pysYmZIPUNKGOP"
            "9gfpEKm8tCuvpUWZvoFsrmxfYQNe9vUznG0PZMhHDSc5C6wDBpFqDBhHEHRdg0KR"
            "Df8CU5RsaaviBtI8yFb0plaRQjzTYg2xZcppX4NANeqB0udVdEdfhIxX6iVXcEb5"
            "lGY0a35dDRjCgL7ePgZn7oQbLuusUurDbprEu2msxDXz94KJPwhnMldretN5bgq7"
            "yK3vQHgUQ1WBUNPGprMSh0o1ZOTpGzC788DMB0OoQytSTDmKo7zeybWdx1DGh3SX"
            "IpfkYHSkX3hQuEUdWC8jWBq6qRAzv4NB79aECZwwUsReylQcJOzZ4NW1rY3xbyya"
```

```
      "ep9DOEWnRsWkjILrSh4crLHlfvmqVLxRjA1dDvHx72JVD4rvhhLbcQ6Gi94lvVF7"
      "0KmnO4Lh7IRGUm37TVXzQXtRnb228WPngoCC5Ge4GZNmBRFhXWtgeuU9Vt2JJbID"
      "jvdEpZVL88RszUn7Ah4pnTC7rkHRft6apfuZCUqo0udcvNbEaUFncwjsU8zkw8j7"
      "mfiD7QxF2A9Kv7XTztxef2Ryj1MbWe0vDAPXUz3yb4AqfgcxPb3TCocDCgAd2F2S"
      "xlAZ69913oxzD0M24Sl0YIztsCnTuUrzrgIOrdXWvjOcEcuEJltiIZMygVx8gxwc"
      "pwY4YNybojiLfuRET4w91tbTgn33IvFcY8J7tu5Y8LZjk5ZfkekJg5zhZs6Bo2Jm"
      "N0mC7eCqYvSBGm4No2TPbLjYD2fB5ERubVuo2rGeZjbnWEx8jcP9jgq049pEjjS9"
      "MRXvJnDtpo8hIZcZpz1HKyXbOXbz60baSbpW5RHOhwg1TBh8wrBTOOORMMCBhl4O"
      "QApYjcf2w4ZlbyfWUjQY6gkGR21599Wb1IjraQL911QeFjiRFGtcDEpxo5GMWL1O"
      "ZKM4Gnkp4LP0A0yK9FlHeopsaCOBxOI0dTaq2gWDD8rRRCbYykck0J5IZfQnrBbv"
          "
  AH1MSzQuBq5BLjPC6KhWj519pymLg11fSvhgWlOnhfuSNlmqq9pysYmZIPUNKGOP"
      "9gfpEKm8tCuvpUWZvoFsrmxfYQNe9vUznG0PZMhHDSc5C6wDBpFqDBhHEHRdg0KR"
      "Df8CU5RsaaviBtI8yFb0plaRQjzTYg2xZcppX4NANeqB0udVdEdfhIxX6iVXcEb5"
      "lGY0a35dDRjCgL7ePgZn7oQbLuusUurDbprEu2msxDXz94KJPwhnMldretN5bgq7"
      "yK3vQHgUQ1WBUNPGprMSh0o1ZOTpGzC788DMB0OoQytSTDmKo7zeybWdx1DGh3SX"
      "IpfkYHSkX3hQuEUdWC8jWBq6qRAzv4NB79aECZwwUsReylQcJOzZ4NW1rY3xbyya"
      "ep9DOEWnRsWkjILrSh4crLHlfvmqVLxRjA1dDvHx72JVD4rvhhLbcQ6Gi94lvVF7"
      "0KmnO4Lh7IRGUm37TVXzQXtRnb228WPngoCC5Ge4GZNmBRFhXWtgeuU9Vt2JJbID"
      "jvdEpZVL88RszUn7Ah4pnTC7rkHRft6apfuZCUqo0udcvNbEaUFncwjsU8zkw8j7"
      "mfiD7QxF2A9Kv7XTztxef2Ryj1MbWe0vDAPXUz3yb4AqfgcxPb3TCocDCgAd2F2S"
      "xlAZ69913oxzD0M24Sl0YIztsCnTuUrzrgIOrdXWvjOcEcuEJltiIZMygVx8gxwc"
      "pwY4YNybojiLfuRET4w91tbTgn33IvFcY8J7tu5Y8LZjk5ZfkekJg5zhZs6Bo2Jm"
      "N0mC7eCqYvSBGm4No2TPbLjYD2fB5ERubVuo2rGeZjbnWEx8jcP9jgq049pEjjS9"
      "MRXvJnDtpo8hIZcZpz1HKyXbOXbz60baSbpW5RHOhwg1TBh8wrBTOOORMMCBhl4O"
      "QApYjcf2w4ZlbyfWUjQY6gkGR21599Wb1IjraQL911QeFjiRFGtcDEpxo5GMWL1O"
      "ZKM4Gnkp4LP0A0yK9FlHeopsaCOBxOI0dTaq2gWDD8rRRCbYykck0J5IZfQnrBbv"
          "
  AH1MSzQuBq5BLjPC6KhWj519pymLg11fSvhgWlOnhfuSNlmqq9pysYmZIPUNKGOP"
      "9gfpEKm8tCuvpUWZvoFsrmxfYQNe9vUznG0PZMhHDSc5C6wDBpFqDBhHEHRdg0KR"
      "Df8CU5RsaaviBtI8yFb0plaRQjzTYg2xZcppX4NANeqB0udVdEdfhIxX6iVXcEb5"
      "lGY0a35dDRjCgL7ePgZn7oQbLuusUurDbprEu2msxDXz94KJPwhnMldretN5bgq";




#if defined(CONFIG_BSD_LIBRARY)
/**@brief Recoverable BSD library error. */
void bsd_recoverable_error_handler(uint32_t err)
{
  LOG_ERR("bsdlib recoverable error: %u\n", (unsigned int)err);
}

#endif /* defined(CONFIG_BSD_LIBRARY) */

/**@brief Configures modem to provide LTE link. Blocks until link is
 * successfully established.
 */
static void modem_configure(void)
{
```

```c
#if defined(CONFIG_LTE_LINK_CONTROL)
  if (IS_ENABLED(CONFIG_LTE_AUTO_INIT_AND_CONNECT)) {
    /* Do nothing, modem is already turned on
     * and connected.
     */
  } else {
    int err;

    LOG_INF("LTE Link Connecting ...\n");
    err = lte_lc_init_and_connect();
    __ASSERT(err == 0, "LTE link could not be established.");
    LOG_INF("LTE Link Connected!\n");
  }
#endif /* defined(CONFIG_LTE_LINK_CONTROL) */
}

/* @brief returns a random "sample"*/
static u8_t sensor_data_get() {
  u8_t random_sample;

  random_sample = sys_rand32_get() % 255;

  return random_sample;
}

/**@brief Callback for button events from the DK buttons and LEDs library. */
static void button_handler(u32_t button_states, u32_t has_changed)
{
  u8_t sample = 0;
  int err;

  if (has_changed & button_states & DK_BTN1_MSK) {
    LOG_INF("DEV_DBG: button 1 pressed\n");

    // alarm inducer
    sample = sensor_data_get();
    err = mqtt_data_publish(&sample,1);

    if (err < 0) {
      LOG_ERR("MQTT_PUBLISH ret %d", err);
      return;
    }
  }
  else if (has_changed & button_states & DK_BTN2_MSK) {

  }

  return;
}

```

```
/**@brief Initializes buttons and LEDs, using the DK buttons and LEDs
 * library.
 */
static void buttons_leds_init(void)
{
  int err;

  LOG_INF("DEV_DBG: Initalizing buttons and leds.\n");

  err = dk_buttons_init(button_handler);
  if (err) {
    LOG_ERR("Could not initialize buttons, err code: %d\n", err);
  }

  err = dk_leds_init();
  if (err) {
    LOG_ERR("Could not initialize leds, err code: %d\n", err);
  }

  err = dk_set_leds_state(DK_ALL_LEDS_MSK, DK_NO_LEDS_MSK);
  if (err) {
    LOG_ERR("Could not set leds state, err code: %d\n", err);
  }
}

/* Requests the configured PSM values from the network */
void setup_psm(void)
{
  /*
  * GPRS Timer 3 value (octet 3)
  *
  * Bits 5 to 1 represent the binary coded timer value.
  *
  * Bits 6 to 8 defines the timer value unit for the GPRS timer as follows:
  * Bits
  * 8 7 6
  * 0 0 0 value is incremented in multiples of 10 minutes
  * 0 0 1 value is incremented in multiples of 1 hour
  * 0 1 0 value is incremented in multiples of 10 hours
  * 0 1 1 value is incremented in multiples of 2 seconds
  * 1 0 0 value is incremented in multiples of 30 seconds
  * 1 0 1 value is incremented in multiples of 1 minute
  * 1 1 0 value is incremented in multiples of 320 hours (NOTE 1)
  * 1 1 1 value indicates that the timer is deactivated (NOTE 2).
  */
  char psm_settings[] = CONFIG_LTE_PSM_REQ_RPTAU;
  printk("PSM bits: %c%c%c\n", psm_settings[0], psm_settings[1],
         psm_settings[2]);
  printk("PSM Interval: %c%c%c%c%c\n", psm_settings[3], psm_settings[4],
         psm_settings[5], psm_settings[6], psm_settings[7]);
  int err = lte_lc_psm_req(true);
  if (err < 0) {
    printk("Error setting PSM: %d Errno: %d\n", err, errno);
  }
}

/* @brief triggers every minute. Set transmit true if one periodic tau period
    has passed */
void app_timer_handler(struct k_timer *dummy)
{
  static u32_t minutes;

  minutes++;
  /* This shall match the PSM interval*/
  if (minutes % actual_tau == 0) {
    LOG_INF("Awake - transmit true");
    transmit = true;
  }
  LOG_INF("Elapsed time: %d\n", minutes);
}

K_TIMER_DEFINE(app_timer, app_timer_handler, NULL);

/* @brief initializes timer that triggers every minute */
void timer_init(void)
{
  k_timer_start(&app_timer, K_MINUTES(1), K_MINUTES(1));
}


#ifdef TEST_PERIODIC
static int run_periodic(void) {
  static bool transmit_finished = false;
  static u8_t sample_cnt = 1;

    if (transmit) {
      //Lighting LED2 to indicate that transmission is initiated
      dk_set_led(DK_LED2, 0);

      //Data upload
      int err = mqtt_data_publish(testData,PERIODIC_MSG_SIZE);
      if (err < 0) {
        LOG_ERR("MQTT publish error: %d", err);
        return err;
      }

      transmit_finished = true;
    }

    k_sleep(K_SECONDS(SAMPLE_INTERVAL));

    if(transmit && transmit_finished) {
      //Transmission phase over.
      dk_set_led(DK_LED2, 1);
      transmit = false;
      transmit_finished = false;


      if(sample_cnt >= MAX_SAMPLES) {
        //exit test
        return -1;
      }

      sample_cnt++;
    }

    return 0;
}
#endif

#ifdef TEST_SWEEP
static int run_size_sweep(void) {
  static size_t test_index = 0;

  //Lighting LED2 to indicate that transmission is initiated

  LOG_INF("Current test_index: %d", test_index);
  if(test_index < TEST_DATA_SIZE) {
    LOG_INF("Transmit");
    int err = mqtt_data_publish(testData,test_index);
    if (err < 0) {
        LOG_ERR("MQTT_PUBLISH ret %d", err);
        return err;
    }
    test_index += SIZE_STEP;
  } else {
    //Ending test
    test_index = 0;
    return -1;
  }

  return 0;
}
#endif

void main(void)
{
  LOG_INF("\nDT MQTT application example started\n");

  int err;
```

```

buttons_leds_init();

#ifdef WITH_PSM
setup_psm();
#else
/* Force PSM off in case the modem uses old PSM settings */
err = lte_lc_psm_req(false);
if (err) {
      LOG_ERR("ERROR: set psm %d\n", err);
      return;
  }
#endif

modem_configure();

