Master's thesis

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

Ådne Karstad

Evaluating the Energy Consumption of Asset Tracking Applications

Master's thesis in Natural Science with Teacher Education
Supervisor: Magnus Jahre
Co-supervisor: Ketil Erichsen, Nicolai Berthelsen
June 2022

NTNU
Norwegian University of Science and Technology

NORDIC SEMICONDUCTOR

Ådne Karstad

Evaluating the Energy Consumption of Asset Tracking Applications

Master's thesis in Natural Science with Teacher Education
Supervisor: Magnus Jahre
Co-supervisor: Ketil Erichsen, Nicolai Berthelsen
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Evaluating the Energy Consumption of Asset Tracking Applications

Ådne Karstad

Problem Description

Asset tracking applications periodically collect interesting pieces of information about an asset and make it available to a back-end system. Asset tracking applications are typically implemented on Ultra-Low-Power (ULP) platforms since the system needs to either be shipped with sufficient energy to last throughout its operational lifetime or rely on the energy that it can harvest from its environment. These constraints result in a fundamental trade-off between performance, e.g., how frequently information about the asset can be retrieved and transmitted, and energy consumption. A useful asset tracking system must also be secure, but security requirements vary significantly across applications - meaning that the amount of energy and time the system can invest into improving security varies widely.

The objective of this master thesis project is to investigate the trade-off between performance, energy consumption, and security in the context of asset tracking applications. The starting point is the generic, yet unoptimized, asset tracking application the candidate implemented in his autumn project. The first task is to optimize this application by reducing the energy overhead the application incurs while executing system code and while idle. This will result in an optimized baseline that the candidate should then use to investigate the trade-off between performance, energy, and security for one or more security mechanisms.

Abstract

With the emergence of energy-efficient cellular IoT it is vital to research the relationship between security and energy consumption. In this thesis, I have implemented a Generic Asset Tracking Application in order to evaluate the trade-off between security and energy consumption on a nRF9160 System in Package platform. The Asset Tracking Application that was used in the experiments is implemented using LTE-M and MQTT over TCP, and utilizes Power Saving Mode and a keep-alive feature for power saving that enables the asset tracker to remain connected to the network and to the MQTT server through its sleep cycles. The results from the experiments show that connecting to the server using Transport Layer Security (TLS) is 480% more energy demanding than without TLS. However, the experiments also indicate that the variation in the cost of connecting to the network using LTE-M is greater than the added cost of connecting to the server using TLS. Furthermore, the results seem to indicate that the cost of establishing connection is negligible, assuming that it is possible to use both PSM and the keep-alive feature. Lastly, the results of the experiments suggest that platforms that are hardware accelerated (DSA) for cryptography, like the nRF9160, have a negligible relationship between energy consumption and the security protocols that were tested in this thesis.

Sammendrag

Med en fremvekst av energieffektiv cellulær IoT er det avgjørende å forske på forholdet mellom sikkerhet og energiforbruk. I denne oppgaven har jeg implementert en Generisk Asset Tracking Applikasjon for å evaluere avveiningen mellom sikkerhet og energiforbruk på en nRF9160 System in Package-platform. Asset Tracking-applikasjonen som er brukt i eksperimentene er implementert med LTE-M og MQTT over TCP, og bruker Power Saving Mode og en keep-alive funksjon for strømsparing som gjør det mulig for asset trackeren å forbli koblet til nettverket og MQTT-serveren gjennom søvnsyklene. Resultatene fra eksperimentene viser at tilkobling til serveren med TLS er 480% mer energikrevende enn uten TLS. Eksperimentene indikerer imidlertid også at variasjonen i kostnadene ved tilkobling til nettverket med LTE-M er større enn tilleggskostnaden ved å bruke TLS nå enheten etablerer forbindelse med serveren. Videre ser resultatene ut til å indikere at kostandene ved å etablere forbindelse er neglisjerbare, forusatt at det er mulig å bruke både PSM og keep-alive-funksjonaliteten. Avslutningsvis tyder resultatene av eksperimentene på at plattformer som er maskinvareakselerert (DSA) for kryptografi, slik som nRF9160, har et neglesjerbart forhold mellom energiforbruk og sikkerhetsprotokollene som ble testet i denne oppgaven.

Contents

Figures

Tables

Code Listings

Acronyms

AI Artificial Intelligence (AI). 16

AIATA All-Intensive Asset Tracking Applications (AIATA). 16

AT Asset Tracker (AT). xix, 5–7, 9, 10, 13, 14, 16–20, 23, 32, 33, 35, 36

CIATA Compute-Intensive Asset Tracking Applications (CIATA). 16

CoAP Constrained Application Protocol (CoAP). 8, 36

DSA Domain-Specific Hardware Acceleration (DSA). v, vii, 33, 35, 36

DUT Device Under Test (DUT). 18, 20, 21, 25, 28

E2EE End-to-end encryption (E2EE). xi, 9–11, 26, 33

GATA Generic Asset Tracking Application (GATA). xi, xix, 1–3, 5–7, 9, 10, 13, 14, 17–19, 23, 25–28, 33, 35, 36

GPIO General Purpose Input Output (GPIO). xiii, 20, 21, 24, 32

GUI Graphical User Interface (GUI). 18

IEFT Internet Engineering Task Force (IEFT). 8

IoT Internet of Things (IoT). 1

LPWAN Low-Power Wide-Area Network (LPWAN). 5, 7, 10, 36

MITM attack Man-in-the-Middle (MITM) attack. xi, 9, 10

NIATA None-Intensive Asset Tracking Applications (NIATA). 16

OSI model Open Systems Interconnection model (OSI model). 7–9

PIATA Payload-Intensive Asset Tracking Applications (PIATA). 16

PP Power Profiler (PP). xi, 18, 20–23

PSA Platform Security Architecture (PSA) framework. 33

PSM Power Saving Mode. v, vii, 7, 8, 10, 11, 33, 35, 36

RAM Random Access Memory. 24

SiP System in a package (SiP). 23, 24, 35

TAU Tracking Area Update (TAU). 6, 7

TCP Transmission Control Protocol (TCP). xi, 8, 9, 11, 35

TLS Transport Layer Security (TLS). v, vii, xi, 1, 9–11, 18, 26, 28, 32, 33, 35

UDP User Datagram Protocol (UDP). 8

UE User Equipment (UE). 6, 7

Glossary

3GPP The 3rd Generation Partnership Project (3GPP) unites seven telecommunications standard development organizations and provides their members with a stable environment to produce the Reports and Specifications that define 3GPP technologies [1] . 1

Main loop The main loop of a Generic Asset Tracking Application (GATA) is the essential part of an AT application and consists of a compute, send, and sleep cycle defined in Chapter 3. xi, 14, 18–20, 26, 28, 33–35

TLS Handshake TLS Handshake is the process of establishing a secure encrypted connection between a client and a server, and is described in Chapter 2. 9, 11, 18

Chapter 1

Introduction

1.1 Motivation

Asset Tracking applications based on cellular IoT are becoming more common, and utilizing the advancement of cellular technology standards, developed by 3GPP, asset trackers are able to send environment data using the new technology and still remain energy-efficient [2–4]. Still, an emerging security concern in the IoT industry has been known for years [5], and with the emergence of a new cellular IoT industry there seems to be a lack of research on the relationship between security and energy consumption.

Some of the research that have been conducted have found that Transport Layer Security (TLS) has a negative impact on the performance, compared to similar protocols [6]. However, the research is more concerned with the efficiency in terms of latency and throughput of messages, and at the time of writing I have not found any research that investigates the relationship between security and energy consumption on cellular asset tracking technology.

Hence, the aim of this thesis will be to create a basis for application specific recommendations in the future, and I will start by developing a Generic Asset Tracking Application (GATA), illustrated in Figure 1.1, in order to assess the impact of security implementations on asset tracking applications with different performance requirements.

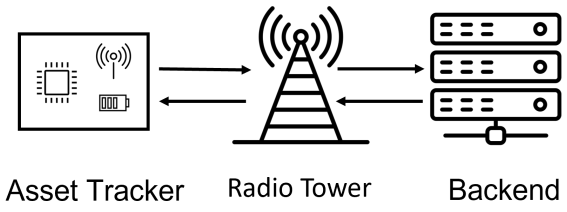


Figure 1.1: Generic Asset Tracking Application (GATA)

1.2 Assignment Interpretation

In this thesis, I will first continue my work from last semester by developing a GATA that will be used as a basis in order to investigate the relation between security and energy consumption on an asset tracking application. The GATA will be developed and energy-optimized with different security protocols, and will have a baseline implementation called **none**, in order to compare the impact of security on the energy-consumption. The assignment also asks to investigate the trade-off between performance, energy consumption and security, and I have decided to included performance in order to differentiate various asset tracking application from a Generic Asset Tracking Application (GATA) based on different performance requirements shown in Figure 1.2. With the performance requirements, I want to mainly investigate the trade-off between security and energy consumption, and secondly see if the performance requirements impact the relationship between security and energy consumption.

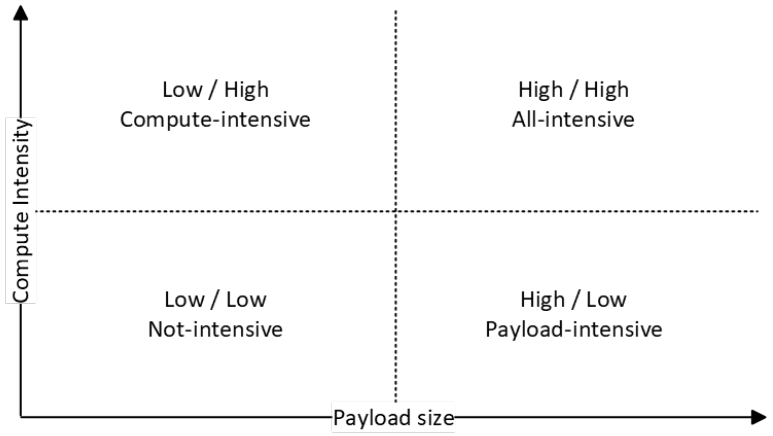


Figure 1.2: Requirement matrix for Generic Asset Tracking Applications. Different asset tracking applications are identified by adjusting the requirements with regards to payload size and compute intensity, and will be further explained in chapter 3.

The second task will be to develop a Framework in order to automate the experiments and to analyse the energy consumption in a way that allows me to investigate the relationship between security and energy consumption.

To meet the requirements of the problem description, I hence have to complete the following tasks:

- T1 Implement and optimize a GATA in order to conduct a set of experiments.
- T2 Implement a Framework in order automate testing of GATA with different parameters, and to evaluate the relationship between security and energy consumption for different asset tracking requirements.
- T3 Analyse the results from the experiments.

1.3 Contributions

I have addressed T1 by creating a Generic Asset Tracking Application (GATA) implementation which I will present in Chapter 3. I initially started to create the GATA implementation using a sample provided by Nordic Semiconductor ASA, and over iterations, implemented new parameters in order to capture different asset tracking requirements. In parallel I worked on implementing the framework, which has been named the GATA Framework. The GATA Framework not only enables automatic power measurements and analysis of the results, but it also contributes to improve the development experience by providing simple parameterization of builds, automates the building process, and flashing to device. Hence, the GATA Framework have exceeded the expectations I initially had for T2. The results show that, for the specific platform that the experiements was conducted on, the security parameters may impact the initial stage of an asset tracking application. However, the results also seem to indicate that over time the difference in energy consumption is overshadowed by variation in the energy that the platform's modem consume, and hence that the security protocol have little impact on the total energy consumption, which concludes my contribution to T3.

