

# Bachelor Project Thesis

From energy measurement to big data

D. Hoonhout, M. Straatman, A.N.I. Tarcy,  
J.J.C. Vlekke, A.D. de Vos

# Bachelor Project Thesis

From energy measurement to big data

by

Douwe Hoonhout  
Martijn Straatman  
A.N.I. Tarcy  
Jimmy Johannes Cornelis Vlekke  
Adriaan Daan de Vos

to obtain the degree of Bachelor of Science  
at Delft University of Technology,  
to be defended publicly on Thursday 4 July 2019 at 14:00.

Project duration: April, 2019 – July, 2019  
Thesis committee: Dr. M. F. Aniche, TU Delft, coach  
Dr. H. Wang, TU Delft, bachelor coordinator  
H. James, Adviesgroep Strategisch Gebouwbeheer B.V., client

*This thesis is confidential and cannot be made public until July 4, 2019.*

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.

# Contents

<b>Preface</b>	<b>iii</b>
<b>Summary</b>	<b>iv</b>
<b>Acknowledgements</b>	<b>v</b>
<b>List of Figures</b>	<b>vi</b>
<b>List of Tables</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Structure . . . . .	1
<b>2 Problem Definition</b>	<b>2</b>
<b>3 Research</b>	<b>4</b>
3.1 Hardware . . . . .	4
3.1.1 Single-board computer . . . . .	4
3.1.2 Input Device: Serial Dongle . . . . .	6
3.1.3 Output Device: SIM Card Dongle . . . . .	7
3.1.4 Storage: USB Flash Drive . . . . .	7
3.2 Software . . . . .	8
3.2.1 Operating System . . . . .	8
3.2.2 Framework . . . . .	10
3.2.3 Programming language . . . . .	10
3.2.4 Data serialization . . . . .	11
3.2.5 Testing framework . . . . .	11
3.2.6 Mocking framework . . . . .	11
3.3 Web server . . . . .	12
3.4 Protocols . . . . .	12
3.4.1 Open Metering System (OMS) . . . . .	12
3.4.2 Advanced Encryption Standard . . . . .	15
3.4.3 Virtual Private Network (VPN) . . . . .	16
3.5 Databases . . . . .	17
3.5.1 MySQL . . . . .	17
3.5.2 OpenTSDB . . . . .	17
<b>4 Design</b>	<b>18</b>
4.1 Overview . . . . .	18
4.2 Challenges . . . . .	18
4.2.1 OMS Protocol . . . . .	18
4.2.2 Robustness . . . . .	20
4.2.3 Data Security . . . . .	21
4.3 Final Design Technologies . . . . .	22

<b>5</b>	<b>Implementation</b>	<b>23</b>
5.1	Overall Structure. . . . .	23
5.1.1	Supporting the OMS protocol. . . . .	24
5.1.2	Supporting a whitelist . . . . .	25
5.1.3	Reliability and data backup . . . . .	26
5.1.4	Other functionality . . . . .	27
<b>6</b>	<b>Process</b>	<b>28</b>
6.1	Planning. . . . .	28
6.2	Scrum . . . . .	29
6.3	Draft Report . . . . .	29
6.4	Software Improvement Group . . . . .	29
<b>7</b>	<b>Testing and Code Quality</b>	<b>30</b>
7.1	Unit and integration testing . . . . .	30
7.2	Risk analysis . . . . .	31
7.3	System and acceptance testing . . . . .	32
7.4	Static Analysis . . . . .	32
7.5	Code reviews . . . . .	33
7.6	Software Improvement Group . . . . .	33
<b>8</b>	<b>Discussion</b>	<b>34</b>
8.1	Evaluation of our product . . . . .	34
8.2	Recommendations . . . . .	35
<b>9</b>	<b>Conclusion</b>	<b>36</b>
<b>A</b>	<b>Project Description</b>	<b>37</b>
<b>B</b>	<b>Info Sheet</b>	<b>38</b>
<b>C</b>	<b>Software Improvement Group Report</b>	<b>40</b>
C.1	First Submission. . . . .	40
C.2	Second Submission . . . . .	41
<b>D</b>	<b>Ethical Evaluation</b>	<b>42</b>
D.1	Customer's Perspective. . . . .	42
D.2	Client's Perspective . . . . .	42
<b>E</b>	<b>MoSCoW requirements</b>	<b>43</b>
E.0.1	Description. . . . .	43
E.0.2	Naming conventions . . . . .	43
E.1	Must Have . . . . .	43
E.2	Should Have . . . . .	45
E.3	Could Have . . . . .	46
E.4	Won't Have . . . . .	46
<b>F</b>	<b>Gantt Chart</b>	<b>47</b>
<b>G</b>	<b>OMS Example</b>	<b>50</b>
	<b>Bibliography</b>	<b>52</b>

# Preface

This report consummates the Bachelor Project TI3806 course. To accumulate a Bachelor's degree in Computer Science and Engineering at the Delft University of Technology, it is compulsory to pass this course. The project, spanning a duration of ten weeks, is described in this report. The project client was Adviesgroep Strategisch Gebouwbeheer Nederland B.V. (ASG). This is a Dutch energy consultancy company based in Delft that uses big data innovations for energy management and for gaining insight into sustainability issues. The client encountered a real-life problem regarding the transmission of the data from their measuring devices. We were tasked to solve this problem. The project goal was to create a better data transmitter for the client. For naming convention purposes, we call the transmitter a gateway, as the device acts as a port between the measuring devices and the servers. It receives data from the measuring devices within its proximity and after filtering and processing this data, it is transmitted to the servers. The main objective was to design and develop a gateway that is better than the previously used gateway, the one that the client encountered numerous issues with. This improved gateway would then be deployed throughout the country such that the client enjoys numerous improvements. The most important improvements include cost reduction and a more robust, feature-rich and configurable solution.

Furthermore, the goal of this report is to inform the reader about the work that has been completed during the duration of the project at ASG. This report also contains possible future recommendations for this project. The objective of the future recommendations is to act as advice, guidance and instructions for the continuity of this project as the client might decide to further expand the solution by co-workers.

# Summary

Adviesgroep Strategisch Gebouwbeheer B.V. (ASG) requested five students from Delft University of Technology to create a custom gateway. This gateway is responsible for receiving the data collected by measuring devices. This data is filtered and processed and eventually sent to the servers. The custom gateway will replace the gateway that the client used to buy from a manufacturer. As this device is expensive, not robust and has limited functionality, it was not a suitable solution for the client. Therefore, to create a more optimal and suitable solution for the client, the custom gateway should solve these issues.

The custom gateway should surpass the gateways currently found on the market. To accomplish this, something unique has to be created. This means that the custom gateway must be much more robust, reliable, feature-rich and relatively cheaper compared to the solutions found on the market today. The client highly values an improved and custom gateway, as their core business will gain a significant boost from all the benefits that the custom gateway brings. This includes a cost reduction that allows the client to improve their services for their customers without having to charge them more. It is expected that the use of the custom gateway will contribute to an improved market position and an increase of market share for ASG.

The project consisted of two main phases; the research phase and the development phase. During the research phase, the client introduced the bottlenecks of the gateway currently in use and explained their desired solution. Research was done on the technologies involved in the gateway, such as the Open Metering System (OMS) protocol used by the measuring devices. The requirements for the development of the custom gateway and a clear project schedule were formulated after combining the client's interviews with the conclusions from the research phase. These documents formed the building blocks of the development phase. During the development phase, the SCRUM developing methodology was followed.

As an important requirement for our custom solution was for it to be robust, it was important to thoroughly test it. An extensive suite of automated unit and integration tests was written and comprehensive system testing was performed. Furthermore, vigorous system testing was conducted: the client installed our solution in a customer's building in [REDACTED], allowing it to be effectively tested in a real-life scenario for 4 weeks. Load testing was also performed at the office with over 1000 measuring devices, which our solution consistently processed successfully. The system testing was combined with acceptance testing and the client was very happy with the quality of our solution. The client plans to replace their existing, old gateways with our solution. As our solution is three times cheaper than the client's old gateway, significant financial savings will also be made. Furthermore, the client plans to reuse parts of our software in future projects.

# Acknowledgements

This project owes its accomplishment to many people. Many thanks go to all the people that helped us realise this. Especially our client Humphrey James, the CTO of Adviesgroep Strategisch Gebouwbeheer Nederland B.V., who made the proposal for this project, was always available to help us, guided us through the development process and gave us the freedom to experiment. Ronald van Westering, the owner of 1FOCUS, supported us in creating and maintaining our development planning, and he provided additional comments to make our software development process more efficient. And Patrick Liefink, for explaining the steps required to set up OpenVPN tunnels and provided us with a script for setting up these tunnels. Additionally, other employees of Adviesgroep Strategisch Gebouwbeheer B.V. were kind enough to help us. Moreover, our coach Mauricio Aniche mentored us and helped us by reviewing our report and answering our questions.

# List of Figures

3.1	Basic message OMS message. Retrieved from wireless dongle [3]. . . . .	13
4.1	OMS Protocol Class Diagram . . . . .	19
7.1	C# Code Coverage by Automated Tests . . . . .	31



# List of Tables

3.1	Structure of the Data Link Layer . . . . .	13
3.2	Structure of the Application Layer . . . . .	15

# Introduction

With diminishing energy sources [10] and increasing global warming [23], it is becoming more important to get insight into utility (water, electricity and gas) consumption. Reliable data aggregation creates extensive insight into current utility consumption and will allow people to create and deploy targeted solutions to reduce utility consumption and improve sustainability of energy and utility sources. Minimizing utility consumption is environmentally friendly and cost-efficient. Adviesgroep Strategisch Gebouwbeheer Nederland B.V. (ASG) is our client. ASG is a Dutch energy consultancy company based in Delft that uses big data innovations to gain and provide insight into utility consumption in buildings and apartment complexes. It also provides consultancy to improve utility consumption, which in turn, improves sustainability and cost-efficiency. This is done by collecting data, with a high frequency, from measuring devices in buildings of their customers and then analyzing the incoming data. To collect the data, ASG currently uses a gateway that aggregates data from measuring devices and sends this to their servers. However, this gateway is quite expensive, unreliable and has limited functionality, which is why ASG requested the design and development of an improved gateway. ASG aims to improve its business, market position and market share by using the new custom gateway.

This report contains the documentation of the research and development process of this project including the design choices that have been made for the final solution.

## 1.1. Structure

Firstly, the problem definition will be formulated in chapter 2. Chapter 3 contains our preliminary research on optimal hardware and software choices. This chapter also covers research on the protocols, which include the Open Metering System protocol for the data that is sent by measuring devices, and encryption standards. Chapter 4 will define our design choices based on the research and challenges that we encountered. Chapter 5 will discuss the implementation and structure of the code. Our development process will be discussed in chapter 6. Our code quality and software testing process is described in chapter 7. Finally, chapter 8 covers the discussion and provides future recommendations and chapter 9 will cover the final conclusion.

# 2

## Problem Definition

The goal of this project is to develop a robust and scalable solution that collects energy measurement data on a large scale and sends it to a database. This custom solution is developed in cooperation with the client Adviesgroep Strategisch Gebouwbeheer Nederland B.V. (ASG). There are a lot of solutions on the market that can receive data from measuring devices and process and send the data over the internet to a server. But when this needs to be done simultaneously in near real-time for over 65,000 measuring devices in various locations that send multiple measurement values, a new, custom solution is required.

Currently, the client collects data of many water, gas and electricity measuring devices. However, the problem with the current implementation is that the devices responsible for sending the data, called gateways, are very expensive and do not satisfy the client's needs in terms of reliability, scalability, consistency, durability, performance and functionality. Some problems of the current gateway include; it cannot be restarted remotely, it can only receive data from a limited number of measuring devices at the same time, it does not automatically restart and it does not allow incoming data to be filtered and temporarily stored. The objective of this project is to design and develop a solution that is more cost-efficient, more robust, more reliable, more consistent, more durable, has a better performance, and has more functionality. To create a solution that suits the client's needs, it is important to select appropriate hardware and write reliable software. Our solution must act as a receiver of the signals of various measuring devices within its proximity. Making smart use of encrypting and decrypting data is essential, as the data is privacy sensitive and must be encrypted by law [11]. Furthermore, our solution needs to allow for basic remote commands, such as a command to allow for the system to be restarted. All the received measurement data needs to be filtered, the data that passes the filter needs to be processed and sent to a server in near real-time, where the data is fed to a time-based database to be further processed. The data that does not pass the filter will be temporarily saved on the gateway for data recovery. To ensure data integrity, data validation is necessary. The gathered data is used to measure real-world energy sustainability solutions and their performance level in the specifically measured situations. The gathered data is also used to analyse the energy usage efficiency of an entire building, but also of individual apartments, rooms and offices. This analysis enables customers to determine the exact consumption of each individual user, room, office or building for billing and to make evidence-based changes to improve

energy and cost efficiency.