#ifdef WITH_PSM
// The network can provide other PSM values. So we fetch the actual values of
   the network
int curr_active;
err = lte_lc_psm_get(&actual_tau, &curr_active);
LOG_INF("Reqested: TAU = %s | AT = %s", log_strdup(CONFIG_LTE_PSM_REQ_RPTAU),
   log_strdup(CONFIG_LTE_PSM_REQ_RAT));
LOG_INF("Got: TAU = %d | AT = %d", actual_tau, curr_active);
#endif

/* Initialize MQTT connection */
mqtt_start_thread();
while(!mqtt_connected()) {
  k_sleep(100);
}

#ifdef TEST_PERIODIC
// Converting TAU to minutes
actual_tau = actual_tau/60;
timer_init();
#endif

if(err) {
  LOG_ERR("Initialization error");
  return;
}

#ifdef TEST_SWEEP
k_sleep(K_SECONDS(SWEEP_INTERVAL));
#endif

//Lighting LED1 to indicate that the application is connected and enterin
  main loop.
dk_set_led(DK_LED1, 0);
```

```

  while(1) {
    #ifdef TEST_PERIODIC
      err = run_periodic();
      if (err < 0) {
        dk_set_led(DK_LED3, 0);
        LOG_ERR("Error or finished periodic");
        return;
      }
    #elif defined(TEST_SWEEP)
      dk_set_led(DK_LED2, 0);
      if (run_size_sweep() < 0) {
        dk_set_led(DK_LED3, 0);
        LOG_ERR("Error or finished sweep");
        return;
      }
      dk_set_led(DK_LED2, 1);
      k_sleep(K_SECONDS(SWEEP_INTERVAL));
    #else
      k_sleep(K_SECONDS(SAMPLE_INTERVAL));
    #endif
  }
}
```

Listing C.2: The MQTT header file from Telenor

```
/*
- file: mqtt_module.h
- desc: contains the mqtt thread and the means to start it
*/

#ifndef _MQTT_MODULE_H_
#define _MQTT_MODULE_H_

// Initiates MQTT connection and thread
void mqtt_start_thread();

// Publish data to topic specified in prj.conf
int mqtt_data_publish(u8_t *data, size_t len);

// Returns connection status of MQTT
int mqtt_connected(void);

#endif
```

Listing C.3: Project configuration file for MQTT

```
#
# Copyright (c) 2019 Nordic Semiconductor ASA
#
# SPDX-License-Identifier: LicenseRef-BSD-5-Clause-Nordic
#

# General config
CONFIG_TEST_RANDOM_GENERATOR=n
CONFIG_BSD_LIBRARY_TRACE_ENABLED=n
CONFIG_NET_SOCKETS_POSIX_NAMES=y

# Logging
CONFIG_SERIAL=n #set to "n" for no logging. Saves power
CONFIG_LOG=y
CONFIG_APP_LOG_LEVEL_DBG=y
CONFIG_LTE_LINK_CONTROL_LOG_LEVEL_DBG=y


# AT Host
CONFIG_UART_INTERRUPT_DRIVEN=y
CONFIG_AT_HOST_LIBRARY=n       #set to "n" when serial is "n".


# Network
CONFIG_NETWORKING=y
CONFIG_NET_SOCKETS=y
CONFIG_NET_SOCKETS_OFFLOAD=y

# BSD library
CONFIG_BSD_LIBRARY=y

# LTE link control
CONFIG_LTE_LINK_CONTROL=y
CONFIG_LTE_AUTO_INIT_AND_CONNECT=n
CONFIG_LTE_NETWORK_MODE_NBIOT=y

#PSM test
CONFIG_LTE_PSM_REQ_RPTAU="10100101" #5 min
CONFIG_LTE_PSM_REQ_RAT="00000000"   #0 min

CONFIG_LTE_EDRX_REQ=n
CONFIG_LTE_EDRX_REQ_VALUE="0010"
CONFIG_LTE_PTW_VALUE="0000"
CONFIG_MODEM_INFO=n

# MQTT
CONFIG_MQTT_LIB=y
CONFIG_MQTT_LIB_TLS=n

```

```
# Library for buttons and LEDs
CONFIG_DK_LIBRARY=y

# Application
CONFIG_MQTT_PUB_TOPIC="/nrf91/publish/touch"
CONFIG_MQTT_SUB_TOPIC="/nrf91/subscribe/topic1"
CONFIG_MQTT_CLIENT_ID="DT_nRF91_2020_local"
CONFIG_MQTT_BROKER_HOSTNAME="84.210.212.77"
CONFIG_MQTT_BROKER_PORT=1883
CONFIG_MQTT_KEEPALIVE=7200

CONFIG_APP_MQTT_PASSWORD="dt_spring2020"
CONFIG_APP_MQTT_USERNAME="nrf9160"

# Main thread
CONFIG_MAIN_THREAD_PRIORITY=7
CONFIG_MAIN_STACK_SIZE=8192
CONFIG_HEAP_MEM_POOL_SIZE=16384

# Disable native network stack to save some memory
CONFIG_NET_IPV4=n
CONFIG_NET_IPV6=n
CONFIG_NET_UDP=n
CONFIG_NET_TCP=n
```

Appendix D

Processing notebook

processing_sweep

June 29, 2020

```
[ ]: #### Sweep processing code ####
     # This file contains the code handling the measurements from the payload sweep␣
      ↪experiments.
```

```
[ ]: import pandas as pd
     import numpy as np
     import scipy as sp
     import matplotlib.pyplot as plt

     from scipy import signal as sig
     from scipy import stats as stat
     from sklearn import preprocessing
     from sklearn.linear_model import LinearRegression
     from sklearn.model_selection import train_test_split
     from numpy import fft
     import importlib

     import processing_helpers as ph
     from processing_helpers import do_linear_regression
     from processing_helpers import get_residuals
     from processing_helpers import segment_data
     from processing_helpers import plot_pred_interval
```

```
[ ]: ## For reloading processing_helpers when changes are made
     importlib.reload(ph)
```

```
[ ]: #initial constants
     delta_t = 0.00025 #s
     V = 3.7 #V

     to_uWh = (1000*1000*delta_t*V)/3600 #uWh

     nb_active_energy = 28.48 #uWh Based on 20 measurements from test data
     nb_active_time = 20.48 #s

     ltem_active_energy = 101.95 #uWh Based on 20 measurements from test data
     ltem_active_time = 10.24 #s
```

[ ]:
```
## CoAP measurment data
coap_byte_step = 41

sweep_file1 = "../measurements/otii/sweep_coap_psm_test01.csv"
sweep_file2 = "../measurements/otii/sweep_coap_psm_test02.csv"
sweep_file3 = "../measurements/otii/sweep_coap_psm_test03.csv"


coap_nb_data = []
coap_nb_data.append(pd.read_csv(sweep_file1))
coap_nb_data.append(pd.read_csv(sweep_file2))
coap_nb_data.append(pd.read_csv(sweep_file3))


sweep_file1 = "../measurements/otii/sweep_ltem_coap_psm_test01.csv"
sweep_file2 = "../measurements/otii/sweep_ltem_coap_psm_test02.csv"
sweep_file3 = "../measurements/otii/sweep_ltem_coap_psm_test03.csv"


coap_ltem_data = []
coap_ltem_data.append(pd.read_csv(sweep_file1))
coap_ltem_data.append(pd.read_csv(sweep_file2))
coap_ltem_data.append(pd.read_csv(sweep_file3))
```

[ ]:
```
## MQTT measurement data
mqtt_byte_step = 64

sweep_file1 = "../measurements/otii/sweep_mqtt_psm_test01.csv"
sweep_file2 = "../measurements/otii/sweep_mqtt_psm_test02.csv"
sweep_file3 = "../measurements/otii/sweep_mqtt_psm_test03.csv"


mqtt_data = []
mqtt_data.append(pd.read_csv(sweep_file1))
mqtt_data.append(pd.read_csv(sweep_file2))
mqtt_data.append(pd.read_csv(sweep_file3))

sweep_file1 = "../measurements/otii/sweep_ltem_mqtt_psm_test01.csv"
sweep_file2 = "../measurements/otii/sweep_ltem_mqtt_psm_test02.csv"
sweep_file3 = "../measurements/otii/sweep_ltem_mqtt_psm_test03.csv"

mqtt_ltem_data = []
mqtt_ltem_data.append(pd.read_csv(sweep_file1))
mqtt_ltem_data.append(pd.read_csv(sweep_file2))
#mqtt_ltem_data.append(pd.read_csv(sweep_file3)) #major outlier that affects␣
↪segmenting
```

[ ]:
```
## Getting the Energy and Current data from the measurements

coap_nb_energy = []
coap_nb_current = []
for i in coap_nb_data:
    coap_nb_energy.append(i["Arc Main Energy (J)"])
    coap_nb_current.append(i["Arc Main Current (A)"])

mqtt_nb_energy = []
mqtt_nb_current = []
for i in mqtt_data:
    mqtt_nb_energy.append(i["Arc Main Energy (J)"])
    mqtt_nb_current.append(i["Arc Main Current (A)"])

coap_ltem_energy = []
coap_ltem_current = []
for i in coap_ltem_data:
    coap_ltem_energy.append(i["Arc Main Energy (J)"])
    coap_ltem_current.append(i["Arc Main Current (A)"])

mqtt_ltem_energy = []
mqtt_ltem_current = []
for i in mqtt_ltem_data:
    mqtt_ltem_energy.append(i["Arc Main Energy (J)"])
    mqtt_ltem_current.append(i["Arc Main Current (A)"])
```

[ ]:
```
## Plotting total energy

plt.figure()
for i in coap_nb_energy:
    i.plot()

for i in coap_ltem_energy:
    i.plot()

for i in mqtt_nb_energy:
    i.plot()

for i in mqtt_ltem_energy:
    i.plot()
```

[ ]:
```
#### CoAP over NB-IoT - START   ####
```

[ ]:
```
## Segmenting the CoAP over NB-IoT measurements
[segments_coap_nb, timing_coap_nb] = segment_data(coap_nb_current, 160000, 0.
↪01, 120000, 20000)
```

[ ]:
```
## Calculating the energy and time spent in each segment of each data set

fig = plt.figure()
ax = fig.add_subplot(111)

idx_seg = 0
coap_nb_sums = []

bytes_coap = range(0,coap_byte_step*len(segments_coap_nb[0]), coap_byte_step)

for i in segments_coap_nb:
    sum_acc = []
    idx_sample = 0
    for j in i:
        curr_sum = j.sum()*to_uWh - nb_active_energy
        sum_acc.append(curr_sum)
        idx_sample += 1

    print(np.size(sum_acc))
    plt.plot(bytes_coap,sum_acc, label = str(idx_seg))
    coap_nb_sums.append(sum_acc)
    idx_seg += 1


plt.grid()

plt.ylabel("Power consumption [uWh]")
plt.xlabel("Payload size [Bytes]")
```

[ ]:
```
## Calulating the average time and energy spent for each payload step
coap_nb_stds = []
coap_nb_avg = []

timing_coap_nb_avg = []

for i in range(len(coap_nb_sums[0])):
    curr_sample = []
    curr_time = []
    for j in range(len(coap_nb_sums)):
        curr_sample.append(coap_nb_sums[j][i])
        curr_time.append(timing_coap_nb[j][i])
    coap_nb_avg.append(np.mean(curr_sample))
    coap_nb_stds.append(np.std(curr_sample))

    timing_coap_nb_avg.append(np.mean(curr_time))

print(np.mean(coap_nb_stds))
print(coap_nb_avg[1])
```

```
[ ]: #### CoAP over NB-IoT - END   ####


     #### CoAP over LTE-M  - START ####
```

```
[ ]: ## Segmenting the CoAP over LTE-M measurements
     [segments_coap_ltem, timing_coap_ltem] = segment_data(coap_ltem_current,␣
      ↪100000, 0.01, 60000, 20000)
```