Chapter 2

Background

In this Chapter I will present relevant theory about the technologies and concepts that will be used in the thesis. I will first present LTE-M, which is a Low-Power Wide-Area Network, and is used for cellular IoT communication, and I will mainly focus on it's Power Saving Mode feature. Then, I will present the messaging protocol MQTT, and explain how it may be used as part of a cellular IoT system. Last, I will describe some security challenges and present two possible security additions that will also be used in the experiments.

2.1 Low-Power Wide-Area Networks

In this section I will provide an overview of Low-Power Wide-Area Network (LPWAN) technology and how it fits into the GATA model. LPWAN is aimed specifically at constrained devices that have requirements for communication type protocol that is longer than what short-range protocols like Bluetooth and WiFi are able to provide. On the other hand, conventional telecommunication technologies like LTE and 4G are able to satisfy the range requirements, but are too energy demanding to be used by constrained devices, like an Asset Tracker (AT). Hence, 3GPP who has the responsibility to develop compulsory telecommunication protocols, have developed the LPWAN technologies LTE-M and NB-IoT. Common for the two technologies is that they enable the AT to communicate with a backend service by providing a connection through a radio tower, as illustrated in Figure 2.1.

LTE-M is specified in 3GPP in Release 13, 14 and 15 [3] and differs from NB-IoT by providing high bandwidth and low latency in comparison, but for a constrained device it may require more sophisticated equipment. 3GPP defined in 2016 that NB-IoT is aimed at lasting more than 10 years on a single 5 watt hour battery (depending on traffic and coverage), and LTE-M is aimed at lasting about 10 years [7]. To restrict energy consumption both LPWAN technologies utilize the same power saving features, and I will focus on a feature called Power Saving Mode.

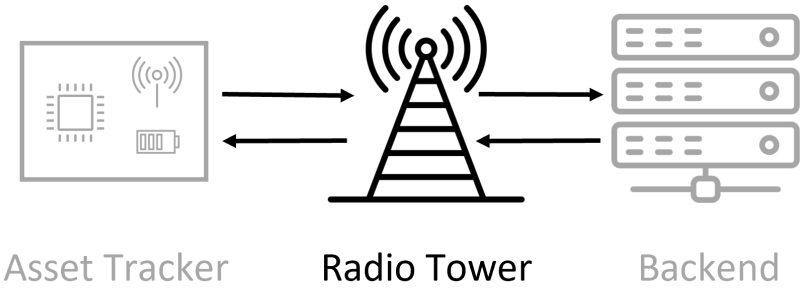


Figure 2.1: Telecommunication aspect of GATA.
Telecommunication is dependant on the network that radio towers can provide, and the Figure is meant to illustrate that in order for an AT to communicate with a backend it must use the radio tower as an intermediary.

Power Saving Mode (PSM) is a feature specifically designed for constrained devices in order to conserve battery power and achieve a longer battery life time [3]. PSM essentially grants the User Equipment (UE) the ability to stay attached to the network when it enters sleep and is unreachable by the network. An illustration of how PSM works is shown in Figure 2.2. The UE is able to transfer data in cycles, and enter sleep between data transfer phases, however the specifications require the UE to remain reachable for a specified duration after data transferal is complete. The shortest duration that the UE must remain reachable is first determined by the network, and is called an RRC inactivity timer, but may also be extended by the developer by asking for an active timer (T_{3324}) when initiating the connection with the network.

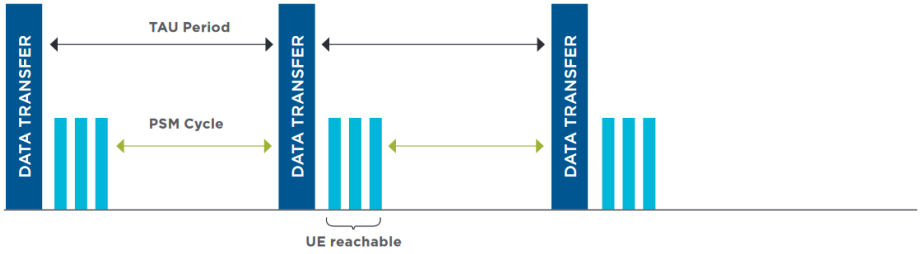


Figure 2.2: An illustration of PSM intervals [3].
The Figure shows the TAU Period, which is the duration between when the UE sends a TAU to the network to notify that it still wants to be attached to the network. The TAU Period must be shorted than the specified period TAU timer (T_{3412}). The PSM cycle illustrated is the duration that the device is allowed to sleep and therefore be unreachable by the network, and the period between data transfer and PSM cycle is the required duration the UE must remain reachable by the network.

Another timer specified by PSM, is called the periodic TAU timer (T_{3412}), or often referred to as the PSM mode sleep interval, because it specifies how often the UE must provide a Tracking Area Update (TAU) [8], the TAU period is illustrated in Figure 2.2. The minimum value of the periodic TAU timer is zero, which would effectively disable PSM, and the maximum value, as of 3GPP Release 13, is 413 days [3]. Note that every time the UE wakes up to transfer data it will perform a new TAU, effectively resetting the periodic TAU timer. This means that there is no real drawback if the periodic TAU timer provided by the network is longer than the data transfer interval of the application, because it will send a TAU whenever the UE starts to interact with the network, regardless of how far the timer has come when the UE starts to interact with the network again.

2.2 Communication Protocols

In the last section, I described how LPWAN enables long-range communication between an Asset Tracker (AT) and a backend server as illustrated by the Generic Asset Tracking Application (GATA) model, and now I want to describe a communication protocol that an AT and a backend can use in order to communicate regardless of the network they are using. Figure 2.3 shows a simple illustration of what aspects of the GATA model this section targets.

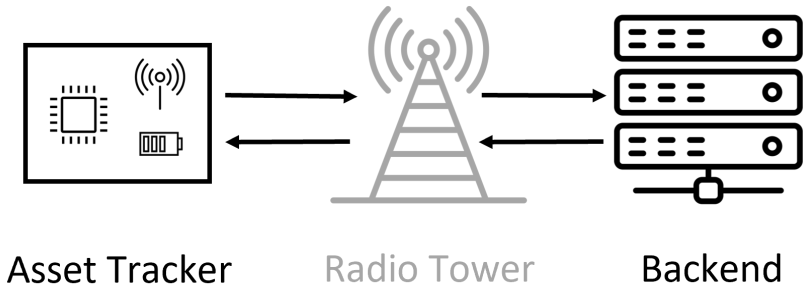


Figure 2.3: Aspects of GATA model that are affected by the communication protocols.

The AT and the backend are the only parties in the GATA illustration that relies on the MQTT protocol, and it enables developers to abstract away the telecommunication aspect of it.

How a communication protocol may be understood is through the Open Systems Interconnection model (OSI model), which is a conceptual model describing the standard communication layers of a communication system, and it is normal to map a specific protocol to different layers of the OSI model.

The communication protocol that I will present is called MQTT, formerly an initialism for MQ Telemetry Transport, and is a protocol that resides in the application layer of the OSI model. MQTT is a lightweight messaging protocol that uses a publish/subscribe pattern. The protocol consists of a server, called a broker,

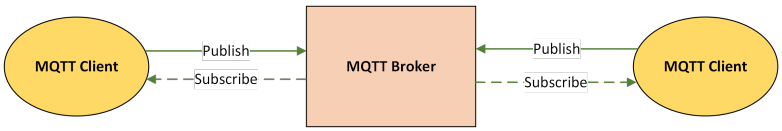


Figure 2.4: Illustration of MQTT.
The MQTT broker works as an intermediary for all clients that operate in the network. Producers share data by publishing to topics on the Broker, and consumers get data by subscribing to topics on the Broker.

that works as an intermediary for clients on the network. Clients can subscribe to topics on the server, and when a message arrives the broker, because a client published it to the topic, the broker will publish/send the message to all the clients that subscribe to it. Figure 2.4 shows how clients typically interact with the broker.

MQTT requires a transport protocol that is ordered, lossless, and bi-directional, and is therefore often run over Transmission Control Protocol (TCP). TCP is located in the transport layer of the OSI model, and was first specified in RFC793 [9] by the Internet Engineering Task Force (IEFT). TCP is intended to be reliable connection-oriented communication protocol, but adds a cost in terms of an increase in the number of messages that must be sent in order to establish the reliable connection. When a client wants to establish connection to a server, using TCP, it performs a 3-way handshake in order to synchronize the communication, and the procedure is shown in Figure 2.5. The essential part of the 3-way handshake is to synchronize separate sequence numbers for the client and the server, which is used in order to detect whether parts of the communication is lost, and to reorder packets if they arrive out of order.

A common substitute for MQTT over TCP is Constrained Application Protocol (CoAP) over User Datagram Protocol (UDP). CoAP, which is the application layer substitute for MQTT, was developed specifically for constrained environments. CoAP over UDP allows constrained devices to disregard any connection, and therefore, save the cost of establishing and maintaining a connection. However, with MQTT the client is able to initiate the connection with a keep-alive value, which enables the client to remain connected to the broker for a given duration without exchanging data. The keep-alive feature is compatible with PSM and enables a client to sleep during a keep-alive interval, and may therefore avoid the cost of reconnecting to the server after sleep. Keep-alive is, however, restricted to a maximum of 18 hours 12 minutes and 15 seconds [10], which is a short time compared to the 413 days that are enabled by PSM, and is therefore a bottleneck in the possible sleep time for a production environment.

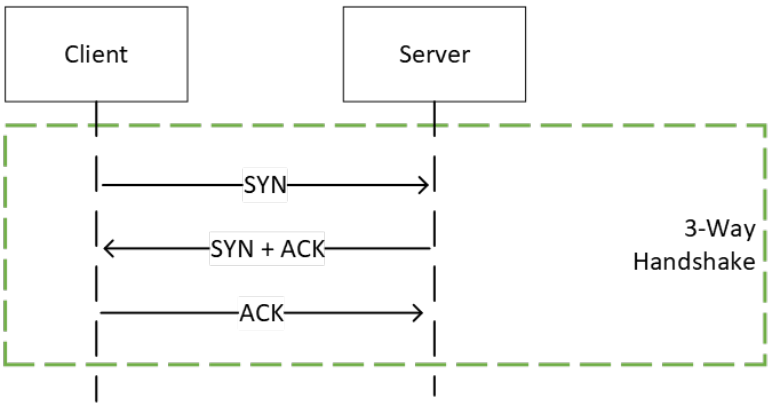


Figure 2.5: TCP 3-Way Handshake.

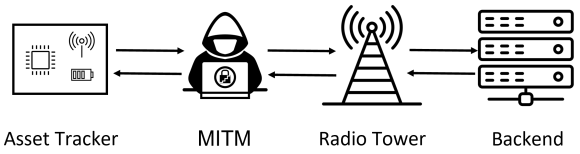
When a client tries to establish connection with a server it will initialize the 3-Way Handshake by sending a SYN message which is a synchronization of the sequence number that will be incremented for each new packet sent by the client. The server will respond with an acknowledgement (ACK) to the SYN message together with it's own sequence number that will be incremented for each new message the server sends. Last, the client responds with an ACK to the server's SYN message before the connection is fully established.