All the collected data will result in an analysis that will help determine and improve the effectiveness of energy usage in a given environment. Our solution will act as the gateway between the measuring devices and the client's servers.

# Research

### 3.1. Hardware

### 3.1.1. Single-board computer

First of all, the hardware is required to have a relatively small form factor that can easily be placed in a casing and installed in a building's fuse box. Moreover, the computer needs

multiple USB ports to connect several devices such as the serial dongle that will receive the data of the measuring devices nearby, a SIM card dongle to send the data to the servers of the client, and a flash drive for reliable data storage. Additionally, the storage space on the computer should store the operating system and the program files. The data it receives from the measuring devices will be placed on the USB flash drive. The computer needs to be able to store this data on a permanent location to create a buffer as it might happen that the data cannot immediately be sent to the server. This buffer will prevent data from being completely lost. Thus, the computer preferably needs at least several gigabytes of storage space. Another important factor is that our solution will be produced hundreds of times because almost a thousand of these products will be deployed in the Netherlands. Thus, the price of the hardware needs to be taken into consideration too. Additionally, the solution will process a lot of data simultaneously and thus needs relatively high processing power. Therefore, the decision was easily made to use an SBC as these computers are relatively small, have several input and output ports, are available with storage of at least several gigabytes, are relatively cheap and can handle a lot of data.

[illegible]

[illegible][illegible]

### 3.1.2. Input Device: Serial Dongle

The serial dongle that will be used is the [REDACTED]. There are just a few serial dongles on the market that support the Open Metering System protocol, so that made the decision easier. This dongle is already in use by the client's mechanics and they have not reported any problems with it so far. This dongle allows the SBC to receive the wireless serial data generated by the measuring devices. It can easily be connected through USB and it supports an additional antenna to increase the range of the device. The basic range of the dongle is 100 meters according to the specification. However, in practice this range is less because it does not account for walls or other structures between the measuring device and the dongle. The first estimate was that a range of 100 meters would be enough because these devices are only used in close range and in high density areas such as flats. System testing has shown that 100 meters was not enough, so the optional antenna was used to increase the range to 800 meters. Moreover, the [REDACTED] USB Dongle supports the reception of data from measuring devices that have different operating modes [3]. This increases the compatibility of the dongle with the broad range of measuring devices that need to be supported. One important thing to note is that the serial USB dongle will receive many serial signals from all measuring devices within range, also the ones from competitors. That means a white-list will be implemented in a later stage to

ignore messages that are not from the client.

### 3.1.3. Output Device: SIM Card Dongle

The SIM card or 4G dongle that will be used is the [REDACTED]. This dongle was chosen because the embedded router functionality would make the dongle operating system independent. This allows the client to be flexible with their hardware and software if they want to make changes to it in the future. It also allowed for quick development because there was no additional configuration required in the operating system and it works out-of-the-box. The data collected by the SBC has to be sent to the servers of the client for analysis and storage. The SBC cannot depend on the client's customers to provide a stable wired or wireless internet connection. A SIM card dongle provides internet connectivity via a telephone provider. The [REDACTED] network is relatively stable, but there have been outages in the last few years [20]. This means that we need to build software that is resilient in case of internet outages. The SIM card dongle is connected to the single-board computer with USB.

### 3.1.4. Storage: USB Flash Drive

The USB flash drive that will be used is the [REDACTED]. The client requested a very durable USB flash drive that has a small form factor. Research has shown that the [REDACTED] is one of the best tiny USB flash drives [21]. As mentioned in the section about the SBC, using the SD-card to store all incoming data before it will be sent to the servers is a bad idea because it is not durable enough to work for 10 years. Many SBC users report issues with their SD-card durability. Therefore, it has been decided to use a USB flash drive to store the data and not use an SD-card at all. Even though a USB flash drive also uses flash storage, just like an SD-card, the experience of the client and other SBC users with the durability of a USB stick is better [6, 28]. Additionally, USB sticks have less electrical connections that could cause problems compared to SD-cards [28]. Moreover, might the durability of a USB flash drive turn out to be mediocre, it can easily be replaced. And, in order to increase the durability of the USB stick, it is important to use a USB stick with a lot of memory such that the wear levelling has more media to operate over and the USB stick should be of high (industrial) quality [28]. Furthermore, the USB stick does not need to be taped to the SBC to avoid it from falling off due to possible vibrations. An SD-card, however, needs to be taped to the SBC to avoid it from falling off. Additionally, The [REDACTED] is water resistant, shock resistant and x-ray resistant, ensuring it is suitable for use in the various environments the solution will be used in.

Flash memory endurance was an important consideration when taking the memory into account, because the gateway is expected to run correctly for 10 years. The endurance of flash memory is based on multiple factors. The most important factors are: The type of flash memory, the temperature of the flash memory and the size of the flash memory. The USB Flash Drive that was chosen uses Multi-Level Cell memory [34]. This type of memory is less durable than Single-Level Cell memory. It is also believed that Single-Level Cell memory is more reliable and has a lower amount of bit errors, but a study [12] shows that this is not the case. Multi-Level Cell memory is also considerably cheaper, which is ultimately the reason that a Multi-Level Cell device was chosen over a Single-Level Cell. To validate that the flash memory will work for 10 years, we made a calculation based on the memory usage of the gateway during the development phase and the expected number of write cycles of



the memory, which is about 1000 expected write cycles in the worst case [34].

Since the chosen USB Flash Drive has 32 GB of storage, the expected total number of bytes that can be written is 32 TB. During the development phase, we found that every message that our gateway can correctly receive and decode will write 1 kB to the flash drive. Generally, measuring devices send messages every 20 minutes, meaning that every measuring device will cause 3 kB of data to be written every hour. In a worst case, about 250 measuring devices will send their data to the gateway. Therefore, it is expected that the gateway will write the following amount of data in 10 years:

$$3kB * 250 * 24 * 365 * 10 = 65,70GB$$

Our solution will also keep track of incoming raw data of some devices and the received measuring devices and when they were last received. This does not take up as much memory as the correctly incoming messages and are therefore left out of the calculation. By taking a broad assumption of the memory usage, we expect that about 100-150 GB of data will be written to the flash drive over 10 years in the worst case. This amount of data does not come close to the expected 32 TB of storage that can be used. Therefore, the flash drive durability should not be an issue. There are some external factors that have not been taken into account, which may negatively affect the durability, like temperature and unexpected power loss. Therefore, the durability may turn out to be lower than expected, but that cannot be predicted yet at this point.

### 3.2. Software

In this section, we research and compare different viable choices of software for our solution, including operating systems and programming languages.

### 3.2.1. Operating System

We have chosen to use the [REDACTED] operating system. In this section, we outline our research into various viable operating systems and compare them.

## Windows 10 Internet of Things

[illegible]

compared to the others and is slow to start up and use. Additionally, the added benefit of a clear user interface in Win10IoT - at the cost of performance - is negligible in the context of our solution.

### Ubuntu Core

The goal of our solution is simple, receive data from measuring devices and send data to servers. Many existing operating systems provide a plethora of functionality that is unneeded in our use-case. Making an embedded device would be best, but that is out of scope for this project. The second best option would be to choose an operating system that provides the bare necessities to run our code, has a small memory, power and performance footprint, and allows for easy development. Ubuntu Core fits exactly these needs. It contains just the kernel and a packet manager. It is up to the developer to install the tools needed to run the code. Over 70% of the current IoT devices run a version of the Linux operating system [8]. The client uses Windows and Mac devices, so they might not be very familiar with a Unix based operating system. However, that should not be a problem because the application code is platform independent, and the configuration of the operating system will be relatively minor. The configuration of the operating system will be documented and preferably automated. This operating system focuses on security, reliability and gives complete control of the update process. Ubuntu Core has a memory impact of 85 MB for the operating system and the install image is 500MB.

### Ubuntu Server

Ubuntu Server is also an OS we looked in to as an alternative to Ubuntu Core. It is similar to Ubuntu Core, but it has added functionality that makes it easier to have the system act as a server. For this to work, the system needs access to the internet, though. In practice, the system that we will be developing will not have access to the internet and the server functionality will not be needed. Also, our solution cannot rely on a constant internet connection as this will make the solution less robust and data could be lost if the system relies on the internet and there is no connection. Because of this, we decided that Ubuntu Server would not be a suitable alternative to Ubuntu Core.

[REDACTED]

[REDACTED] [REDACTED]. [REDACTED] [REDACTED] [REDACTED]  
[REDACTED] [REDACTED] [REDACTED] [REDACTED]. [REDACTED] [REDACTED] [REDACTED] [REDACTED]  
[REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED]. [REDACTED] [REDACTED] [REDACTED]  
[REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED]. [REDACTED] [REDACTED] [REDACTED]  
[REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED].

### 3.2.2. Framework

In this subsection, we will describe the frameworks we have considered for our solution.

## .NET Framework

We have chosen to use the .NET Framework because it is developer-friendly and works well with our chosen operating system. Furthermore, it also works optimally with our chosen method of encryption, as we will discuss later [7]. Additionally, should the client ever choose to change the operating system to Win10IoT, the .NET framework allows for optimal compatibility. Moreover, the version of the .NET Framework that will be used is 4.7 this is, at the time of writing, not the most recent version. The reason for this choice is that this was most convenient for all the programmers.

Note that in order to run a .NET Framework program on Linux, Mono should be installed on the Linux device.

## .NET Core

We also considered using .NET Core because it allows for the creation of a multi-targeted .NET Core project that targets multiple frameworks. "This is a great option because a single project is able to compile for different frameworks. It also has the power to handle different compilation options and dependencies per targeted framework" [13]. The most preferable version of .NET Core to be used would be version 3.0 which is, at the time of writing, the most recent version of .NET Core. However, this version is a preview build and therefore not a suitable option for developing a program that must be robust. The other version of .NET Core that could have been a possible option was .NET Core version 2.2 but this version of .NET Core is not compatible with the serial port class. This class is required to read data from the serial input device and therefore also version 2.2 and lower of .NET Core was not an option. Therefore it was decided that .NET Core would not be used as the framework.

### 3.2.3. Programming language

In this subsection, we will describe the considered programming languages that are supported by the chosen code framework. The programming languages that are supported by .Net Framework are C#, F#, Visual Basic and C++. Visual Basic and F# were not considered because these programming languages are not known by any of the programmers. Therefore the decision between C# and C++ was made.

## C# or C++

As a programming language, we have chosen C#. We compared both C# and C++. C# is a powerful, object-oriented language and all team members are familiar with the syntax. Furthermore, memory management is done automatically and it supports garbage collection. Additionally, C# is the higher-level language of the two. Therefore, the programming language we have chosen to use is C#. An added benefit of this choice is that it supports a number of excellent testing frameworks.

### 3.2.4. Data serialization

To store various data on the SBC, such as encryption keys for measuring devices and general configuration settings, a data storage format is used to serialize data. We have chosen to use XML.

#### JSON

One option for data serialization is JSON. This popular format is lightweight and quite easy to understand. However, it is best suited to be used in dynamically typed languages, which is unsurprising given that JSON is based on Javascript. As C# is traditionally a statically typed language and does not include support for JSON other than through external libraries, we felt that JSON was not best suited for our needs.

#### XML

Another option for data serialization is XML. XML is also lightweight, although slightly less so than JSON. However, XML is arguably far easier to understand for a person with a limited technical background than JSON. Furthermore, the .NET framework offers excellent built-in support for XML. Therefore, we have chosen to use XML for data serialization and storage on the SBC.