```
[ ]: ## Calculating the energy and time spent in each segment of each data set
     fig = plt.figure()
     ax = fig.add_subplot(111)

     idx_seg = 0

     coap_ltem_sums = []

     for i in segments_coap_ltem:
         sum_acc = []
         idx_sample = 0
         for j in i:
             curr_sum = j.sum()*to_uWh - ltem_active_energy
             sum_acc.append(curr_sum)
             idx_sample += 1

         print(np.size(sum_acc))
         plt.plot(bytes_coap,sum_acc, label = str(idx_seg))
         coap_ltem_sums.append(sum_acc)
         idx_seg += 1

     plt.grid()

     plt.ylabel("Power consumption [uWh]")
     plt.xlabel("Payload size [Bytes]")
```

```
[ ]: ## Calulating the average time and energy spent for each payload step
     coap_ltem_stds = []
     coap_ltem_avg = []

     timing_coap_ltem_avg = []

     for i in range(len(coap_ltem_sums[0])):
         curr_sample = []
         curr_time = []
         for j in range(len(coap_ltem_sums)):
```

```
        curr_sample.append(coap_ltem_sums[j][i])
        curr_time.append(timing_coap_ltem[j][i])
    coap_ltem_avg.append(np.mean(curr_sample))
    coap_ltem_stds.append(np.std(curr_sample))

    timing_coap_ltem_avg.append(np.mean(curr_time))


print(np.mean(coap_ltem_stds))
print(coap_ltem_avg[1])
```

[ ]:
```
#### COAP over LTE-M  -  END  ####

#### MQTT over NB-IoT - START ####
```

[ ]:
```
## Segmenting the MQTT over NB-IoT measurements
[segments_mqtt_nb, timing_mqtt_nb] = segment_data(mqtt_nb_current, 180000, 0.
 ↪01, 140000, 20000)
```

[ ]:
```
## Calculating the energy and time spent in each segment of each data set
fig = plt.figure()
ax = fig.add_subplot(111)

idx_seg = 0

mqtt_nb_sums = []
bytes_mqtt = range(0,mqtt_byte_step*len(segments_mqtt_nb[0]), mqtt_byte_step)

for i in segments_mqtt_nb:
    sum_acc = []
    idx_sample = 0
    for j in i:
        curr_sum = j.sum()*to_uWh - nb_active_energy
        sum_acc.append(curr_sum)
        idx_sample += 1

    print(np.size(sum_acc))
    plt.plot(bytes_mqtt,sum_acc, label = str(idx_seg))
    mqtt_nb_sums.append(sum_acc)
    idx_seg += 1

plt.grid()

plt.ylabel("Power consumption [uWh]")
plt.xlabel("Payload size [Bytes]")
```

[ ]:
```
## Calulating the average time and energy spent for each payload step
mqtt_nb_stds = []
mqtt_nb_avg = []

timing_mqtt_nb_avg = []

for i in range(len(mqtt_nb_sums[0])):
    curr_sample = []
    curr_time = []
    for j in range(len(mqtt_nb_sums)):
        curr_sample.append(mqtt_nb_sums[j][i])
        curr_time.append(timing_mqtt_nb[j][i])
    mqtt_nb_avg.append(np.mean(curr_sample))
    mqtt_nb_stds.append(np.std(curr_sample))

    timing_mqtt_nb_avg.append(np.mean(curr_time))


print(np.mean(mqtt_nb_stds))
print(mqtt_nb_avg[1])
print(len(mqtt_nb_avg))
```

[ ]:
```
#### MQTT over NB-IoT -  END  ####

#### MQTT over LTE-M  - START ####
```

[ ]:
```
## Segmenting the MQTT over LTE-M measurements
[segments_mqtt_ltem, timing_mqtt_ltem] = segment_data(mqtt_ltem_current,␣
 ↪120000, 0.01, 100000, 20000)
```

[ ]:
```
## Calculating the energy and time spent in each segment of each data set
fig = plt.figure()
ax = fig.add_subplot(111)

idx_seg = 0

mqtt_ltem_sums = []
bytes_mqtt = range(0,mqtt_byte_step*len(segments_mqtt_ltem[0]), mqtt_byte_step)

for i in segments_mqtt_ltem:
    sum_acc = []
    idx_sample = 0
    for j in i:
        curr_sum = j.sum()*to_uWh - ltem_active_energy
        sum_acc.append(curr_sum)
        #mqtt_ltem_sums[idx_sample][idx_seg] = curr_sum
        idx_sample += 1
    print(np.size(sum_acc))
    plt.plot(bytes_mqtt,sum_acc, label = str(idx_seg))
    mqtt_ltem_sums.append(sum_acc)
    idx_seg += 1

plt.grid()

plt.ylabel("Energy consumption [uWh]")
plt.xlabel("Payload size [Bytes]")
```

```
[ ]: ## Calulating the average time and energy spent for each payload step

mqtt_ltem_stds = []
mqtt_ltem_avg = []

timing_mqtt_ltem_avg = []

for i in range(len(mqtt_ltem_sums[0])):
    curr_sample = []
    curr_time = []
    for j in range(len(mqtt_ltem_sums)):
        curr_sample.append(mqtt_ltem_sums[j][i])
        curr_time.append(timing_mqtt_ltem[j][i])

    mqtt_ltem_avg.append(np.mean(curr_sample))
    mqtt_ltem_stds.append(np.std(curr_sample))

    timing_mqtt_ltem_avg.append(np.mean(curr_time))

print(np.mean(mqtt_ltem_stds))
print(mqtt_ltem_avg[1])
print(len(mqtt_ltem_avg))
```

```
[ ]: ## Smoothing the average measurements for pretty plotting
#  Applying a Savitzky-Golay filter:
#  - window length   : 9
#  - polynomial order: 3

coap_ltem_smooth = sig.savgol_filter(coap_ltem_avg,9,3)
coap_nb_smooth   = sig.savgol_filter(coap_nb_avg,9,3)
mqtt_nb_smooth   = sig.savgol_filter(mqtt_nb_avg,9,3)
mqtt_ltem_smooth = sig.savgol_filter(mqtt_ltem_avg,9,3)

timing_coap_ltem_smooth = sig.savgol_filter(timing_coap_ltem_avg,9,3)
timing_coap_nb_smooth   = sig.savgol_filter(timing_coap_nb_avg,9,3)
timing_mqtt_nb_smooth   = sig.savgol_filter(timing_mqtt_nb_avg,9,3)
```

```
timing_mqtt_ltem_smooth = sig.savgol_filter(timing_mqtt_ltem_avg,9,3)
```

[ ]:
```
## Plotting the smoothed energy averages
fig, (ax1, ax2) = plt.subplots(1,2)

plt.suptitle("Message energy comparison")


ax1.set_title('NB-IoT')
ax2.set_title('LTE-M')

ax1.set(ylabel="Energy [uWh]")
ax1.set_xlabel("Payload size [Bytes]")
ax2.set_xlabel("Payload size [Bytes]")
ax1.set_ylim([20,200])
ax2.set_ylim([20,200])

ax1.axvline(536)
ax2.axvline(536, label = "536 Bytes (TCP default MSS)")


#plt.plot(bytes_coap, coap_nb_avg   ,     label = "CoAP NB-IoT average")
ax1.plot(bytes_coap, coap_nb_smooth, "g" ,label = "NB-IoT")
ax1.plot(bytes_coap, coap_nb_smooth+nb_active_energy, "g--" ,label = "NB-IoT")


#plt.plot(bytes_mqtt, mqtt_nb_avg   ,       label = "MQTT 2300 average")
ax1.plot(bytes_mqtt, mqtt_nb_smooth, "r" ,label = "NB-IoT")
ax1.plot(bytes_mqtt, mqtt_nb_smooth+nb_active_energy, "r--" ,label = "NB-IoT")


#plt.plot(bytes_coap, coap_ltem_avg   ,     label = "CoAP LTE-M average")
ax2.plot(bytes_coap, coap_ltem_smooth, "g", label = "CoAP")
ax2.plot(bytes_coap, coap_ltem_smooth+ltem_active_energy, "g--", label = "CoAP␣
 ↪with cDRX energy")

#plt.plot(bytes_mqtt, mqtt_ltem_avg   ,       label = "MQTT 2300 average")
ax2.plot(bytes_mqtt, mqtt_ltem_smooth, "r" ,label = "MQTT")
ax2.plot(bytes_mqtt, mqtt_ltem_smooth+ltem_active_energy, "r--" ,label = "MQTT␣
 ↪with cDRX energy")

ax2.legend(loc='center left', bbox_to_anchor=(1, 0.5))

plt.savefig("../dt_thesis/plots/tx_energy_comparison.pdf", bbox_inches='tight')
```

[ ]:
```
## Plotting the smoothed time averages
fig, (ax1, ax2) = plt.subplots(1,2)

ax1.set_title('NB-IoT')
ax2.set_title('LTE-M')

plt.suptitle("Transmission time comparison")

ax1.set(ylabel="Transmission time [s]")
ax1.set_xlabel("Payload size [Bytes]")
ax2.set_xlabel("Payload size [Bytes]")
ax1.set_ylim([0,35])
ax2.set_ylim([0,35])

#plt.plot(bytes_coap, coap_nb_avg   ,      label = "CoAP NB-IoT average")
ax1.plot(bytes_coap, timing_coap_nb_smooth-20.48, "g" ,label = "NB-IoT")
ax1.plot(bytes_coap, timing_coap_nb_smooth, "g--" ,label = "NB-IoT")


#plt.plot(bytes_mqtt, mqtt_nb_avg   ,        label = "MQTT 2300 average")
ax1.plot(bytes_mqtt, timing_mqtt_nb_smooth-20.48, "r" ,label = "NB-IoT")
ax1.plot(bytes_mqtt, timing_mqtt_nb_smooth, "r--" ,label = "NB-IoT")


#plt.plot(bytes_coap, coap_ltem_avg   ,      label = "CoAP LTE-M average")
ax2.plot(bytes_coap, timing_coap_ltem_smooth-10.24, "g", label = "CoAP")
ax2.plot(bytes_coap, timing_coap_ltem_smooth, "g--", label = "CoAP with RRC␣
 ↪inactive time")


#plt.plot(bytes_mqtt, mqtt_ltem_avg   ,        label = "MQTT 2300 average")
ax2.plot(bytes_mqtt, timing_mqtt_ltem_smooth-10.24, "r" ,label = "MQTT")
ax2.plot(bytes_mqtt, timing_mqtt_ltem_smooth, "r--" ,label = "MQTT with RRC␣
 ↪inactive time")

ax2.legend(loc='center left', bbox_to_anchor=(1, 0.5))

plt.savefig("../dt_thesis/plots/tx_time_comparison.pdf", bbox_inches='tight')
```

[ ]:
```
## OLS linear regression analysis of the calculated message energy
class Results:
    def __init__(self, results):
        self.intercept = results.params[0]
        self.slope     = results.params[1]
        self.results   = results
```

```
timing_coap_nb_res = Results(do_linear_regression(bytes_coap, timing_coap_nb))
timing_mqtt_nb_res = Results(do_linear_regression(bytes_mqtt, timing_mqtt_nb))
timing_coap_ltem_res = Results(do_linear_regression(bytes_coap,␣
 ↪timing_coap_ltem))
timing_mqtt_ltem_res = Results(do_linear_regression(bytes_mqtt,␣
 ↪timing_mqtt_ltem))
energy_coap_nb_res = Results(do_linear_regression(bytes_coap, coap_nb_sums))
energy_mqtt_nb_res = Results(do_linear_regression(bytes_mqtt, mqtt_nb_sums))
energy_coap_ltem_res = Results(do_linear_regression(bytes_coap, coap_ltem_sums))
energy_mqtt_ltem_res = Results(do_linear_regression(bytes_mqtt, mqtt_ltem_sums))
```