2.3 Security

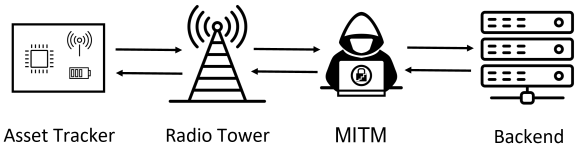
Now that I have presented the relevant communication technologies I want to provide some background for the security principles that will be assessed in this thesis. The scope in terms of security is limited to the communication, and what the AT is able to contribute with locally. In addition, I want to limit the scope to involve protocols that reduce the communication's vulnerability to Man-in-the-Middle (MITM) attacks, which is an umbrella term for attacks where the attacker relays or alters what is sent between two parties. Hence, I will present two security protocols that will be used, Transport Layer Security (TLS) that is a security improvement of the transportation layer of the OSI model, and End-to-end encryption (E2EE) which is platform-specific security protocol that I will define below.

TLS is a protocol intended to increase security for client/server communication by providing privacy and data integrity between the two, and it is designed to prevent eavesdropping, tampering, and message forgery, which are all different forms of MITM attack [11]. At the lowest layer TLS is layered on top of another reliable transport protocol, usually TCP, and is therefore a natural addition to the baseline setup of MQTT over TCP that I will explore in this thesis. The way TLS works is that both the client and server has to agree to enable TLS. The procedure of setting up a TCP communication with TLS between a client and a server, called the TLS Handshake is illustrated in Figure 2.7.

The security effect of TLS is limited to the communication between the AT and the server, which limits the impact of a potential MITM attack in the GATA model,



(a) MITM attack between AT and the radio tower.



(b) MITM attack between the radio tower and server.

Figure 2.6: Man-in-the-Middle (MITM) attack inside the GATA model

MITM attack is a an umbrella term for all attacks where an attacker compromises the communication by working as an intermediary, and enables the attacker to read, and potentially modify the data being transmitted between a client and a server. In the GATA model a MITM attack can be placed either between the AT and the radio tower, shown in Figure 2.6a, or it between the radio tower and the server, shown in Figure 2.6b.

shown in Figure 2.6. However, once the message is received by the server, and if the developer of the Asset Tracker (AT) do not make any assumption on the security carried out by the server, then the server may store the message in plain text after it is decrypted by the TLS protocol, which is a security vulnerability because attackers may get access to the server. Furthermore, there is no guarantee that other clients that have legitimate access to the data on the server utilize TLS, and then the communication is still vulnerable to a potential MITM attack, illustrated in Figure 2.8a. Assuming that I can only guarantee that TLS improves the security in communication between the AT and the server, I want to add another security metric that the AT can perform in order to secure the message payload from client to client, as illustrated by Figure 2.8b.

End-to-end encryption (E2EE) is the process of using data encryption in order to make the data unreadable for any party that has access to it, but does not have the key in order to decrypt the data. This enables securing the message payload from initial transmission from the client, and all the way to another client. If the payload is intercepted anywhere between the originator of the message and the receiving client, the message will be unreadable. The process of E2EE is illustrated in Figure 2.9.

As a summary of the background chapter, I have presented the LPWAN technology LTE-M which enables devices to communicate with a network from long ranges, and enables IoT devices to utilize an efficient power saving feature called PSM, which grants a power-efficient device to potentially run on a single battery for up to 10 years [3]. Then I presented a messaging protocol called MQTT, which

is run on top of a connection-oriented transport protocol called TCP. MQTT has the potential to maintain a connection through sleep cycles by using a keep-alive value that is compatible with the PSM feature granted by LTE-M, but the keep-alive feature is limited to a maximum of circa 18 hours of sleep at a time, whereas PSM would allow a device to sleep for 413 days. Last, I introduced two different security metrics called TLS and E2EE.

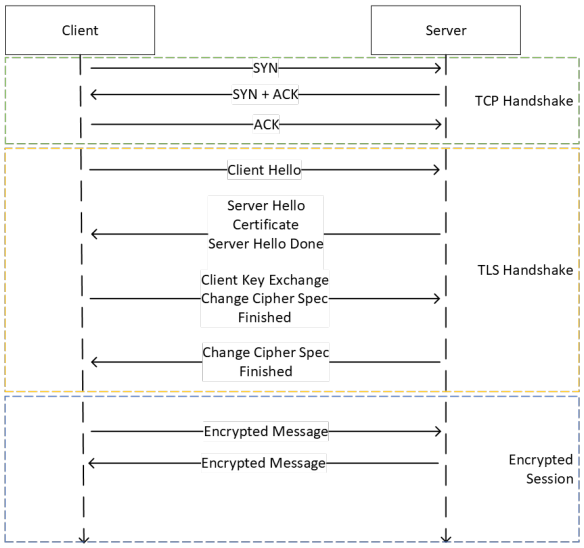


Figure 2.7: Process diagram of TLS handshake.
The TLS Handshake is the process exchanging certificates and negotiating encryption specifications between client and server, and when the process is complete the communication will be fully encrypted [12].

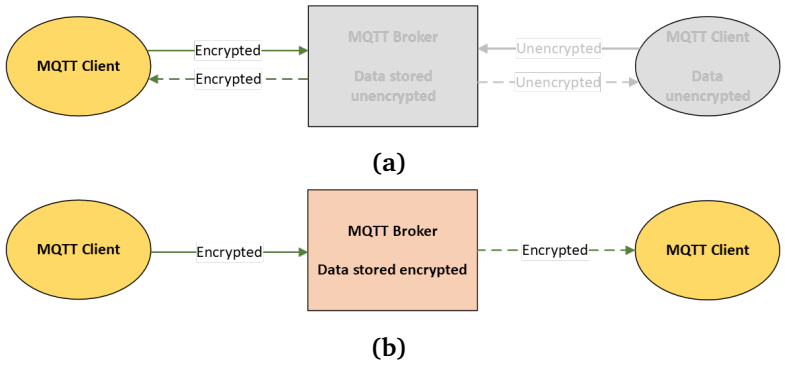


Figure 2.8: The scope of message-encryption using TLS versus E2EE
Figure 2.8a shows that only the traffic between the original client and the broker is encrypted using TLS, in contrast to using E2EE, shown in Figure 2.8b, where the message is encrypted by originater and decrypted by another client.

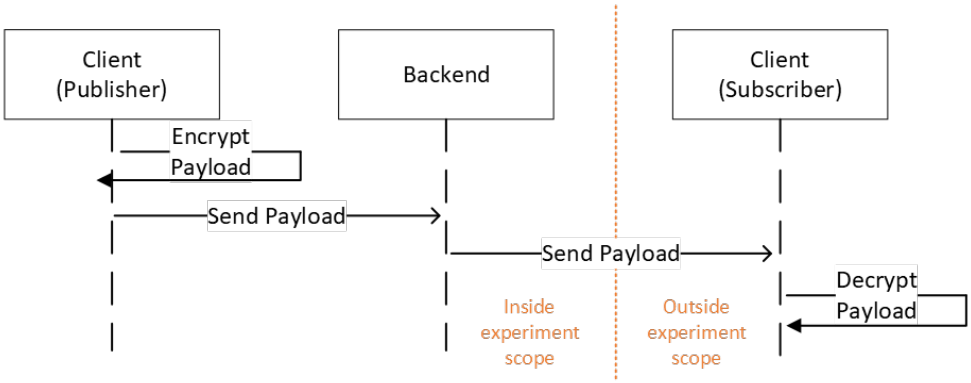


Figure 2.9: Process diagram of an end-to-end encryption of a message payload. A publishing client encrypts a message and sends it to a backend, the payload remains encrypted until someone with the encryption key may decrypt it. The Figure illustrates that a different client may receive the message, still encrypted, from the backend and, if the client has the encryption key, it may decrypt message.

Chapter 3

The Generic Asset Tracking Application (GATA)

A Generic Asset Tracking Application (GATA), or the GATA model, is an attempt to capture a generic application that utilizes an Asset Tracker (AT) without limiting its scope, and the idea is to assess it by using various configurations in order to capture different requirements that developers may have when creating an AT.

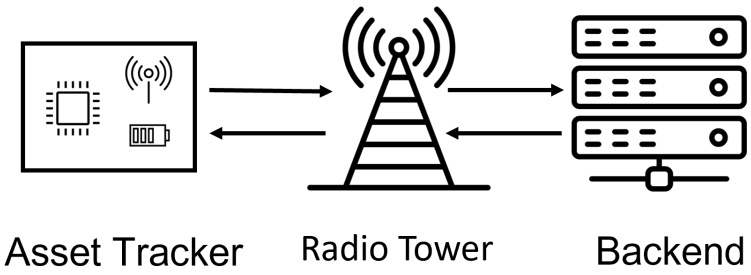


Figure 3.1: The Generic Asset Tracking Application (GATA) model
In this thesis I define that a GATA is an Asset Tracker (AT) that collects some environment data that it communicates to a Backend through a Radio Tower. A more detailed illustration of a generic AT can be viewed in Figure 3.2.

A simple illustration of the GATA model is shown in Figure 3.1, and explains the relationship between an AT, a radio tower, and a backend server. An AT is used to collect data about its environment, and sends the data to a backend using a radio tower in order to utilize long-range cellular technology. A generic AT is illustrated in Figure 3.2.

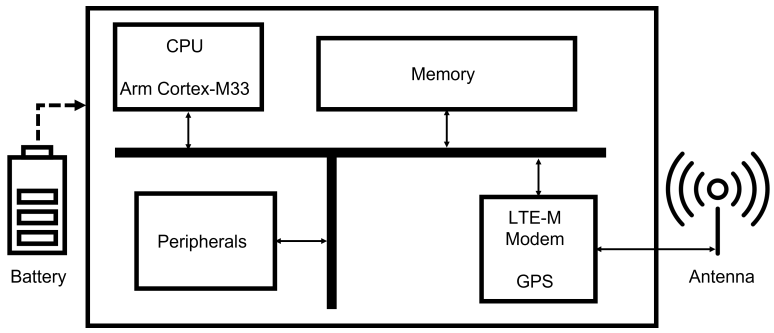


Figure 3.2: A Generic Asset Tracker.
A Generic AT is supplied with power by an external battery, and includes a CPU that has access to memory, peripherals and a Modem through a bus. The Modem is connected to an external antenna, and the modem controls both telecommunication and the GPS, and it is possible to use separate antennas for GPS and telecommunication.

3.1 Implementing GATA

In order to create a generic application I have tried to capture the essence of an AT application, and have derived three key features that I would like to give more attention to. The three key features are what I call compute, send, and sleep. Compute is the part of the application that collect environment data, and may include running analytical computations on the data. Send will typically be performed after harvesting environment data and consists of communicating the data to a backend server. The last key feature is sleep, which is essentially what an AT does whenever it is not performing compute or send. Ultimately, a GATA, will run a cycle of compute, send, and sleep, for as long as it is in use, as illustrated by Algorithm 1. I will refer to the compute, send, and sleep cycle as the Main loop of a GATA.

