LUT University
School of Energy Systems
Degree Programme in Electrical Engineering

Toni Naukkarinen

MODULAR IOT DEVICE FOR SOLAR POWER PLANT MONITORING

Examiners: Professor Pertti Silventoinen
M.Sc. Kimmo Huoman

ABSTRACT

LUT University
LUT School of Energy Systems
Degree Programme in Electrical Engineering

Toni Naukkarinen

Modular IoT device for solar power plant monitoring
2019

Master's Thesis
Pages 58, pictures 19, tables 3.

Examiners: Professor Pertti Silventoinen
M.Sc. Kimmo Huoman

Keywords: solar power, Internet of Things, NB-IoT, LoRaWAN, Sigfox, energy management

GreenEnergy Finland Oy (GEF) is a Finnish company founded in 2010 which sells solar power plants mainly through its network of retailers. GEF develops its own solar monitoring and energy management platform called GEF Vision. Connecting a power plant into Vision currently requires installing an embedded Linux based computer, which uses the customers internet connection in order to exchange data between the cloud server.

Customer internet connection has proven to be problematic for multiple reasons, which cause unnecessary work for GEF and its retailers. This led to an idea of creating an IoT-network based device for solar power plant monitoring. The design was made modular, so additional functionality can be added via interconnected modules.

This master's thesis compares three commercial IoT-network technologies available in Finland: LoRaWAN, Sigfox and NB-IoT. This thesis also describes the mechanical implementation, system level design and software architecture for the next-generation modular monitoring device.

TIIVISTELMÄ

Lappeenrannan-Lahden teknillinen yliopisto LUT
LUT Energiajärjestelmät
Sähkötekniikan koulutusohjelma

Toni Naukkarinen

Laajennettava IoT-laiteratkaisu aurinkosähkön tuotannonseurantaan
2019

Diplomityö
Sivumäärä 58, kuvia 19, taulukoita 3.

Tarkastajat: Professori Pertti Silventoinen
Diplomi-insinööri Kimmo Huoman

Hakusanat: aurinkosähkö, Internet of Things, LoRaWAN, Sigfox, NB-IoT, tuotannonseuranta, kulutuksenohjaus

GreenEnergy Finland Oy (GEF) on suomalainen vuonna 2010 perustettu yritys, joka maahantuo ja myy aurinkosähköjärjestelmiä. GEF kehittää sen omaa Vision-nimistä alustaa, joka on tarkoitettu aurinkosähköjärjestelmien tuotannonseurantaan ja kulutuksenohjaukseen. Voimalan yhdistäminen Visioniin tapahtuu tällä hetkellä Linux-pohjaisella sulautetulla tietokoneella, joka käyttää loppuasiakkaan internet-yhteyttä kommunikointiin GEF:n pilvipalvelun kanssa.

Loppuasiakkaiden internet-yhteydet ovat osoittautuneet ongelmallisiksi useista eri syistä, joka herätti ajatuksen luoda uusi IoT-verkkoa yhteysmuotona käyttävä tuotannonseurantalaite. Laitteen haluttiin olevan modulaarinen, jotta siihen voitaisiin lisätä ominaisuuksia asiakkaan tarpeiden mukaan.

Työssä vertaillaan kolmea Suomessa toimivaa kaupallista IoT-verkkoa, jotka ovat LoRaWAN, Sigfox ja NB-IoT. Lisäksi työssä on esitetty modulaarisen laiteratkaisun mekaniikkasuunnittelu, järjestelmäsuunnittelu ja ohjelmistosuunnittelu.

Preface

This thesis was made for GreenEnergy Finland Oy as part of the process of creating a next-generation solar power plant monitoring device for their energy management platform.

I'd like to thank GreenEnergy Finland Oy for providing the opportunity to work in this very interesting project. I would also like to thank my examiners Pertti Silventoinen and Kimmo Huoman for all their help and encouragement during this project. Last, but not least, I'd like to thank my family and friends who have supported me through this journey.

These past seven years have indeed been an epic quest towards graduation.

Lappeenranta, August 29th, 2019

Toni Naukkarinen

TABLE OF CONTENTS

USED SYMBOLS AND ABBREVIATIONS

Acronyms

2G	2nd Generation
4G	4th Generation
ADC	Analog to Digital Converter
AES	Advanced Encryption Standard
API	Application Programming Interface
APN	Access Point Name
ASCII	American Standard Code for Information Interchange
CoAP	Constrained Application Protocol
ECDSA	Elliptic Curve Digital Signature Algorithm
GEF	GreenEnergy Finland Oy
GPIO	General-purpose input/output
GSM	Global System for Mobile Communications
IoT	Internet of Things
JTAG	Joint Test Action Group
LTE	Long Term Evolution
MCU	Microcontroller unit
MQTT	Message Queuing Telemetry Transport
MQTT-SN	Message Queuing Telemetry Transport for Sensor Networks
NB-IoT	Narrowband Internet of Things
RAM	Random-Access Memory
RTOS	Real-Time Operating System
SPI	Serial Peripheral Interface
SWD	Serial Wire Debug
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
USART	Universal Synchronous/Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
WLAN	Wireless Local Area Network

1. INTRODUCTION

GreenEnergy Finland Oy (GEF) is a Finnish company founded in 2010. Its main business consists of importing and selling solar power plants. GEF mainly operates through its network of retailers but it also designs and sells larger projects independently. In addition, GEF develops and maintains its own solar power monitoring and load control platform called GEF Vision. Vision consists of on-site solar power plant monitoring devices, the map view and the monitoring views for end customers. Devices installed on-site send data to the Vision cloud service and this data is then processed and made available to the user interfaces. Retailers, installers and customers with multiple solar power plants have access to the map view, which they can use to view the status of installed systems collectively. A screenshot of the map view is displayed in Figure 1.

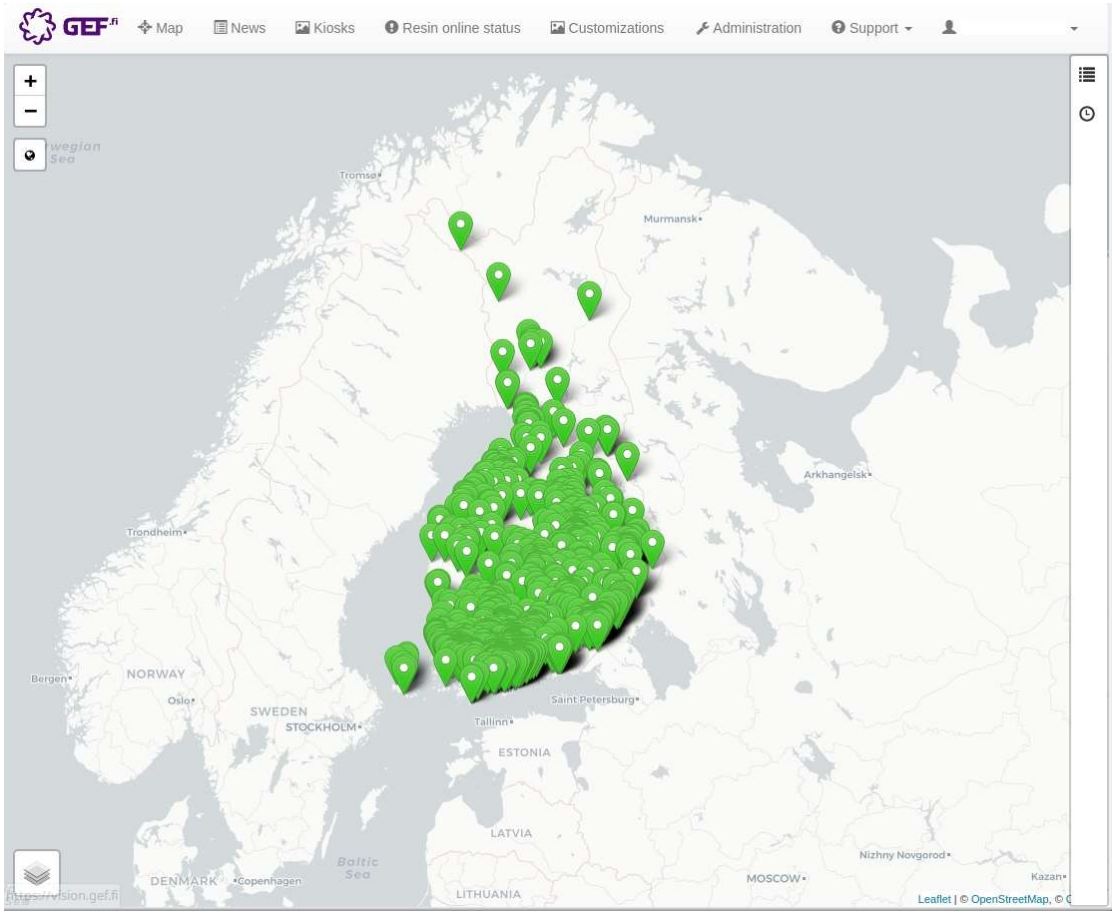

Figure 1. The map view of GEF Vision

The development of Vision has occurred in a relatively rapid pace. For this reason, no application-specific hardware has been developed. The current off-the-shelf device, which is called GEF Reader is based on Raspberry Pi 3 and installed in a DIN-compatible housing. The device needs internet connectivity which can be provided via Ethernet or Wireless Local Area Network (WLAN). The internet connectivity must be provided by the customer. Reader runs a Linux distribution called balenaOS, which is designed for high-reliability embedded applications. The current device sends measurement data and receives control commands via the Message Queuing Telemetry Transport (MQTT) protocol. The device is also connected to balenaCloud service, which is used for remote access, device configuration and software update distribution. The current communication diagram for GEF Reader and Vision is shown in Figure 2.

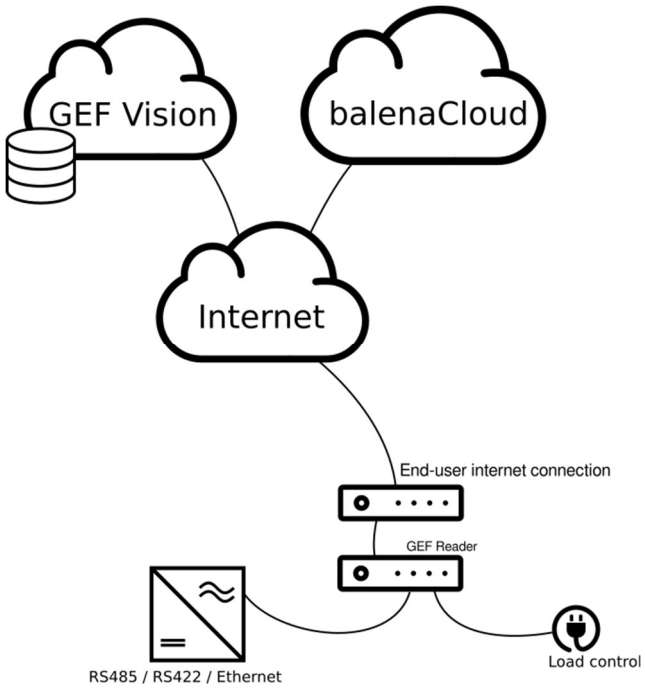

Figure 2. Communication diagram for GEF Reader. The current solution connects to the cloud using customer internet connectivity. Bus adapters are required for device connections.

The main troublemaker in this system has proved to be the end-customer internet connections. Among private customers changing wireless network credentials, badly behaving

routers and low signal levels generate workload for retailers and GEF employees. Bad WLAN signals are the most common problem, and it cannot be improved with the current hardware, since it only has a small integrated antenna on its circuit board. In corporate networks firewalls and IT-departments cause unnecessary workload and delays. These problems quickly accumulate to tens of hours of wasted time. Another problem with the current device is the connection between the GEF Reader and the target device. The connection requires different kinds of Universal Serial Bus (USB)-connected bus converters which vary depending on the target device and this adds difficulty to the installation. The most common installation includes a Fronius inverter and the RS-422-cable used with these installations has proved to be very expensive for GEF.

The system has now been developed to a point where it can be considered stable. The most common features required in a typical installation are known, along with gained experience from typical problems with the current device solution. With this knowledge the next generation device was planned. This new device concept consisted of a microcontroller based, modular device which would connect to the cloud using an Internet of Things (IoT) network. This solution would modify the communication diagram in Figure 2 to one presented in Figure 3.