```
[ ]: ## Writing the results to file in order for later use (in the model)

     result_matrix = {
         'timing_coap_nb'   : [  timing_coap_nb_res.slope   , timing_coap_nb_res.
      ↪intercept],
         'energy_coap_nb'   : [  energy_coap_nb_res.slope   , energy_coap_nb_res.
      ↪intercept],
         'timing_coap_ltem' : [timing_coap_ltem_res.slope, timing_coap_ltem_res.
      ↪intercept],
         'energy_coap_ltem' : [energy_coap_ltem_res.slope, energy_coap_ltem_res.
      ↪intercept],
         'timing_mqtt_nb'   : [  timing_mqtt_nb_res.slope   , timing_mqtt_nb_res.
      ↪intercept],
         'energy_mqtt_nb'   : [  energy_mqtt_nb_res.slope   , energy_mqtt_nb_res.
      ↪intercept],
         'timing_mqtt_ltem' : [timing_mqtt_ltem_res.slope, timing_mqtt_ltem_res.
      ↪intercept],
         'energy_mqtt_ltem' : [energy_mqtt_ltem_res.slope, energy_mqtt_ltem_res.
      ↪intercept]
     }

     regression_results = pd.DataFrame(data = result_matrix)
     regression_results.to_csv("reg_res.csv")
```

```
[ ]: ## Regression line plot for both NB-IoT and LTE-M data.

     fig, (ax1,ax2) = plt.subplots(1,2)

     fig.suptitle("Message energy regression lines")
     x_mqtt = np.linspace(0,4096,4096)
     x_coap = np.linspace(0,1439,1439)

     ## CoAP
     ax1.set_title("CoAP")
     ax1.set_ylabel("Energy [uWh]")
```

```
ax1.set_xlabel("Payload size [Bytes]")
ax1.set_ylim([0,170])

y_coap_nb =   energy_coap_nb_res.slope*x_coap+energy_coap_nb_res.intercept
y_coap_ltem = energy_coap_ltem_res.slope*x_coap+energy_coap_ltem_res.intercept

ax1.plot(x_coap,y_coap_nb,    "g", label="NB-IoT regression line")
ax1.scatter(bytes_coap, coap_nb_sums[0], color = "g", s=5, alpha=0.3,␣
 ↪label="NB-IoT data")
ax1.scatter(bytes_coap, coap_nb_sums[1], color = "g", s=5, alpha=0.3)
ax1.scatter(bytes_coap, coap_nb_sums[2], color = "g", s=5, alpha=0.3)

ax1.plot(x_coap,y_coap_ltem,  "r", label="LTE-M regression line")
ax1.scatter(bytes_coap, coap_ltem_sums[0], color = "r", s=5, alpha=0.3,␣
 ↪label="LTE-M data")
ax1.scatter(bytes_coap, coap_ltem_sums[1], color = "r", s=5, alpha=0.3)
ax1.scatter(bytes_coap, coap_ltem_sums[2], color = "r", s=5, alpha=0.3)

## MQTT
ax2.set_title("MQTT")
ax2.set_xlabel("Payload size [Bytes]")
ax2.set_ylim([0,170])


y_mqtt_nb   = energy_mqtt_nb_res.slope  * x_mqtt+energy_mqtt_nb_res.intercept
y_mqtt_ltem = energy_mqtt_ltem_res.slope* x_mqtt+energy_mqtt_ltem_res.intercept

ax2.plot(x_mqtt,y_mqtt_nb,    "g", label="NB-IoT regression line")
ax2.scatter(bytes_mqtt, mqtt_nb_sums[0], color = "g", s=5, alpha=0.3,␣
 ↪label="NB-IoT data")
ax2.scatter(bytes_mqtt, mqtt_nb_sums[1], color = "g", s=5, alpha=0.3)
ax2.scatter(bytes_mqtt, mqtt_nb_sums[2], color = "g", s=5, alpha=0.3)

ax2.plot(x_mqtt,y_mqtt_ltem,  "r", label="LTE-M regression line")
ax2.scatter(bytes_mqtt, mqtt_ltem_sums[0], color = "r", s=5, alpha=0.3,␣
 ↪label="LTE-M data")
ax2.scatter(bytes_mqtt, mqtt_ltem_sums[1], color = "r", s=5, alpha=0.3)
#ax2.scatter(bytes_mqtt, mqtt_ltem_sums[2], color = "r", s=5, alpha=0.3)




ax2.legend(loc='center left', bbox_to_anchor=(1, 0.5))

plt.savefig("../dt_thesis/plots/energy_reg_lines.pdf", bbox_inches='tight')

intersect_coap = (energy_coap_nb_res.intercept-energy_coap_ltem_res.intercept)/
 ↪(energy_coap_ltem_res.slope-energy_coap_nb_res.slope)
```

```
intersect_mqtt = (energy_mqtt_nb_res.intercept-energy_mqtt_ltem_res.intercept)/
 ↪(energy_mqtt_ltem_res.slope-energy_mqtt_nb_res.slope)


print("CoAP Energy consumption intersection at: " + str(intersect_coap))
print("MQTT Energy consumption intersection at: " + str(intersect_mqtt))
```

[ ]:
```
## Regression line plot for both NB-IoT and LTE-M data.
fig, (ax1,ax2) = plt.subplots(1,2)

fig.suptitle("Transmission time regression lines")
x_mqtt = np.linspace(0,4096,4096)
x_coap = np.linspace(0,1439,1439)

y_min_t = 0
y_max_t = 14

ax1.set_ylim([y_min_t,y_max_t])
ax2.set_ylim([y_min_t,y_max_t])


## CoAP
ax1.set_title("CoAP")
ax1.set_ylabel("Time [s]")
ax1.set_xlabel("Payload size [Bytes]")

y_coap_nb_t =   timing_coap_nb_res.slope  *x_coap+timing_coap_nb_res.intercept
y_coap_ltem_t = timing_coap_ltem_res.slope*x_coap+timing_coap_ltem_res.intercept

ax1.plot(x_coap,y_coap_nb_t-20.48, "g", label="NB-IoT regression line")
ax1.scatter(bytes_coap, np.array(timing_coap_nb[0])-20.48, color = "g", s=5,␣
 ↪alpha=0.3, label="NB-IoT data")
ax1.scatter(bytes_coap, np.array(timing_coap_nb[1])-20.48, color = "g", s=5,␣
 ↪alpha=0.3)
ax1.scatter(bytes_coap, np.array(timing_coap_nb[2])-20.48, color = "g", s=5,␣
 ↪alpha=0.3)

ax1.plot(x_coap,y_coap_ltem_t-10.24, "r", label="LTE-M regression line")␣
 ↪#subtracting the RRC Connected time of LTE-M. This is not consistent though
ax1.scatter(bytes_coap, np.array(timing_coap_ltem[0])-10.24, color = "r", s=5,␣
 ↪alpha=0.3, label="LTE-M data")
ax1.scatter(bytes_coap, np.array(timing_coap_ltem[1])-10.24, color = "r", s=5,␣
 ↪alpha=0.3)
ax1.scatter(bytes_coap, np.array(timing_coap_ltem[2])-10.24, color = "r", s=5,␣
 ↪alpha=0.3)
```

```
## MQTT
ax2.set_title("MQTT")
ax2.set_xlabel("Payload size [Bytes]")

y_mqtt_nb_t   =   timing_mqtt_nb_res.slope  *x_mqtt+timing_mqtt_nb_res.intercept
y_mqtt_ltem_t = timing_mqtt_ltem_res.slope*x_mqtt+timing_mqtt_ltem_res.intercept

ax2.plot(x_mqtt,y_mqtt_nb_t-20.48, "g", label="NB-IoT regression line")
ax2.scatter(bytes_mqtt, np.array(timing_mqtt_nb[0])-20.48, color = "g", s=5,␣
 ↪alpha=0.3, label="NB-IoT data")
ax2.scatter(bytes_mqtt, np.array(timing_mqtt_nb[1])-20.48, color = "g", s=5,␣
 ↪alpha=0.3)
ax2.scatter(bytes_mqtt, np.array(timing_mqtt_nb[2])-20.48, color = "g", s=5,␣
 ↪alpha=0.3)

ax2.plot(x_mqtt,y_mqtt_ltem_t-10.24, "r", label="LTE-M regression line")␣
 ↪#subtracting the RRC Connected time of LTE-M. This is not consistent though
ax2.scatter(bytes_mqtt, np.array(timing_mqtt_ltem[0])-10.24, color = "r", s=5,␣
 ↪alpha=0.3, label="LTE-M data")
ax2.scatter(bytes_mqtt, np.array(timing_mqtt_ltem[1])-10.24, color = "r", s=5,␣
 ↪alpha=0.3)
#ax2.scatter(bytes_mqtt, np.array(timing_mqtt_ltem[2])*delta_t-10.24, color =␣
 ↪"r", s=5, alpha=0.3)

ax2.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.savefig("plots/time_reg_lines.pdf", bbox_inches='tight')


intersect = (energy_mqtt_nb_res.intercept-energy_mqtt_ltem_res.intercept)/
 ↪(energy_mqtt_ltem_res.slope-energy_mqtt_nb_res.slope)
print("Energy consumption intersection at: " + str(intersect))
```

[ ]:
```
### The following cells plots the calculated transmission times and energy ###
```

[ ]:
```
plt.title("NB-IoT message energy")
plt.ylabel("Message energy [uWh]")
plt.xlabel("Payload size [Bytes]")

xMax = 4096
yMin = 30
yMax = 110

plt.scatter(bytes_coap, coap_nb_sums[0], color = "g", s=5)
plt.scatter(bytes_coap, coap_nb_sums[1], color = "g", s=5)
plt.scatter(bytes_coap, coap_nb_sums[2], color = "g", s=5, label = "CoAP")
plt.scatter(bytes_mqtt, mqtt_nb_sums[0], color = "r", s=5)
plt.scatter(bytes_mqtt, mqtt_nb_sums[1], color = "r", s=5)
plt.scatter(bytes_mqtt, mqtt_nb_sums[2], color = "r", s=5, label = "MQTT")

nb_diffs = []
for i in range(len(timing_coap_nb)):
    curr_coap = np.array(coap_nb_sums[i])
    curr_mqtt = np.array(mqtt_nb_sums[i][:len(curr_coap)])

    nb_diffs.append(100*(1-curr_coap/curr_mqtt))

print(np.mean(nb_diffs))
print(np.std(nb_diffs))

plt.legend()

plt.savefig("../dt_thesis/plots/energy_nbiot.pdf", bbox_inches='tight')
```

```
[ ]: plt.title("LTE-M message energy")
plt.ylabel("Message energy [uWh]")
plt.xlabel("Payload size [Bytes]")

plt.scatter(bytes_coap, coap_ltem_sums[0], color = "g", s=5)
plt.scatter(bytes_coap, coap_ltem_sums[1], color = "g", s=5)
plt.scatter(bytes_coap, coap_ltem_sums[2], color = "g", s=5, label = "CoAP")

plt.scatter(bytes_mqtt, mqtt_ltem_sums[0], color = "r", s=5)
plt.scatter(bytes_mqtt, mqtt_ltem_sums[1], color = "r", s=5, label = "MQTT")
#plt.scatter(bytes_mqtt, mqtt_ltem_sums[2], color = "g", s=5)

plt.savefig("../dt_thesis/plots/energy_ltem.pdf", bbox_inches='tight')
```

```
[ ]: plt.title("NB-IoT transmission time")
plt.ylabel("Transmission time [s]")
plt.xlabel("Payload size [Bytes]")

plt.scatter(bytes_coap, np.array(timing_coap_nb[0])-20.48, color = "g", s=5,␣
 ↪label="CoAP")
plt.scatter(bytes_coap, np.array(timing_coap_nb[1])-20.48, color = "g", s=5)
plt.scatter(bytes_coap, np.array(timing_coap_nb[2])-20.48, color = "g", s=5)

plt.scatter(bytes_mqtt, np.array(timing_mqtt_nb[0])-20.48, color = "r", s=5,␣
 ↪label="MQTT")
plt.scatter(bytes_mqtt, np.array(timing_mqtt_nb[1])-20.48, color = "r", s=5)
plt.scatter(bytes_mqtt, np.array(timing_mqtt_nb[2])-20.48, color = "r", s=5)
```