### 3.2.5. Testing framework

To test our code, we should use a testing framework that is compatible with the chosen framework and programming language. We have chosen to use NUnit.

#### MSUnit

The first natural choice for a C# testing framework is MSUnit. It comes with Visual Studio and offers deep integration with Visual Studio. However, one of our development machines runs Linux and thus MonoDevelop. Therefore, it is impractical to use MSUnit as it does not integrate properly with MonoDevelop.

#### NUnit

We have chosen to use the NUnit framework to test our code. NUnit is open-source and more feature-rich than MSUnit. It supports fluent Assert syntax, that is `Assert.That`. However, the primary motivation for selecting NUnit is that it integrates well with both Visual Studio (Windows) and MonoDevelop (Linux).

### 3.2.6. Mocking framework

To test our code, it is important to use a mocking framework for better, more isolated unit tests. The mocking framework must be compatible with our previous choices. We have chosen to use Moq.

#### Microsoft Fakes

Microsoft Fakes is Microsoft's own mocking framework, with support for stubs and shims. Shims allow compiled code to be modified to mock third-party assemblies. Stubs require your own code to implement an interface [14]. However, it is not compatible with MonoDevelop (Linux). Moreover, it requires the costly Enterprise edition of Visual Studio.

**Moq**

We have chosen to use Moq as our mocking framework. Moq is free and open-source [15] and integrates with both Visual Studio and MonoDevelop. It supports mocking virtual methods and properties, as well as interface implementations. Unlike Microsoft Fakes, though, it doesn't support shims, meaning that static and non-virtual members cannot be mocked. Therefore, it is important that our code is written with testability in mind. For our purposes, Moq is ideal.

**3.3. Web server**

We have chosen to use Apache as our web server software. The reason for this is simple - it is nicely compatible with the OS and requires very little setup to get working. Furthermore, it is widely used and documented and is more than robust enough for our purposes.

**3.4. Protocols**

In this section, we research various protocols that we can use for our solution, including the Open Metering System (OMS) and the Advanced Encryption Standard (AES).

**3.4.1. Open Metering System (OMS)**

For the communication between measuring devices (electricity, gas, water) a common standard is used, called Open Metering System (OMS). This standard was driven by the European Union. They desired for the creation of a common standard that satisfied all the rules and requirements of the European Union. They formed a group of specialists from measuring device manufacturers, manufacturers of communication devices, energy suppliers, communication companies and scientific institutions and it resulted in the Open Metering System. It is vendor independent and is now widely used in Europe. Since this is the standard used in measuring devices, we cannot deviate from it [19]. There are manufacturers that created their own protocol but we will not support this. In this section, we will explain the OMS protocol in our own words since the original documents are very technical. Note that there will always be spoken about hexadecimal number instead of writing out the whole byte-string. That means that instead of writing 0011 1100, 3C will be written.

**Serial communication**

For communication between measuring devices, serial communication is used. In this communication type, information is sent or received one bit at a time. This technique is very old but is still used for a lot of applications, like Ethernet. In order to receive the serial communication of the measuring devices, an [REDACTED] [REDACTED] [REDACTED] USB dongle is used [3]. This USB has some extra functionality with respect to receiving serial data. First of all, it puts a [REDACTED] [REDACTED] in front of each message. This makes it easy to determine the beginning of a message which would otherwise be very hard to find. Also, the USB automatically checks for errors. This error detection is done by a 16-byte cyclic redundancy check following the requirements set in [REDACTED]. This way of detecting errors works by dividing a block of 16 bytes by a polynomial that is used for all parties. The remainder of this division will then be added after the 16 bytes before sending. If the receiving party now does the same 16-byte division with the same polynomial, it should have the same remainder as the remainder that was added just after the 16 bytes. During the project, this CRC-16 check

Length n+10 (1 byte)	Block 1 (9 bytes)	Payload (n bytes)	RSSI (1 byte)
----------------------------	----------------------	----------------------	------------------

Figure 3.1: Basic message OMS message. Retrieved from wireless dongle [3].

was implemented in case the client wants to switch to another wireless m-bus device. One last functionality that was found to be very useful is the RSSI value. This value is added to the message as the last byte. The RSSI value represents the weight of the link between the gateway and a measuring device. If the client installs multiple gateways in a building, this value can be used to determine which gateway should process the outgoing signal of a measuring device. In figure 3.1 the basic structure of a message is shown. However, the starting bytes are not shown here and should come before the length byte.

### Basic structure

In this section, the basic structure of the OMS-protocol will be explained. After the basic structure is clear we will go into more detail for each specific layer of the protocol. G contains an example of an OMS message and can be used to make this subchapter more clear. In each message, three layers are mostly used. There are two additional layers that can be used but these are not within the scope of our project. The three layers that are always represented in a message are the Data Link Layer, the Transport Layer and the Application layer. The Data Link Layer contains information about the sender of the message. This layer corresponds to the first 2 sections in Figure 3.1. The Transport Layer contains information about the message and the Application Layer contains the actual data of the message. These layers correspond with the 3<sup>rd</sup> block in Figure 3.1.

### Data Link Layer

The Data Link Layer is the first layer of bytes in a message. This layer always consists of ten bytes that are unencrypted. The structure of these bytes is as follows:

L-Field	C-Field	M-Field	A-Field
1 byte	1 byte	2 bytes	6 bytes

Table 3.1: Structure of the Data Link Layer

The first byte in a message is the L-Field. It specifies the length of the complete message excluding itself, the CRC bytes and the RSSI byte at the end of the message. The second byte is the C-Field which specifies the type of message that is received. This is used to determine whether the incoming message is sending new information or if it is requesting information. For the project, only SND-NR messages are decoded. SND-NR stands for Send No Request. The third byte is the M-Field which specifies the manufacturer of the meter. This field is used for verifying the incoming message and for storing the message correctly in the database. Only messages of which the manufacturers are known will be decoded by the program. Each manufacturer ID is represented by three characters. In order to get the Manufacturer ID, the bytes need to be decoded in a specific way. First, the bytes need to be represented as an integer. The first character is then calculated by an

integer division with 1024, addition with 64 and looking up the result in the ASCII table. In order to get the other two characters, we need the remainder of the integer division. The second character is then calculated by an integer division between the remainder and 32, addition with 64 and looking up the result in the ASCII table. The last character is calculated by taking the remainder of the division between the first remainder and 64, addition with 64 and looking up the result in the ASCII table. After the M-Field, we have the A-Field. It consists of six bytes in total and contains three types of information. The first four bytes of the A-Field specify the serial number of the measuring device that sent the message. This is used for verifying the device and finding the correct decryption keys. The fifth byte specifies the version number of the device and the last byte of the A-Field specifies the type of device. A table containing the different byte values for different kinds of measuring devices can be found in the OMS specification, Volume 2 [19].

### **Transport Layer**

The Transport Layer follows the Data Link Layer and it can be of a variable length. This layer consists of a CI-Field followed by the TPL-Header. There are 3 possible structures for the TPL-header. The first of these structures is a short TPL-Header. This header contains the following information:

- Access Number: This number is used to track successful transmissions. It is incremented by 1 for every successful synchronous transmission.
- Status: This byte gives the current status of the measuring device and can be used to prevent errors in the device.
- Configuration Field: This field of 2 bytes specifies the method of encryption of the data. In the scope of this project, only 2 modes of encryption need to be accounted for, which is no data encryption and AES-128 encryption.

The second type of TPL-Header is a long TPL-Header. This header contains all the information of the short TPL-Header and the identification data of the meter/actuator, which can also be found in the Data Link Layer.

The third type of header is no TPL-Header, which means that no more information is present in the Transport Layer and the next byte of data will be the start of the Application Layer.

The first byte in the Transport Layer is a CI-Field. This field specifies the type of header of the Transport Layer. If there is an Extended Link Layer or Authentication and Fragmentation layer present, the CI-field will contain information about the structure of those layers instead. However, these layers are not within the scope of our project.

### **Application Layer**

The application layer contains all the meter data that is sent along with the message. Although this sounds easy, there are a lot of different units and some different data formats that can be used. That means that sometimes the actual data can take up more bytes than another data field. The OMS-group have found something to solve this problem. It will first send a DIB field which contains information about how the data will be formatted and how many bytes it will use. After the DIB field, a VIB field will follow. The VIB field contains information about the unit of the value and the description that corresponds to it. After these fields, the actual data is stored. This structure is then repeated until the end of the

message.

DIF	DIFE	VIF	VIFE	Data
1 byte	0-10 bytes	1 byte	0-10 bytes	x bytes
DIB		VIB		

Table 3.2: Structure of the Application Layer

The DIB field always contains a DIF-byte. This byte contains the information of the number of data bytes and the way the data is encoded. This byte can also specify whether 1 or more DIFE-bytes follow the DIF-byte. DIFE-bytes act as an extension of a DIF-byte and mainly contains information about the storage number of the concerned data. This storage number can be used by meters to differentiate between historic states or clarify the order in which the data should be read. A DIFE-byte also has an extension bit to indicate whether another DIFE-byte follows for more specific storage numbers. A maximum of 10 DIFE-bytes is allowed, which gives a maximum size of 11 to the complete DIB field.

The VIB field always contains a VIF-byte. This byte usually indicates the unit and scale of the data. 1 of the bits of this byte also indicates whether 1 or more VIFE-bytes follow this VIF-byte. VIFE-bytes are used to extend the VIF-byte. This can be done by adding a factor to the scale of the VIF-byte or changing the unit. VIFE-bytes can also specify errors in the messages. The last functionality of a VIFE-byte is to replace the functionality of the VIF-byte. This is applied when the needed unit and scale combination could not be encoded in a VIF-byte. This occurs occasionally since 1 byte is not enough to encode all the necessary unit and scale combinations. As with the DIFE-bytes, the VIFE-bytes indicate whether another VIFE-byte follows and a maximum of 10 VIFE-bytes is allowed per data field.

3.4.2. Advanced Encryption Standard

Encrypting the sensitive data generated by the measuring devices is important to prevent eavesdropping. Encryption is necessary to secure the data transfer between the measuring devices and the single-board computer and to prevent the neighbours from snooping on the usage of gas, electricity, and water. Encryption from the single-board computer to the servers is also necessary because it ensures that our data arrives unchanged and unseen by malicious actors. Advanced Encryption Standard (AES) is a new encryption standard recommended by the National Institute of Standards and Technology (NIST) to replace Data Encryption Standard (DES) [7]. AES is a block cipher that uses the Rijndael algorithm [27]. Research has shown that Blowfish and AES have the best performance when comparing all modern encryption algorithms [7]. The performance of AES increases with the data block size. [REDACTED].

[REDACTED]. The implementation of AES in .NET is considered to be the best in the market [7]. Blowfish might have better performance in theory, but, in practice, many modern CPUs include hardware implementation of a cryptography module that supports AES. [REDACTED].



**Encryption: Measuring Device to SBC**

The decision for encryption of the communication between the measuring devices and the single-board computer is an easy one. The majority of the measuring devices used by the client are [REDACTED] meters, and these meters only support AES-128 [2]. The single-board computer that will be used, the SBC also supports AES and even has an efficient hardware implementation for improved performance. This part of the encryption will use AES-128.

**Encryption: The middle man**

The single-board computer is the middle man for this project. It receives a message from the meter that contains an unencrypted header, and an encrypted payload. The single-board computer could, in theory, decrypt this payload and encrypt it to transfer it to the server, but this is not practical. The best option for this use-case is to send the unencrypted header with the encrypted payload to the server, and the server will then be able to decrypt it. The first reason for this is that it enables us to only have the encryption keys in our database, this removes the risk of having the keys/certificates stolen from the single-board computer. The second reason is because of performance. The single-board computer has limited computing power and might be busy receiving many messages at once. The received messages will not be decrypted on the single-board computer and the amount of processing on this data will be limited to what is necessary.

This paragraph will only be included in the project if we have enough time because it is a "should have" in our MoSCoW requirements document. For the "must have", the decryption keys and the decryption process will be on the single-board computer instead of the server.