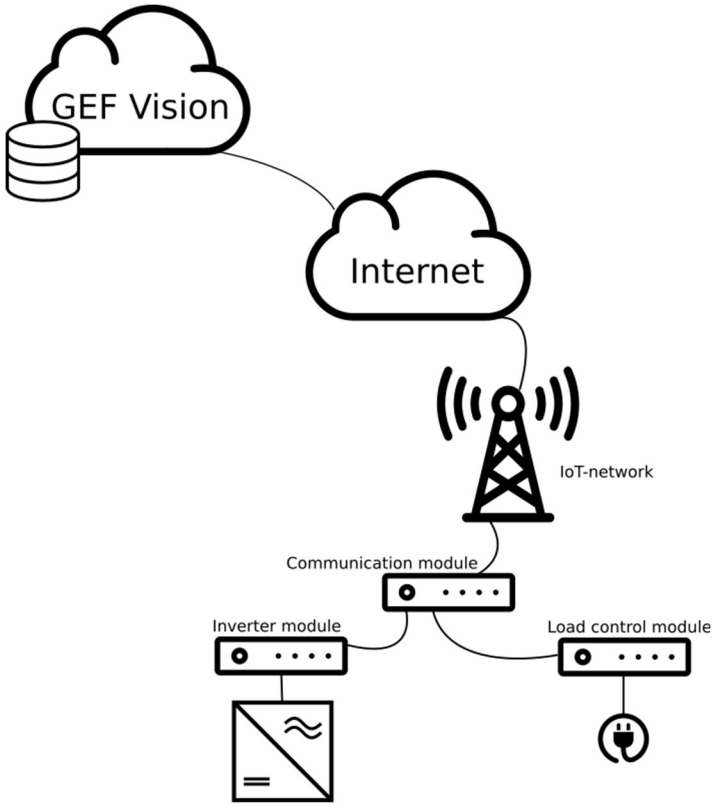

Figure 3. Communication diagram for the planned device. Communication module is the device presented in this thesis.

Modularity would cut off costs for a typical installation, since only the most common features are included in the main module. The modular design would consist of one master module, that takes care of communication with the Vision cloud server. This master module could then be extended with different and cheap feature modules that are interconnected with the master module using a communication bus. These modules can then be used to add different features depending on the customer's needs. Existing installations can also be easily extended with new features. This idea was combined with the rise of different IoT-networks in Finland, which in turn would solve the problem with end customer internet connections.

1.1 Objectives of the work

The objective for this thesis was to design the communication module for the device concept described in the introduction. The starting point was to find out what kind of different IoT-networks are be available for the product and which one is the most fitting for this use case. There were also some requirements given by GEF. These requirements include:

- The communication latency between the cloud service and the device must be 15 seconds at maximum.
- Master module should have a width of 1 DIN module (17.5 mm).
- Modules must be attachable to each other mechanically and electrically.
- Device can be powered from the power supply the Fronius RS-422 bus offers.
- The modules must be installable in a common household electrical switchboard.
- The master module should have the most commonly used features.
- The minimum message payload to the cloud service is 128 bytes.

1.2 Outline of the thesis

Chapter two introduces different IoT-networks available in Finland that were considered for this device. These three networks and their features are compared and the network for the device implementation is chosen.

Chapter three covers the mechanical design of the master module. This includes the selection for the housing used and description on how it makes possible to connect the modules together mechanically and electrically.

Chapter four focuses on the general system design in a block diagram level and explains the different main components that were determined to be required for the master module. The chapter also describes which main components were chosen.

Chapter five describes the software architecture of the device. This includes the software components required for the system and a description on how they are designed.

Chapter six gives a summary of the topics in this thesis and gives a brief review on the state of the prototype as well as ideas for future development.

2. STATE OF INTERNET OF THINGS NETWORKS IN FINLAND

Three commercially available networks for internet of things devices in Finland were selected for comparison: LoRaWAN, Sigfox and Narrowband Internet of Things (NB-IoT). All the networks are based on different technical solutions and they provide different features and restrictions. From these three network solutions Sigfox and LoRaWAN operate on unlicensed industrial, scientific and medical (ISM) frequency bands and require building of new base stations for coverage. NB-IoT is based on existing 4th generation (4G) cellular networks and can be done as an upgrade to the existing network. Unlike Sigfox and LoRaWAN it operates on the licensed 4G frequency bands. This chapter introduces these three networks, their features and their differences.

2.1 LoRaWAN

LoRaWAN is an open standard that defines the communication protocol and system architecture that is used with compatible devices. The standard is developed by the LoRa Alliance. LoRaWAN uses the Long Range (LoRa) protocol as it's physical communication layer, which is a patented product by Semtech. LoRa devices can communicate directly with each other in a peer-to-peer manner without LoRaWAN: it just specifies the protocol layer. Even though the LoRaWAN standard is open, the LoRa physical layer protocol itself is not. (Silva, et al., 2017)

Data rates of around 290 bits per second to 50 kilobits per second are possible in a LoRaWAN network with an extremely low power consumption. Communication ranges can vary from about 5 kilometers in an urban area to almost 50 kilometers in rural areas. It is commonly advertised that LoRaWAN devices can reach up to 10 years of battery life. (Kais, et al., 2018)

LoRaWAN network consists of 5 different components. These components are the LoRaWAN connected devices, base stations, gateways, network servers and application servers. Communication path for the LoRaWAN network is presented in Figure 4. (Kais, et al., 2018)

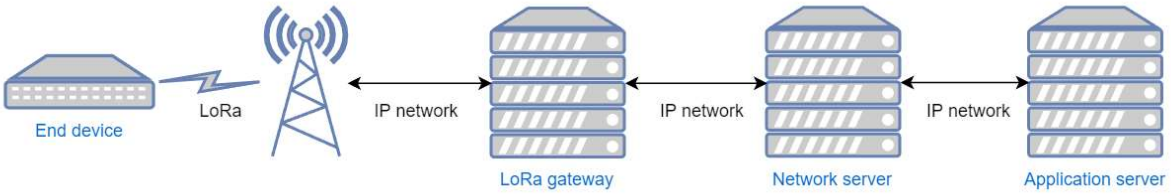

Figure 4. Communication path in a LoRaWAN network. The end device communicates with the LoRaWAN gateway using LoRa modulation. LoRaWAN network servers transfer data between the application server and the device.

The end devices communicate with the LoRaWAN network via the base station. This communication utilizes the LoRa modulation. Gateways are connected via faster links to network servers, usually using the Internet Protocol (IP). The network servers perform authentication checks and control the adaptive data rate of the protocol. The network server then determines what data should be sent to the actual connected device (if any). The network servers dispatch data from the device to the application server which in turn provides the network servers with the data that should be sent to the connected device. (Sinha, et al., 2017)

The maximum payload for one LoRaWAN transmission is 243 bytes. This payload can be sent once every couple of minutes, which makes LoRaWAN not suitable for real-time communication. The connection between the end device and the base station is encrypted using a 128-bit Advanced Encryption Standard (AES) keys. There are separate keys for encrypting the traffic between the base station and the device as well as encrypting the message payload itself. This means that the network cannot see the contents of the message payload. (Kais, et al., 2018; LoRa Alliance, Inc, 2017)

LoRaWAN network in Finland is operated by Digita. Current LoRaWAN coverage is presented in Figure 5. (Digita Oy, 2019)

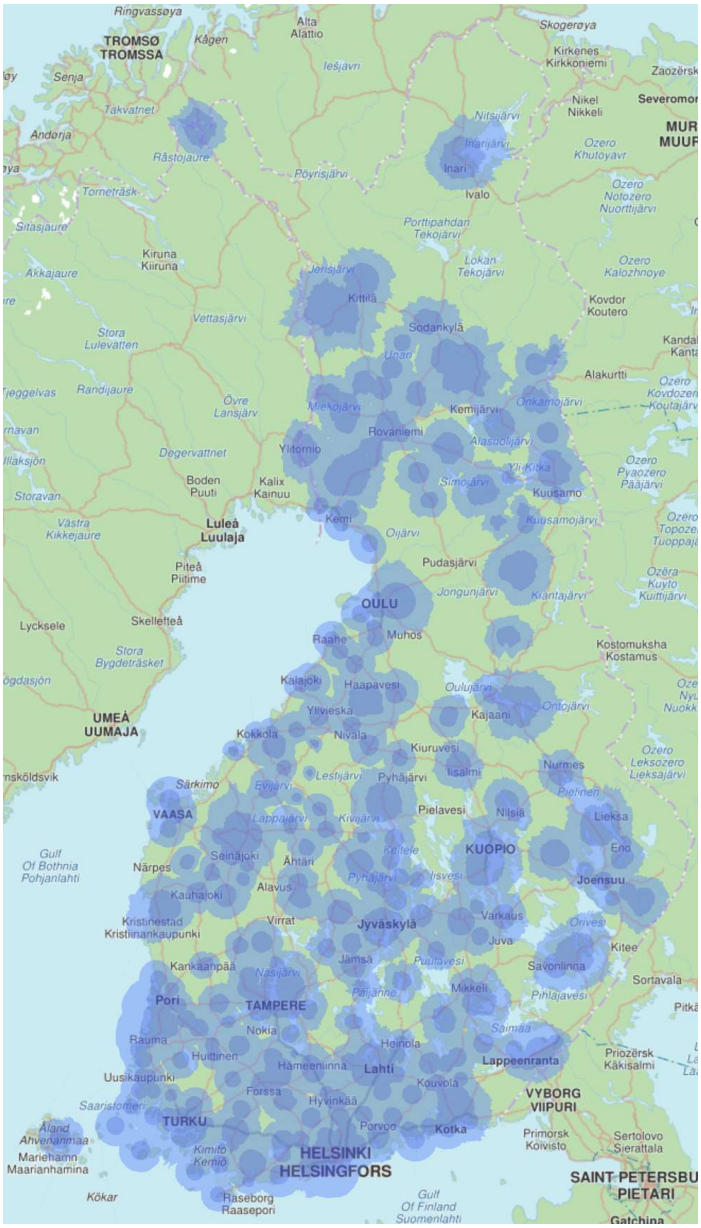

Figure 5. LoRaWAN coverage for network operated by Digita as of July 2019. Light blue areas indicate outdoor coverage, while the dark blue areas indicate indoor coverage. (Digita Oy, 2019)

Devices in the LoRaWAN network are divided into three classes: class A, class B and class C. The device class determines how often the end device can receive data:

- **Class A**: When the end device initiates communication with the network, there are two short windows when the network can transfer data to the device. The end device determines when data can be sent to the device. Class A is usually used with low-power applications.
- **Class B**: This class adds scheduled delivery windows in addition to the receive slots used in class A. The device periodically wakes up to listen for a message.
- **Class C**: The end devices have almost constant window open for reception. This is usually used with non-battery powered devices that require near constant bidirectional communication.

Duty cycle limitations limit the number of messages that can be sent from the device. For LoRaWAN a message can be sent every couple of minutes at maximum. This number is further decreased by the number of nodes connected to the the base station. (Adelantado, et al., 2017)

2.2 Sigfox

Sigfox is a commercial IoT-network solution developed and maintained by Sigfox S. A. Unlike LoRaWAN, which is an open standard, Sigfox is a closed system. Both the cloud service and communication protocol are proprietary. The cloud service which communicates with Sigfox connected devices and application servers is operated by the Sigfox S. A., but the physical networks are operated by different licensed commercial operators in different countries. In Finland the Sigfox network is operated by Connected Finland Oy. (Connected Finland Oy, 2019)

The end devices communicate with the Sigfox base stations, which in turn use Internet Protocol (IP) networks to communicate with the Sigfox Cloud. Sigfox Cloud handles passing

data and receiving data from the application server and therefore acts as an application firewall as well. The communication path for Sigfox network is displayed in Figure 6. (Sigfox SA, 2017)

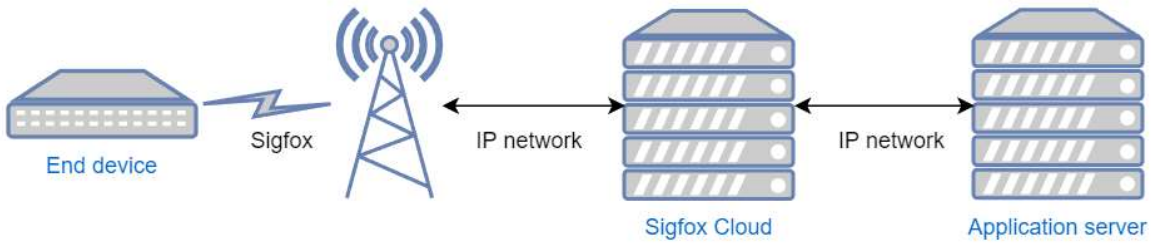

Figure 6. Communication path for Sigfox network. End devices communicate with Sigfox base station using the Sigfox protocol. Sigfox cloud handles communications between the application server and the device.