```
nb_diffs = []
for i in range(len(timing_coap_nb)):
    curr_coap = np.array(timing_coap_nb[i])
    curr_mqtt = np.array(timing_mqtt_nb[i][:len(curr_coap)])

    nb_diffs.append(100*(1-curr_coap/curr_mqtt))

print(np.mean(nb_diffs))
print(np.std(nb_diffs))
plt.savefig("../dt_thesis/plots/time_nbiot.pdf", bbox_inches='tight')
```

```
[ ]: plt.title("LTE-M transmission time")
plt.ylabel("Transmission time [s]")
plt.xlabel("Payload size [Bytes]")

plt.scatter(bytes_coap, np.array(timing_coap_ltem[0])-10.24, color = "g", s=5,␣
 ↪label="CoAP")
plt.scatter(bytes_coap, np.array(timing_coap_ltem[1])-10.24, color = "g", s=5)
plt.scatter(bytes_coap, np.array(timing_coap_ltem[2])-10.24, color = "g", s=5)

plt.scatter(bytes_mqtt, np.array(timing_mqtt_ltem[0])-10.24, color = "r", s=5,␣
 ↪label="MQTT")
plt.scatter(bytes_mqtt, np.array(timing_mqtt_ltem[1])-10.24, color = "r", s=5)


plt.savefig("../dt_thesis/plots/time_ltem.pdf", bbox_inches='tight')
```

```
[ ]: ### Following this cell is calculation and plotting of residuals ###
# This was not used directly in the thesis #
```

```
[ ]: residuals_coap_nb   = get_residuals(coap_nb_sums,   energy_coap_nb_res,   ␣
 ↪bytes_coap)
residuals_coap_ltem = get_residuals(coap_ltem_sums, energy_coap_ltem_res, ␣
 ↪bytes_coap)
residuals_mqtt_nb   = get_residuals(mqtt_nb_sums,   energy_mqtt_nb_res,   ␣
 ↪bytes_mqtt)
residuals_mqtt_ltem = get_residuals(mqtt_ltem_sums, energy_mqtt_ltem_res, ␣
 ↪bytes_mqtt)
```

```
[ ]: fig, ((ax1, ax2),(ax3,ax4)) = plt.subplots(2,2)

ax1.set_title("Coap NB-IoT residuals")
ax2.set_title("Coap LTE-M residuals")
ax3.set_title("MQTT NB-IoT residuals")
ax4.set_title("MQTT LTE-M residuals")
```

```
ax1.scatter(residuals_coap_nb[0],   residuals_coap_nb[1])
ax2.scatter(residuals_coap_ltem[0], residuals_coap_ltem[1])
ax3.scatter(residuals_mqtt_nb[0],   residuals_mqtt_nb[1])
ax4.scatter(residuals_mqtt_ltem[0], residuals_mqtt_ltem[1])
```

[ ]:
```
fig, (ax1, ax2) = plt.subplots(1,2)

fig.suptitle("Message energy regression residuals")

ax1.set(ylabel="Energy [uWh]")
y_min = -70
y_max = 110

ax1.set_ylim([y_min,y_max])
ax2.set_ylim([y_min,y_max])

ax1.set_title("CoAP")
ax1.set_xticks([1,2])
ax1.set_xticklabels(['NB-IoT', "LTE-M"])
ax2.set_title("MQTT")
ax2.set_xticks([1,2])
ax2.set_xticklabels(['NB-IoT', "LTE-M"])

ax1.violinplot(residuals_coap_nb[1], positions = [1])
ax1.violinplot(residuals_coap_ltem[1], positions = [2])

ax2.violinplot(residuals_mqtt_nb[1], positions = [1])
ax2.violinplot(residuals_mqtt_ltem[1], positions = [2])

plt.savefig("plots/energy_reg_res.pdf", bbox_inches='tight')
```

[ ]:
```
residuals_time_coap_nb   = get_residuals(timing_coap_nb,   timing_coap_nb_res, ␣
 ↪   bytes_coap)
residuals_time_coap_ltem = get_residuals(timing_coap_ltem,␣
 ↪timing_coap_ltem_res,   bytes_coap)
residuals_time_mqtt_nb   = get_residuals(timing_mqtt_nb,   timing_mqtt_nb_res, ␣
 ↪   bytes_mqtt)
residuals_time_mqtt_ltem = get_residuals(timing_mqtt_ltem,␣
 ↪timing_mqtt_ltem_res,   bytes_mqtt)
```

[ ]:
```
fig, ((ax1, ax2),(ax3,ax4)) = plt.subplots(2,2)

ax1.set_title("Coap NB-IoT time spent - residuals")
ax2.set_title("Coap LTE-M time spent - residuals")
ax3.set_title("MQTT NB-IoT time spent - residuals")
```

```
ax4.set_title("MQTT LTE-M time spent - residuals")

ax1.scatter(residuals_time_coap_nb[0]  , residuals_time_coap_nb[1])
ax2.scatter(residuals_time_coap_ltem[0], residuals_time_coap_ltem[1])
ax3.scatter(residuals_time_mqtt_nb[0]  , residuals_time_mqtt_nb[1]    )
ax4.scatter(residuals_time_mqtt_ltem[0], residuals_time_mqtt_ltem[1])
```

[ ]:
```
fig, (ax1, ax2) = plt.subplots(1,2)

fig.suptitle("Transmit time regression residuals")

ax1.set(ylabel="Time [s]")
y_min = -5
y_max = 11

ax1.set_ylim([y_min,y_max])
ax2.set_ylim([y_min,y_max])

ax1.set_title("CoAP")
ax1.set_xticks([1,2])
ax1.set_xticklabels(['NB-IoT', "LTE-M"])
ax2.set_title("MQTT")
ax2.set_xticks([1,2])
ax2.set_xticklabels(['NB-IoT', "LTE-M"])

ax1.violinplot(residuals_time_coap_nb[1]  , positions = [1])
ax1.violinplot(residuals_time_coap_ltem[1], positions = [2])

ax2.violinplot(residuals_time_mqtt_nb[1]  , positions = [1])
ax2.violinplot(residuals_time_mqtt_ltem[1], positions = [2])

plt.savefig("plots/time_reg_res.pdf", bbox_inches='tight')
```

Appendix E

Model notebook

processing_model

June 29, 2020

```
[92]: #### Model test code ####
      # This file contains the code handling the results of the model testing.
      # It calculates the model using functions from processing_helpers.py and
      # plots the model togheter with measured data.
      # Some error calculations as well as a case study is also performed.
```

```
[ ]: import pandas as pd
     import numpy as np
     import scipy as sp
     import matplotlib.pyplot as plt

     from scipy import signal

     from processing_helpers import get_energy
     from processing_helpers import get_con_energy
     from processing_helpers import get_residuals
```

```
[ ]: ### Constants ###

     V = 3.7             #V
     hour = 3600         #s
     delta_t = 0.00025   #s
     joule_to_uWh = 0.000277777778*1000*1000 #uWh
```

```
[ ]: ### Read OTII measurment data ###

     coap_256_nb    = pd.read_csv("../measurements/otii/model_test_coap_256_nb.csv" ␣
      ↪ , usecols = ["Arc Main Energy (J)"]).to_numpy()*joule_to_uWh
     coap_256_ltem  = pd.read_csv("../measurements/otii/model_test_coap_256_ltem.
      ↪csv" , usecols = ["Arc Main Energy (J)"]).to_numpy()*joule_to_uWh
     coap_1280_nb   = pd.read_csv("../measurements/otii/model_test_coap_1280_nb.csv"␣
      ↪ , usecols = ["Arc Main Energy (J)"]).to_numpy()*joule_to_uWh
     coap_1280_ltem = pd.read_csv("../measurements/otii/model_test_coap_1280_ltem.
      ↪csv", usecols = ["Arc Main Energy (J)"]).to_numpy()*joule_to_uWh

     mqtt_256_nb    = pd.read_csv("../measurements/otii/model_test_mqtt_256_nb.csv" ␣
      ↪ , usecols = ["Arc Main Energy (J)"]).to_numpy()*joule_to_uWh
```

```
mqtt_256_ltem  = pd.read_csv("../measurements/otii/model_test_mqtt_256_ltem.
↪csv" , usecols = ["Arc Main Energy (J)"]).to_numpy()*joule_to_uWh
mqtt_4096_nb   = pd.read_csv("../measurements/otii/model_test_mqtt_4096_nb.csv"␣
↪ , usecols = ["Arc Main Energy (J)"]).to_numpy()*joule_to_uWh
mqtt_4096_ltem = pd.read_csv("../measurements/otii/model_test_mqtt_4096_ltem.
↪csv", usecols = ["Arc Main Energy (J)"]).to_numpy()*joule_to_uWh
```

[ ]:
```
### Read the regression coefficients calculated in processing_sweep.ipynb ###

regression_results = pd.read_csv("reg_res.csv")

timing_coap_nb = regression_results["timing_coap_nb"].to_numpy()
energy_coap_nb = regression_results["energy_coap_nb"].to_numpy()
timing_coap_ltem = regression_results["timing_coap_ltem"].to_numpy()
energy_coap_ltem = regression_results["energy_coap_ltem"].to_numpy()
timing_mqtt_nb = regression_results["timing_mqtt_nb"].to_numpy()
energy_mqtt_nb = regression_results["energy_mqtt_nb"].to_numpy()
timing_mqtt_ltem = regression_results["timing_mqtt_ltem"].to_numpy()
energy_mqtt_ltem = regression_results["energy_mqtt_ltem"].to_numpy()
```

[ ]:
```
#device parameters
p_idle       = 157.7*V #uW
coap_p_sleep = 108*V   #uW
mqtt_p_sleep = 117*V   #uW
```

[ ]:
```
## Function for getting the start segment of the data ##
def get_data_start(data):

    diff_array = np.diff(data, axis=0)

    i = 160000
    while(diff_array[i] < 0.01):
        i -= 1
    return [i*delta_t, data[i][0]]
```

[ ]:
```
### Estimating the cDRX energy for NB-IoT and LTE-M ###

#NB-IoT
E_cdrx_nb = get_con_energy(
    t_inactive   = 20.48, #s
    t_cycle      = 2.048, #s
    t_onDuration = 0.200, #s
    E_monitor    = 2.580, #uWh
    E_release    = 2.171, #uWh
    p_idle  = p_idle)

#LTE-M
```

```
E_cdrx_ltem = get_con_energy(
    t_inactive   = 10.24, #s
    t_cycle      = 0.320, #s
    t_onDuration = 0.100, #s
    E_monitor    = 3.227, #uWh
    E_release    = 0.448, #uWh
    p_idle  = p_idle
)
```

[ ]:
```
### The different tests are defined and put in arrays for easy iteration ###

class Test:
    def __init__(self, data, T_msg, n_bytes, t, E, E_start, cdrx):
        self.data = data
        self.duration = np.linspace(0,len(data)*0.00025,len(data))
        self.T_msg = T_msg
        self.n_bytes = n_bytes
        self.t = t
        self.E = E
        self.E_start = E_start
        self.cdrx = cdrx

coap_test_array = []
coap_test_array.append(Test(coap_256_nb,    300,256, timing_coap_nb,  ␣
 ↪energy_coap_nb,   get_data_start(coap_256_nb  ), E_cdrx_nb))
coap_test_array.append(Test(coap_256_ltem,  300,256, timing_coap_ltem,␣
 ↪energy_coap_ltem, get_data_start(coap_256_ltem ), E_cdrx_ltem))
coap_test_array.append(Test(coap_1280_nb,   300,1280,timing_coap_nb,  ␣
 ↪energy_coap_nb,   get_data_start(coap_1280_nb ), E_cdrx_nb))
coap_test_array.append(Test(coap_1280_ltem, 300,1280,timing_coap_ltem,␣
 ↪energy_coap_ltem, get_data_start(coap_1280_ltem), E_cdrx_ltem))


mqtt_test_array = []
mqtt_test_array.append(Test(mqtt_256_nb,    300,256,  timing_mqtt_nb,  ␣
 ↪energy_mqtt_nb,   get_data_start(mqtt_256_nb), E_cdrx_nb))
mqtt_test_array.append(Test(mqtt_256_ltem,  300,256,  timing_mqtt_ltem,␣
 ↪energy_mqtt_ltem, get_data_start(mqtt_256_ltem), E_cdrx_ltem))
mqtt_test_array.append(Test(mqtt_4096_nb,   300,4096, timing_mqtt_nb,  ␣
 ↪energy_mqtt_nb,   get_data_start(mqtt_4096_nb), E_cdrx_nb))
mqtt_test_array.append(Test(mqtt_4096_ltem, 300,4096, timing_mqtt_ltem,␣
 ↪energy_mqtt_ltem, get_data_start(mqtt_4096_ltem), E_cdrx_ltem))
```

[ ]:
```
### Calculating the coefficients of estimated total energy consumption for all␣
 ↪the tests ###
```