**Encryption: SBC to Servers**

Finally, the single-board computer should send all the meter data to the servers. In the ideal case, the payload data is still encrypted by AES, but the header data is not. To ensure confidentiality, Transport Layer Security (TLS) v1.2 is used for communicating with the server. The cipher best used for TLS is

TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256, this provides good performance with incredible security [22].

**3.4.3. Virtual Private Network (VPN)**

The communication between the gateway and the server will be through a VPN tunnel. This choice had to be made due to an issue which will be elaborated soon. However, using a VPN for communication has major privacy and security advantages.

The gateway will use a SIM card dongle to send data to the server. Using the internet connection from the 3G/4G network provided by a telephone provider. Internet service provider networks use the Dynamic Host Configuration Protocol (DHCP). This protocol uses dynamic allocation of an IP address, this means that the gateway gets a different IP address assigned (every time it restarts) [33]. Therefore, it does not have a fixed IP address. This is not practical at all since the gateway needs to have a fixed IP address in order for the server to be able to communicate with the gateway. If the server knows the IP address of the gateway, it can communicate, otherwise it cannot. For the gateway to have a fixed IP address, a VPN tunnel will be set up between the gateway and the server. This allows the gateway to have a fixed IP address for the communication between the server and the

gateway.

**VPN type: Site-to-Site VPN**

The type of VPN that will be used is the Site-to-Site VPN and not the Remote Access VPN. The Site-to-Site VPN type allows for a virtual bridge (tunnel) between the gateway and the server. This allows for private communication between the gateway and the server. Moreover, in this type of VPN, encryption and security are added at the router level rather than the user's computer [1].

**VPN protocol: OpenVPN**

There are numerous of different VPN protocols, the VPN protocol that we have chosen is OpenVPN. OpenVPN is open-source and has, as of late, become somewhat of a standard. It is one of the best VPN protocols considering security. Moreover, it is supported by all the major operating systems [1]. Additionally, the client's servers already run OpenVPN and, as the SBC supports OpenVPN, this is a perfect choice for the client.

## 3.5. Databases

Two databases will be used to correctly process incoming data: MySQL and OpenTSDB.

### 3.5.1. MySQL

The first database contains information about all the installed meters and will be used to validate the source of incoming data. For this, our choice was to use an RDBMS over a DBMS, because the database will have to handle large amounts of data and RDBMS is more suited for handling large amounts of data. We compared MySQL and PostgreSQL as options. In the end, we chose MySQL because it puts a larger focus on handling large databases [9].

### 3.5.2. OpenTSDB

The second database uses OpenTSDB and contains energy usage data sent by the meters. This is the database where the incoming data will be stored on and this database will be used by our client to analyze the energy usage of their clients. OpenTSDB is interesting to use because it is a time series database. This will allow for modelling energy usage over time, which will be useful for the client since the client wants to measure energy usage over time. OpenTSDB also runs on Hadoop, which makes the database more scalable [18]. This will allow for processing the big amounts of data our client receives.

Another reason for us to choose these systems for the databases is that our client already uses them and using the same systems will make it easier for the client to continue with their current systems rather than to adapt to new systems.

# 4

## Design

This chapter provides a high-level overview of the design of the solution. It describes the system specifications and discusses the details required for implementation. An UML diagram is used to explain how the OMSDecoder should be implemented. Also, a number of challenges relating to designing the solution are discussed. Finally, based on the research in Chapter 3, the final design decisions are clarified.

### 4.1. Overview

A high-level overview of the design of the solution can be seen in Figure ?? . This figure depicts how an incoming message is received, processed and sent to the company server.

*This figure has been removed due to confidentiality.*

Figure ?? on the previous page shows the usage of a whitelist, a received measuring devices list and the raw data. Chapter 5.1 contains an extensive writeup on how this design has been implemented in the application. It is important to note that this figure has been very helpful in implementing all the request features. It is important for a project of this size with multiple members to have an agreed upon design to streamline the development process.

### 4.2. Challenges

This section contains several challenges that came up during the design process. Each of these challenges has been discussed extensively within the development group and in consideration with the client's requirements. Each subsection will discuss a challenge by describing the process and solution that has been used to resolve it.

#### 4.2.1. OMS Protocol

The OMS protocol is a comprehensive protocol that defines the communication and data exchange between measuring devices and collectors. Publicly available documentation on the inner-workings of this protocol was minimal and it was not structured in a clear way. It proved to be a challenge to figure out the exact structure of the OMS messages and to fully support the decoding of the various OMS messages. There are numerous exceptions within

the OMS protocol that differ from the European norm and specifications. Each manufacturer has a slightly different implementation of the protocol, and the solution that is being developed needs to support them all. It was necessary, due to the complex structure of the protocol, to come up with a smart design for decoding the messages. Figure 4.1 depicts the structure of the code responsible for decoding the OMS messages.

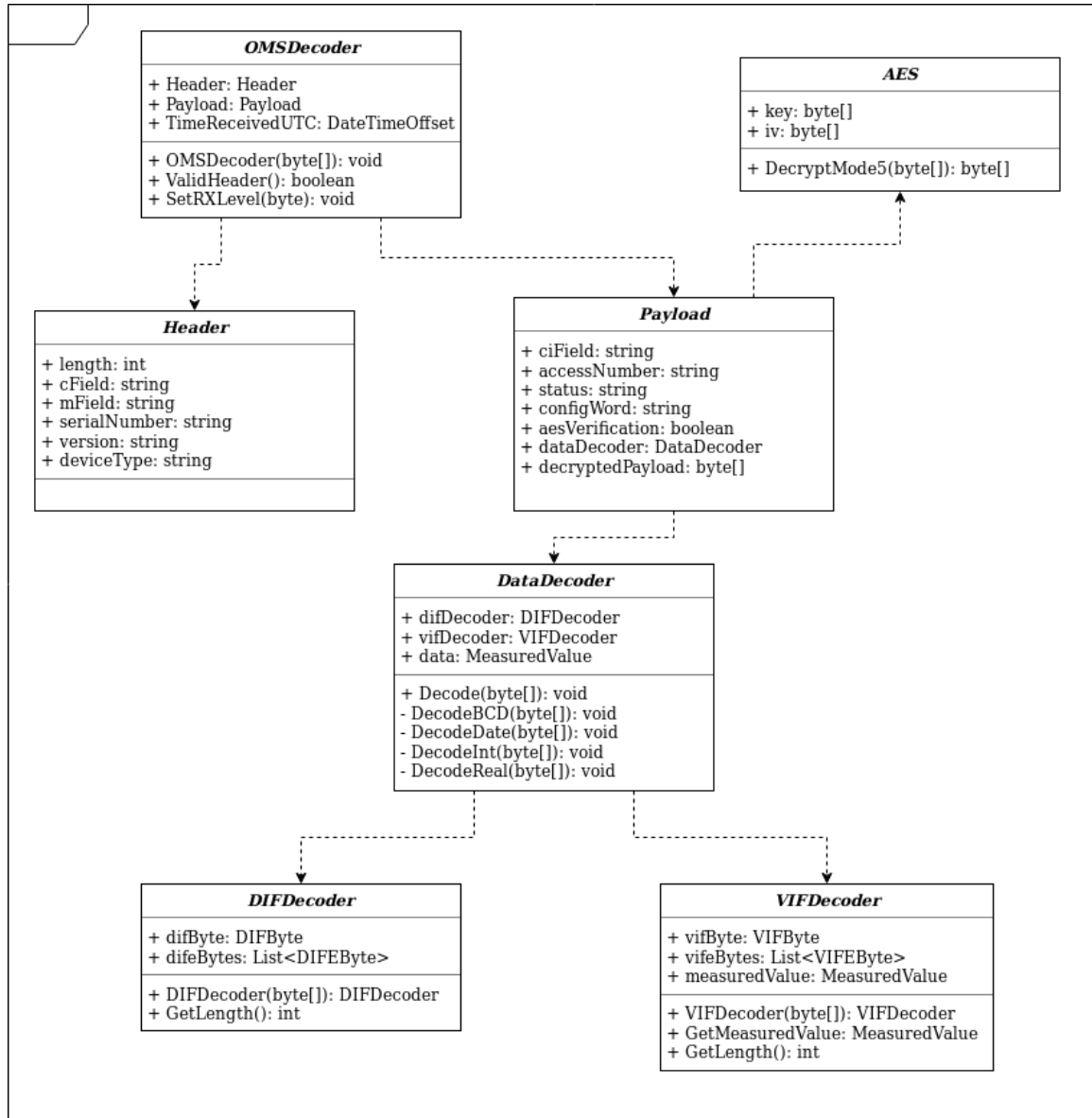


Figure 4.1: OMS Protocol Class Diagram

Whenever the Serial device receives a message, it is sent to the `OMSDecoder` class. This class is responsible for splitting up the message into a header and a payload, which is sent to the `Header` and `Payload` classes respectively. The `Header` class then decodes its part of the message following the structure described in Chapter 3.4.1, Data Link Layer. The part of the message that is sent to the `Payload` class is decoded in multiple steps. First, the bytes that represent the Transport Layer (see Chapter 3.4.1, Transport Layer) are decoded to determine the encryption mode and status of the measuring device. If the data is encrypted,

the rest of the payload is decrypted by the AES class. All unencrypted data is then sent to the DataDecoder. The DataDecoder loops over all of the leftover data. In every loop, the data is first sent to the DIFDecoder, which is responsible for reading the DIFByte and the DIFBytes of the current data field. These bytes specify the length and encoding of the data field and are mapped using Dictionaries in C#. Then, the VIF- and VIFBytes are read. These specify the unit and scale of the value and are also mapped using Dictionaries. Using the encoding and length of the data, the data is then decoded by the DataDecoder using the appropriate decode methods.

### 4.2.2. Robustness

The solution will be widely distributed to various cities across the country. Therefore, it is of great importance that the solution is robust because if the solution does not work as intended, a mechanic will have to travel all the way to the gateway to solve the issue. Deploying a mechanic to gateways across the country, costs time, effort and money. Not only should it be robust for cost saving, but also because the client does not want to lose any data. If the solution is not robust enough and for some reason stops and thus does not receive and process any data, the client will never get this data. Thus, by ensuring that the solution is stable and robust, the risk of missing data is minimized. The possible challenges that are accounted for in making the solution robust were:

- Incorrect OMS messages
- Losing connection to the server
- Corrupted SD cards
- Unexpected system crashes

#### Incorrect OMS Messages

During the development of the solution, numerous messages were retrieved that caused crashes due to radio signal failures or deviation from the European norm. To ensure a robust solution, resilience against these inconsistencies should be taken into account. This has been done by adding multiple checks during the decoding of the messages to validate the structure of the message. These validations check the remaining length of the message or data fields, and it checks whether the received bytes contain expected values. Whenever a validation fails, the remaining part of the message is ignored and discarded. This is done because the validity of the remaining bytes cannot be checked after an error and could contain arbitrary nonsense. The correct part of the message will still be processed and sent to the database. Implementing resilience against arbitrary failures within the OMS protocol greatly reduced the number of errors and crashes while running the application.

#### Corrupted SD Cards

As explained in Chapter 3.1.4, the SD card delivered with the [REDACTED] will most likely not be reliable enough to run for 10 years without getting corrupted. For this reason, the decision was made to use a USB flash drive specified in Chapter 3.1.4. By implementing this change, the chance of a faulty storage device is significantly decreased. Also, since the chosen USB flash drive has more memory than the SD card, the chance of the storage

device becoming full is reduced too. These two benefits greatly add to the robustness of the solution.

## Server Connection Loss

Due to external factors such as network outage or bad reception, the solution may not be able to connect to the server at all times. The solution is therefore designed to be resilient against internet connection failures. To account for this, the data is saved to the storage device and it will try to send all the completed files at a regular interval to the FTP server. The data will only be removed from the storage device if it receives a confirmation from the server that the data arrived correctly. By doing this, the data will not get lost in case the solution cannot connect to the server.

However, saving data to the storage device introduced another problem, which was the possibility of running out of storage. This should only happen if the solution is unable to send data to the server for a long time (about a month), but should be taken into account. Therefore, the solution will detect when the data storage is almost full and delete the oldest data when this happens.