Messaging between the end device and the Sigfox base station is based on binary phase-shift keying (BPSK) modulation on an ultra-narrow bandwidth of 100 Hz. The narrow bandwidth makes Sigfox use the frequency band very efficiently and with high noise immunity, but it sacrifices in transfer speeds which are only around 100 bits per second. (Kais, et al., 2018)

The data between the end device and the Sigfox base station is not encrypted by default. Sigfox provides a method to encrypt communications between the base station and the device. Customer can also implement custom payload encryption. Data transfers between the base stations and the Sigfox Cloud and the transfers between the Sigfox Cloud and the application servers are always encrypted. If the message payload itself is not encrypted with some custom method, the message payload is visible for the Sigfox Cloud service. (Sigfox SA, 2017)

Sigfox networks provide the largest coverage per base station of the compared technologies. The coverage for a single base station can be as high as 40 kilometers. Sigfox is also very resistant to interference. (Kais, et al., 2018)

The current coverage of the Sigfox network in Finland is presented in Figure 7. (Sigfox, 2019)

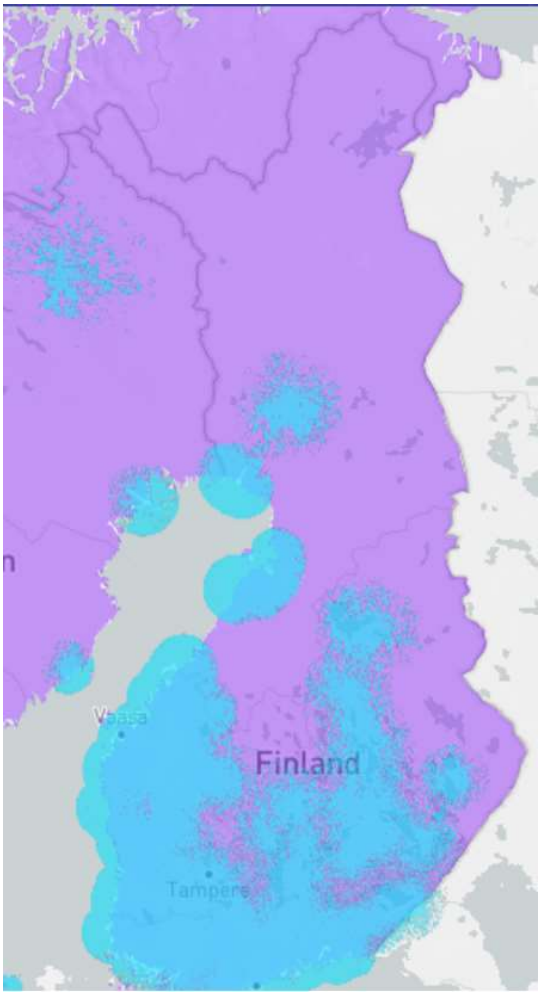

Figure 7. Sigfox coverage in Finland as of July 2019. The cyan area represents current coverage and the purple area means that the country is currently under roll-out (Sigfox, 2019).

The payloads of Sigfox messages are very small. Payloads can have a maximum length of 12 bytes from device to network and 8 bytes from network to device. As Sigfox operates on the unlicensed ISM bands like LoRaWAN, it is also subject to the duty cycle regulations. In Sigfox network, there is a limit on how many messages can be delivered daily. The limit is 140 messages from device to the network. From network to device, Sigfox guarantees

delivery of 4 messages per day. Sigfox is therefore unsuitable for real-time communications. The receive window for the device is open only when it has transmitted first. (Kais, et al., 2018)

2.3 Narrowband Internet of Things

Narrowband Internet of Things (NB-IoT) is an extension standard for cellular 4G networks that are specifically designed for IoT-devices. It is based on Long Term Evolution (LTE) standard and therefore operates under licensed LTE frequency bands, unlike LoRaWAN or Sigfox which operate on unlicensed ISM bands. Even though these protocols extend the Global System for Mobile Communications (GSM) standard, they are not compatible with existing GSM devices. NB-IoT is designed for coexistence with existing LTE networks. (Rohde & Schwarz GmbH & Co. KG, 2016)

Compared to Sigfox and LoRaWAN, NB-IoT uses the most power. Low power usage with NB-IoT can however be achieved by taking advantage of two LTE features. These features are called Extended Discontinuous Reception (eDRX) or and LTE Power Saving Mode (LTE PSM) . LTE network normally has an around 1.28 second paging window in which the base station can contact the end device and send any data that has been queued for it. The eDRX feature allows the device to sleep for extended periods of time between these windows. The PSM mode is used to achieve similar functionality as in LoRaWAN or Sigfox devices: the device can inform the base station that it is going to sleep indefinitely. When the end device wakes up and sends data to the base station, there is a short receive window in which the base station can send any queued data back to the device. During the sleep period the device still stays registered to the network, but it is not reachable. (Sultania, et al., 2018)

A bandwidth of 180 kHz is required by the NB-IoT transmission, which is very high compared to LoRaWAN or Sigfox. The same bandwidth is shared for uplink and downlink. NB-IoT can be deployed to cellular networks in three different operating modes which are shown in Figure 8. (Rohde & Schwarz GmbH & Co. KG, 2016)

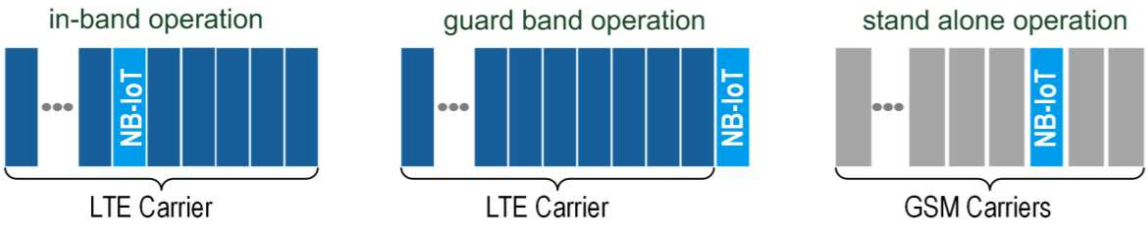

Figure 8. NB-IoT operation modes (Rohde & Schwarz GmbH & Co. KG, 2016). NB-IoT can be deployed in three different ways with or without an existing LTE network.

The three deployment options compare as follows. The deployment method does not matter to the end device and it is completely transparent. (Wang, et al., 2017)

- **In-band:** LTE network operator can replace one 200 kHz LTE carriers with NB-IoT. Because the technology is designed with coexistence in mind, in-band deployment has no effect on the performance of the regular LTE network nor the NB-IoT network.
- **Guard band:** LTE uses guard bands between carrier signals in order to reduce interference between carrier frequencies. These guard bands are at least 180 kHz, which is enough for NB-IoT deployment. In guard-band deployments interference from the existing LTE network is possible, but its effect is very small. (Adhikary, et al., 2016)
- **Standalone:** The third option is to deploy NB-IoT in standalone mode. In this mode NB-IoT carrier is used to replace one of the regular 200 kHz GSM carriers in a network. (Mangalvedhe, et al., 2016)

NB-IoT provides the smallest coverage per base station of the three technologies compared. The coverage is around 10 kilometers per base station. Due to lower bandwidth and data rate, NB-IoT offers much better penetration than traditional LTE networks. This means it is more reliable for indoor data sensors and monitoring devices. (Kais, et al., 2018)

NB-IoT offers significantly larger payload sizes than LoRaWAN and Sigfox with the maximum payload size being 1600 bytes. In addition to the largest payload size, NB-IoT also offers the possibility for real-time communication with a maximum latency of 10 seconds. The connections are encrypted using LTE encryption, which is the encryption standard in 4G networks. As regular internet protocols can be used with NB-IoT payloads can be encrypted using common protocol encryption methods such as Transport Layer Security (TLS). (Kais, et al., 2018)

Of the three major cellular operators in Finland, Elisa and DNA didn't provide a coverage map that shows the current NB-IoT coverage specifically. Elisa advertises to have NB-IoT coverage in every municipality in Finland (Mareti Media Oy, 2018). Telia however provides an NB-IoT specific coverage map which can be seen in Figure 9.

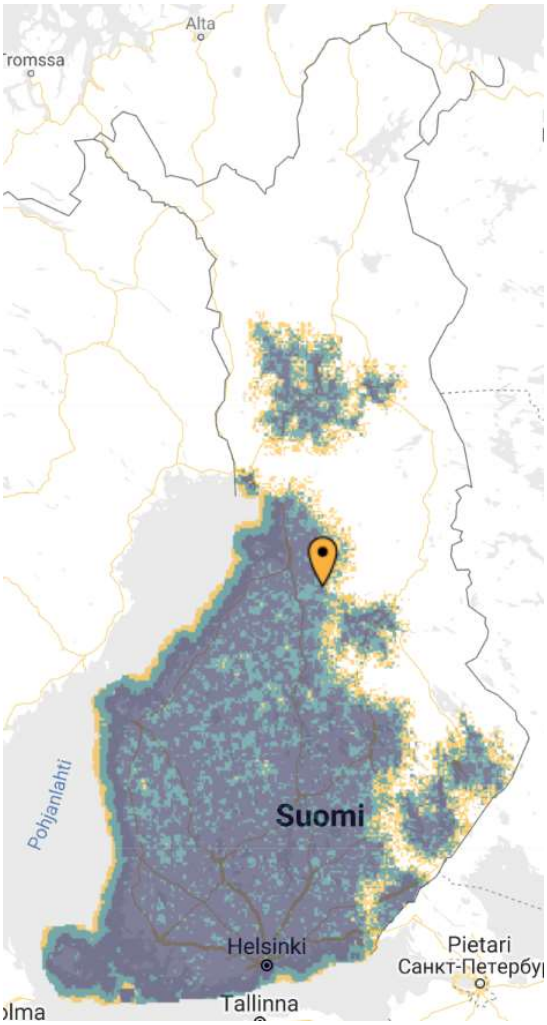

Figure 9. Coverage map for NB-IoT network operated by Telia in July 2019 (Telia Finland Oyj, 2019)

If the coverage map of Telia is compared to the map view of GEF Vision presented in Figure 1, it can be seen that majority of the currently installed power plants are under the coverage of the NB-IoT network operated by Telia.

2.4 Choosing the network technology for the device

Sigfox offered very small payloads and low message throughput, which didn't make it usable in this case. The maximum number of daily messages and the lack of real-time communication don't fulfill the requirements for the master module.

LoRaWAN was a potential candidate, but since the system requires real-time communication, it can't be utilized. The maximum payload size of 243 bytes is enough, but the communication interval is not.

The limitations of Sigfox and LoRaWAN leave NB-IoT as the only possible choice. The 10 second maximum latency along with the maximum data rate of 62.5 kbps are enough for data transmission requirements of this application. Because NB-IoT uses the existing 4G cellular networks, which have very good coverage in Finland, it was also thought to provide nationwide coverage faster than other technologies. The estimated battery life or power consumption was not the most important factor, but the low-power nature of NB-IoT is enough for the device to be powered from the Fronius RS-422 bus.

Because NB-IoT devices can use IP-based connections directly and don't rely on an external cloud service to dispatch messages, it also offers the benefit of using existing infrastructure that GEF has. Sigfox or LoRaWAN would have required designing and building additional Application Programming Interfaces (API) for receiving and sending data from and to the device.