3.2 Application requirements

I believe that three different parameters, or requirements, are natural with regards to the GATA model illustrated by Algorithm 1. First, is the computational intensity required by collecting environment data and performing some potential analysis of the data. Second, is the payload size of the data that is sent to the backend, and the third is the duration of sleep. I want to focus on the former two in order to assess the impact on energy consumption when comparing different security protocols on four different types of AT applications. The four different type of applications, or requirements, are illustrated in Figure 3.3, and will be used in order to find suitable testing-parameters for the experiments.

Algorithm 1 Simple Pseudocode of a Generic Asset Tracking Application.

```
1: setup()
2: for ever do
3:     compute()
4:     send()
5:     sleep()
6: end for
```

A Generic Asset Tracker will need to connect to a backend, and will run different variations of compute, send, and sleep depending on the specific implementation of the application. The setup function will be setting up connection with the backend and all dependencies related to establishing connection to it, and may be performed before the loop with premiss of maintaining the connection, or it may connect and disconnect between each send. The compute functionality is a representation of all computational intensive activity an Asset Tracker performs when it is not sleeping, and may include collecting environment data and possibly computations on the data. The send functionality includes the cost of transmitting data and potentially receiving data from the backend. The goal with GATA is therefore to implement a Generic Application that is easily parameterized in order to represent different applications. Hence the natural parameters may be compute intensity, payload size, or duration of sleep. In this thesis, as will be described in the next Chapter, I have chosen to parameterize compute intensity and payload size.

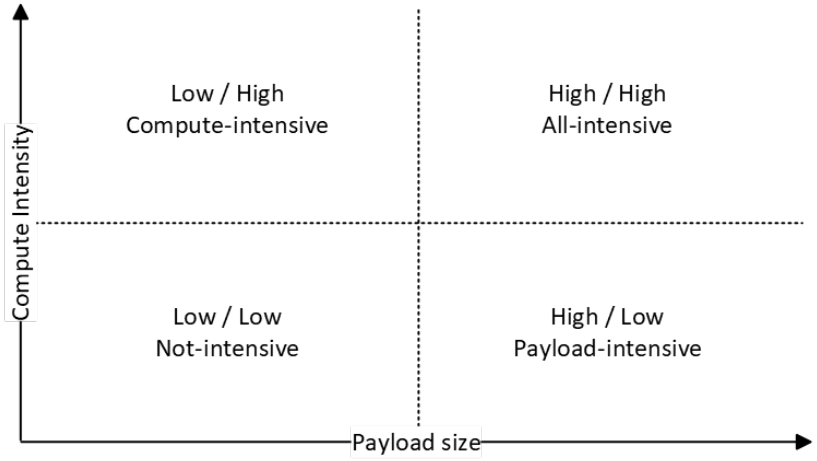


Figure 3.3: Requirement matrix for Generic Asset Tracking Applications. The Figure illustrates different requirements with regards to Payload size and Compute intensity, and four different applications are derived from the requirements.

None-Intensive Asset Tracking Applications None-Intensive Asset Tracking Applications (NIATA) are applications that does not require much of the CPU and sends small payloads to the backend, and could be ATs that use cheap sensors, in terms of energy cost, and send single bytes of data to the backend.

Payload-Intensive Asset Tracking Applications Payload-Intensive Asset Tracking Applications (PIATA) are applications that sends relatively big payloads to the backend, but does not require much compute-intensity by the CPU before sending the payload. A general example could be an AT that sends a picture to the backend.

Compute-Intensive Asset Tracking Applications Compute-Intensive Asset Tracking Applications (CIATA) are applications that requires relatively high compute-intensity, but only sends a few bytes to the backend. An example could be ATs that use Artificial Intelligence (AI) in order to make some decision about the environment, and hence only needs to send a few bytes to the backend.

All-Intensive Asset Tracking Applications All-Intensive Asset Tracking Applications (AIATA) are applications that require both high payload sizes and high compute-intensity. This could be a requirement if the AT needs to run several diagnostics on the environment and needs to send a longer report to the backend.

Chapter 4

The GATA Framework

In this chapter I will present the GATA Framework, illustrated in Figure 4.1, which is the process I have developed in order to automate power measurements of different configurations and evaluate the resulting power profiles. The Framework requires three physical devices and a few different software tools. The physical devices are a computer, an Asset Tracker (AT) with debug features, and a power measurement device. I have personally developed the software tools in the framework, and the automatic power profiling was developed on top of code provided by Nordic Semiconductor ASA.

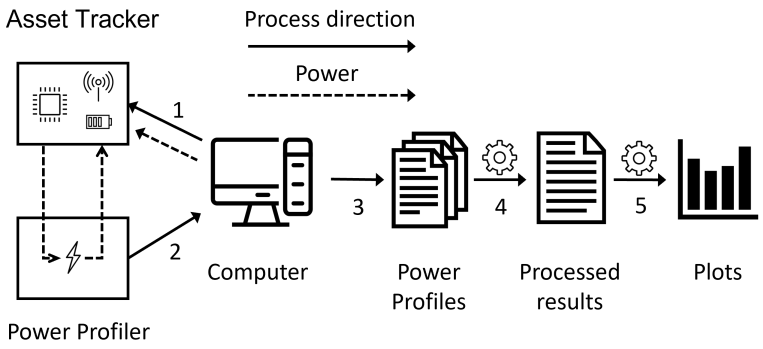


Figure 4.1: The GATA Framework

The Computer runs a script in order to flash each configuration to the Asset Tracker, and then starts the power measurements and provides the Asset Tracker with power. The Power Profiler sends the measurement data to the Computer, which stores each test result in separate files. When all tests have been run the computer can start to analyse the test results. First it reads through all power profiles and outputs a processed file that summarizes the results, and then it runs a separate script to generate plots.

4.1 Automated Power Measurements

The process described by the GATA Framework in Figure 4.1 process 1 through 3 illustrates the automated power measurement work flow. The computer runs a script that takes care of building the AT application with different parameters, in terms of different configurations. When all solutions are built, the script loops through all the solutions, and flashes them to the DUT. When the DUT is flashed with a new build, the script synchronizes the Power Profiler (PP) and the DUT, and starts the measurements. The script receives all the data in real-time, and when the measurement is finished saves the power profile in two different formats, .ppk format for the Graphical User Interface (GUI), and .csv files for further analysis.

4.2 Analysis of Power Profiles

Process 4 through 5 of the GATA Framework, shown in Figure 4.1, consists of running two different tools I have developed. I call the first tool uAnalyser, which takes all the power profiles from the automated power measurements and outputs a single file with a summary of time and energy consumption for all the relevant sections of the application. The second tool I call uAplotter, and it is responsible for plotting the results that I will present in chapter 6. Now, I want to present how I will derive the different sections in order to assess the power profiles, and I have chosen to assess the cost of setup, compute, modem, and sleep.

Setup The setup part of the application consists of connection to the network using LTE-M, as mentioned in the Background chapter, and connecting to the MQTT broker. Both are controlled by semaphores in order for the application to wait until connection has been established before moving on, and hence the AT needs to connect to the LTE-M network before it can connect to the MQTT broker, and needs to establish connection to the broker before starting the Main loop.

Connection to the LTE-M network should not be affected by the security parameter that is used by the application, connecting to the MQTT broker on the other hand, will be affected if Transport Layer Security (TLS) is used. Then a certificate is read into memory, and after completing the 3-Way handshake associated with TCP, the AT and the broker will complete a TLS Handshake. As a result, I will mainly assess how TLS may affect the energy consumption in the setup section.

Compute The compute part of the application will mainly be assessed by performing a simulation of some computational work composed of a loop with a single `ADD` operation. The computational intensity will be varied by adjusting the number of iterations of the loop. The C-code, shown in Listing 4.1, is deliberately written to not be optimized by the compiler in order to ensure it will perform all the iterations, and the assumed compiled Assembly code is shown in Listing 4.2.

```
#pragma GCC push_options
#pragma GCC optimize ("O0")
void do_compute() {
  int result = 3;
  for (int i=0; i<CONFIG_COMPUTE_OPERATIONS; ++i) {
    result += result;
  }
}
#pragma GCC pop_options
```

Code listing 4.1: Compute section of main.c

```
do_compute():
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], 3
        mov     DWORD PTR [rbp-8], 0
.L3:
        cmp     DWORD PTR [rbp-8], 1023
        jg      .L4
        sal     DWORD PTR [rbp-4]
        add     DWORD PTR [rbp-8], 1
        jmp     .L3
.L4:
        nop
        pop     rbp
        ret
```

Code listing 4.2: Assembly code after compilation with gcc -O0 optimization

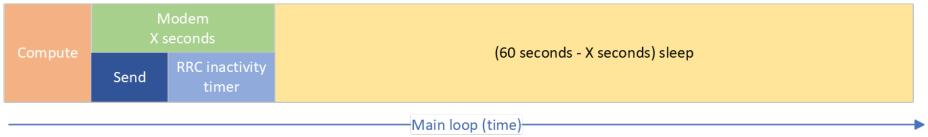
Modem The modem part of the application will be used to assess the cost of the send feature of a GATA, explained in Chapter 3. The section is called modem because the samples are categorized as modem only if the modem is on, and I have done it this way because I have determined that I want to assess the cost of compute based on the energy the AT consumes with the modem off, and the cost of send based on the energy the AT consumes with the modem on.

Sleep The sleep section of the application is when the CPU is idle and the modem is off. With this, I have presented how I will assess the different features of the GATA in my experiments. Setup will be used to asses the cost of connecting an AT to the network and server, compute will be used to assess the compute intensity, modem will be used to assess the cost of send (and essentially all communication), and sleep will be used to analyse sleep activity compared to the other sections. As mentioned in Chapter 3, compute, send, and sleep compose the Main loop of a GATA, and in Figure 4.2, I have illustrated a comparison between an ideal and a realistic iteration of the Main loop. I have included Figure 4.2 because it will explain why the sleep time may vary in the experiments, and how it correlates to the variations in modem-activity.

Now that I have presented the sections that will be used by the Framework in order to assess the application's energy consumption, I want to explain how I

analyse the power profiles in order to assess the different sections.

(a) An illustration of ideal sleep time of 60 seconds which would be realistic only if the DUT would be able to transmitt the message and turn off the modem before the *sleep* feature is initiated, as illustrated by Algorithm 1 in Chapter 3.



(b) An illustration of a realistic iteration of the Main loop, since it is unlikely that the message will be transmitted and the RRC inacitivty timer has finished before the application initiates the *sleep* feature. Therefore, it's more realistic to assume the sleep time will be 60 seconds minus the time the modem is reachable by the network.

Figure 4.2: An illustration of an ideal iteration of the Main loop compared to a realistic iteration of the Main loop.

4.2.1 Verbose Samples

When the Power Profiler (PP) samples the Asset Tracker (AT), it will also read a set of eight GPIO-pin states, as shown in Figure 4.3, and these pin states are used by the uAnalyser to determine what section the application is in at the time the sample is read by the PP. The uAnalyser uses four different pins and the current drawn in order to derive the application section, and four pins are used to debug the DUT both in production and in the post analysis. An overview of the pin states with its associated section is shown in Table 4.1, and the error codes will not be discussed. The more detailed explanation of how the PP is used will be given in Section 5.2.

I have labeled the four different pins Modem, MAIN, PIN_1, and PIN_2. If MAIN is off, then the application has not started. If the application has started and both PIN_1 and PIN_2 are off, then the application is in the setup section. If the setup section is finished, meaning the Main loop has begun and the Modem pin is on, then the AT is using the Modem. If both PIN_1 and PIN_2 are on, the Modem is off, and current drawn in the sample is below the average sleep current plus 3 times the standard deviation of sleep current, then the sample is counted as sleep. If none of the above are valid, then the application is in a compute section.