Moreover, the client is able to detect that a gateway lost its connection with the server. The client may act by sending an employee (mechanic) to the gateway to investigate what is wrong. This will most likely happen within a couple of days, thus data loss will be a rare event. As soon as the mechanic is able to fix the internet connection, all stored data will be sent to the server.

## Unexpected System Crashes

All software systems contain bugs and could contain errors that may cause crashes. Given the short scope of the project, there is a reasonable chance that a bug or issue has not been found during development. While it is desired to detect these issues during development with extensive testing and code reviews. There should also be a fail-safe in the form of crash detection and automatic restart functionality. This should not be done manually by an engineer or mechanic but it should be automated fully. To make sure the solution keeps running, a service script has been configured that will start the solution on boot and restart the solution automatically when the solution stops running. This service script has been created using systemd. This service script runs at boot, which means that the solution will also automatically start in case of a temporary power loss.

### 4.2.3. Data Security

Securing the data received from measuring devices is important because it is classified as private data of the customers. The law requires encryption of privacy-sensitive data [11]. Determining how to handle this private data proved to be a challenge, since there was a choice to be made at what point in the process the incoming data should be decrypted. Also, since the solution saves the private data to the storage device, an educated choice had to be made on how to encrypt this data. Lastly, the transmission of the data from the gateway to the database should also be secure.

The most secure option would be to send the encrypted OMS messages to the server and decrypt the data server-side. By doing this, the gateway should never have access to any private information and therefore can never leak private information in any way.

**9** **1**

**1**

**9** **9**

[REDACTED]

To ensure the private data is sent in a secure way to the server, every gateway opens a secure VPN connection to the server using OpenVPN. The gateway will be able to send the data over the secure VPN connection using the SIM Card Dongle specified in Chapter 3.1.3.

### 4.3. Final Design Technologies

Extensive research into the available technology options established the final design. The details of this research can be found in Chapter 3. The research is concluded by selecting the most appropriate technologies for the final design. An overview of these technology choices that have been selected for the development of the solution are summarised in the following list:

- Single-board computer: [REDACTED]
- Input Serial Dongle: [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED].
- Output SIM card (4G) dongle: [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED]
- Storage USB flash drive: [REDACTED] [REDACTED] [REDACTED] [REDACTED]
- Operating system: [REDACTED] [REDACTED]
- .NET Framework and C#
- XML
- OpenVPN
- Apache
- Open Metering System (OMS) protocol
- Advanced Encryption Standard (AES)
- MySQL
- OpenTSDB

The client may wish to substitute certain pieces of hardware (such as the single-board computer or the serial input device), or certain pieces of software (such as operating system or XML) when the solution is fully deployed. The solution supports substitution of every part of the selected design, except the programming language and framework.

# 5

## Implementation

After researching the best solution and designing the system, the implementation of the required functionality began. During the implementation, the focus was on creating our software with stability, fault resilience and code maintainability in mind. These factors are of big importance to this project because the final product needs to run for at least ten years while making sure to retrieve and send all the available measurement data. The focus on code maintainability is important because the client wants to extend and update the software after this project is finished.

In section 5.1 the overall structure and basic functionality of the system are discussed. This section gives a detailed description of the implementation of the required features. If time permitted, secondary and tertiary features would also be implemented based on the MoSCoW document created in consideration with the client. These features mainly consist of functionality that extends our basic functionality and focuses more on the off-site maintainability and two-way communication with the server.

### 5.1. Overall Structure

The implementation of the features that our improved gateway contains can be split up in the following tasks:

1. Supporting the OMS protocol
  - Decoding the OMS messages
  - Decrypting the OMS payloads
  - Converting decoded OMS messages to CSV
  - Configuring the wireless m-bus dongle
2. Supporting a whitelist
  - Filtering OMS messages
  - Saving non-whitelisted messages as raw data
3. Reliability and data backup
  - Sending data to the FTP server



- Automatically restarting the improved gateway

#### 4. Other functionality

- Keeping track of all received devices and allow this to be securely accessed via a web browser
- Configuration files can be changed during run-time

All these features have been discussed with, and agreed upon, by the client and the project members. Details on the planning process and details on the inclusion or exclusion of certain features can be found in Section 6.1.

An in-depth description of the implementation process of supporting the OMS protocol is discussed in section 5.1.1. In order to only support measuring devices from the client, our application contains a whitelist. This allows our improved gateway to save unnecessary CPU-cycles and data bandwidth caused by devices from competitors. The details on this are elaborated in section 5.1.2. To ensure reliability, the improved gateway should automatically restart and all received measuring data that is whitelisted should be sent to the server. Implementation details on this can be found in section 5.1.3. Finally, the implementation of other functionality such as configuration files and keeping track of received measuring devices is described in section 5.1.4.

### 5.1.1. Supporting the OMS protocol

Decoding the OMS protocol has been the biggest challenge during this project. As documentation and specification documents were sparse, much of the implementation had to be done based on reverse engineering real-life data or given example messages. Section 3.4.1 contains an extensive write-up on the OMS protocol. This preliminary research helped us to implement this feature quicker, but it was still a challenging task.

The physical layer of the OMS protocol is completely handled by the XXXXXXXXXX Wireless M-Bus dongle. The serial data received from the dongle is scanned by our application for the start bytes of a message, 0xFF and 0x03. When these bytes are found, the improved gateway reads the next byte to find out the length of the message and then waits until the buffer contains as many bytes as the length of the incoming message before retrieving them all at once. The bytes are sent to the OMSDecoder class where the OMS message is split into three parts, the header, the payload, and the signal strength.

The header class uses the first 10 bytes of the message to determine certain information from the message such as the manufacturer, the device ID and the device type. Many of these fields use helper functions to decode the raw bytes to usable values.

The payload class receives all bytes from ten until the end of the message except the last byte. Firstly, it determines certain information on the encryption method, configuration, and message type. Then, depending on this information, it decides if and how to decrypt the payload with the AES class.

The AES class calculates the IV based on information from the OMS header and information from the payload header. It retrieves the AES key from the key dictionary included in the program. It then decrypts the message based on the specification in EN 13757 [30].

After successfully decrypting the payload, the data is sent to the DataDecoder class that contains the functionality to read the data protocol. This class uses the decrypted payload

bytes to retrieve the data values from the OMS message. Each data value consists of one DIFbyte with zero or multiple DIFEbytes that are handled by the DIFDecoder class. Following the DIFEbytes there is one VIFbyte and zero or multiple VIFEbytes that are handled by the VIFDecoder class. The decoding of DIF(E) and VIF(E) bytes is described extensively in section 3.4.1. And finally, there is a data value that should be decoded based on the configuration of the DIF(E) and VIF(E) bytes. The payload contains multiple data values so the DataDecoder class continues until the end of the message is reached.

Finally, the last byte of the OMS message is converted in the OMSDecoder class to describe the signal strength of the received message.

It is clear to see that the implementation of the OMS decoding described in the paragraphs above follows the design of the OMS decoding in figure 4.1. Every OMS message that is successfully decoded is converted to the CSV format so it can easily be processed by the server. This is done in the application by giving the decoded OMSDecoder object containing the decoded OMS message to the CSVTransformer class. This class generates a CSV header before adding OMS messages to the file. The transformation of an OMS message to CSV is very precise because it must conform to the standard set by the client so it allows seamless integration within the current system. However, converting the data is efficiently implemented because all information about the OMS message is easily retrievable from the OMSDecoder object. Every OMS message generates a row in the CSV file. When a specified time-out is completed, the file is moved to another location to signal it is ready to be uploaded to the server.

The measuring devices send serial signals around using radio waves. These serial signals are detected by the [REDACTED] Wireless M-Bus Dongle[3], which will be attached to the gateway. Measuring devices sent using specific operating modes, where different modes use different frequencies. Since the [REDACTED] Wireless M-BUS dongle can only receive up to one frequency at a time, some messages of the measuring devices cannot be received. Because of this, we included a settings file in our solution where the operating mode can be specified by the client, so the client can choose which operating mode it wants to receive messages from. The program reads this settings file and compares the operating mode of the settings with the actual operating mode on the dongle. This is done by sending commands to the dongle which will then respectively return an answer. If the operating modes are not the same, a command is sent to the dongle to change the operating mode and store it in the non-volatile memory. The client knows about the different operating modes and is planning to use measuring devices that use the same operating mode, so all the desired data can be received at all times.

### 5.1.2. Supporting a whitelist

The improved gateway should only decode the OMS messages of measuring devices from the client and not the devices from competitors. This functionality is implemented on two different levels.

The first level is based on the manufacturer of the measuring devices and it allows the improved gateway to ignore all measuring devices that are from different manufacturers than the client is using. At the time of writing, the client only uses three manufacturers, while there are over 1000 different manufacturers available [26] and in-use by competitors.

This first check is done in the OMSDecoder class after decoding the header. It checks the manufacturer against a specified manufacturer list and it skips the decoding of the payload for unknown manufacturers. By doing this, the gateway will not waste time and power on processing messages that are not wanted by the client anyways.

The second level of whitelisting is done based on the serial number of the measuring device that has sent the OMS message. This whitelist enables the client to have multiple gateways in the same physical location and link each measuring device to a distinct gateway. If the serial number is not on the whitelist, the gateway does not completely discard the OMS message. It stops decoding, but it saves the raw data in the RawDataHandler and it still updates the received measuring devices list. This allows the client to see the received strength for all measuring devices and improve the gateway/measuring device distribution. It also allows the reprocessing of non-whitelisted OMS messages if the whitelist is updated at a later point in time.

The RawDataHandler class contains the functionality to save OMS messages that are not whitelisted to the disk as raw received bytes. It saves messages that are from the correct manufacturer but do not have a matching serial number. 0xFF 0x03 is the delimiter used by the [REDACTED] Wireless M-Bus Dongle and it is also prefixed to every raw saved message to allow easier decoding at a later point of time. The raw data is saved in a separate directory for every serial number. This class also supports the processing of stored raw data after the whitelist has been changed. This can be CPU intensive so it is done on a separate thread, allowing parallel processing of incoming OMS messages. This is done whenever changes are made to the whitelist and when the program starts.

### 5.1.3. Reliability and data backup


All decoded OMS messages are sent to the client's server to allow processing and backups. This has to be done in a CSV format specified by the client to allow a seamless integration in the current system. The CSVTransformer class moves finished CSV files to a directory to signal that the files are finished and ready to be uploaded. The FTP thread within the application is responsible for checking this directory and uploading the files to the FTP server. This is done in a separate thread to allow asynchronous uploading of the data without blocking the OMS decoding process. The thread configures the connection to the FTP server and it operates on a specified timeout to check and upload all files.

To improve reliability, it first checks the CSV directory for files that have not been finished properly due to a system or application crash. This enables the recovery of CSV files and it adds them to the upload list. Secondly, it checks the ready for upload directory where the CSVTransformer stores all the finished CSV files. It also adds them to the upload list. Finally, it attempts to upload all the files from the aggregated upload list to the FTP server. This is asynchronously done with an FTPState to monitor the progress and result of the FTP request. After a successful upload, the files are deleted from the improved gateway to free disk space.

The improved gateway should also be resilient against power loss, crashes of the operating system or the application code. This is mainly done through [REDACTED] but the application itself should also check if it still receives messages.

#### 5.1.4. Other functionality

Our application saves the timestamp, serial number, and signal strength of all received measuring devices that pass the manufacturer whitelist. This is done in the `ReceivedMeasuringDevice` class and it saves, and updates on a regular time interval, the received measuring devices to an XML file. The list contains, for every device, the last time it has been received and the signal strength of the last received message. Through an Apache web-server and an OpenVPN internet connection, the client can access this XML file through a nicely formatted web page to identify which devices are being received. The XML serializing contains a lock to make sure it does not write and read at the same time.

Finally, the configuration files contain many configurable values such as timeouts, IP addresses, serial numbers, manufacturer IDs and the receive mode of the  Wireless M-Bus dongle. The client requires these configuration values to be easily configurable to allow efficient maintaining and configuring of hundreds of devices. Usually, the program should be restarted for these changes to take effect. However, while the program is being restarted, the gateway will not receive incoming data, therefore causing data loss. That is why all configuration files have file watchers that notify the application if they are changed. The file watchers enable the client to update the configuration of the application by just replacing the configuration file. Our application notices this change and it reads and processes the new configuration files automatically, so no data will be lost.