3. MECHANICAL DESIGN

One of the requirements for the device is that it could be installed to a standard DIN rail. The device housing must be chosen in a way it can be installed in a typical household switch-board. Two options for the casing are evaluated*:* Modulbox One from ITALTRONIC and the BC-series from Phoenix Contact.

From the DIN-rail installable housings produced by Italtronic, the Modulbox One series is considered. The 1M housing from the series is presented in Figure 10. (ITALTRONIC Srl, 2019)

Figure 10. Italtronic Modulbox One (ITALTRONIC Srl, 2019)

The Italtronic housing doesn’t offer any kind of ready solution for connecting the modules together. This means that a custom implementation for interconnecting the modules would need to be implemented. This leads to additional costs in both manufacturing and design of the device.

The other housing considered is the BC-series of modular housings from Phoenix Contact. In general, the housings are similar, but the BC-series offers a ready solution for connecting the cases together mechanically and electrically with a connector called HBUS. The HBUS connector is installed to the bottom of the case while retaining their ability to be installed to a standard DIN rail. The housings and bus connectors are available in variable sizes from one to nine DIN width units (17.5 to 157.5 millimetres). Two different types of housings from the BC series is displayed in Figure 11. (Phoenix Contact Oy, 2019)

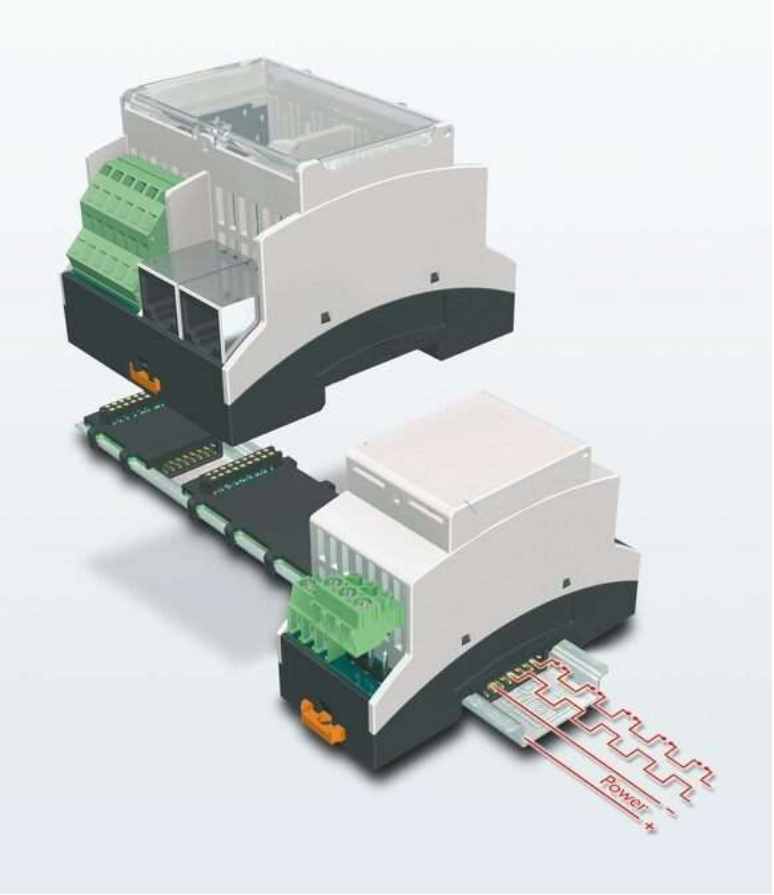

Figure 11. Modular housings and HBUS-connector from the BC series (Phoenix Contact Oy, 2019). HBUS allows the device to provide an extension connector while retaining the possibility to be installed in a DIN rail.

The HBUS-connector has a total of 16 pins, which can be used to transfer power and signals between the modules (Phoenix Contact Oy, 2019). If a custom connector solution is

designed, there is a risk of not making it extendable enough for future use. The Phoenix Contact housing is selected for the device, instead of developing a custom solution for the Italtronic housing.

SolidWorks was used to model the master module in 3D. This model is used to determine whether the required connectors can fit inside the housing and what kind of modifications the housing requires. The model is also used to evaluate the remaining surface area of the circuit board that could be used for components, A rendering of this 3D model is presented in Figure 12.

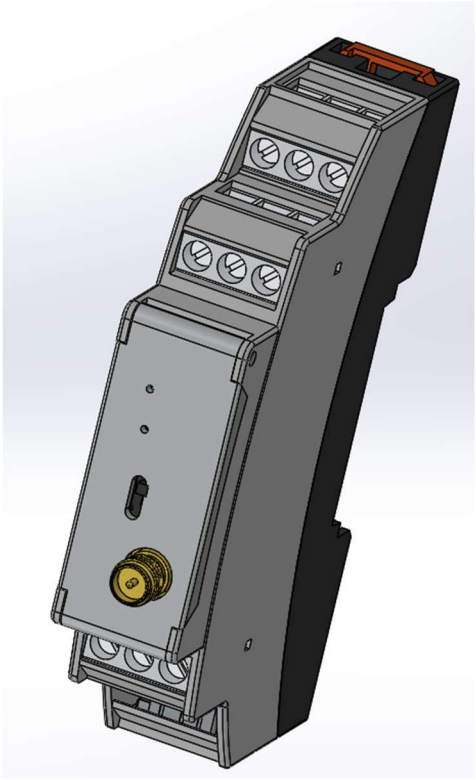

Figure 12. Concept picture of the NB-IoT master module

With the aid of 3D-modelling it can be determined that required functionality is achieved with the selected housing. Minor modifications need to be done in order to fit all the necessary connectors and user interface components.

4. GENERAL SYSTEM DESIGN

The system design is based on a master-slave architecture. The system consists of one master module, which handles communication with the cloud service, and multiple slave modules. Slave modules are used to add support for different devices and features to the modular system. For example, additional modules can be made for electric car chargers, energy metering or different inverter models. A block-level diagram of an example system configuration is presented in Figure 13.

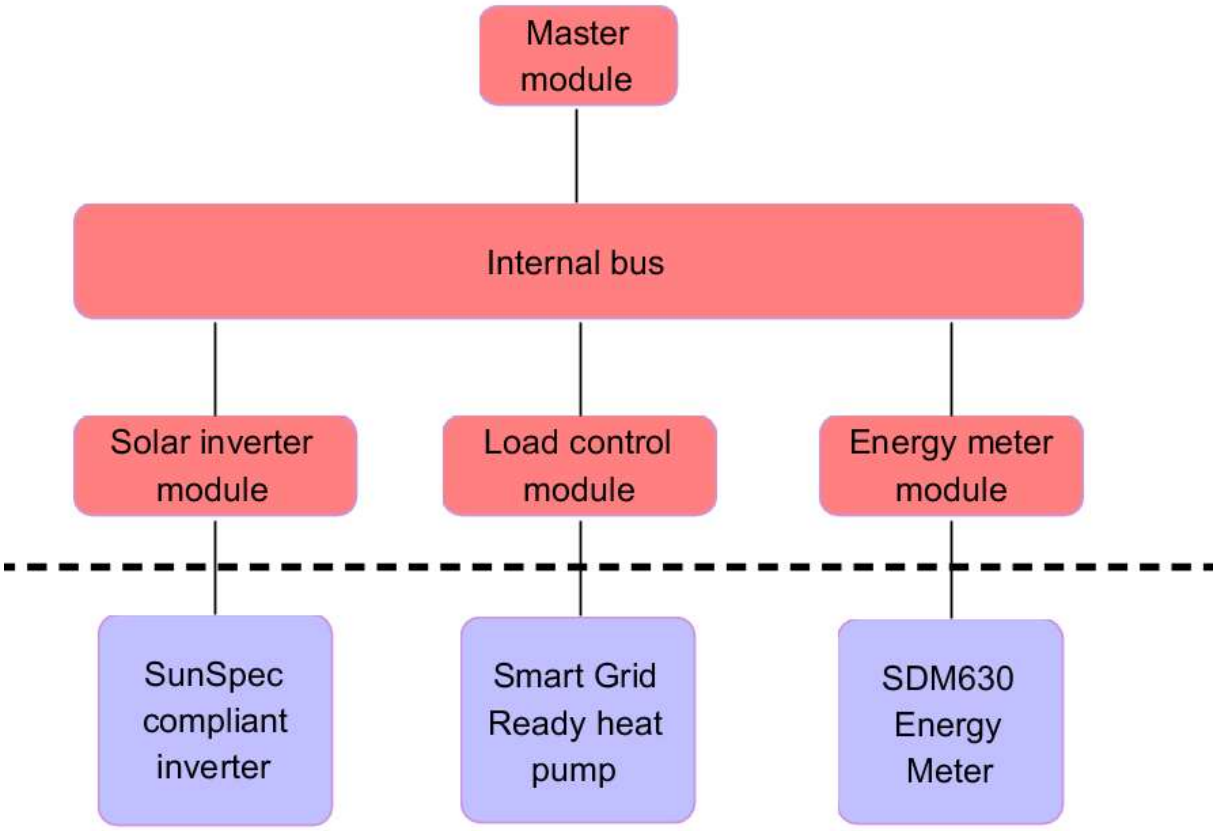

Figure 13. Block diagram of an example system with modules. Cyan blocks represent external third-party devices and red blocks functionality made by GEF.

The functionality required in a most common household installations delivered currently by GEF is reading data from a Fronius Symo -inverter and controlling the domestic water heater. Based on this typical installation scenario, the features required for the master module are chosen.

Fronius inverters have a RS-422 bus for communication, which uses a custom protocol (Fronius International GmbH, 2019). For this reason, a RS-422 converter is required. In order to turn a water heater (or other connected load) on and off, a single solid-state-relay is added to the requirements for controlling a contactor.

In addition to these, an input for a single current transformer was added. A current transformer can be used to measure if the water heater is drawing power, or whether the required water temperature has been reached and the power draw has been stopped by a thermostat. The current measurement is used to visualize the water heater run time but also with demand response. In demand response, the grid operator (Fingrid in Finland) can control connected loads remotely in order to adjust the grid frequency. In domestic households this load is commonly a water heater. Fingrid requires confirmation on whether the load activated or deactivated successfully or not. With these requirements a block diagram of the main components used in the master module is presented in Figure 14. (Fingrid Oyj, 2018)

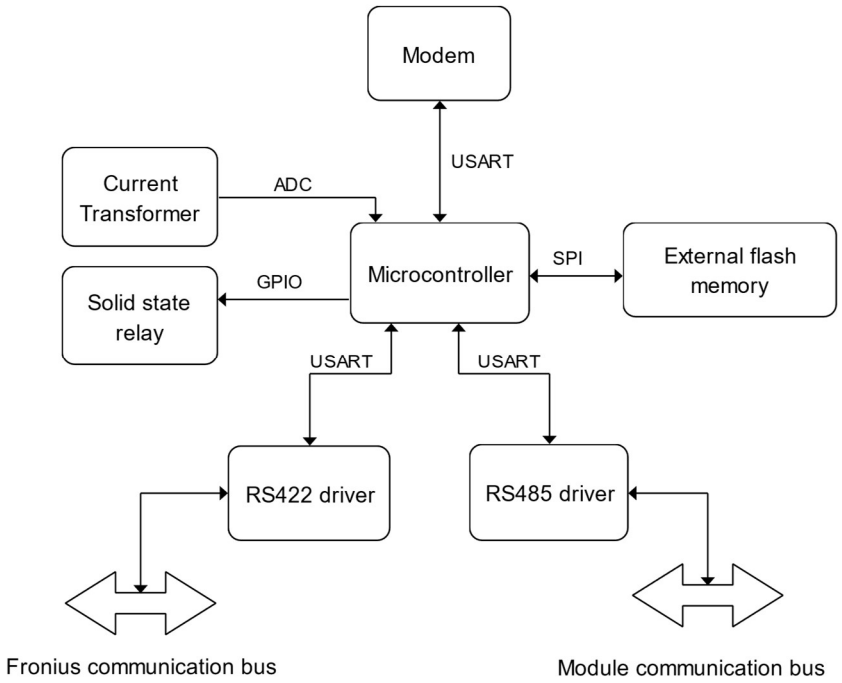

Figure 14. Block diagram of the master module components. Arrows represent the direction of the data flow. Arrow descriptions indicate the peripheral used for communication.