I ran a separate experiment to find the average sleep current and standard deviation of sleep current for the Device Under Test (DUT), and I want to present my findings below.

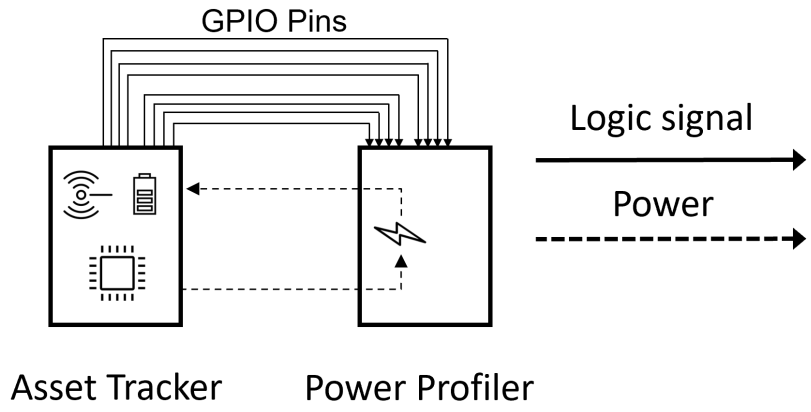


Figure 4.3: Illustration of logical pins connected to the Power Profiler (PP). Reading the pin states of the Device Under Test (DUT) while measuring the current, the PP is able to output the pin states together with the power measurements.

Modem	**MAIN**	**PIN 1**	**PIN 2**	**Description**
x	0	x	x	Application has not started or is finished
x	1	0	0	Setup
0	1	0	1	Compute
0	1	1	0	Compute
0	1	1	1	Compute (if $current > \overline{x} + 3 \times \sigma$)
0	1	1	1	Sleep (if $current \leq \overline{x} + 3 \times \sigma$)
1	1	0	1	Modem
1	1	1	0	Modem
1	1	1	1	Modem

Table 4.1: Overview of what GPIO-pins are associated to what application section.

When the application has started the four different GPIO-pins are used in order to derive what section the application is in for the given sample. In addition the current drawn is assessed in order to separate compute from sleep.

Sleep Experiment

It may be difficult to separate when the CPU is active from when it is idle using only the logical pins explained above. However, I have identified that using both the logical pins and the current drawn in the specific sample may provide enough information in order to accurately separate the two sections. Hence, in order to more accurately separate compute from sleep, I have conducted a simple sleep experiment to find a sleep threshold that can be used as an addition to the logical pins.

The code for the sleep experiment is trivial since I only want to measure the current drawn during sleep, hence the code will simply attempt to make the device sleep for the entire test. With this I can calculate the average current drawn and its variation. I conduct the experiment five times and end up with four tests with circa $4\mu A$ average current drawn and a standard deviation, σ, ranging from circa $5.1\mu A$ to $5.9\mu A$, and one test with average current of $\sim 8.848\mu A$ and σ of $\sim 11.423\mu A$. I end up choosing the latter result in order to calculate the sleep threshold. I calculate the threshold by adding $3 \times \sigma$ to the average, which is 43.117, and round up the sleep threshold to $50.0\mu A$.

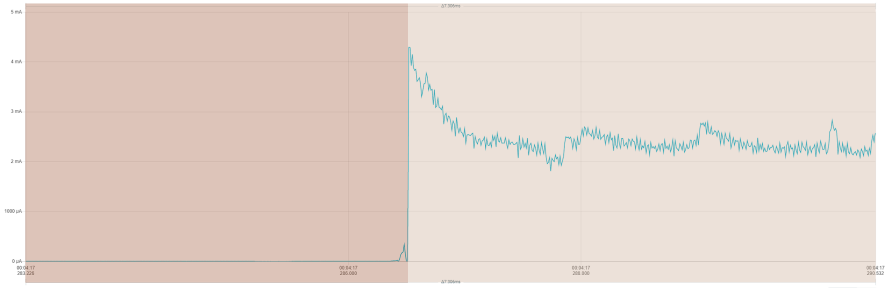


Figure 4.4: Power profile snippet to illustrate difference between sleep and CPU activity in terms of current drawn.

Sleep
Compute

The power profile shows that the current drawn whilst device is sleeping is significantly lower than when it is awake. When it is awake it draws well above $1mA$, and when it is sleeping it draws only a few μA. The power profile should be a confirmation that $50\mu A$ is a valid threshold, because if it's sleeping it should generally draw well under $10\mu A$, and if the CPU is active it draws significantly more than the sleep threshold.

I decided to go with the highest average sleep current because the Product Specification of the Power Profiler (PP) reveal that expected power consumption when the CPU is active is in the mA range, in contrast to μAs when the CPU is idle [13]. In addition I have included a power profile, Figure 4.4, that shows the difference. This is also the reason why I decided to round up to $50.0\mu A$ rather than choosing to strictly keep to $\overline{X} + 3 \times \sigma$.

Chapter 5

Experimental Setup

The experimental setup will give an overview of the platform I will use in order to run the experiments, and that includes what device I will use as an Asset Tracker (AT), an overview of the configurations that the device will parameterized with during the experiments, and what device was will be used for the power measurements.

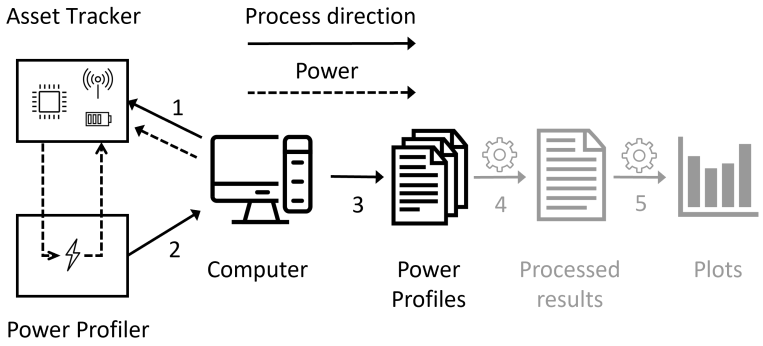


Figure 5.1: Experimental setup scope with regards to the GATA framework

This Chapter will give an overview of what platform is used for the Asset Tracker (AT) and the Power Profiler (PP) in the Figure, and will also give a specific list of configurations that will be used as parameters.

5.1 nRF9160 System in Package

The device that will be used as an Asset Tracker (AT) is the nRF9160 System in a package (SiP) provided by Nordic Semiconductor ASA. It is a fully integrated SiP for cellular IoT that supports the latest low power LTE technology and has access to multiple security features. nRF9160 also comes with an integrated modem that supports both LTE-M and NB-IoT with the power-saving features eDRX and PSM supported as well as TCP/TLS transport security. Both SIM and eSIM are supported for connection and authentication with mobile network operators, and the

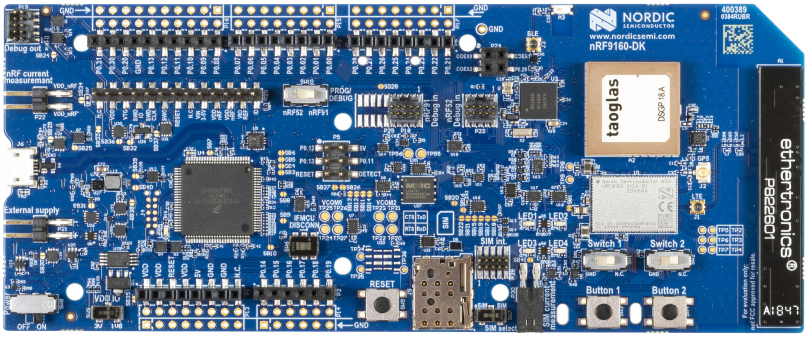


Figure 5.2: nRF9160 Development Kit. Picture credit: Nordic Semiconductor ASA

SIM-card that will be used is an iBasis card that is able to connect to both Telenor and Telia in Trondheim, Norway. Moreover, it comes with an Arm Cortex-M33 CPU which is optimized for cost and power sensitive applications developed for IoT, and provides 1MB of flash and 256 KB of RAM. Arm also provides the nRF9160 SiP with a rich security Platform Security Architecture (PSA) framework which include Arm TrustZone technology providing Security by Separation, meaning that the memory is separated into a Non-Secure partition and a Secure partition. Additionally it provides secure boot, trusted firmware updates and a Root of Trust implementation. The framework also includes Arm CryptoCell which provides security enhancements regarding cryptographic and security resources. Arm CryptoCell is designed to provide high performance cryptography solutions optimized for energy-constrained devices [13, 14]. In this thesis I will utilize the nRF9160 Development Kit (DK), see Figure 5.2, which is designed by Nordic Semiconductor ASA specifically to give customers a platform to evaluate the nRF9160 SiP and to use during the development process. The DK is equipped with all features that are necessary when developing and testing a cellular IoT application. The more relevant features in this thesis are the antenna, the easily accessed power supply pins, and its support for external devices through General Purpose Input Output (GPIO).

5.2 Power Profiler Kit II

Power Profiler Kit II (PPK2), shown in Figure 5.3, is an affordable and flexible real-time power consumption measuring tool designed and built by Nordic Semiconductor ASA. The PPK2 is equipped with an advanced analog measurement unit with a high dynamic measurement range. This enables us to perform accurate power consumption measurements for the entire measurement range that I expect the nRF9160 will work in, ranging from just a few μAs to a multiple of mAs. It comes with two measuring modes Source Meter Mode and Ampere Meter Mode. The Source Meter Mode supplies power to the development kit through

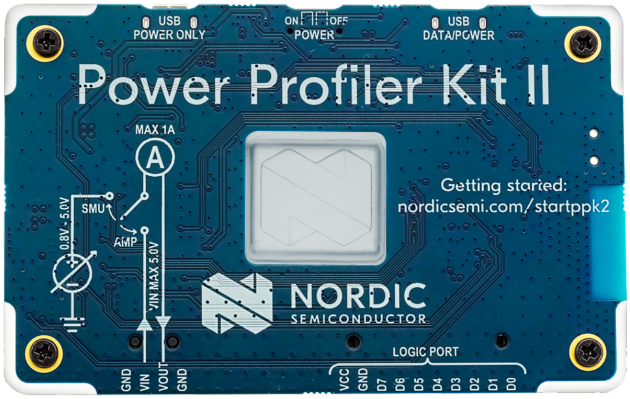


Figure 5.3: Power Profiler Kit II. Picture Credit: Nordic Semiconductor ASA

the PPK2, and allows the tester to set a specific voltage. The Ampere Meter Mode uses an external source for power and measures the power consumption by acting as an ampere meter [15].

In the experiments I will utilize the Ampere Meter Mode, because in order to run the automated power measurements enabled by the GATA Framework, explained in Chapter 4, I need to have the nRF9160DK connected to the computer in order to flash each of the test builds. A weakness to this setup is that I am not able to set the specific supply voltage to what I would like to, but the DUT is supplied by the computer through the USB that connects the nRF9160DK to the computer. I depend on knowing the supply voltage to be able to calculate the total energy consumption have therefore measured the supply voltage using a multimeter on several occasions. The multimeter shows that the computer supplies the DUT with 4.81 Volt.