```
coap_predictions = []

for i in coap_test_array:
    coap_predictions.append(
        get_energy(
            n_bytes = i.n_bytes,
            max_bytes = 1439,
            T_msg = i.T_msg,
            E_cdrx = i.cdrx,
            p_sleep = coap_p_sleep,
            reg_coeffs_t = i.t,
            reg_coeffs_e = i.E,
            start_params = i.E_start
        )
    )

mqtt_predictions = []
for i in mqtt_test_array:
    mqtt_predictions.append(
        get_energy(
            n_bytes = i.n_bytes,
            max_bytes = 4096,
            T_msg = i.T_msg,
            E_cdrx = i.cdrx,
            p_sleep = mqtt_p_sleep,
            reg_coeffs_t = i.t,
            reg_coeffs_e = i.E,
            start_params = i.E_start
        )
    )
```

[ ]:
```
### Plotting the CoAP model test results ###

fig, (ax1, ax2) = plt.subplots(1,2)
plt.subplots_adjust(wspace = 0.3)

fig.suptitle("CoAP - 5 min interval")

ax1.set_ylabel("Energy [uWh]")
ax1.set_xlabel("Time [s]")
ax2.set_xlabel("Time [s]")

ax1.set_title("256 byte message")
ax2.set_title("1280 byte message")

style = [["g--" , "g"],["r--","r"]]
```

```
labels = [["Model (NB-IoT)", "Data (NB-IoT)"], ["Model (LTE-M)", "Data␣
↪(LTE-M)"]]


for i in range(2):
    curr_durr = coap_test_array[i].duration
    ax1.plot(curr_durr, coap_predictions[i][0]*curr_durr +␣
↪coap_predictions[i][1], style[i][0])
    ax1.plot(curr_durr, coap_test_array[i].data, style[i][1])

for i in range(2,4):
    curr_durr = coap_test_array[i].duration
    ax2.plot(curr_durr, coap_predictions[i][0]*curr_durr +␣
↪coap_predictions[i][1], style[i-2][0], label = labels[i-2][0])
    ax2.plot(curr_durr, coap_test_array[i].data, style[i-2][1], label =␣
↪labels[i-2][1])

ax2.legend(loc='center left', bbox_to_anchor=(1, 0.5))

plt.savefig("../dt_thesis/plots/model_test_coap.pdf", bbox_inches='tight')
```

```
[ ]: ### Plotting the MQTT model test results ###

fig, (ax1, ax2) = plt.subplots(1,2)

plt.subplots_adjust(wspace = 0.3)

fig.suptitle("MQTT - 5 min interval")

y_min = -50
y_max = 1500

ax1.set_ylim([y_min,y_max])
ax2.set_ylim([y_min,y_max])

ax1.set_ylabel("Energy [uWh]")
ax1.set_xlabel("Time [s]")
ax2.set_xlabel("Time [s]")

ax1.set_title("256 byte message")
ax2.set_title("4096 byte message")

style = [["g--" , "g"],["r--","r"]]
labels = [["Model (NB-IoT)", "Data (NB-IoT)"], ["Model (LTE-M)", "Data␣
↪(LTE-M)"]]

for i in range(2):
    curr_dur = mqtt_test_array[i].duration
    ax1.plot(curr_dur, mqtt_predictions[i][0]*curr_dur +␣
↪mqtt_predictions[i][1], style[i][0])
    ax1.plot(curr_dur, mqtt_test_array[i].data, style[i][1])

for i in range(2,4):
    curr_dur = mqtt_test_array[i].duration
    ax2.plot(curr_dur, mqtt_predictions[i][0]*curr_dur +␣
↪mqtt_predictions[i][1], style[i-2][0], label = labels[i-2][0])
    ax2.plot(curr_dur, mqtt_test_array[i].data, style[i-2][1], label =␣
↪labels[i-2][1])

ax2.legend(loc='center left', bbox_to_anchor=(1, 0.5))

plt.savefig("../dt_thesis/plots/model_test_mqtt.pdf", bbox_inches='tight')
```

```
[ ]: ### Function for finding the point of transmission in the measured data ###

def get_msg_idxs(test_data):

    test_msg_idxs = []

    diff_array = np.diff(test_data, axis=0)
    i = len(diff_array)-1

    while(i > 0):
        curr_diff = diff_array[i]
        if (curr_diff > 0.010):
            test_msg_idxs.append(i)
            i-=960000
        else:
            i-=1
    test_msg_idxs.reverse()
    return test_msg_idxs
```

```
[ ]: ### Getting the model error in terms of difference from the measured CoAP␣
↪transmissions ###


idx = 0
test_tuples_coap = []

for i in coap_test_array:
    test_msg_idxs = get_msg_idxs(i.data)

    msg_energy_tuples = []
    for x in test_msg_idxs:
        #new_tuple = ()
        msg_energy_tuples.append(
            100*(1-float(i.data[x])/float(coap_predictions[idx][0]*x*0.00025 +␣
 ↪coap_predictions[idx][1])))
    idx += 1
    test_tuples_coap.append(msg_energy_tuples)
```

[ ]:
```
### Getting the model error in terms of difference from the measured MQTT␣
 ↪transmissions ###

idx = 0
test_tuples_mqtt = []

for i in mqtt_test_array:
    test_msg_idxs = get_msg_idxs(i.data)

    msg_energy_tuples = []

    for x in test_msg_idxs:
        #new_tuple = ()
        msg_energy_tuples.append(
            100*(1-float(i.data[x])/float(mqtt_predictions[idx][0]*x*0.00025 +␣
 ↪mqtt_predictions[idx][1])))
    idx += 1
    test_tuples_mqtt.append(msg_energy_tuples)
```

[ ]:
```
### Plotting the model error in terms of difference from measured data ###

fig, (ax1, ax2) = plt.subplots(1,2)

fig.suptitle("Model error - data points")

ax1.set_title("CoAP")
ax2.set_title("MQTT")

ax1.set_ylabel("Difference [%]")
ax1.set_xlabel("Message number")
ax2.set_xlabel("Message number")

y_min = -1
y_max = 7

ax1.set_ylim([y_min,y_max])
ax2.set_ylim([y_min,y_max])
```

```
labels = ["256 B (NB-IoT)", "256 B (LTE-M)", "4096 B (NB-IoT)", "4096 B␣
↪(LTE-M)",]
styles = ["g","r","g--","r--"]
markers = ["o","+","o","+"]

for i in range(len(test_tuples_coap)):
    ax1.plot(test_tuples_coap[i], styles[i], marker = markers[i])
    ax2.plot(test_tuples_mqtt[i], styles[i], marker = markers[i],label =␣
↪labels[i])

ax1.axhline()
ax2.axhline(label = "Zero difference")
ax2.legend(loc='center left', bbox_to_anchor=(1, 0.5))

plt.savefig("../dt_thesis/plots/model_error_diff.pdf", bbox_inches='tight')
```

```
[ ]: ### Getting the model error in terms of slope difference between model and␣
↪mesured data for CoAP ###

idx = 0
test_tuples_coap = []

for i in coap_test_array:
    test_msg_idxs = get_msg_idxs(i.data)
    msg_energy_tuples = []

    acc = []

    real_prev_energy = i.data[test_msg_idxs[0]]
    pred_prev_energy = float(coap_predictions[idx][0]*test_msg_idxs[0]*delta_t␣
↪+ coap_predictions[idx][1])

    for x in range(1,len(test_msg_idxs)):
        curr_msg_idx = test_msg_idxs[x]

        real_curr_energy = i.data[curr_msg_idx]
        real_energy_diff = real_curr_energy-real_prev_energy
        real_prev_energy = real_curr_energy

        pred_curr_energy = float(coap_predictions[idx][0]*curr_msg_idx*delta_t␣
↪+ coap_predictions[idx][1])
        pred_energy_diff = pred_curr_energy - pred_prev_energy
        pred_prev_energy = pred_curr_energy

        acc.append(100*(1-real_energy_diff/pred_energy_diff))

        msg_energy_tuples.append(
            100*(1-real_energy_diff/pred_energy_diff))
    idx += 1
    test_tuples_coap.append(msg_energy_tuples)
    print("AVG %d: %f (+/- %f)" %(idx, np.mean(acc), np.std(acc)))
```

```
[ ]: ### Getting the model error in terms of slope difference between model and␣
      ↪mesured data for CoAP ###

     idx = 0
     test_tuples_mqtt = []

     for i in mqtt_test_array:
         test_msg_idxs = get_msg_idxs(i.data)
         msg_energy_tuples = []

         acc = []

         real_prev_energy = i.data[test_msg_idxs[0]]
         pred_prev_energy = float(mqtt_predictions[idx][0]*test_msg_idxs[0]*delta_t␣
      ↪+ mqtt_predictions[idx][1])

         for x in range(1,len(test_msg_idxs)):
             curr_msg_idx = test_msg_idxs[x]

             real_curr_energy = i.data[curr_msg_idx]
             real_energy_diff = real_curr_energy-real_prev_energy
             real_prev_energy = real_curr_energy

             pred_curr_energy = float(mqtt_predictions[idx][0]*curr_msg_idx*delta_t␣
      ↪+ mqtt_predictions[idx][1])
             pred_energy_diff = pred_curr_energy - pred_prev_energy
             pred_prev_energy = pred_curr_energy

             acc.append(100*(1-real_energy_diff/pred_energy_diff))

             msg_energy_tuples.append(
                 100*(1-real_energy_diff/pred_energy_diff))
         idx += 1
         test_tuples_mqtt.append(msg_energy_tuples)
         print("AVG %d: %f (+/- %f)" %(idx, np.mean(acc), np.std(acc)))
```

```
[ ]: ### Plotting the slope error ###

     fig, (ax1, ax2) = plt.subplots(1,2)

     fig.suptitle("Model error - slope")
```

```
ax1.set_title("CoAP")
ax2.set_title("MQTT")

ax1.set_ylabel("Difference [%]")
ax1.set_xlabel("Message number")
ax2.set_xlabel("Message number")

y_min = -10
y_max = 20

ax1.set_ylim([y_min,y_max])
ax2.set_ylim([y_min,y_max])

labels = ["256 B (NB-IoT)", "256 B (LTE-M)", "4096 B (NB-IoT)", "4096 B␣
↪(LTE-M)",]
styles = ["g","r","g--","r--"]
markers = ["o","+","o","+"]

for i in range(len(test_tuples_coap)):
    ax1.plot(test_tuples_coap[i], styles[i], marker = markers[i])
    ax2.plot(test_tuples_mqtt[i], styles[i], marker = markers[i],label =␣
↪labels[i])

ax1.axhline()
ax2.axhline(label = "Zero difference")
ax2.legend(loc='center left', bbox_to_anchor=(1, 0.5))

plt.savefig("../dt_thesis/plots/model_error_slope.pdf", bbox_inches='tight')
```

```
[ ]: ### Calculation and plotting of battery life in years for a NB-IoT application␣
     ↪5wh battery with 4uA sleep current ###

     psm_length_test_array = [900,1800,3600,7200,86400]
     msg_size_test_array = [1,8,16,256,512,1280]
     msg_size_test_array.reverse()

     sleep_power = 4*V #uW
     capacity = 5000000 #wh

     coap_test_times = []
     mqtt_test_times = []