For future needs an external flash memory is added. This adds the possibility to easily save configuration data, firmware updates or any other data. Using an external flash chip also makes it possible to use the whole internal flash memory capacity of the microcontroller unit (MCU) to the firmware. Without an external flash memory, the internal flash of the microcontroller would have to be split, because the firmware updates must be downloaded somewhere before flashing and the random-access memory of the microcontroller is not big enough to store the whole firmware file for an application of this scale.

The communication between the master module and the extension modules is implemented using RS-485 and a master-slave architecture. RS-485 defines the electrical standard for a multiple device bus. The bus is half-duplex, which means that only one device can send at a time while the others act as receivers. Since RS-485 provides only the electrical standard,

the communication protocol must be defined in software by the application developer. This gives flexibility, since the communication can be implemented completely as required by the application. In this application the master module is be the bus master that polls the information from the extension modules, which act as slaves. RS-485 bus can be extended for a relatively long range, so the modules can be installed physically separately if needed. (Texas Instruments, 2016)

4.1 Choosing the main components

After the main components are recognized the actual components are selected for the master module implementation. The two main components for the project that define the rest of the development process are the microcontroller and the NB-IoT modem. There exists many different RS-422/RS-485 drivers on the market, as well as solid state relays and Serial Peripheral Interface (SPI) flash chips. They have relatively little difference in the implementation, so specific chip models are not locked and specified in this thesis.

4.1.1 Microcontroller

For speeding up the development an MCU architecture that the software developer has previous experience is chosen. Because the requirements for the MCU are mainly the peripherals needed for the device the selection can be made mainly based on the price. Power consumption is not a major factor for the device, since it is not battery powered. However, power consumption is considered because the device is designed to be able to be powered from the power supply provided by the Fronius inverter RS-422 bus. The bus offers a 12-volt power supply with a maximum current of 300 milliamperes. (Fronius International GmbH, 2019)

Two possible choices are compared for the microcontroller platform. These were the Nordic Semiconductor nRF9160 and a more traditional solution with a separate modem and a microcontroller. The nRF9160 includes a Cortex-M33 microcontroller and a NB-IoT compatible modem in a single IC chip, achieving the lowest footprint in the market. The nRF9160

was released in December, which makes it a relatively fresh product. The roadmap for the software development kit makes it not possible to use within the timespan for this project. (Nordic Semiconductor, 2019)

After consideration the ST Microelectronics STM32L4-series is determined to have required communication capabilities with small enough power consumption. The firmware size is unknown during the start of the software development so the STM32L476 was chose as the development MCU, because it has the largest amount of flash memory and random-access memory (RAM) from the series. The series is compatible with each other (pin and peripheral counts are different between MCU's), so the chip could be scaled down to a model with less RAM and flash after the size requirements are known, reducing the price. (ST Microelectronics, 2018)

4.1.2 Modem

In addition to the MCU, the NB-IoT modem is the other main component of the device that is specified in the early stage of the project. Two types of modems are considered for the master module. These include modems with 2nd generation (2G) cellular network fallback capability, and NB-IoT only modems. Modems with 2G-fallback option attempt to register to a regular GSM network in case a NB-IoT network cannot be found. These modems are generally more expensive and have a larger footprint. (SIMCom Wireless Solutions Co., Ltd, 2017; Quectel Wireless Solutions Co., Ltd, 2019)

The modem selected to the master module is the BC66 manufactured by Quectel. It is a NB-IoT only modem with no 2G-fallback. The modem offers a variety of communication protocols with very small footprint and low power usage. The documentation of the modem also proved to be in a good shape, which supported its selection. The choice was locked after a phone call with a sales representative of a large semiconductor distributor in April of 2019. (Quectel Wireless Solutions Co., Ltd., 2019)

5. SOFTWARE ARCHITECTURE

The main requirement of the software architecture is that the software should be modular and therefore unit testable. The software also requires abstraction so it can be run with a personal computer or a Raspberry Pi, and not necessarily just the target hardware. This makes it possible to start development of the modem driver using a Linux-based computer, which is simpler in practice than being developed directly using the target hardware.

Software is designed in a way that the most common parts of its functionality can be unit tested. Unit testing is a way to automate testing of software. In unit testing a separate test program is written that calls functions with specified input values. The output values can then be compared to known-good values. Comparing of values is done with assertion functions. If the values match, the test is passed. Unit tests were added by using the Unity-framework, which is a unit testing framework for embedded software. Unit testing is used to improve software quality and security and can be done in an automated manner. (ThrowTheSwitch.org, 2019)

Unity can be used together with a framework called *CMock* which is also developed by *ThrowTheSwitch.org*. *CMock* can be used to mock hardware specific functions for unit tests. Mock functions make it possible to test the code without running it on the actual hardware. (ThrowTheSwitch.org, 2019)

The code base is made in a way that it can be reused as much as possible between the master module and the extension modules. This is done in order to save time in the future when developing extension modules for the system. Common drivers allow the same MCU to be used in these modules as well. In software all the buffer sizes are checked before operations and standard C library functions that don't check the buffer size are not used (for example *sprintf* vs. *snprintf*). This reduces the risk of buffer overflow vulnerabilities.

Because the software consists of different interconnected functionality that needs to be run in a parallel manner, a Real-Time Operating System (RTOS) is required. Real-Time Operating Systems provide task scheduling and different options for synchronizing them. They also include functionality for sharing data between tasks using queues. FreeRTOS is chosen to be the RTOS for this device, since its widely used. Furthermore, it is MIT-licensed and therefore royalty-free and has a large community. There also exists a simulator for FreeRTOS, which makes it possible to run it under Linux (Becker, 2019). This further contributes to hardware-independent development and makes it possible to run the actual application code with a PC. The simulator also removes the need of an OS abstraction layer. (Amazon Web Services, 2019)

Based on these decisions a draft for the software architecture can be designed. This architecture is presented in Figure 15.

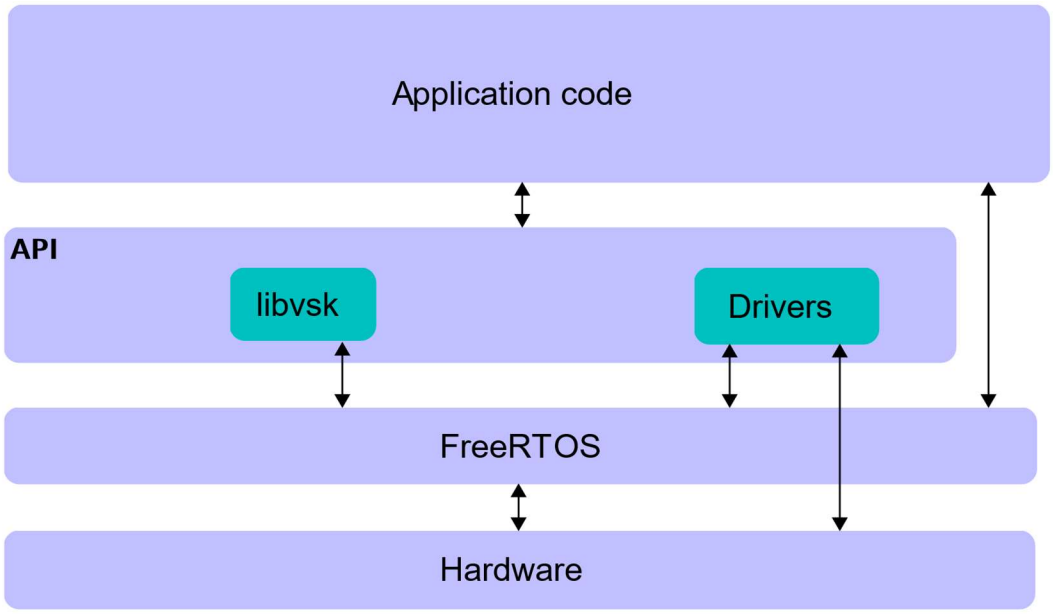

Figure 15. Software architecture. Blue blocks indicate different parts of the software. The arrows represent data flow and direction between these parts. Application code doesn't interact directly with the hardware.

The API design is concentrated on hardware abstraction. Common functionality among the bootloader, master module and extension modules are concentrated in a library named

"libvsk" in Figure 15. This library includes functionalities such as receiving and validating communication frames used in communication between the modules, or validation of firmware updates. As can be seen from the diagram, the application itself doesn't have hardware-specific functionality, but uses the underlying hardware through driver application programming interfaces (API).

The driver API works by utilizing virtual method tables. This means defining a structure with function pointers. These function pointers define the API for a specific driver. For example, the driver API for a flash device is displayed in Listing 1.

```
 1  struct vsk_flash_api
 2  {
 3          void (*vsk_flash_init)(struct vsk_flash *dev);
 4          int (*vsk_flash_write_protection)(struct vsk_flash *dev, vsk_flash_wp_t level);
 5          int (*vsk_flash_read)(struct vsk_flash *dev, uint32_t address, uint8_t *dst, int len);
 6          int (*vsk_flash_write)(struct vsk_flash *dev, uint32_t address, uint8_t *src, int len);
 7          int (*vsk_flash_erase_sector)(struct vsk_flash *dev, uint32_t sector);
 8          int (*vsk_flash_is_busy)(struct vsk_flash *dev);
 9          uint32_t (*vsk_flash_get_size)(struct vsk_flash *dev);
10  };
```

Listing 1. Example of a virtual method table. Structure defines the function pointers, their return value type and arguments.

The driver structure itself includes a pointer to an instance of this driver API structure. Using this pointer, it is possible to call the member functions specified in the instance of the function pointer structure. An example of the flash driver structure with the API pointer is presented in line 10 of Listing 2.

```
 1  struct vsk_flash
 2  {
 3          StaticSemaphore_t __flash_sem_mem[2];
 4          SemaphoreHandle_t mutex;
 5          SemaphoreHandle_t req_cplt_sem;
 6          uint32_t block_size;
 7          uint32_t block_count;
 8          uint32_t size;
 9
10          struct vsk_flash_api *api;
11          void *ctx;
12  };
```

Listing 2. Device structure for a flash device. A pointer is included to the virtual method table on line 10.

In the application the pointer is changed based on the device type used. For example, in this design there are two flash drivers sharing the same API functions. The other is for the internal flash memory of the microcontroller and the other one for the external SPI flash memory. For both device structures the pointer to the API structure point to a different instance: the other instance is defined for internal flash functions and the other for handling the external SPI flash. An example of this is assignment based on device type is shown in Listing 3.

```
1  switch (type) {
2  case VSK_FLASH_EXTERNAL:
3          dev->api = &stm32_qspi_flash_api;
4          break;
5  case VSK_FLASH_INTERNAL:
6          dev->api = &stm32_int_flash_api;
7          break;
8  }
```

Listing 3. Assignment of the virtual method table pointer based on the flash device type.

After the pointer has been assigned, it is possible to make function calls through the *"api"* pointer variable. The functions itself and the structure definition, which is either *"stm32_qspi_flash_api*" and *"stm32_int_flash_api"* in this example are defined in a separate C-file. These instances assign the function pointers to point to their corresponding member functions, which define the actual functionality. An instance definition for a virtual method table and a single member function for the example described here is displayed in Listing 4.

```
 1  static int stm32_qspi_flash_busy(struct vsk_flash *dev)
 2  {
 3          ASSERT(dev != NULL);
 4          ASSERT(dev->ctx != NULL);
 5
 6          uint8_t status_reg;
 7          struct vsk_qspi *qspi_dev = (struct vsk_qspi *)dev->ctx;
 8          status_reg = __stm32_qspi_flash_read_status(qspi_dev);
 9          return ((status_reg & 0x1) != 0) ? 1 : 0;
10  }
11
12  struct vsk_flash_api stm32_qspi_flash_api = {
13          .vsk_flash_init = &stm32_qspi_flash_init,
14          .vsk_flash_write_protection = &stm32_qspi_write_protection,
15          .vsk_flash_read = &stm32_qspi_flash_read,
16          .vsk_flash_write = &stm32_qspi_flash_write,
17          .vsk_flash_get_size = &stm32_qspi_flash_get_size,
18          .vsk_flash_erase_sector = &stm32_qspi_flash_erase_sector,
19          .vsk_flash_is_busy = &stm32_qspi_flash_busy
20  };
```

Listing 4. An example definition for a virtual method table. Member functions are declared static, since they do not have to be visible from outside the source file. This structure definitions fulfills the interface defined in listing 1.