In addition to provide us with a power profile the PPK2 have eight digital input pins labeled D0 through D7 [16], which can be seen in the lower right of Figure 5.3 under Logic Port, and was illustrated in Figure 4.3 in Chapter 4. With this, the PPK2 is able to read any of the GPIO pins that are found on the nRF9160DK, and map them to any of the digital input pins on the PPK2. This enables the possibility to review the state of each of the pins in parallel with the power profile. I am able to utilize this feature in the GATA Framework in order to analyse each sample individually by deriving the section from the pin state, as was explained in Chapter 4.

5.3 Application Code Parameters

The main goal of the experiments is to assess what affect different security metrics have on the energy-consumption of the Generic Asset Tracking Application (GATA), and I have run the application with four different security configurations. The name of the security configuration will be called protocol, and the baseline protocol is called **none** to indicate it has no added security metric. The next protocol is called **e2e**, which adds End-to-end encryption (E2EE) in the form of running encryption of the message payload using the symmetric key encryption AES Cipher Block Chaining, for each iteration of the Main loop. The third protocol is called **tls** and adds Transport Layer Security (TLS) to the application. And the fourth and last protocol is called **tls+**, and it includes both the **e2e** protocol and **tls** protocol. I have summarized the protocol parameters as a list below.

- **none** - Baseline protocol configuration, TCP without any added security
- **e2e** - End-to-end encryption of application message using AES CBC mode
- **tls** - TLS for secure communication between client and server
- **tls+** - Combination of **e2e** and **tls** configurations

In the parameterization I have also included configurations that vary the compute-intensity and the payload size in order to assess the different types of Asset Tracking application that was presented in Chapter 3. The parameters are selected in order to attempt to affect the different sections that was presented in Chapter 4.1. The compute parameters affect the number of iterations for the loop that was presented in Listing 4.1, and will be given by a power of 2, and on the form X-OPS, where X is 2^X. The different number of compute-intensities are:

- **10-OPS** - Performs 2^{10} number of iterations in the compute stage
- **15-OPS** - Performs 2^{15} number of iterations in the compute stage
- **20-OPS** - Performs 2^{20} number of iterations in the compute stage
- **25-OPS** - Performs 2^{25} number of iterations in the compute stage

The payload size will be provided in terms of number of bytes that the message contains, and will be a placeholder string that will be sent to the MQTT broker. The different number of payload sizes are:

- **256B**
- **708B**
- **1024B**
- **1416B**
- **2048B**

The protocol, compute, and payload size parameters will be provided in the project as separate configuration files that directly affect how the application is built. The script run by the computer in the GATA Framework, illustrated in Chapter 4, will combine all the configurations in order to create 80 different solutions. The experiment will consist of running each of the 80 solutions five times.

Chapter 6

Results

In this Chapter I will present the results after testing and analysing all the 80 configurations presented in Chapter 5. As a summary, the configurations are made up of four different security protocols, four compute-intensity parameters, and five different payload-size parameters, and they have all been combined in order to make up 80 different versions of the Generic Asset Tracking Application (GATA) that was presented in Chapter 3. As a reminder the different security protocols are as follows:

- **none** - Baseline TCP solution without any added security features.
- **e2e** - End-to-End encryption of message payload
- **tls** - Transport layer security, which means that all communication is encrypted between client and server.
- **tls+** - Both TLS and End-to-End encryption.

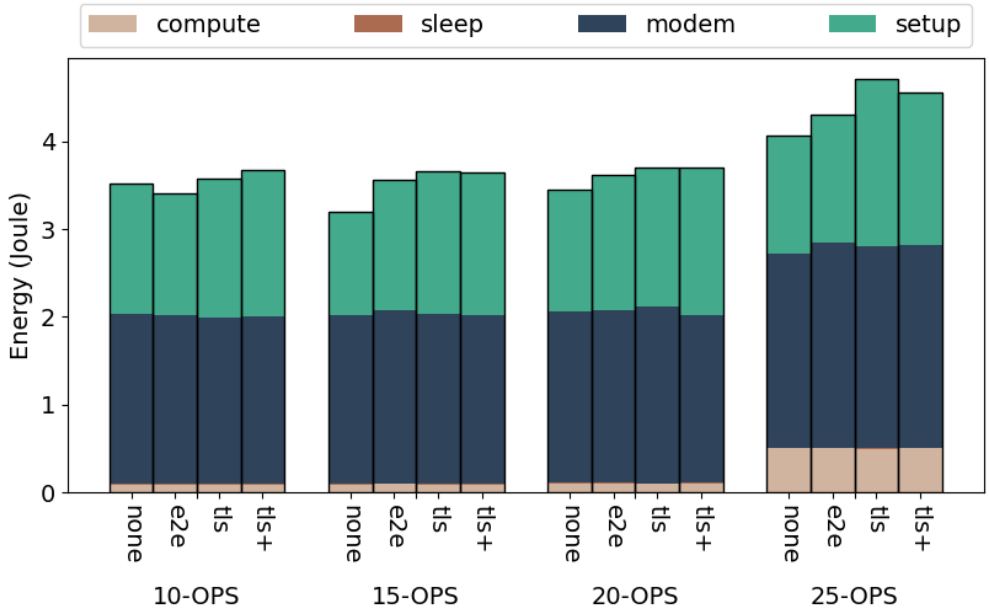
6.1 Results

Figure 6.1 shows the results of the different tests where payload-sizes for the given protocol and compute-intensity have been averaged in order to create a bar representing all the payload tests holding protocol and compute-intensity constant. What can be derived from the Figure is that there looks to be an increase in the total energy-consumption when adding security protocol. However, the increase seem to be located in the setup section of the application, and setup aside, it looks like the modem activity is inconsistent with regards to the different security protocols. It is therefore difficult to derive any pattern explaining the variation in energy consumption of the Main loop to the security protocols seen in Figure 6.1.

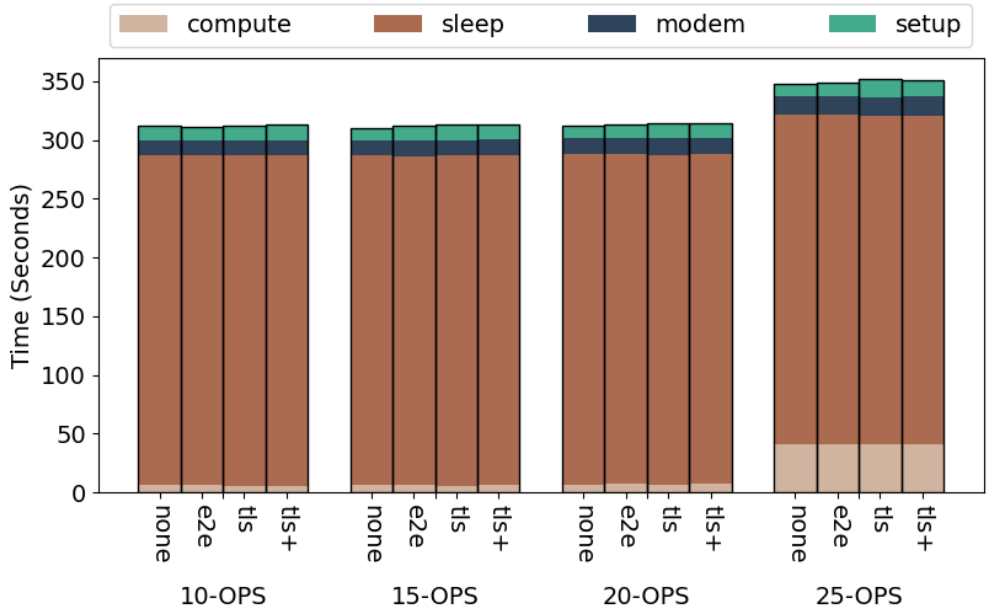
Figure 6.1 confirms that the Device Under Test (DUT) sleeps most of the time, but while sleeping it consumes little energy. In contrast modem activity is almost not visible in Figure 6.1b, which shows the time consumption, but takes up about half the energy consumption seen in Figure 6.1a. This means that the modem activity is significantly more expensive than the other sections in the Main loop of the GATA. In addition, it is possible theorize that the modem activity could overshadow the setup section if the Main loop was to continue more than the five loops that was measured in experiments.

The compute section in the more compute-intensive tests, with 25-OPS, still looks to be relatively small in comparison to the energy consumption in the modem section. Moreover, Figure 6.2 shows that energy consumption of the application's Main loop is not noticeably affected by the payload size. There is one outlier on **e2e** protocol for 1416B payload size, but may be explained by a resend due to a missed acknowledgement from the server, and is correlated to the lower sleep time and therefore higher modem time in Figure 6.2b.

With this, it looks like there is a small difference in total energy consumption when TLS is used, but looking at Figure 6.3 it does not look like there is any significant relationship between the protocol and higher energy consumption. Additionally, it looks like the 25-OPS configuration have increased the energy consumption in the modem section. A possible explanation is that the compute section immediately after setup is complete somehow restrict the modem to be turned off, and would explain why the modem is also on for a longer time in 25-OPS tests.