     #CoAP
     for i in psm_length_test_array:
         test_time_coap = []
         test_time_mqtt = []
         for j in msg_size_test_array:
```

```
        curr_mqtt_test   = Test(mqtt_256_nb, i,j, timing_mqtt_nb,␣
↪energy_mqtt_nb, get_data_start(mqtt_256_nb), E_cdrx_nb)
        curr_coap_test   = Test(coap_256_nb, i,j, timing_coap_nb,␣
↪energy_coap_nb, get_data_start(coap_256_nb), E_cdrx_nb)

        curr_mqtt_energy = get_energy(
            n_bytes      = curr_mqtt_test.n_bytes,
            max_bytes    = 1439,
            T_msg        = curr_mqtt_test.T_msg,
            E_cdrx       = curr_mqtt_test.cdrx,
            p_sleep      = sleep_power,
            reg_coeffs_t = curr_mqtt_test.t,
            reg_coeffs_e = curr_mqtt_test.E,
            start_params = curr_mqtt_test.E_start
        )

        curr_coap_energy = get_energy(
            n_bytes      = curr_coap_test.n_bytes,
            max_bytes    = 1439,
            T_msg        = curr_coap_test.T_msg,
            E_cdrx       = curr_coap_test.cdrx,
            p_sleep      = sleep_power,
            reg_coeffs_t = curr_coap_test.t,
            reg_coeffs_e = curr_coap_test.E,
            start_params = curr_coap_test.E_start
        )

        test_time_mqtt.append(((capacity - curr_mqtt_energy[1])/
↪curr_mqtt_energy[0])/3600/24/365)
        test_time_coap.append(((capacity - curr_coap_energy[1])/
↪curr_coap_energy[0])/3600/24/365)

    mqtt_test_times.append(test_time_mqtt)
    coap_test_times.append(test_time_coap)

plt.title("NB-IoT case study\n4uA sleep current")
plt.ylabel("Battery lifetime [years]")
plt.xlabel("Payload size [bytes]")
plt.ylim([0,13])

labels = ["CoAP - 900s", "CoAP - 1800s", "CoAP - 3600s", "CoAP - 7200s", None]
markers = ["+","o","<", "s", None]
idx = 0
for i in coap_test_times:
    plt.plot(i, color="g", marker=markers[idx], label = labels[idx])
    print(i)
    idx += 1
```

```
labels = ["MQTT - 900s", "MQTT - 1800s", "MQTT - 3600s", "MQTT - 7200s", None]
idx = 0
for i in mqtt_test_times:
    plt.plot(i, color="r", marker=markers[idx], label = labels[idx])
    print(i)
    idx += 1

plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.xticks(range(len(msg_size_test_array)), msg_size_test_array)

for i in range(len(psm_length_test_array)):
    print(100*(coap_test_times[i][5]/coap_test_times[i][0]-1))
for i in range(len(psm_length_test_array)):
    print(100*(mqtt_test_times[i][5]/mqtt_test_times[i][0]-1))


plt.savefig("../dt_thesis/plots/case_study_nb.pdf", bbox_inches='tight')
```

```
[ ]: ### Calculation and plotting of battery life in years for a LTE-M application␣
     ↪5wh battery with 4uA sleep current ###

     psm_length_test_array = [900,1800,3600,14400,86400]
     msg_size_test_array = [1,8,16,256,512,1280]
     msg_size_test_array.reverse()

     sleep_power = 4*V
     capacity = 5000000 #ah

     coap_test_times = []
     mqtt_test_times = []


     #CoAP
     for i in psm_length_test_array:
         test_time_coap = []
         test_time_mqtt = []
         for j in msg_size_test_array:
             curr_mqtt_test    = Test(mqtt_256_ltem, i,j, timing_mqtt_ltem,␣
     ↪energy_mqtt_ltem, get_data_start(mqtt_256_ltem), E_cdrx_ltem)
             curr_coap_test    = Test(coap_256_ltem, i,j, timing_coap_ltem,␣
     ↪energy_coap_ltem, get_data_start(coap_256_ltem), E_cdrx_ltem)

             curr_mqtt_energy = get_energy(
                 n_bytes       = curr_mqtt_test.n_bytes,
                 max_bytes     = 1439,
                 T_msg         = curr_mqtt_test.T_msg,
                 E_cdrx        = curr_mqtt_test.cdrx,
                 p_sleep       = sleep_power,
                 reg_coeffs_t = curr_mqtt_test.t,
                 reg_coeffs_e = curr_mqtt_test.E,
                 start_params = curr_mqtt_test.E_start
             )

             curr_coap_energy = get_energy(
                 n_bytes       = curr_coap_test.n_bytes,
                 max_bytes     = 1439,
                 T_msg         = curr_coap_test.T_msg,
                 E_cdrx        = curr_coap_test.cdrx,
                 p_sleep       = sleep_power,
                 reg_coeffs_t = curr_coap_test.t,
                 reg_coeffs_e = curr_coap_test.E,
                 start_params = curr_coap_test.E_start
             )

             test_time_mqtt.append(((capacity - curr_mqtt_energy[1])/
      ↪curr_mqtt_energy[0])/3600/24/365)
             test_time_coap.append(((capacity - curr_coap_energy[1])/
      ↪curr_coap_energy[0])/3600/24/365)

         mqtt_test_times.append(test_time_mqtt)
         coap_test_times.append(test_time_coap)

     plt.title("LTE-M case study\n4uA sleep current")
     plt.ylabel("Battery lifetime [years]")
     plt.xlabel("Payload size [bytes]")
     plt.ylim([0,13])

     labels = ["CoAP - 900s", "CoAP - 1800s", "CoAP - 3600s",  "CoAP - 14400s", None]
     markers = ["+","o","<", "D", None]
     idx = 0
     for i in coap_test_times:
         print(i)
         plt.plot(i, color="g", marker=markers[idx], label = labels[idx])
         idx += 1

     labels = ["MQTT - 900s", "MQTT - 1800s", "MQTT - 3600s", "MQTT - 14400s", None]

     idx = 0
     for i in mqtt_test_times:
         print(i)
         plt.plot(i, color="r", marker=markers[idx], label = labels[idx])
         idx += 1
```

```python
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.xticks(range(len(msg_size_test_array)), msg_size_test_array)

for i in range(len(psm_length_test_array)):
    print(100*(coap_test_times[i][5]/coap_test_times[i][0]-1))
for i in range(len(psm_length_test_array)):
    print(100*(mqtt_test_times[i][5]/mqtt_test_times[i][0]-1))

plt.savefig("../dt_thesis/plots/case_study_ltem.pdf", bbox_inches='tight')
```

[ ]:

```python

```

Appendix F

Processing helper functions

Listing F.1: t

```python

### This file contains helper functions used in processing ###

import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt

from sklearn import preprocessing
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn import metrics

## Linear Regression ##
## Params: byte_range  : the range of bates for the specific sample_list
##         sample_list : an array of datasets with either energy or time data
## Based on: https://towardsdatascience.com/a-beginners-guide-to-linear-
    regression-in-python-with-scikit-learn-83a8f7ae2b4f
def do_linear_regression(byte_range, sample_list):
    y = []
    X = []

    for i in range(len(sample_list[0])):
        for j in range(len(sample_list)):
            X.append(byte_range[i])
            y.append(sample_list[j][i])

    X_const = sm.add_constant(X)
    model = sm.OLS(y, X_const)
    results = model.fit()
    prediction = results.get_prediction()

    return results

## Gets the residuals with the given regression coefficients
def get_residuals(sample_list, reg_coeffs, byte_range):
    y = []
    X = []

    for i in range(len(sample_list[0])):
        for j in range(len(sample_list)):
            X.append(byte_range[i])
            y.append(sample_list[j][i])


    y_pred = reg_coeffs.slope*np.array(X)+reg_coeffs.intercept

    return [X, y-y_pred]

## Returns the message duration given the regression coefficients
def get_message_duration(n_bytes, max_bytes, reg_coeffs):
    x = np.linspace(0,max_bytes+1,max_bytes+1)
    y = [reg_coeffs[0]]*x+reg_coeffs[1]

    return y[n_bytes]

## Returns the message energy given the regression coefficients
def get_message_energy(n_bytes, max_bytes, reg_coeffs):
    x = np.linspace(0,max_bytes+1,max_bytes+1)
    y = [reg_coeffs[0]]*x+reg_coeffs[1]

    return y[n_bytes]

## Returns the coefficients for the estimation of total energy given the model
    parameters
def get_energy(n_bytes,max_bytes,E_cdrx,T_msg,p_sleep,reg_coeffs_t,reg_coeffs_e
    ,start_params):
    t_psm = T_msg-get_message_duration(n_bytes, max_bytes, reg_coeffs_t)

    E_msg = get_message_energy(n_bytes, max_bytes, reg_coeffs_e) #uWh
    E_sleep = p_sleep*t_psm/3600
    E_start = start_params[1]

    E_tot_coef = (E_msg+E_cdrx + E_sleep)/(T_msg)
    E_tot_intercept = E_start-E_tot_coef*start_params[0]

    return [E_tot_coef, E_tot_intercept]

## Returns an approximation of the cDRX energy
def get_con_energy(t_inactive, t_cycle, t_onDuration, E_monitor, E_release,
    p_idle):
    return ((p_idle*(t_cycle-t_onDuration)/3600 + E_monitor) * t_inactive/
    t_cycle + E_release)

## Returns an array of segments and an array of message durations. Parameter
    description follows:
# data              : the measurement data from the OTII
# batch_start       : an index at some point after the device has
#                     entered PSM and before the first test transmission.
# rrc_con_threshold: given in mA. Used to determine that a transmission is
    afoot.
# jump_past         : an value to be added to the start index of segment so that
     the index
#                     jumps past a transmission and can begin to iterate
    backwards to find
#                     the end.
# jump_between      : used to shorten processing time by moving closer to the
    next
#                     transmission.
def segment_data(data, batch_start, rrc_con_threshold, jump_past, jump_between)
    :
    segments = []
    timing = []