# 6

## Process

This chapter will provide an overview of the process of the project along with the decisions that have been made regarding the process. In section 6.1, the planning of the project will be discussed. Section 6.2 will be about Scrum, which is the software development method that has been used during the project and section 6.4 will review the feedback that has been received from the Software Intelligence Group (SIG).

### 6.1. Planning

During the research phase, a MoSCoW document has been made in cooperation with the client as a basis for the product, which can be found in Appendix E. A primary planning has been made based on the MoSCoW document, where the individual must-haves were grouped together to form milestones. The milestones that have been used in this project are as follows:

- OMS Decryption. This milestone contains the must-haves regarding the OMS protocol and data decryption, so that it is possible to read incoming data.
- Send Data to Server. This milestone focuses on putting the data in the right format for the database, connecting to the database and sending the data to the database.
- Whitelist. This milestone adds functionality to include or exclude specific measuring devices on choice.
- Fault Resilience. The last milestone focuses on the robustness of the product. An automatic restart functionality was added along with the possibility to change files containing settings while the program is running. Also, incoming data messages of devices that are not whitelisted are saved in case the client decides to whitelist the device later on.

Using these milestones and must-haves, a Gantt Chart has been created, which can be found in Appendix F. This chart shows all must-haves and their expected complexity, along with a time estimate and the responsible team members for the task. By using the Gantt Chart, our team could dynamically change the planning if a task was finished earlier than expected or if a task took longer due to unforeseen circumstances. This created a clear

overview for both the team and the client and showed the progress on the product. This chart became the basis during the development phase for determining the tasks to be done every week.

## 6.2. Scrum

For the development process, the decision was made to work with the Scrum method [25]. Scrum is an agile method of working by using small development cycles called sprints. At the start of each sprint, a planning would be made for that sprint and task would be assigned to all members. These tasks were defined in a Scrum board on Trello. At the end of the sprint, the team would have a meeting with the client to discuss and show the progress. Also, the sprint would be recapped where we would list possible improvements for the next sprint and update the Gantt Chart.

By using this agile method, our team could act on arising problems before the problems would get out of hand. This along with the Gantt Chart made the development process very dynamic.

While this method worked out for us, we did encounter some small problems during the development phase. During the sprint planning, tasks were assigned to members based on the Gantt Chart. However, working on the report was not covered in the planning at the start of the development phase. This caused our product to progress faster than initially planned. However, the report was falling behind. Because of this, one week has been dedicated to bringing the report back on track with little focus on software development. While this did fix our problem, it was not desired and led us to taking the report into account during the future sprint plannings.

## 6.3. Draft Report

We agreed with our TU Delft Coach that we would deliver a Draft version of our report on Friday 14 July 2019. This would allow the coach to give us feedback on our report and indicate any points to address to help us improve the quality of our report. This draft report was submitted to our coach, who said that he was really impressed with the quality of our report. The coach also suggested adding summary boxes to some sections to ensure readers were quickly able to grasp the main ideas of that section. This feedback was addressed by improving and elaborating on the opening, core sentences of sections. Furthermore, a full, clear overview of the design choices based on the Research chapter is included in the Design chapter.

## 6.4. Software Improvement Group

During the development phase, we delivered our code twice to the Software Improvement Group (SIG). Once for midterm feedback and the other as a final product. The feedback we received from SIG for our first delivery can be found in Appendix C. During our planning, we reserved time to discuss the feedback and implement suggested changes by SIG in the last two weeks of the development phase. By doing this, the workload in the last two weeks did not turn out unexpectedly high, because these tasks were not taken into account.

## Testing and Code Quality

The client's usage of our solution is of such a nature that the robustness and reliability of our solution is paramount. Ultimately, the client will deploy many different instances of our solution in various locations in The Netherlands. If our solution breaks, the client has to deploy a company engineer to that location, which could be hundreds of kilometres away. Furthermore, other developers must be able to further develop the solution after the project ends. Therefore, it is of great importance that our solution is thoroughly tested and that the code is of high quality. In this chapter, the testing process and quality assurance process is explained and analysed.

### 7.1. Unit and integration testing

The C# code is extensively tested with unit tests and integration tests. The testing framework used is NUnit. This open-source testing framework was selected over other popular C# testing frameworks, such as MSUnit, because it seamlessly integrates with both Visual Studio and MonoDevelop. Furthermore, it is feature-rich, well-documented and widely used and provides support for a code coverage tool. Because it was an agreed requirement that all new classes were properly tested before being merged, unit tests were created for all classes. The only exceptions to this are the entry point, Program, the serial input device handler, SerialProcessor, the system info class, SystemInfo and the Configuration class, Config. These classes were not tested, either because their non-virtual dependencies were unable to be mocked (Program, SerialProcessor), the code was system dependent (SystemInfo) or no methods were present to be tested (Config). To enhance the unit tests further, a C# mocking framework was used: Moq. This ensured that more complex classes that have dependencies, such as the CSV Transformer, could be tested effectively and efficiently by stubbing elements of other classes. This removes the dependency on other classes to function well for the tests to pass.

At the end of every sprint, a full code coverage report was generated and the overall line and branch coverage was added to the retrospective. The code coverage reports were generated using OpenCover [35]. These figures were stable, with an upwards trend occurring over the weeks. A graph displaying the line and branch code coverage at the end of every week is displayed in Figure 7.1. The line coverage of the final codebase submission was 91.4%. The branch coverage of the final codebase submission was 85.8%.

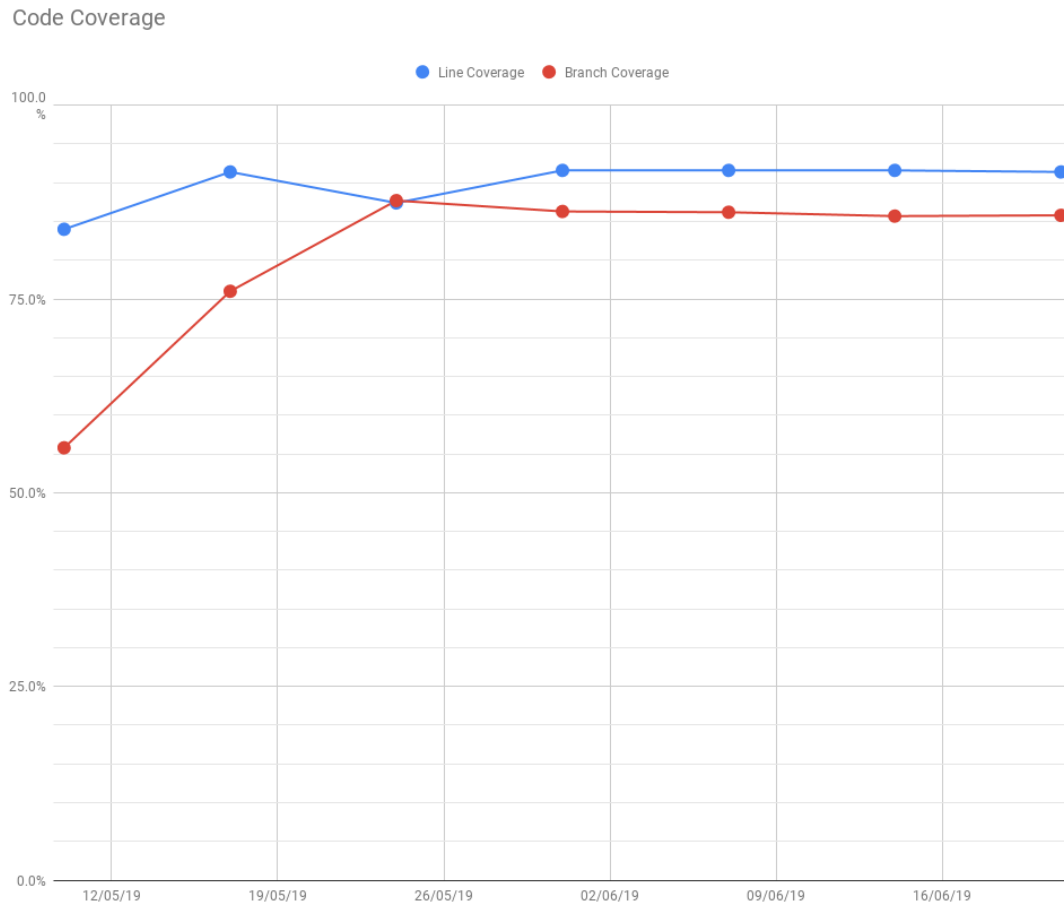


Figure 7.1: C# Code Coverage by Automated Tests

## 7.2. Risk analysis

A risk analysis of the code was also conducted by the OpenCover report. This risk analysis identified a number of potential risk hotspots in our code. One of these hotspots was the following method:

```
System.Xml.SerialSettings.GetModeByte(string receiveMode)
```

Initially, this method contained a switch statement and had a high cyclomatic complexity [32], NPath complexity [17] and CRAP score [24] as a result. The code was refactored to make use of `if`, `else if` and `else` statements and the complexity was significantly reduced. The cyclomatic complexity and CRAP score were brought to acceptable levels of 9. However, it still has an NPath complexity of 256, which is an improvement on the previous score but still above what is regarded as an acceptable level. Thus, the method was refactored again to make use of a readonly dictionary to map `receiveMode` strings to their corresponding bytes. By doing this, the method significantly reduced in complexity and was thereby no longer considered a risk hotspot by the risk analysis.



## 7.3. System and acceptance testing

The solution has been system tested. In the development location (the client's office), many different measuring devices were present. These measuring devices transmitted data to an instance of our solution. Our solution processed this data. We then manually checked whether the output of our solution was as expected by reviewing the generated CSV file and viewing the incoming data on the server. At the office, over 1000 measuring devices were present, which provided an excellent opportunity for load testing our solution. Our solution handled the continuous data from these 1000 devices without problems.

We were also given the opportunity to test our solution at one of the client's customers in [REDACTED]. We found out that radio signals are pretty sensitive to the environment as we only received one third of the measuring devices that were in that area. This problem can be fixed with repeaters and using antennas. Repeaters resend messages coming from measuring devices with more power, thus covering more area. We tested this solution at the office by putting a repeater in between a measuring device, that was out of range to be normally received, and the gateway. Only with the repeater we were able to receive the data. The client accepted the functionality and was content with the data received from our solution. The client was able to use this data. Our solution was still stable and running after 4 weeks of it being live. The system testing was combined with acceptance testing and the client was very happy with the quality of our solution, determining that all of the must-have requirements and many of the should-have requirements had been met.

Overall, system testing our solution meant that the solution's functionality was tested by simulating real-life situations. Thus, the behaviour of the system was validated. System testing was the final step for us to be able to guarantee to the client that our solution satisfied the client's needs - which it did. The client plans to reuse parts of the software in future projects.

## 7.4. Static Analysis

We decided to use static analysis to ensure our code is of high quality. The package we used to do this with is the StyleCop.Analyzers package for the .NET compiler platform. StyleCop.Analyzers extends the default Microsoft code analysis functionality by adding numerous rules to the code ruleset, related to code style and consistency constraints [31]. When editing code, violations of the ruleset are displayed as warnings in the integrated development environment. As a group, we agreed upon a slightly modified ruleset. The following default StyleCop.Analyzers rules were fully disabled:

- SA1639: File header should have summary. We considered this redundant as all classes and other code elements already have Summary documentation.
- SA1609: Property documentation should have value. We considered this redundant as all properties already have summary documentation starting with 'Gets or sets a value indicating whether', meaning that the setter is already documented.

A select few warnings were suppressed at specific locations in the code. In all cases, the justification for this is included in the suppression attribute. One example of this is to suppress rule SA1124 (Do not use regions) in the Config.cs file, as lengthy config files benefit from categorizing variables into expendable regions for improved readability.

We have ensured all our code complies with the modified ruleset and that there are no violations of the ruleset in the codebase.