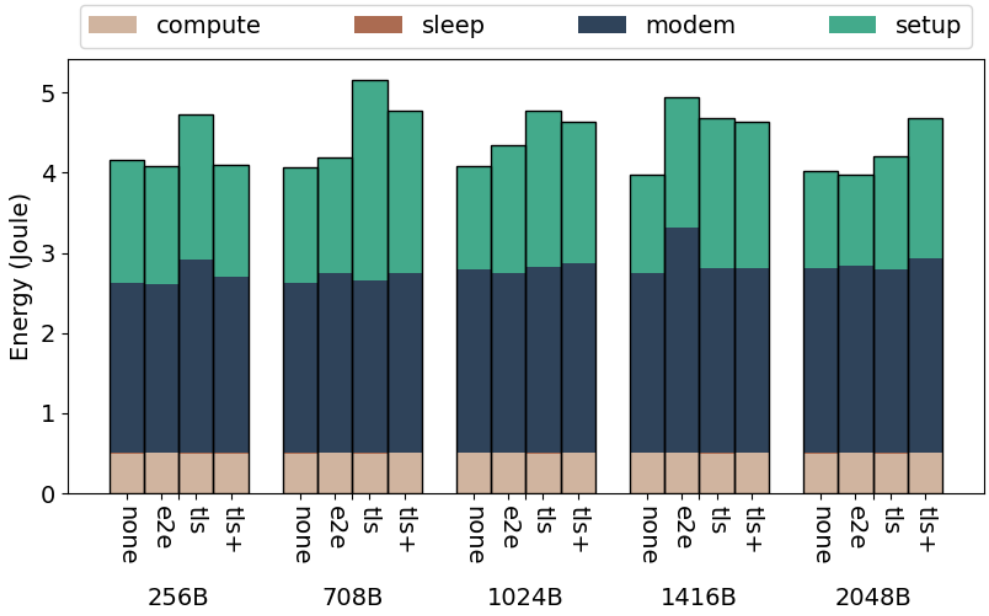
(a) Energy consumption



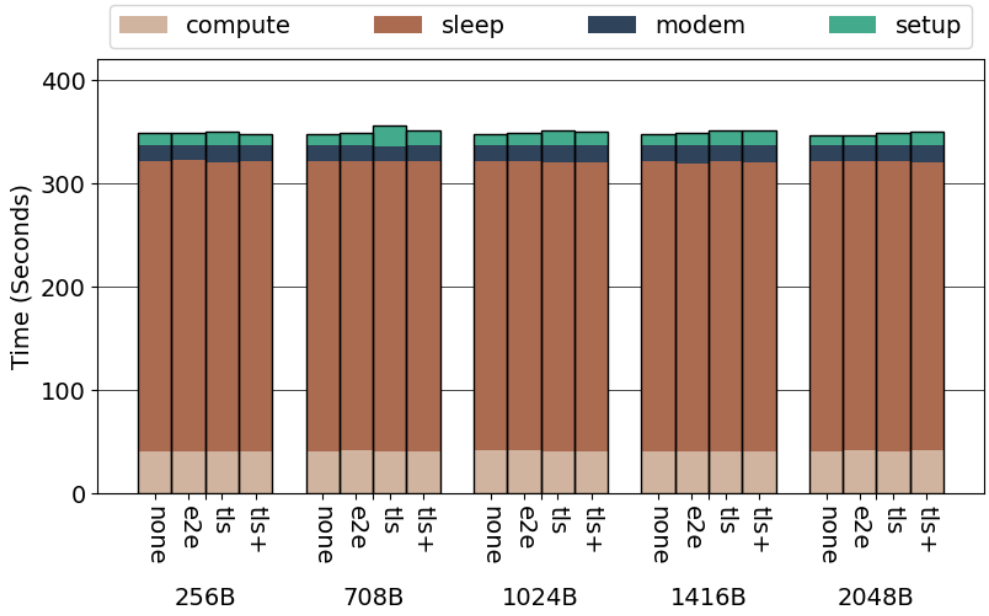
(b) Time consumption

Figure 6.1: Energy and time consumption of all results.

Each bar shows the average of all payloads for the given protocol. The results seem to indicate that none is generally the least expensive in terms of time and energy consumption. It also looks like the energy consumption is correlated to the time. Moreover, it seems like the most variability in terms of time consumption is found in the setup section. For lower compute intensity, 10-OPS specifically, it seems that the compute section has a significant impact on the total energy consumption. For all tests it seems that both modem and setup

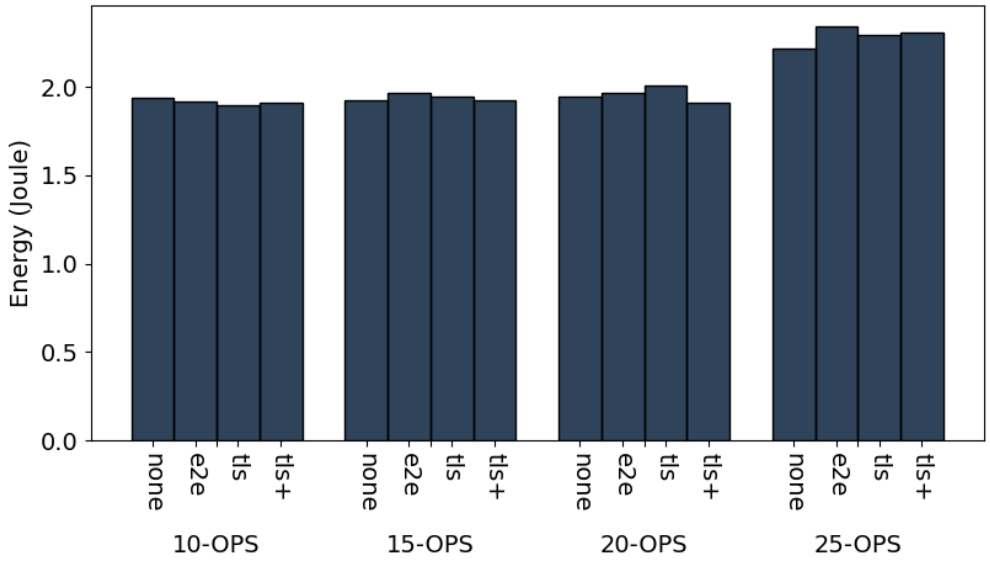


(a) Energy consumption

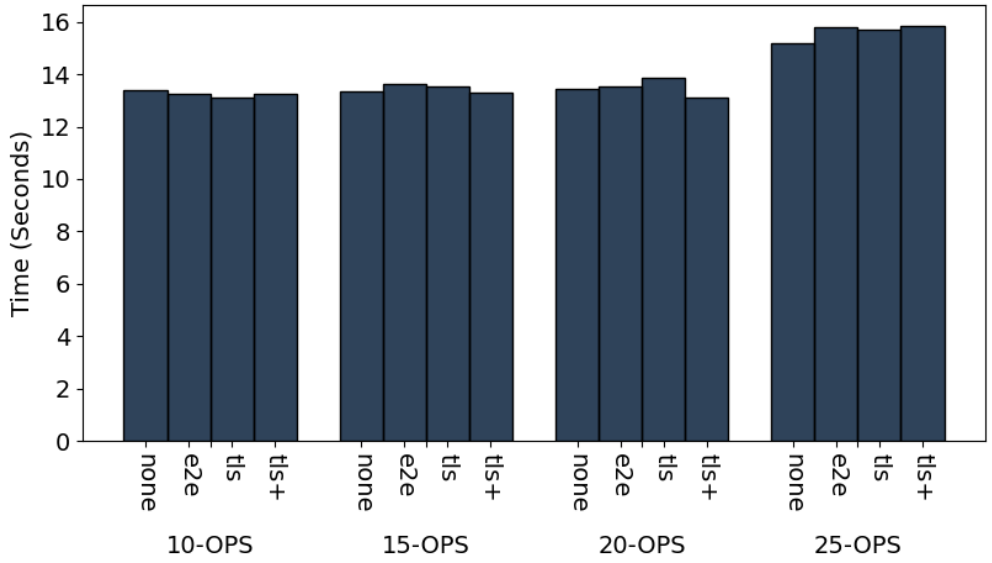


(b) Time consumption

Figure 6.2: Energy and time consumption for 25-OPS compute-intensity parameter.



(a) Energy consumption



(b) Time consumption

Figure 6.3: The energy consumed by modem.
The Figure is an excerpt from Figure 6.1a, but includes only the Modem activity.

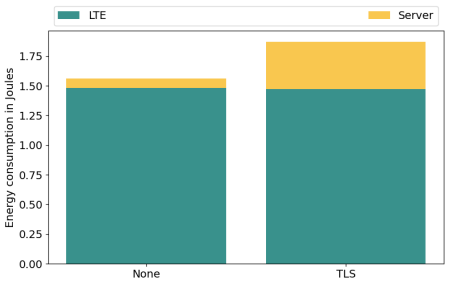
6.2 Setup Experiment

Having seen significant variation in the setup cost, both in terms of energy and time in Figures 6.1 and 6.2, I have decided to run an experiment with the purpose of explaining this variability. The goal will be to compare the **none** and **tls** security protocols, and I will look at both energy and time consumption of connecting to LTE and the Server.

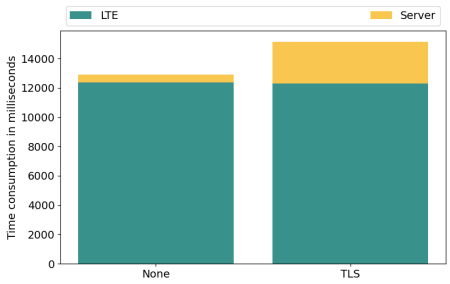
To conduct the experiment I have run the same process as described in Chapter 4, only with a new application code that has mapped the GPIO-pins differently in order to analyse the setup section specifically. I have labeled the subsection LTE and Server. The LTE section sets up the LTE-M connection with the network, and the Server section connects the AT to the MQTT broker. It is in the Server section where I expect to see a difference in energy consumption because TLS should not impact the process of establishing an LTE link.

After running the experiment 20 times with the **none** protocol, and 20 times with the **tls** protocol, the results show that connecting to the LTE network is significantly more expensive than connecting to the server, as shown in Figure 6.4a and Figure 6.4b. Additionally, it confirms that connecting to LTE is equally energy- and time consuming with and without TLS. The results also show that connecting to the server with TLS enabled is 479.57% more expansive than without TLS, in terms of energy consumption.

The setup experiment also showed a standard deviation of almost 9% in terms of the energy cost of connecting to LTE, which indicates that the variation in the cost of connecting to LTE is significant. This result helps explain some of the variations in the setup cost shown in Figures 6.1 and 6.2, and the reason why the protocols are not always a good indication of the cost of setup. This is because, as seen in Figure 6.4a, the cost of establish LTE connection is equal for **none** and **tls**, but you can assume the variation is the same. Furthermore, the cost of LTE is 371.79% more expensive than the cost of establishing connection to the server using TLS. I, therefore, believe that the expected variation in establishing LTE connection may potentially overshadow the cost of connecting to the server.



(a) The average energy it took to finish the setup section running 20 experiments wihtout TLS and 20 experiments with TLS.



(b) The average time it took to finish the setup section running 20 experiments wihtout TLS and 20 experiments with TLS.

Chapter 7

Discussion

7.1 Domain-Specific Hardware Acceleration

In Chapter 5, I presented the nRF9160 SiP used in the experiments, which comes with a Platform Security Architecture (PSA) framework. With the Arm CryptoCell that is utilized in the PSA all cryptography performed by the nRF9160 is hardware accelerated, which means that the encryption performed while having the End-to-end encryption (E2EE) and Transport Layer Security (TLS) protocols enabled is accelerated, and may therefore perform encryption significantly faster than platforms without Domain-Specific Hardware Acceleration (DSA) [17]. This means that I cannot claim that the Main loop of the Generic Asset Tracking Application (GATA) is more, less or equally energy consuming for any of the protocols, other than for platforms that use the same DSA used in the experiments. This is a weakness in my experiments, and if time had allowed it, I should have performed separate experiments that did not utilize DSA.

7.2 Theoretical Energy Consumption

In my experiments I have measured the energy consumption for approximately 5 to 6 minutes, which is an insignificant amount of time compared to how long a realistic Asset Tracker (AT) would run. Therefore, I claim that the cost of the setup section, that is presented in the Results Chapter, is unrealistically high when the AT is able to utilize Power Saving Mode and the keep-alive feature provided by MQTT. Hence, I have made an estimate of how much energy the different sections would consume, assuming that the Main loop runs a total of 5 minutes in the experiments, in Figure 7.1.

Provided that the AT is able to maintain connection to both the network and the server throughout a day, then the cost of the setup section looks to be negligible compared to the Main loop.