Using this method presents some overhead to the application code but the extra benefit from this method is that multiple different device types can be used with the same API. Similarly, if for example a flash driver would not require control of the write protection, the *"vsk_flash_write_protection"* function can be set to have a NULL value in the virtual method table instance. Then from the main application, the pointer value can be checked and run, if it's not set to a NULL value. The method described in this chapter is used to achieve virtual functions and polymorphism in C, which is not an object-oriented language. (Reese, 2013)

5.1 Bootloader

In modern internet connected devices it is common practice to develop firmware in a way that it can be updated over-the-air. The possibility of updates can be utilized to bring new features to the device and more importantly to fix bugs or security issues. A bootloader serves as a reliability factor. If the firmware update fails for some reason, for example to due to power loss, the bootloader still stays intact and is be able to start the upgrade again using the update data previously stored to the external flash. Bootloaders are commonly used to

validate the integrity of the installed application. They are also used to ensure firmware signatures. Digital signing of firmware removes the possibility of tampering with the application code or running unauthorized firmware on the device. (IETF Trust, 2019)

Firmware updates are downloaded from the main application to the external flash and then written to the main application area in a separate bootloader. This is done in order to prevent the microcontroller from executing application code from the flash memory while it is being overwritten.

The bootloader is a separate program from the main application and is stored to the beginning of the flash area where the microcontroller starts its execution. In practice this means that the bootloader is always executed first when the MCU is reset. In this application, the bootloader checks if a software update is pending and whether a valid firmware is present. The flowchart for the bootloader designed is shown in Figure 16.

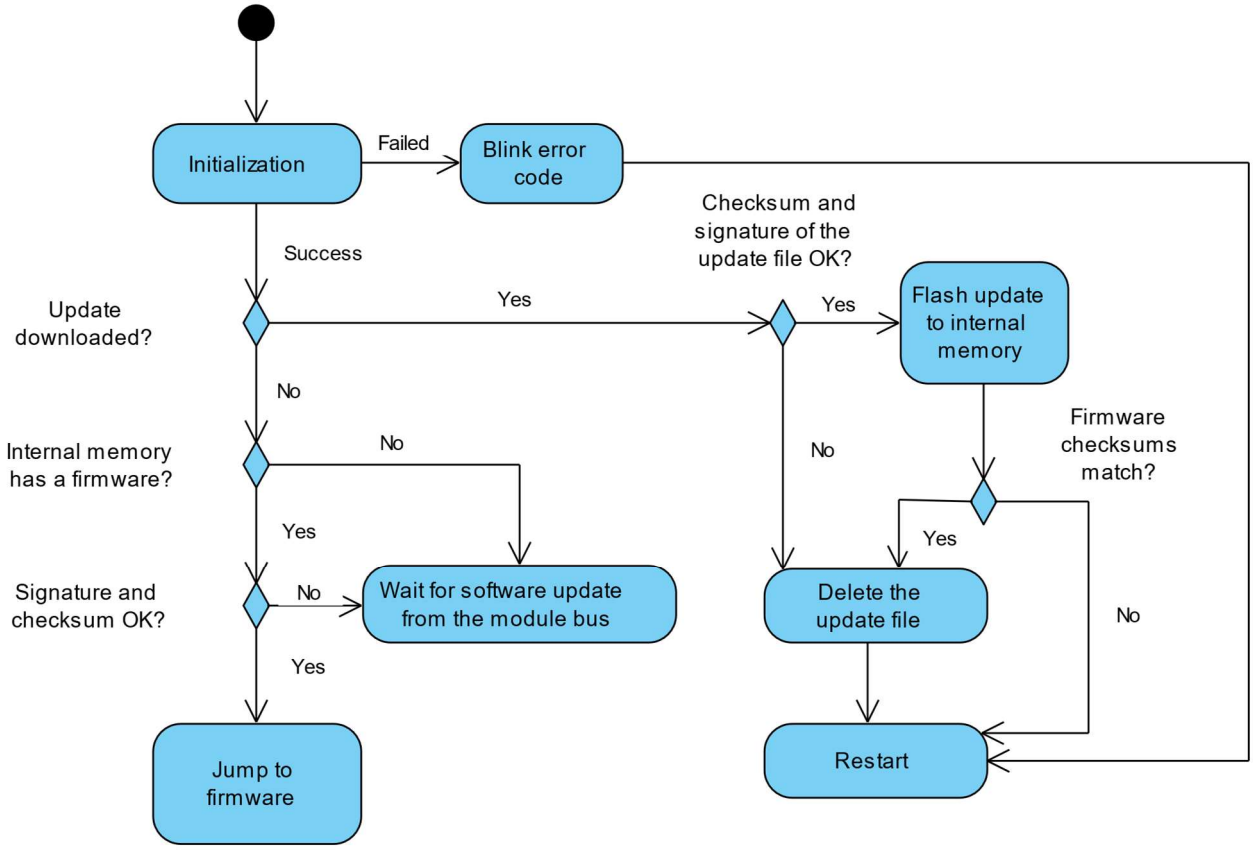

Figure 16. Flowchart for the bootloader. The bootloader determines whether a valid firmware is present or if a software update is requested and acts accordingly.

The bootloader initializes the Universal Synchronous/Asynchronous Receiver-Transmitter (USART) peripheral used for the module bus, the Serial Peripheral Interface (SPI) peripheral used for the external flash memory as well as the General-purpose input/output (GPIO) pins required to drive the status Light Emitting Diodes (LED). Then it proceeds to read the size of the flash chip. If communication with the chip fails an error is presented using a LED blink code. After this the device waits for 10 seconds and restarts.

For the bootloader to determine if a valid firmware is present, the firmware always starts with a fixed-size header. The header is added to the firmware after compilation and it consists of a magic value, timestamp of the compilation (software version), length of the

firmware, a 256-bit Secure Hash Algorithm (SHA) based hash and the digital signature. The header data is presented in Table 5.1.

Table 5.1. Firmware header fields and their sizes.

Field	**Size**
Magic value 0x47454656534B	8 bytes
Compilation timestamp (UNIX timestamp)	8 bytes
Firmware length	4 bytes
SHA-256-checksum	32 bytes
Digital signature	128 bytes

The magic value is used to determine the type of the file. This can be used to identify whether the firmware file is meant to be used in this device. The main module for example can use a different magic value than an extension module. Timestamp is used for software versioning. This version number is sent to the server during device communication and the server can decide whether a firmware update is required. This can also be used to determine whether the device has successfully updated its firmware. Firmware length is used during the update and validation processes. The length provides the information on how many bytes of data must be written in the internal flash and how many bytes must be read in order to form the SHA-256 hash of the firmware. It also prevents the flashing of too big firmware files for the device. The digital signature is used to validate the origin of the firmware. If the digital signature does not match the public key of GEF, the firmware is not run. The digital signature is generated using Elliptic Curve Digital Signature Algorithm (ECDSA).

After a successful peripheral initialization, the device checks whether there is an update file on the external flash filesystem. If an update exists, the hash is calculated and compared to the one provided in the firmware header. If the hashes match the firmware signature is checked. If it matches the preinstalled public key the firmware is flashed to the internal memory of the MCU. If the write is successful, the update file is deleted, and the system is restarted.

In case an update file does not exist, same checks are done for the firmware in the internal memory. If the firmware passes validation the bootloader jumps into the firmware. Jumping on ARM Cortex-M4 is done by setting up the stack pointer to the memory address where the firmware starts. After setting the stack pointer the interrupt vector table must be relocated. This is done so the correct interrupt handlers will be used after the jump to the main application. If the table isn't relocated, the main program would use the interrupt handlers of the bootloader which is not desirable. Interrupt vector table always starts at the 4th byte of the program and its location can be set using the *VTOR* register. After these values have been set, the MCU jumps to the main program by setting it's program counter accordingly. (IAR Systems, 2016)

Because the update file is deleted only after a successful firmware update the process is resilient to unexpected behaviors during flashing. If for example power to the device is cut during flashing, the firmware update still exists on the external flash memory and the write process is reinitiated. If the checksums don't match after writing the new firmware the device is restarted without deleting the new firmware file. This forces the device to reattempt flashing because the update file still exists.

In case the bootloader detects that there is no valid firmware present it goes into a software update mode. This allows new firmware to be transferred via the RS-485 bus used for module communication. During normal operation this situation should never occur. Regardless a utility tool is also being developed for updating the device via the RS-485 bus.

In order to increase security, the device was designed to be only able to run firmware provided by GEF. For this reason, a signature check is added. A SHA-256 checksum of the firmware files is calculated and the signed using an ECDSA private key. The bootloader calculates the SHA-256 hash for the firmware and checks if the firmware signature matches the programmed public key. The SHA-256 hash also ensures that the firmware file did not suffer from corruption during transfer.

If an attacker modifies the firmware and updates the SHA-256 hash to match the modifications, it would cause the ECDSA signature not match the hash signed with the private key. A matching ECDSA signature cannot be practically produced without knowing the private key used to sign the firmware. (Maxim Integrated Products Inc., 2017)

The production version of the bootloader disables the Joint Test Action Group (JTAG) and Serial Wire Debug (SWD) debug ports on the MCU during the boot process. Debug ports enable an attacker to alter memory on the device. This means an attacker could send arbitrary data to the server, input arbitrary data on the device or dump the flash memory contents.

For additional security, the bootloader should be immutable. This means any attempts to modify the bootloader (and hence attempts to tamper with the firmware validation) are mitigated. (IETF Trust, 2019)

5.2 Main application

The main program is started by the bootloader. It consists of multiple different RTOS tasks which provide the core functionality of the software. These tasks are interconnected with the use of inter-process communication methods and share data with each other. The tasks along with the data transferred between them are shown in Figure 17.

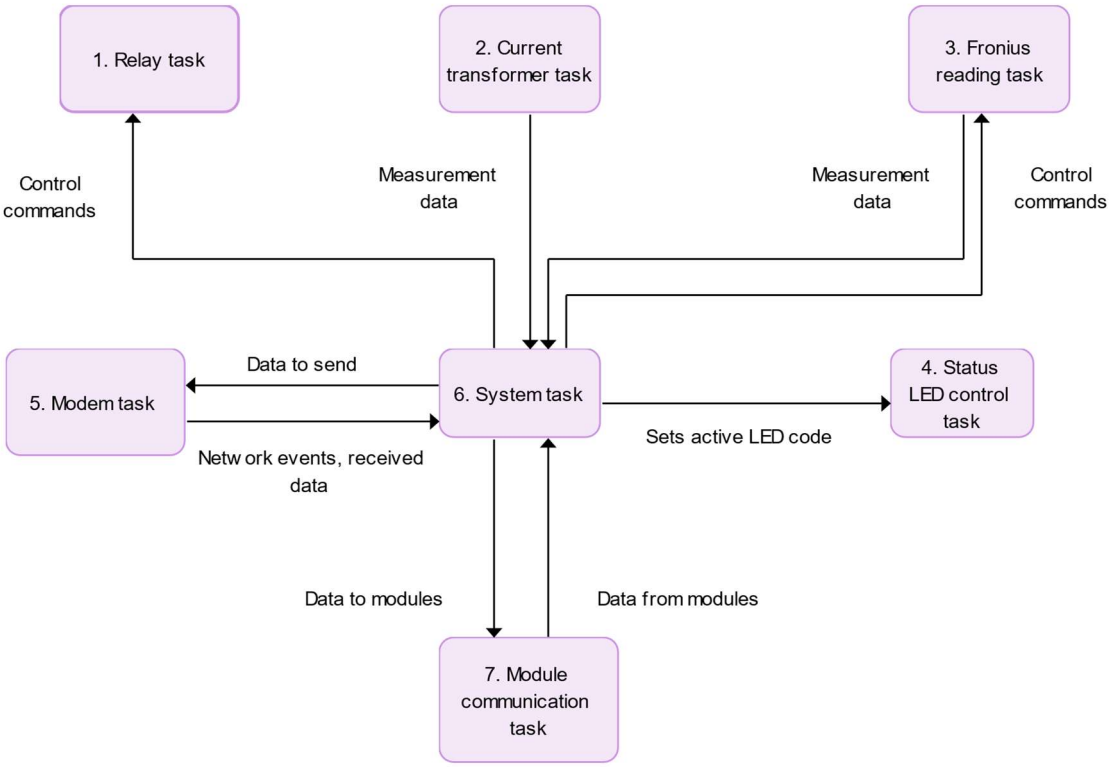

Figure 17. Software block diagram. The numbered blocks represent RTOS tasks. The arrows and their descriptions represent data flow and its direction between the tasks.