    segment_triggered = False

    delta_t = 0.00025

    idx = 0
    for batch in data:
        print("Batch: " + str(idx))
        curr_set = []
        i = batch_start
        curr_timing = []
        endpoint = (batch.index.size)
        while(i < endpoint):
            curr_meas = batch.iloc[i]
            if(segment_triggered):
                if(curr_meas > rrc_con_threshold):
                    segment_end = i
                    curr_timing.append((segment_end-segment_start)*delta_t)
                    curr_set.append(batch.iloc[segment_start:segment_end])
                    i+= jump_between
                    segment_triggered = False
                else:
                    i -= 1
            else:
                if(curr_meas > rrc_con_threshold):
                    segment_start = i
                    i+=jump_past
                    segment_triggered = True
                else:
                    i += 1
        timing.append(curr_timing)
        segments.append(curr_set)
        idx += 1
    return [segments, timing]
```

References

[1] GSMA. (Jun. 2019). NB-IoT deployment guide v3, [Online]. Available: https://www.gsma.com/iot/wp-content/uploads/2019/07/201906-GSMA-NB-IoT-Deployment-Guide-v3.pdf (visited on 12/06/2019).

[2] ——, (Jun. 2019). LTE-M deployment guide v3, [Online]. Available: https://www.gsma.com/iot/wp-content/uploads/2019/08/201906-GSMA-LTE-M-Deployment-Guide-v3.pdf (visited on 12/06/2019).

[3] Z. Shelby, K. Hartke, and C. Bormann. (Jun. 2014). The constrained application protocol (CoAP), [Online]. Available: https://tools.ietf.org/html/rfc7252 (visited on 12/16/2019).

[4] J. Postel. (Aug. 1980). User datagram protocol, [Online]. Available: https://tools.ietf.org/html/rfc768 (visited on 12/16/2019).

[5] Oasis/ISO. (Jun. 2016). ISO/IEC 20922:2016, ISO. Library Catalog: www.iso.org, [Online]. Available: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/06/94/69466.html (visited on 06/12/2020).

[6] J. Postel. (Sep. 1981). Transmission control protocol. Library Catalog: tools.ietf.org, [Online]. Available: https://tools.ietf.org/html/rfc793 (visited on 04/24/2020).

[7] Eclipse. (Apr. 2018). IoT developer survey results, [Online]. Available: https://iot.eclipse.org/resources/iot-developer-survey/iot-developer-survey-2018.pdf (visited on 12/15/2019).

[8] C. R. S. Fosse, "A survey of communication protocols for a low power embedded cellular device," Dec. 2019.

[9] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, "A comparative study of LPWAN technologies for large-scale IoT deployment," *ICT Express*, vol. 5, no. 1, pp. 1–7, Mar. 1, 2019.

[10] P. Reininger, "3gpp standards for IoT," Smart Summit, Singapore, Nov. 2016.

[11] 3GPP. (). Specification # 36.323, [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2439 (visited on 06/29/2020).

[12] S.-H. Hwang and S.-Z. Liu, "Survey on 3gpp low power wide area technologies and its application," in *2019 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS)*, ISSN: null, Aug. 2019, pp. 1–5.

[13] C. B. Mwakwata, H. Malik, M. Mahtab Alam, Y. Le Moullec, S. Parand, and S. Mumtaz, "Narrowband internet of things (NB-IoT): From physical (PHY) and media access control (MAC) layers perspectives," *Sensors (Basel, Switzerland)*, vol. 19, no. 11, Jun. 8, 2019.

[14] GSA. (Mar. 2019). Global narrowband IoT - LTE-m networks – march 2019, GSA, [Online]. Available: https://gsacom.com/paper/global-narrowband-iot-lte-m-networks-march-2019/ (visited on 12/17/2019).

[15] M. Lauridsen, I. Z. Kovacs, P. Mogensen, M. Sorensen, and S. Holst, "Coverage and capacity analysis of LTE-m and NB-IoT in a rural area," in *2016 IEEE 84th Vehicular Technology Conference (VTC-Fall)*, ISSN: null, Sep. 2016, pp. 1–5.

[16] A. K. Sultania, P. Zand, C. Blondia, and J. Famaey, "Energy modeling and evaluation of NB-IoT with PSM and eDRX," in *2018 IEEE Globecom Workshops (GC Wkshps)*, ISSN: null, Dec. 2018, pp. 1–7.

[17] 3GPP. (). Specification # 36.331, [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2440 (visited on 06/29/2020).

[18] A. K. Sultania, C. Delgado, and J. Famaey, "Implementation of NB-IoT power saving schemes in ns-3," in *Proceedings of the 2019 Workshop on Next-Generation Wireless with ns-3*, ser. WNGW 2019, Florence, Italy: Association for Computing Machinery, Jun. 21, 2019, pp. 5–8.

[19] 3GPP. (). Specification # 36.211, [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2425 (visited on 06/29/2020).

[20] ——, (). Specification # 38.331, [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3197 (visited on 06/29/2020).

[21] ——, (). Specification # 24.301, [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1072 (visited on 06/29/2020).

[22] P. Andres-Maldonado, P. Ameigeiras, J. Prados-Garzon, J. J. Ramos-Munoz, and J. M. Lopez-Soler, “Optimized LTE data transmission procedures for IoT: Device side energy consumption analysis,” *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 540–545, May 2017. arXiv: `1704.04929`.

[23] ISO. (). I35.100 - open systems interconnection (OSI), [Online]. Available: `https://www.iso.org/ics/35.100/x/` (visited on 06/29/2020).

[24] C. Gomez, A. Arcia-Moret, and J. Crowcroft, “TCP in the internet of things: From ostracism to prominence,” *IEEE Internet Computing*, vol. 22, no. 1, pp. 29–41, Jan. 2018, Conference Name: IEEE Internet Computing.

[25] C. Gomez. (Mar. 9, 2019). TCP usage guidance in the internet of things (IoT). Library Catalog: tools.ietf.org, [Online]. Available: `https://tools.ietf.org/id/draft-ietf-lwig-tcp-constrained-node-networks-05.html` (visited on 06/11/2020).

[26] J. Postel. (Sep. 1981). Internet protocol. Library Catalog: tools.ietf.org, [Online]. Available: `https://tools.ietf.org/html/rfc791` (visited on 06/12/2020).

[27] A. Stanford-Clark and H. L. Truong, “MQTT for sensor networks (MQTT-SN) protocol specification,” p. 28, Nov. 2013.

[28] A. Keranen. (Sep. 14, 2017). RESTful design for internet of things systems, [Online]. Available: `https://tools.ietf.org/id/draft-keranen-t2trg-rest-iot-05.html` (visited on 12/19/2019).

[29] A. Larmo, A. Ratilainen, and J. Saarinen, “Impact of CoAP and MQTT on NB-IoT system performance,” *Sensors (Basel, Switzerland)*, vol. 19, no. 1, Dec. 20, 2018.

[30] J. Wirges and U. Dettmar, “Performance of TCP and UDP over narrowband internet of things (NB-IoT),” in *2019 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS)*, ISSN: null, Nov. 2019, pp. 5–11.

[31] J. C. Mogul and S. E. Deering. (Nov. 1990). Path MTU discovery. Library Catalog: tools.ietf.org, [Online]. Available: `https://tools.ietf.org/html/rfc1191` (visited on 06/09/2020).

[32] Z. Shelby and C. Bormann. (Aug. 2016). Block-wise transfers in the constrained application protocol (CoAP). Library Catalog: tools.ietf.org, [Online]. Available: `https://tools.ietf.org/html/rfc7959` (visited on 05/10/2020).

[33] R. Braden. (Oct. 1989). Requirements for internet hosts - communication layers. Library Catalog: tools.ietf.org, [Online]. Available: `https://tools.ietf.org/html/rfc1122` (visited on 06/23/2020).

[34] B. Martinez, M. Montón, I. Vilajosana, and J. D. Prades, "The power of models: Modeling power consumption for IoT devices," *IEEE Sensors Journal*, vol. 15, no. 10, pp. 5777–5789, Oct. 2015, Conference Name: IEEE Sensors Journal.

[35] 3GPP. (). Specification # 45.820, [Online]. Available: `https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2719` (visited on 06/29/2020).

[36] P. P. Moletsane, T. J. Motlhamme, R. Malekian, and D. C. Bogatmoska, "Linear regression analysis of energy consumption data for smart homes," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, May 2018, pp. 0395–0399.

[37] J. Thrane, K. M. Malarski, H. L. Christiansen, and S. Ruepp, "Experimental evaluation of empirical NB-IoT propagation modeling in a deep-indoor scenario," *arXiv:2006.00880 [cs, eess]*, Jun. 1, 2020. arXiv: `2006.00880`.

[38] P. B. Palmer and D. G. O'Connell, "Regression analysis for prediction: Understanding the process," *Cardiopulmonary Physical Therapy Journal*, vol. 20, no. 3, pp. 23–26, Sep. 2009.

[39] GSMA. (). Mobile IoT modules, Internet of Things. Library Catalog: www.gsma.com, [Online]. Available: `https://www.gsma.com/iot/mobile-iot-modules/` (visited on 06/11/2020).

[40] N. Semiconductor. (2019). nRF9160 SiP. Library Catalog: www.nordicsemi.com, [Online]. Available: `https://www.nordicsemi.com/en/Products/Low%20power%20cellular%20IoT/nRF9160` (visited on 06/29/2020).

[41] ——, (2019). nRF9160 DK. Library Catalog: www.nordicsemi.com, [Online]. Available: `https://www.nordicsemi.com/en/Software%20and%20tools/Development%20Kits/nRF9160%20DK` (visited on 06/29/2020).

[42] ——, (2019). Nordic thingy:91 prototyping platform. Library Catalog: www.nordicsemi.com, [Online]. Available: `https://www.nordicsemi.com/en/Software%20and%20tools/Prototyping%20platforms/Nordic%20Thingy%2091` (visited on 06/29/2020).

[43] G. Instek. (). GPD-series multiple output programmable linear d.c. power supply. Library Catalog: www.gwinstek.com, [Online]. Available: `https://www.gwinstek.com/en-GB/products/detail/GPD-Series` (visited on 06/18/2020).

[44] Eclipse. (Jan. 8, 2018). Eclipse mosquitto, Eclipse Mosquitto. Library Catalog: mosquitto.org, [Online]. Available: `https://mosquitto.org/` (visited on 06/15/2020).

[45] Telenor. (). Telenor NB-IoT developer portal, Telenor NB-IoT Developer portal. Library Catalog: nbiot.engineering, [Online]. Available: https://nbiot.engineering (visited on 06/18/2020).

[46] P. Jupyter. (). Project jupyter. Library Catalog: jupyter.org, [Online]. Available: https://www.jupyter.org (visited on 06/25/2020).

[47] S. Bland. (May 7, 2020). Measuring PSM sleep current on the nRF9160-DK - nordic blog - nordic blog - nordic DevZone, Nordic Blog, [Online]. Available: https://devzone.nordicsemi.com/nordic/nordic-blog/b/blog/posts/measuring-psm-sleep-current-on-the-nrf9160-dk (visited on 05/08/2020).

[48] N. Semiconductor. (). nRFConnect SDK - nordic semiconductor. Library Catalog: www.nordicsemi.com, [Online]. Available: https://www.nordicsemi.com/en/Software%20and%20tools/Software/nRF%20Connect%20SDK (visited on 06/14/2020).

[49] T. L. Foundation. (). Zephyr project | home, Zephyr Project. Library Catalog: www.zephyrproject.org, [Online]. Available: https://www.zephyrproject.org/ (visited on 06/14/2020).

[50] N. Semiconductor. (). nRF connect for desktop. Library Catalog: www.nordicsemi.com, [Online]. Available: https://www.nordicsemi.com/en/Software%20and%20tools/Development%20Tools/nRF%20Connect%20for%20desktop (visited on 06/14/2020).

[51] A. Patel. (Jan. 2019). Getting started with nRF9160 DK - getting started - cellular IoT guides - nordic DevZone. Library Catalog: devzone.nordicsemi.com, [Online]. Available: https://devzone.nordicsemi.com/nordic/cellular-iot-guides/b/getting-started-cellular/posts/getting-started-with-nrf9160-dk (visited on 06/14/2020).

[52] C. R. S. Fosse, *Dt_app*, Available at: https://github.com/crfosse/dt_app, Jun. 29, 2020.

[53] 3GPP and ETSI. (2019). TS 127 007, [Online]. Available: https://www.etsi.org/deliver/etsi_ts/127000_127099/127007/15.07.00_60/ts_127007v150700p.pdf (visited on 06/18/2020).

[54] N. Semiconductor. (Apr. 2020). Nrf91 AT commands v1.2, [Online]. Available: https://infocenter.nordicsemi.com/pdf/nrf91_at_commands_v1.2.pdf (visited on 06/14/2020).

[55] ——, (Apr. 2020). nRF9160: Simple MQTT, GitHub. Library Catalog: github.com, [Online]. Available: https://github.com/nrfconnect/sdk-nrf (visited on 06/14/2020).

[56] E. Engineering. (Feb. 2020). Constrained application protocol (CoAP), GitHub. Library Catalog: github.com, [Online]. Available: https://github.com/ExploratoryEngineering/nrf9160-telenor (visited on 06/14/2020).

[57] C. R. S. Fosse, *Dt_data*, Available at: https://github.com/crfosse/dt_data, Jun. 29, 2020.

[58] J. Haukland, "Modeling the energy consumption of nb-iot transmissions," Master thesis, Norwegian University of Science and Technology, Trondheim, Jun. 2019, 115 pp.

Carl Richard Steen Fosse

Power Consumption modeling of TCP and UDP over low power cellular networks for a constrained device

NTNU
Norwegian University of
Science and Technology