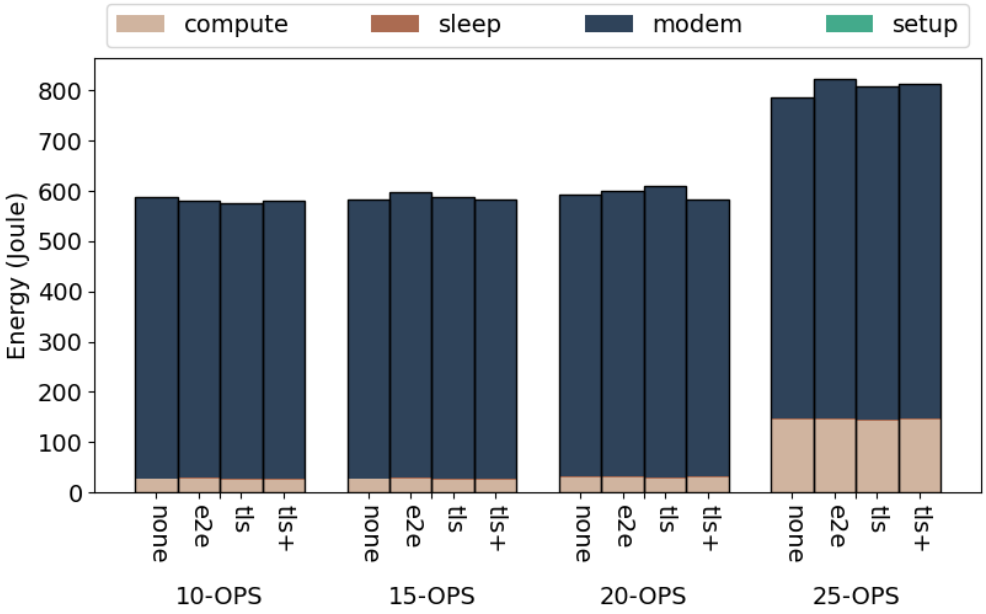


Figure 7.1: Theoretical Energy Consumption of approximately one day.
Assuming the Main loop uses 5 minutes I have multiplied all sections, aside from setup, by a factor of $\frac{60}{5} \times 24$ to approximate a day of actively running the Main loop without losing connection to the network or server.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this thesis, I have presented a Generic Asset Tracking Application (GATA) as one of my contributions addressing T1 in Chapter 3. I have developed and used a GATA Framework, which is presented in Chapter 4, in order to compare the energy consumption running the application with different security and performance parameters on an nRF9160 System in a package (SiP), successfully addressing T2 in the process. The GATA was implemented using LTE-M to access the network, and sent payloads to a server using MQTT over TCP. LTE-M was setup to utilize Power Saving Mode, and MQTT's keep-alive feature in order to avoid reestablishing connection every time the Asset Tracker (AT) woke up to send data.

The results shows that, using Domain-Specific Hardware Acceleration (DSA) for cryptography on a nRF9160 SiP, the difference in total energy consumption looked to be slightly higher when Transport Layer Security (TLS) was enabled. However, when setup was overlooked, there was no significant relation between the security protocol and an increase in energy consumption. Furthermore, there was little variation in energy-consumption of compute, and sleep consumed insignificant amount of energy compared to the other sections. Modem activity, on the other hand, had variations in energy consumption and looked to vary proportionally to the time consumption. The variation in energy consumed by the modem was still cloaked by the setup section in the results of the experiments.

Comparing the setup section with and without TLS, I found that connecting to the network using LTE-M was equally expensive for the two protocols, in terms of energy and time. Furthermore, connecting to the server was almost 480% more expensive using TLS, however, connecting to the network was over 370% more expensive than connecting to the server using TLS. The experiment also showed that connecting to the network had a 9% standard deviation in relation to the expected energy cost, which may also explain why it looks like the setup section is not always more expensive when adding the security protocols.

In the discussion I reason that theoretically one may assume that the setup cost is negligible, because the energy consumption of the Main loop would continue

to grow, whereas the setup cost may be a one-time cost when using the PSM and keep-alive features. Hence, assuming these power saving features, and that the AT is using DSA for cryptography, it is possible to reason that setup is negligible, and therefore that the security protocols I have evaluated in this thesis are negligible in terms of energy consumption.

8.2 Future Work

The first step that would improve the work of this thesis could be to run the same experiments without utilizing Domain-Specific Hardware Acceleration (DSA) for cryptography, which could give the work of this thesis significantly more impact with regards to scope and reliability. Additionally, comparing the results of this thesis with new experiments using Constrained Application Protocol (CoAP) or similar messaging protocols, could be helpful for developers that needs to make decisions based on energy consumption requirements. Furthermore, extending the GATA Framework in order to compare different LPWAN technologies, messaging protocols, and security protocols and map them to different application requirements, like the ones illustrated in Figure 3.3, could really help developers of AT applications to make better decisions in an early stage.

Bibliography

[1] 3GPP, *About 3GPP*, Available at https://www.3gpp.org/about-3gpp (accessed on 2022-05-27), 2022.

[2] 3. G. P. Project, *3GPP TS 36.331, Technical specification group radio access network*, Available at https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2440, version V17.0.0, 2022.

[3] GSMA, *LTE-M deployment guide v3*, Available at https://www.gsma.com/iot/wp-content/uploads/2019/08/201906-GSMA-LTE-M-Deployment-Guide-v3.pdf (accessed on 2022-05-12), Jun. 2019.

[4] GSMA, *NB-IoT deployment guide v3*, Available at https://www.gsma.com/iot/wp-content/uploads/2019/07/201906-GSMA-NB-IoT-Deployment-Guide-v3.pdf (accessed on 2022-05-12), Jun. 2019.

[5] M. Medwed, 'IoT Security Challenges and Ways Forward,' TrustED '16, p. 55, 2016. DOI: 10.1145/2995289.2995298. [Online]. Available: https://doi.org/10.1145/2995289.2995298.

[6] Z. Laaroussi and O. Novo, 'A Performance Analysis of the Security Communication in CoAP and MQTT,' pp. 1–6, 2021. DOI: 10.1109/CCNC49032.2021.9369565.

[7] P. Reininger, *3GPP Standards for the Internet-of-Things*, Available at https://dev.halberdbastion.com/sites/default/files/2017-06/2016_11_3gpp_Standards_for_IoT.pdf (accessed on 2022-05-16), Smart Summit, Singapore, Nov. 2016.

[8] N. S. ASA, *Online Power Profiler for LTE*, Available at https://devzone.nordicsemi.com/power/w/opp/3/online-power-profiler-for-lte.

[9] *Transmission Control Protocol*, RFC 793, Sep. 1981. DOI: 10.17487/RFC0793. [Online]. Available: https://www.rfc-editor.org/info/rfc793.

[10] *MQTT Specification*, Available at https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf, version v3.1.1, OASIS, 2019.

[11] E. Rescorla and T. Dierks, *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC 5246, Aug. 2008. DOI: 10.17487/RFC5246. [Online]. Available: https://www.rfc-editor.org/info/rfc5246.

[12] I. Grigorik, *High Performance Browser Networking: What every web developer should know about networking and web performance*. "O'Reilly Media, Inc.", 2013. [Online]. Available: `https://hpbn.co/`.

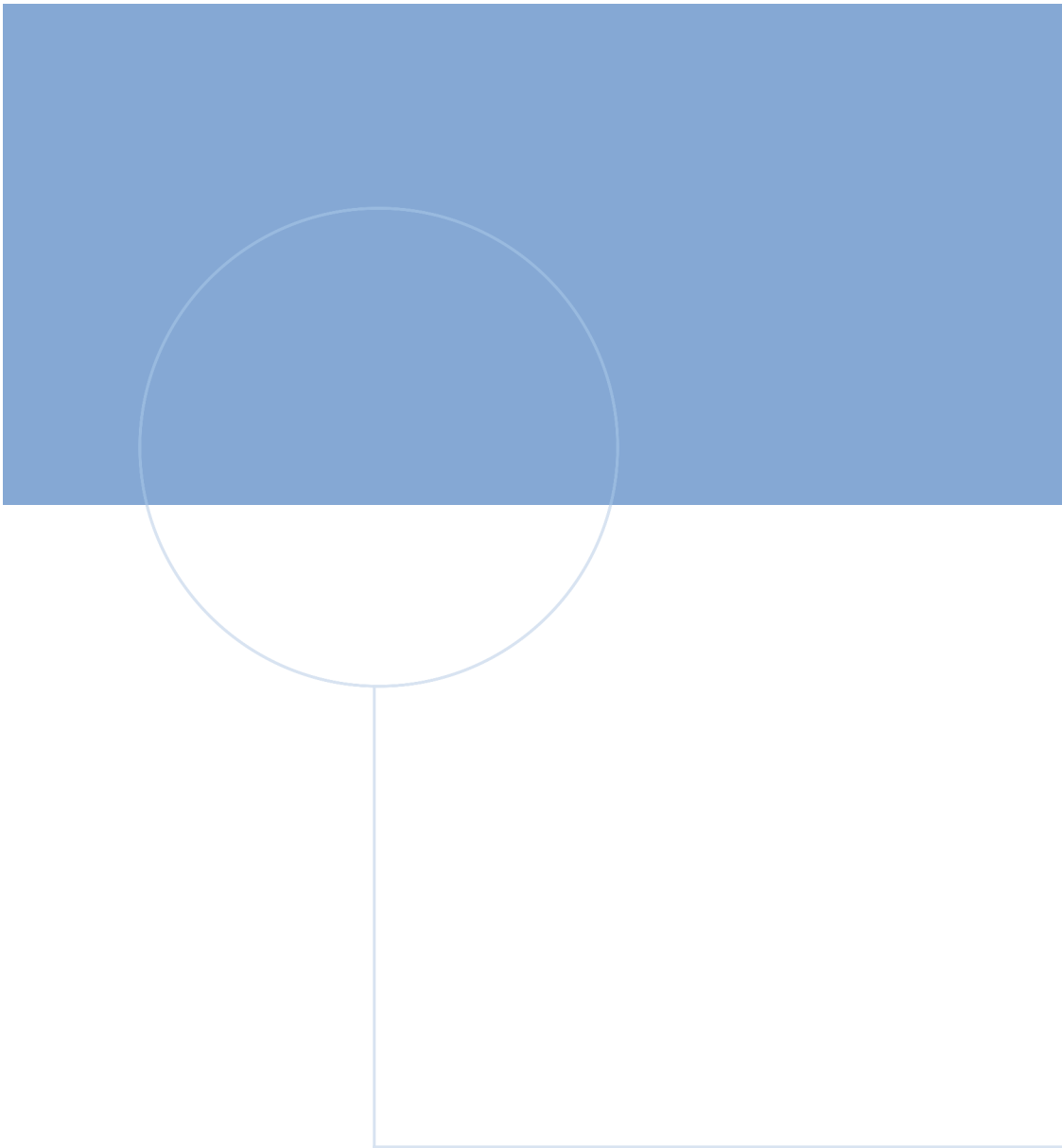
[13] *nRF9160, Product Specification*, version v2.1, 2021. [Online]. Available: `https://infocenter.nordicsemi.com/pdf/nRF9160_PS_v2.1.pdf`.

[14] *Arm Cortex-M33 Processor Technical Reference Manual*, version r1p0. [Online]. Available: `https://developer.arm.com/documentation/100230`.

[15] *Power Profiler Kit II, Power profiling and power optimization of embedded solutions*. [Online]. Available: `https://www.nordicsemi.com/Products/Development-hardware/Power-Profiler-Kit-2`.

[16] *Power Profiler Kit II, User guide*, version v1.0.1, Nordic Semiconductor ASA. [Online]. Available: `https://infocenter.nordicsemi.com/topic/ug_ppk2/UG/ppk/ppk_measure_accuracy.html`.

[17] W. J. Dally, Y. Turakhia and S. Han, 'Domain-specific hardware accelerators,' *Communications of the ACM*, vol. 63, no. 7, pp. 48–57, 2020.



NTNU
Norwegian University of
Science and Technology

NORDIC®
SEMICONDUCTOR