The numbered RTOS tasks in Figure 17 offer the following functionality:

1. Handling of the relay output state. If a watchdog is enabled, and no control command is received for the configured period, it sets the output to a predetermined state.
2. Collects periodic measurements from the current transformer using the analog-to-digital converter (ADC) of the MCU. The value is queued for sending to the server via the system task.
3. Reads measurements values from the Fronius inverter, which include for example AC and DC voltages, active power, energy meter reading and inverter status. Measurements are queued for sending to the cloud via the system task.
4. Handles LED blink codes. System task controls which LED blink code is active, or whether a LED should be on solid.
5. Takes care of modem related functionality.
6. Controls the overall system state and handles the data flow from the modem to the peripherals and vice versa. Also handles periodic update checking from the server as well as writing firmware updates to the external flash.
7. Sends data from and to the extension modules connected to the system task. Sends data received from the modules to the system task.

5.2.1 Modem control

The NB-IoT modem is an essential part of the system. The device is planned in a way that it can be restarted if it is connected to the cloud service. A state machine-based logic is utilized in the modem driver design. This state machine is presented in Figure 18.

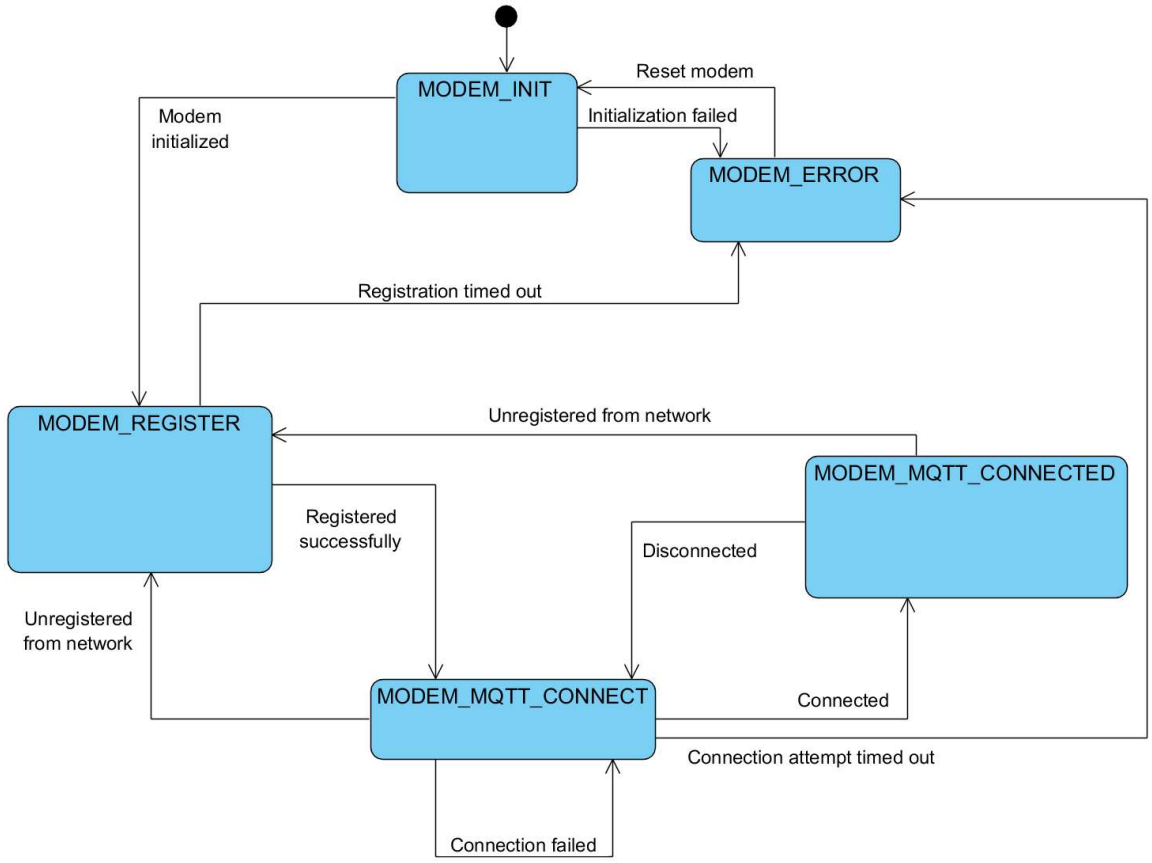

Figure 18. State machine diagram for modem driver. State names correlate with the names in the program code. The arrows describe possible state changes and their conditions.

Modem control is based on a protocol definition called Hayes command set, which are generally called AT-commands. AT-commands are American Standard Code for Information Interchange (ASCII) text-based commands initially developed by Dennis Hayes in 1981. The commands are based on text lines, which end in a control character, which can be the

carriage return character (0x13) and/or the newline character (0x10). (European Telecommunications Standards Institute, 1996)

There are four different types of AT-commands: test, read, set and execute. The test commands are used to check if the modem supports a certain function. Read commands are used to read configuration parameters or for example the state of a connection. The set commands are used to configure settings and parameters. Execute commands are used to run certain functionalities, for example to send a text message or to initiate a connection to a remote server. In addition to these command-based requests the modem can also report changes or give other messages asynchronously without a request. These messages are called URCs (Unsolicited Response Code). An URC is sent for example when the modem registration status changes. Basic AT command set in GSM modems is standardized by the 3GPP TS 27.007 standard. (European Telecommunications Standards Institute, 1996)

In addition to this standard command set the modem manufacturers can implement their own custom commands, and they often do. These commands are used for example to open a socket connection to a remote server. Some AT commands used in the Quectel BC66 are displayed in Table 5.2. (Quectel Wireless Solutions Co. Ltd., 2018)

Table 5.2. Examples of BC66 AT-commands (Quectel Wireless Solutions Co. Ltd., 2018)

Command	**Type**	**Example response**	**Description**
AT+CPIN? <CR>	Read	<CR><LF> +CPIN: READY <CR><LF> <CR><LF> OK <CR><LF>	Queries the SIM card pin code state. READY is responded if additional unlocking is not required.
AT+IPR=987987<CR>	Set	<CR><LF> ERROR <CR><LF>	Attempts to set the device USART baudrate to 987987, which is an invalid value. Therefore, an error is returned.
AT+CCLK=? <CR>	Test	<CR><LF> OK <CR><LF>	Tests if the AT+CCLK command is supported. If not, this command would respond with ERROR.
ATE0 / ATE1 <CR>	Execute	<CR><LF> OK <CR><LF>	If ATE0, data written to modem is not echoed back. If ATE1, data will be echoed back.

The “<CR>” in Table 5.2. represents the carriage return character (ASCII byte 0x13) and the “<LF>” represents the newline character (ASCII byte 0x10).

The BC66 expects the AT commands to end with a carriage return character. Depending on the command the response differs. The response always includes a final response, which can be “OK”, “ERROR” or start with “+CME ERROR” or “+CMS ERROR”. The latter two always include a numerical error code as well.

The modem task notifies the system task in case of any events. These events are created if the MQTT connection state changes (disconnection or established connection) or incoming data is received via MQTT. The system task is responsible for handling the received data or reacting to connection events. Unknown events are discarded.

5.2.2 Communication between modules

Two protocols were considered for the communication between the modules: Modbus-RTU and a custom protocol. Modbus is an industry-standard protocol defined in the 1970s and it's based on 16-bit registers. Modbus protocol was determined to be not flexible enough for this application. The communication between modules require a lot of arbitrary commands which are difficult to implement with a register-based communication, such as firmware updates for the modules. For these reasons a custom frame format is selected because it gives more control and flexibility for the communication. The designed frame format is presented in Table 5.3.

Table 5.3. Structure of communication frames

Value	**C data type**	**Size (bytes)**
Start of frame (0x7C)	uint16_t	1
Sender address	uint16_t	2
Receiver address	uint16_t	2
Packet number	uint16_t	2
Length of payload	uint16_t	2
Payload	Array of uint8_t	0-512
Checksum (CRC16)	uint16_t	2

This framing makes it possible to address frames from certain sender to a certain receiver. It also includes a checksum so the receiver can confirm that the received data was valid. The checksum is calculated from all the bytes in the frame except the start byte and the two bytes for the checksum itself. The receiver uses the receiver address field to determine if it should process the frame or not. The frames use the packet number field for transmissions that are over 512 byte long. In the first frame the first two payload bytes indicate the total number of

frames to be received. If this value is zero, the transmission consists of only one frame. This packet number makes it possible to determine when all the data has been received. If the packet number or the checksum don't match, the receiver doesn't send an acknowledge to the sender. This makes the sender to reattempt the transmission until it's successful or a timeout occurs.

The downside for the framing is that it brings an additional overhead to every transmission. Using this format, the overhead is 11 bytes. If the bus works at a baud rate of 19200 bps, this transfer of 11 additional bytes makes the bus reserved for approximately 4.6 additional milliseconds. This is not considered a problem for the module communication. Receiving of the frames is based on an internal state machine on the receiving side. State machine diagram for reception is presented in Figure 19.

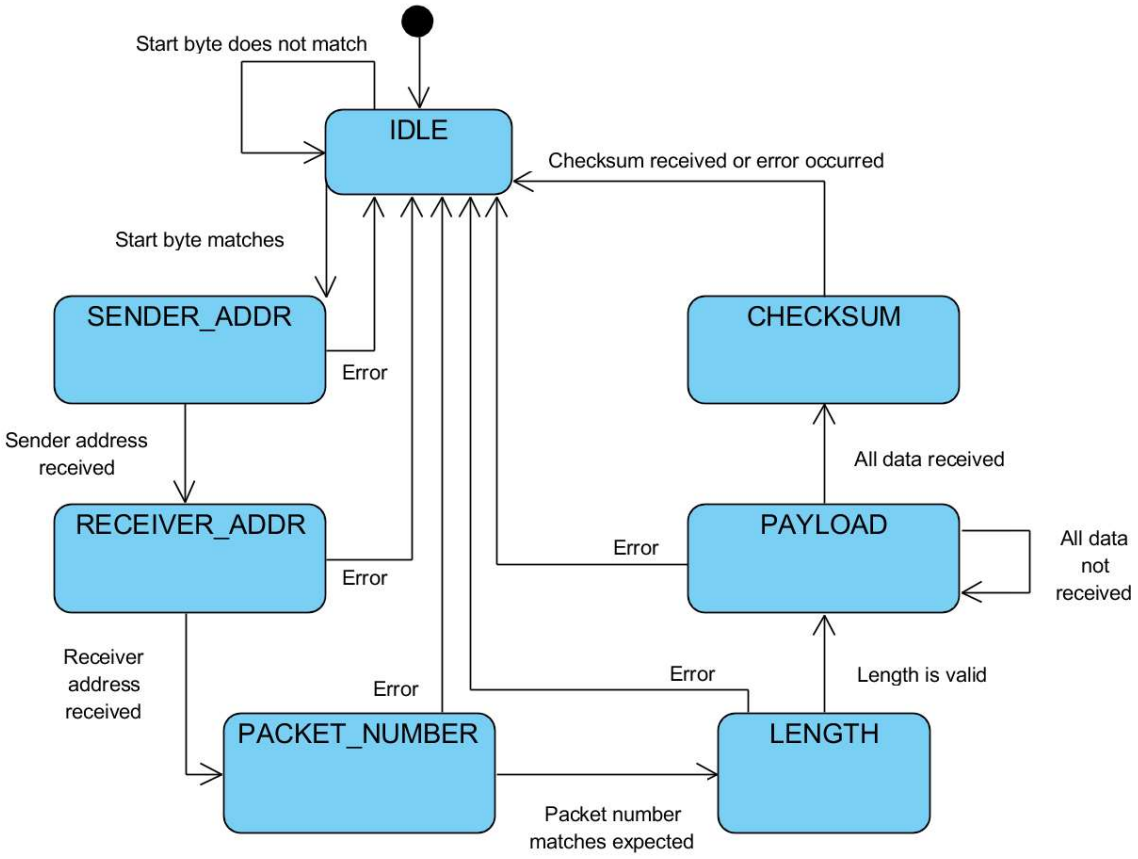

Figure 19. State machine diagram for frame reception. The state names are from the program code. The arrows and their descriptions show possible state changes and their conditions. Every state can lead to error due to for example a timeout.