## 7.5. Code reviews

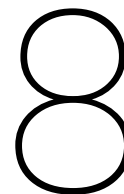
To further ensure the codebase is of high quality, we made extensive use of code reviews. We did this by occasionally pair programming, but most importantly we all agreed to a rule that pull requests were only merged after it had been fully reviewed by at least one other group member. However, in practice, at least 2 group members reviewed pull requests before they were merged. This review process typically involved multiple review cycles. Firstly, one group member would review the code and the author would address any issues raised. Then, a second group member would review the code and the author would address the issues raised in this second review. Lastly, the reviewers would check if their feedback had been implemented and the pull request would be merged. Because of this process, pull requests were typically of a manageable size. Pull requests were not merged until all reviews had been fully processed.

## 7.6. Software Improvement Group

Another important part of the code review process was the Software Improvement Group (SIG) code base review. This had been organized as part of the Delft University of Technology Bachelor Project and involved two submissions of the code base. After the first submission of the draft code base, SIG sent us a report on the quality of our code. Our code was assessed and given a score of 3.5 out of 5, meaning the maintainability of the code is equal to the market average. The two most notable points of improvement were that some methods were too lengthy and others were slightly too complex. Over a period of two weeks, we had the opportunity to work on this feedback from the report and we worked to reduce the length and complexity of our methods insofar possible. For example, as seen above in section 7.2, `SerialSettings.GetModeByte(string receiveMode)` was reduced in complexity by making use of a dictionary data structure, rather than a lengthy if/else if/else construction.

However, one of the methods mentioned, `DataDecoder.DecodeBCD()`, was not changed. It had a cyclomatic complexity of 7 and an NPath complexity of 16. Refactoring the method to artificially reduce this already low complexity was not worthwhile, because it would have a negative impact on the performance. The current method is able to function with one iteration on the BCD array, but reducing the complexity would cause two iterations on the BCD array. We decided to not decouple the validation and calculation within this method as it would lower the performance of one of the most used methods in the application. Furthermore, refactoring the method by splitting functionality would result in tedious and unnecessary variable duplication, as well as a less clear purpose for all the new helper methods. Therefore, the method was not refactored.

All other feedback from SIG was resolved. After the two-week improvement period, a second submission of the final code base was made. The full SIG reports can be found in Appendix C.



## Discussion

This chapter contains a discussion of the project. It includes questions about the correctness of our solution and also adds some recommendations for the future of this product.

### 8.1. Evaluation of our product

The goal that was determined at the beginning of this project was to design a robust, scalable and cheap product to receive data from measuring devices. It would replace a solution that has not worked out for the client in the past and costs a lot of money. In terms of hardware, we think that our solution is a very good one. However, there are some factors that could raise issues with our current solution.

First of all, the product needs to work for around ten years. Since the USB drive is used a lot, it is not fully certain that they will last ten years. It is known that program-erase cycles (P/E cycles) can significantly reduce the lifespan. Different types of USB types can be used, of which Single Life Cell (SLC) USBs might be the best, as they are known for their high P/E cycles per lifespan. However, they are very expensive. Another solution would be to use a hard disk but this will of course also be a more expensive solution and also use a lot more power.

Another possible issue is that our solution only works for a specific wireless m-bus dongle. That dongle contains software that does CRC checks but also uses specific bytes to indicate the beginning of a message. If we switch to another dongle, the beginning of a message might be indicated with different bytes. However, the adjustments that would have to be made in our solution to accommodate this are relatively insignificant.

One last important possible issue is the distance range for receiving the signals of the measuring devices. In the office, we did a test with a wireless m-bus dongle with an antenna that had a range of 800 meters. However it did not receive the measuring device 150 meters away on the other side of the office, most likely explained by the presence of various walls, objects and other obstacles in the office. To resolve this issue, a repeater was used. It was placed in between our gateway and the measuring device. Most of these problems can be fixed by placing repeaters but this will increase the cost for the client.

## 8.2. Recommendations

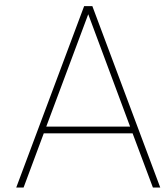
The final product contains a lot of functionality but there are, of course, always features that can be added or improved. Here are some recommendations of features to focus on in the future.

First of all, the preparation of the Single Board Computer can be massively improved. There are a large number of steps that need to be performed before the solution works as it is supposed to. That is why it would be useful if a custom image is created that has all files and scripts needed to automatically set up the Single Board Computer.

[REDACTED]

Lastly, a nice feature would be to make it a bit easier to configure the gateway. A nice user interface for whitelisting certain measuring devices and changing the FTP address, user name and password make it easier and faster to change the gateway in order to satisfy the client's needs.





## Project Description

**Project goal** Develop a robust and scalable solution that will collect energy measuring data on a large scale, and to deploy it to a Big DATA database.

There are a lot of solutions on the market that will measure something and send that information over the internet to a server. But when this needs to be done simultaneously, in near Realtime, for over 65.000 measuring devices, in various locations, that send multiple measuring values, this becomes challenging.

Our project is to develop a solution that will work as a meshed network to make the solution robust. It needs to automatically select the most economical path for transferring the data. The receivers need to be able act as a receiver and as a repeater of the measuring data. Making smart use of encrypting and decrypting data is essential.

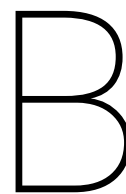
All this data needs to be sent to a datacenter where the measuring data will be fed to the Time-based Database for analysis and etcetera. To assure data integrity data validation is necessary and needs to be addressed both on the backend and on the frontend.

The data is privacy sensitive and therefore needs to be encrypted by law. When the measured data can be connected to a person the AVG applies.

We will use the gathered information to measure real-world sustainability-solutions and their performance-level in this situation. The gathered data will also be used to analyse the usage-efficiency of a building, apartment, room or office, and have the means to determine the exact consumption of each individual user, room, office or building for billing.

All this will result in analysis that will help determine and improve the effectiveness of energy-usage.

**Other information** We have a great accommodation where the students can come and work on the project. An enjoyable workspace with fast internet and a strong Wi-Fi. We expect the students to work on the project in the office, but if necessary, other arrangements can be made. We have got a clear picture of what the end product should look like and we will be working closely with the students on this project. On the journey of making this end product the students will have to figure out some things as they go along. Our Chief Technical Officer will be guiding the students during this process and (he) will monitor if the chosen path is viable and realistic. Our goal is to use this product in our current affairs and continuously improve the product from there. The duration of this project is set to 10 weeks. The students will individually receive €300 per month having earned a total of €800 by the end of the project. A bonus will be included after the finalisation of the product, if indeed we can directly implement it in our current services.



## Info Sheet

*The next page contains the info sheet for this project.*



Before

D. Hoonhout

M. Straatman

**A.N.I. Tarcy**

J.J.C. Vlekke

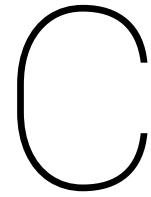
**A.D. de Vos**

**Client:** Humphrey James CTO, ASG Nederland B.V.

**Coach:** Mauricio Aniche    Software Engineering Research group, TU Delft

The final report for this project can be found at: <http://repository.tudelft.nl>





# Software Improvement Group Report

The Software Improvement Group report and the code quality is discussed in greater detail in section 7.6.

## C.1. First Submission

[Feedback]

De code van het systeem scoort 3.5 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code marktgemiddeld onderhoudbaar is. We zien Unit Size en Unit Complexity vanwege de lagere deelscores als mogelijke verbeterpunten.

Bij Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Dit kan verschillende redenen hebben, maar de meest voorkomende is dat een methode te veel functionaliteit bevat. Vaak was de methode oorspronkelijk kleiner, maar is deze in de loop van tijd steeds verder uitgebreid. De aanwezigheid van commentaar die stukken code van elkaar scheiden is meestal een indicator dat de methode meerdere verantwoordelijkheden bevat. Het opsplitsen van dit soort methodes zorgt er voor dat elke methode een duidelijke en specifieke functionele scope heeft. Daarnaast wordt de functionaliteit op deze manier vanzelf gedocumenteerd via methodenamen.

Voorbeelden in jullie project:

- `SerialProcessor.Read()`
- `SerialInputDevice.SearchSerialInputDevice()`

Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Dit betekent overigens niet noodzakelijkerwijs dat de functionaliteit zelf complex is: vaak ontstaat dit soort complexiteit per ongeluk omdat de methode te veel verantwoordelijkheden bevat, of doordat de implementatie van de logica onnodig complex is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, makkelijker te testen is, en daardoor eenvoudiger te onderhouden wordt. Door elk van de functionaliteiten onder te brengen in een aparte methode met een beschrijvende naam kan elk van de onderdelen apart getest worden, en wordt de overall flow van de methode makkelijker te begrijpen. Bij grote en complexe methodes kan dit

gedaan worden door het probleem dat in de methode wordtd opgelost in deelproblemen te splitsen, en elk deelprobleem in een eigen methode onder te brengen. De oorspronkelijke methode kan vervolgens deze nieuwe methodes aanroepen, en de uitkomsten combineren tot het uiteindelijke resultaat.

Voorbeelden in jullie project:

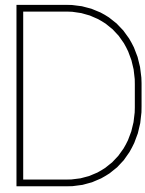
- `SerialSettings.GetModeByte(string)`
- `DataDecoder.DecodeBCD()`

De aanwezigheid van testcode is in ieder geval veelbelovend. De hoeveelheid testcode ziet er ook goed uit, hopelijk lukt het om naast toevoegen van nieuwe productiecode ook nieuwe tests te blijven schrijven.

Over het algemeen is er dus nog wat verbetering mogelijk, hopelijk lukt het om dit tijdens de rest van de ontwikkelfase te realiseren.

## **C.2. Second Submission**

Unfortunately, at the time of writing, we have not yet received feedback on our second submission. The submission was made on Thursday 20 June 2019.



## Ethical Evaluation

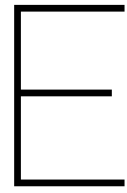
Considering the ethical implications of our work is an important part of making sure the solution will contribute to society. For our current solution, there are two ethical considerations that will be discussed. The first one is from the customer's perspective, and the second one is from the client's perspective.

### **D.1. Customer's Perspective**

Customers do not have the freedom to decide whether they want real-time measuring devices in their house or old-school manual read-out measuring devices. The housing corporation decides whether to place measuring devices in their apartments, and the customer does not have much of a say in it. You could argue that the customers do have freedom of choice, but the housing market has been very limited in recent years [29]. Many people cannot afford to decline housing offers and they are thus forced to take every opportunity of a housing offer, even when they contain real-time measuring devices. The responsibility, however, lies at the housing corporation and not with the client. It is up to them to contact every apartment owner and discuss the options for reducing the impact of real-time measuring devices.

### **D.2. Client's Perspective**

The client has the responsibility to secure the collected data and to use the data to improve the quality of life, minimize the impact on the environment and minimize the costs. The collected data is useful to many people, including surveillance agencies, marketing companies and burglars. It is thus important that the data is safely and securely stored where only the right people have access to the data. Leaking the data is not only a problem for the client, but it could also negatively impact all the customers. The current use for this data within the client's company is very ethical. It is mainly used to give stakeholders a detailed insight into their energy consumption and to identify how energy efficiency solutions can optimally be placed. Thus, it will benefit many different parts in society. The client's solution will decrease the energy costs for both the housing corporations as well as the customers. As the information will also be used to improve or design buildings with efficiency in mind, the client's solution will lead to a smaller carbon footprint for the building. A smaller carbon footprint benefits the climate.



# MoSCoW requirements

## **E.0.1. Description**

The solution concerns a gateway composed of a Single Board Computer with the input (serial dongle), output (SIM card dongle) and memory (USB stick and SD card). The solution should be able to perform numerous actions described in this document. The requirements have been described using the MoSCoW method in collaboration with the client. The main functionality is that the solution functions as a gateway, there is wireless communication between a slave (measuring device) and the stationary, usually mains powered master (gateway), whereas the solution is the master and the slaves are the measuring devices. The master should send the received data in near real-time to the servers where it will be processed.

## **E.0.2. Naming conventions**

- Slave: The measuring device
- Master: The gateway composed of the Single Board Computer, the input (serial dongle) and output (SIM card dongle) and memory (USB flash drive for data storage and SD card for the operating system)

## **E.1. Must Have**

- The master must be able to use the OMS protocol for serial signals
- The master must have a file of specified manufacturers
- The master must only consider the data received from certain manufacturers
- The master must have a file of whitelisted slaves containing the serial numbers for filtering and the keys of the slaves for decryption
- The master must be able to decrypt OMS messages received from the slaves
- The master must only decrypt and process signals from whitelisted slaves
- The master must be able to transform the decrypted message to CSV

- The master must append decrypted data to the active CSV file
- The master must create a new active CSV file every set interval
- The master must send CSV files that are ready to be uploaded in near real-time to the server (for a set interval)
- The CSV file must follow the specified format
- The master must allow its file of whitelisted slaves to be adjusted onsite
- The master must allow its file of whitelisted slaves to be read onsite
- The master must allow its file of manufacturers to be adjusted onsite
- The master must allow its file of manufacturers to be read onsite
- The master must keep track of all received slaves from specified manufacturers by saving the serial number, time of receiving (time stamp) and the signal strength (RSSI) in a file
- The master must store all data from unwhitelisted slaves from specified manufacturers as RAW data to prevent potential data loss
- The master must delete old RAW data, starting with the oldest, if the storage is full
- The master must be able to automatically restart after certain events, such as power loss
- The master must have a secure VPN connection with the server
- The server must be able to view the status of the masters based on received data
- The master must log all warnings, errors and exceptions to a log file.
- All group members must create proper documentation
- All group members must write automated unit tests for their code
- All group members must write StyleCop compliant code
- The group must improve its code based on the received SIG report (feedback)
- The master's settings must be easily configurable

## **E.2. Should Have**

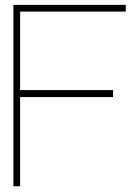
- The master should be able to be accessible through SSH
- The master should allow its file of whitelisted slaves to be adjusted offsite
- The master should allow its file of whitelisted slaves to be read offsite
- The master should allow its file of manufacturers to be adjusted offsite
- The master should allow its file of manufacturers to be read offsite
- The master should be able to check the RAW data and see if any slave that was previously not whitelisted but is now should be processed
- The master should be able to process the RAW data every time it restarts
- The master should process the RAW data every time the whitelisted slaves file changes
- The master should be able to detect changes in the whitelisted slaves file and update its in-memory data structures without having to restart
- The master should be able to detect changes in the manufacturers list file and update its in-memory data structures without having to restart
- The master should have a file with serial settings for the serial input device
- The master should be able to configure the serial input device based on a serial settings file
- The master should allow its file of serial settings to be adjusted on- and offsite
- The master should allow its file of serial settings to be read on- and offsite
- The master should be able to detect changes in the serial settings file and update the serial settings of the serial input device
- The serial device should store the serial settings on its non-volatile memory
- The master should be able to store its data on an encrypted storage device
- The master should be able to safely keep the AES keys in the whitelisted slaves file (XML file) by encrypting this file such that it minimalizes a possible breach. As the master's whitelisted slave file is stored on USB flash drive it and can easily be accessed by a possible intruder, it should keep this security issue in mind
- The master should have its operating system on an SD card and store all its data on a USB flash drive
- The master should synchronise its time with the internet
- The master should be able to receive serial data from the repeaters

### **E.3. Could Have**

- The documentation could be supported by a step-by-step guide for the installation of the master
- The master could output the received measurement devices on a web page
- The server could be able to analyze the data to create custom-tailored whitelists and update the concerning masters. To prevent two masters from measuring the same slave using the RSSI values of the slave messages
- The masters could be managed via a web interface
- The master could support version control such that its software can easily be updated
- The master could also act as a repeater of the signal such that all signals go to one 'main' master

### **E.4. Won't Have**

- Create a (waterproof) casing for the product



## Gantt Chart

*The next two pages contains our Gantt chart for this project.*









## OVS Example

*The next page contains an example of an OVS message with encryption.*

Note that there are still some bytes left but these are left out on purpose. These bytes will only contain some more data but the structure of the message should be clear without these bytes [19].

### N.4.3 wM-Bus Example with partial encryption

#### SND-NR (wM-Bus)

Byte No	OMS wM-Bus frame		Heat cost allocator example		Layer
	Field Name	Content	Bytes [hex]	Bytes [hex]	
			plain	AES coded	
1	L Field	Length of data (45 bytes)		2Dh	Data Link Layer (DLL)
2	C Field	Send - No Reply		44h	
3	M Field	Manufacturer code		93h	
4	M Field	Manufacturer code		44h	
5	A Field	Ident No LSB (BCD)		44h	
6	A Field	Ident No (BCD)		33h	
7	A Field	Ident No (BCD) (= 11223344)		22h	
8	A Field	Ident No MSB (BCD)		11h	
9	A Field	Version (or Generation number)		55h	
10	A Field	Device type (RF-Adapter)		37h	
11	CRC 1			69h	
12	CRC 1			EFh	
13	CI Field	72h (long header)		72h	Transport Layer (TPL)
14	Meter-ID	Ident No LSB (BCD)		88h	
15	Meter-ID	Ident No (BCD)		77h	
16	Meter-ID	Ident No (BCD) (= 55667788)		66h	
17	Meter-ID	Ident No MSB (BCD)		55h	
18	Meter-Man.	Meter Manufacturer code		93h	
19	Meter-Man.	Meter Manufacturer code		44h	
20	Meter-Vers.	Version (or Generation number)		55h	
21	Meter-Med.	Device type (Medium=HCA)		08h	
22	Access No.	Access Number of Meter		00h	
23	Status	Meter state (Low power)		04h	
24	Config Field	NNNNCCHHb (1 encr. block)		10h	
25	Config Field	BASOMMMMb (unidir., async., AES)		05h	
26	AES-Verify	Encryption verification	2Fh	00h	# 1
27	AES-Verify	Encryption verification	2Fh	DFh	
28	DR1	DIF (6 digit BCD)	0Bh	E2h	DLL
29	CRC 2			27h	
30	CRC 2			F9h	
31	DR1	VIF (HCA-units)	6Eh	A7h	# 1 Application Layer (APL)
32	DR1	Value LSB	34h	82h	
33	DR1	Value ( = 001234 HCA-Units)	12h	14h	
34	DR1	Value MSB	00h	6Dh	
35	DR2	DIF (Data type G, StorageNo 1)	42h	15h	
36	DR2	VIF (Date)	6Ch	13h	
37	DR2	Value LSB	FEh	58h	
38	DR2	Value MSB ( = 30.04.2007)	04h	1Ch	
39	DR3	DIF (6 digit BCD, StorageNo 1)	4Bh	D2h	
40	DR3	VIF (HCA-units)	6Eh	F8h	
41	DR3	Value LSB	56h	3Fh	
42	DR3	Value ( = 023456 HCA-Units)	34h	39h	
43	DR3	Value MSB	02h	04h	

# Bibliography

- [1]
- [2]
- [3]
- [4]
- [5]
- [6]
- [7] Abdel-Karim Al-Tamimi. Performance analysis of data encryption algorithms. 2006.
- [8] Devopedia. Iot operating systems. <https://devopedia.org/iot-operating-systems>. Accessed: 2019-06-04.
- [9] Mariella Di Giacomo. Mysql: lessons learned on a digital library. *IEEE software*, 22(3): 10–13, 2005.
- [10] Mikael Höök and Xu Tang. Depletion of fossil fuels and anthropogenic climate change—a review. *Energy Policy*, 52:797–809, 2013.
- [11] Intersoft Consulting. Gdpr encryption. <https://gdpr-info.eu/issues/encryption/>. Accessed: 2019-05-20.
- [12] Justin Meza, Qiang Wu, Sanjev Kumar, and Onur Mutlu. A large-scale study of flash memory failures in the field. In *ACM SIGMETRICS Performance Evaluation Review*, volume 43, pages 177–190. ACM, 2015.
- [13] Microsoft Contributors. Organize your project to support both .net framework and .net core. <https://docs.microsoft.com/en-us/dotnet/core/porting/project-structure>, . Accessed: 2019-05-13.
- [14] Microsoft Contributors. Isolating code under test with microsoft fakes. <https://docs.microsoft.com/en-gb/visualstudio/test/isolating-code-under-test-with-microsoft-fakes?view=vs-2015>, . Accessed: 2019-05-20.
- [15] Microsoft Contributors. Repo for managing moq 4.x. <https://github.com/moq/moq4>, . Accessed: 2019-05-20.
- [16] Microsoft Contributors. Windows 10 iot documentation. <https://docs.microsoft.com/en-us/windows/iot-core/>, . Accessed: 2019-04-23.

- [17] Niklas Modess. Npath complexity and cyclomatic complexity explained. <https://modess.io/npath-complexity-cyclomatic-complexity-explained/>. Accessed: 2019-05-16.
- [18] Mike Olson. Hadoop: Scalable, flexible data storage and analysis. *IQT quarterly*, 1(3): 14–18, 2010.
- [19] OMS-group. Open metering system specification. <https://oms-group.org/en/download4all/oms-specification/>. Accessed: 2019-04-23.
- [20] P3 and connect. The 2019 mobile network test in the netherlands. [https://tweaking.net/files/upload/NL%20Benchmark%202019%20V5\[1\].pdf](https://tweaking.net/files/upload/NL%20Benchmark%202019%20V5[1].pdf). Accessed: 2019-04-23.
- [21] Don Reisinger. Best usb drives 2019. <https://www.tomsguide.com/us/best-usb-drives,review-6150.html>. Accessed: 2019-06-12.
- [22] Ivan Ristić. Ssl and tls deployment best practices. <https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices>. Accessed: 2019-04-23.
- [23] Terry L Root, Jeff T Price, Kimberly R Hall, Stephen H Schneider, Cynthia Rosenzweig, and J Alan Pounds. Fingerprints of global warming on wild animals and plants. *Nature*, 421(6918):57, 2003.
- [24] Alberto Savoia. This-code-is-crap. <https://testing.googleblog.com/2011/02/this-code-is-crap.html>. Accessed: 2019-05-16.
- [25] Ken Schwaber. *Agile project management with Scrum*. Microsoft press, 2004.
- [26] Device Language Message Specification. Flag id list. <https://www.dlms.com/flag-id/flag-id-list>. Accessed: 2019-06-04.
- [27] Jawahar Thakur and Nagesh Kumar. Des, aes and blowfish: Symmetric key cryptography algorithms simulation based performance analysis. *International journal of emerging technology and advanced engineering*, 1(2):6–12, 2011.
- [28] The things network. How can microsd flash memory wear be minimized? <https://www.thethingsnetwork.org/forum/t/how-can-microsd-flash-memory-wear-be-minimized/15896>. Accessed: 2019-05-21.
- [29] Windy Vandevyvere, Andreas Zenthöfer, et al. The housing market in the netherlands. Technical report, Directorate General Economic and Financial Affairs (DG ECFIN), European ..., 2012.
- [30] Apparatuur voor elektrische energiemeting en belastingen regeling. *Communication systems for meters - Part 3: Application protocols*. Stichting Nederlands Normalisatie-Instituut.

- [31] Vincent Weijsters. Stylecop.analyzers documentation. <https://github.com/DotNetAnalyzers/StyleCopAnalyzers/blob/master/DOCUMENTATION.md>. Accessed: 2019-05-15.
- [32] Wikipedia Contributors. Cyclomatic complexity. "[https://en.wikipedia.org/w/index.php?title=Cyclomatic\\_complexity&oldid=883841144](https://en.wikipedia.org/w/index.php?title=Cyclomatic_complexity&oldid=883841144)", . Accessed 2019-05-16.
- [33] Wikipedia Contributors. Dynamic host configuration protocol. [https://nl.wikipedia.org/wiki/Dynamic\\_Host\\_Configuration\\_Protocol](https://nl.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol), . Accessed: 2019-05-20.
- [34] Wikipedia Contributors. Multi-level cell. "[https://en.wikipedia.org/wiki/Multi-level\\_cell](https://en.wikipedia.org/wiki/Multi-level_cell)", . Accessed 2019-06-18.
- [35] Shaun Wilde. Opencover github repository. <https://github.com/OpenCover/opencover>. Accessed: 2019-05-15.