All of the received bytes are processed in the state machine and it changes its internal state accordingly. When all the data is received, the receiver address is checked. If the configured address of the receiving device matches the address, the checksum is calculated. If the checksum matches, an acknowledge response is sent and the received data is passed to the system task for processing. An additional response can be sent after that, if required.

If an error occurs during reception the state machine is reset. These errors are invalid data length, invalid packet number or a timeout. Data bytes for a single frame should arrive within 100 milliseconds between each other. The timeout was determined in a way no processing

on the modules should block the transmission of data for more than this period. If a timeout condition is triggered, the device resets its state machine and waits for a new start byte.

5.2.3 Communication with the cloud server

The existing infrastructure that GEF uses is based on MQTT-protocol. MQTT has turned out to be very reliable and well-suited for this kind of use case. As an option, Constrained Application Protocol (CoAP) was considered. CoAP didn't offer enough benefits when the amount of work that would be needed to implement it into the existing infrastructure was considered. Unlike MQTT which is based on the Transmission Control Protocol (TCP), CoAP uses the User Datagram Protocol (UDP). TCP is based on connections: a connection is always negotiated with the opposing side. UDP however doesn't use the concept of connections. When using the UDP protocol, the sender sends data and receives no confirmation on whether the packet arrived or not unless the application developer implements such feature. TCP connections are more reliable as they use flow control, but on the other side they introduce more overhead. (The Internet Engineering Task Force, 1980; The Internet Engineering Task Force, 1981; The Internet Engineering Task Force, 2014)

In addition to CoAP also Message Queuing Telemetry Transport for Sensor Networks (MQTT-SN) protocol is considered. MQTT-SN acts like MQTT but it uses the UDP protocol instead. MQTT-SN replaces string topics with a 16-bit numerical identifier. Because the topic is sent during every publish in MQTT, it reduces the overhead caused by string topics in cases they are longer than two characters. In practice, MQTT-SN is fairly poorly supported in different MQTT brokers and client libraries and not very widely available. MQTT-SN is not supported in the Quectel BC66 modem that is used in this device (Quectel Wireless Solutions Co., Ltd., 2019; Stanford-Clark & Truong, 2013).

The communication module sends its data periodically to the cloud service. This period is configurable. The data consists of all the measurement data from different modules installed into the system. The MQTT client also listens for a device-specific topic that can be used for transmitting control commands to the device. Control commands can for example be an

instruction to turn on the relay that drivers the water heater. Because the AT-protocol used with the modem is based on the ASCII character set, the binary data sent to the server has to be encoded to ASCII. This process is called binary-to-text-encoding. This type of encoding always causes some overhead to the data, increasing the amount of data transmitted. As a compromise of reliability and amount of overhead the ASCII85 method was chosen and specifically its Z85 variant. This method makes it possible to present four binary bytes as five ASCII-bytes. The difference between Z85 and the original ASCII85 is that Z85 avoids using the backwards slash character and quotes, which are commonly used as escape characters. Z85 requires that the length of the data is divisible by four. Padding must be used in order to achieve this, if the data doesn't fulfill this requirement. (iMatix Corporation, 2013)

6. CONCLUSIONS AND SUMMARY

This thesis introduces a concept for a next-generation solar monitoring device for GreenEnergy Finland Oy. The main difference of this product in comparison to the previous ones was the use of an IoT-network for exchanging the monitoring data and control commands with the GEF Vision cloud platform. This is done to get rid of problematic end user internet connections.

Three IoT-networks compared for this task, with NB-IoT proving to be the most fitting for this use case. It provides a low enough latency for control commands, and a data rate large enough to meet the requirements, even for a system with larger number of modules installed. Coverage maps show that other IoT networks in Finland are comparable in coverage and will probably continue to improve in the future. By dropping the service level requirements for this device, they could be better choices than NB-IoT.

The STM32L4 series MCU chosen for the device has enough processing power for more advanced tasks in the future. The power saving features for the MCU make it possible to develop the software in a way that battery powered application could be possible as well. The BC66 modem from Quectel had all the needed protocol requirements for the device in a very small footprint.

The modular design of the software in addition to the device itself makes the task of creating extension modules fairly easy while reducing development time drastically. Unit testing in general is a way to improve software quality by reducing bugs.

The current communication with the cloud server uses the TCP based MQTT-protocol. Because TCP connections always cause an overhead to the connection due to handshaking and flow control, in order to reduce the amount of data transfers an UDP-based protocol might be used in the future. The need for this will be clarified in the future, when the final prices for the NB-IoT subscriptions are received from the network operators. This of course depends on how much the data amount affects the subscription price.

Security is and will be an issue with IoT devices, and it requires careful planning. STM32 as an MCU platform offers great security features in addition to those that are presented in this thesis. In the future a more in-depth view into them is possible, if deemed necessary. The most obvious security risk in this design would the RS-485 bus. An attacker could connect to this bus and act as an extension module or send control commands to modules. Like exploiting the debugger this attack would require physical access to the device. This was not seen as a huge risk for solar monitoring. For load control the risk would be that an attacker repeatedly toggles a switch relay very fast and breaks it. However, if the attacker already has physical access to the installation site, it would be easier to physically break or disable the relay or control it manually rather than reverse engineering and using the device bus.

If seen necessary, the security can be increased by adding some sort of authentication protocol to the RS-485 bus. This can be distributed in a form of software updates to the master module and the possible extension modules. This requires development of some method in which the modules and the master module share encryption keys with each other and encrypt their communication using these keys.

Two possible housing solutions were compared and the BC-series from Phoenix Contact was selected for the device implementation. The housing offers a ready solution to interconnect the master module with the extension modules both mechanically and electrically while retaining the ability to be installed in a standard DIN-rail. Ideas for the first extension modules are already made, and implementation will start as soon as the master module is ready and proved to be stable.

REFERENCES

Adelantado, F. et al., 2017. *Understanding the Limits of LoRaWAN.* Piscataway, IEEE Press.

Adhikary, A., Lin, X. & Wang, Y.-P. E., 2016. *Performance Evaluation of NB-IoT Coverage.* Montreal, Institute of Electrical and Electronics Engineers.

Amazon Web Services, 2019. *FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions.* [Online] Available at: https://www.freertos.org [Accessed 11 July 2019].

Becker, M., 2019. *Additions to FreeRTOS.* [Online] Available at: https://github.com/michaelbecker/freertos-addons [Accessed 11 July 2019].

Connected Finland Oy, 2019. *Suomen IoT-operaattori - Connected Finland.* [Online] Available at: https://www.connectedfinland.fi [Accessed 25 July 2019].

Digita Oy, 2019. *Digitan IoT LoRaWAN-verkon peittokartta.* [Online] Available at: https://www.digita.fi/yrityksille/iot/iot_lorawan-verkon_peittokartta [Accessed 22 July 2019].

European Telecommunications Standards Institute, 1996. *Digital cellular telecommunications system (Phase 2+); AT command set for GSM Mobile Equipment (ME),* Valbonne: ETSI.

Fingrid Oyj, 2018. *Ehdot ja edellytykset taajuuden vakautusreservin (FCR) toimittajalle.* [Online] Available at: https://www.fingrid.fi/globalassets/dokumentit/fi/sahkomarkkinat/reservit/ehdot-ja-edellytykset-taajuuden-vakautusreservin-fcr-toimittajalle.pdf [Accessed 28 July 2019].

Fronius International GmbH, 2019. *Fronius Interface,* Wels: Fronius International GmbH.

IAR Systems, 2016. *Technical Note 160822 - Creating a bootloader for Cortex-M.* [Online] Available at: https://www.iar.com/support/tech-notes/general/creating-a-bootloader-for-cortex-m [Accessed 11 July 2019].

IETF Trust, 2019. *A Firmware Update Architecture for Internet of Things Devices.* [Online] Available at: https://tools.ietf.org/id/draft-ietf-suit-architecture-02.html [Accessed 24 July 2019].

iMatix Corporation, 2013. *32/Z85 - ZeroMQ RFC,* s.l.: s.n.

ITALTRONIC Srl, 2019. *Modulbox One.* [Online] Available at: https://eng.italtronic.com/products/modulbox_one_en/ [Accessed 11 July 2019].

Kais, M., Eddy, B., Frederic, C. & Fernand, M., 2018. *Overview of Cellular LPWAN Technologies for IoT Deployment: Sigfox, LoRaWAN, and NB-IoT.* Puttelange-aux-Lacs, Elsevier B.V..

LoRa Alliance, Inc, 2017. *LoRaWAN 1.1 Specification.* [Online] Available at: https://lora-alliance.org/sites/default/files/2018-04/lorawantm_specification_-v1.1.pdf [Accessed 28 July 2019].

Mangalvedhe, N., Ratasuk, R. & Ghosh, A., 2016. *NB-IoT Deployment Study for Low Power Wide Area Cellular IoT.* Piscataway, Institute of Electrical and Electronics Engineers.

Mareti Media Oy, 2018. *Elisa toi esineiden internetin NB-IOT-teknologian mobiiliverkkoonsa Suomen jokaisessa kunnassa.* [Online] Available at: https://mobiili.fi/2018/04/25/elisa-toi-esineiden-internetin-nb-iot-teknologian-mobiiliverkkoonsa-suomen-jokaisessa-kunnassa/ [Accessed 22 July 2019].

Maxim Integrated Products Inc., 2017. *Application Note 6426 - The Fundamentals of Secure Boot and Secure Download: How to Protect Firmware and Data within Embedded Devices.* [Online] Available at: https://www.maximintegrated.com/en/app-notes/index.mvp/id/6426 [Accessed 24 July 2019].

Nordic Semiconductor, 2019. *nRF9160 System-In-Package.* [Online] Available at: https://www.nordicsemi.com/Products/Low-power-cellular-IoT/nRF9160 [Accessed 29 July 2019].

Phoenix Contact Oy, 2019. *Modulaariset kotelot BC.* [Online] Available at: https://www.phoenixcontact.com/online/portal/fi?1dmy&urile=wcm:path:/fifi/web/main/products/subcategory_pages/Modular_housings_BC_P-01-12-02/3a371abc-34e5-438f-a723-d2ab8e20eaa1 [Accessed 11 July 2019].

Quectel Wireless Solutions Co. Ltd., 2018. *BC66 AT Commands Manual,* Shanghai: Quectel Wireless Solutions Co. Ltd..

Quectel Wireless Solutions Co., Ltd., 2019. *Quectel BC66 - Compact LTE Cat NB1 Module with Ultra-low Power Consumption.* [Online] Available at: https://www.quectel.com/UploadFile/Product/Quectel_BC66_LPWA_Specification_V1.5.pdf [Accessed 21 July 2019].

Quectel Wireless Solutions Co., Ltd, 2019. *Quectel BG96 LTE Cat M1 & Cat NB1 & EGPRS Module.* [Online] Available at: https://www.quectel.com/UploadFile/Product/Quectel_BG96_LPWA_Specification_V1.7.pdf [Accessed 24 July 2019].

Reese, R., 2013. *Understanding and Using C Pointers.* 1st ed. Sebastopol: O'Reilly Media Inc.

Rohde & Schwarz GmbH & Co. KG, 2016. *Narrowband Internet of Things Whitepaper.* [Online] Available at: https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/dl_application/application_notes/1ma266/1MA266_0e_NB_IoT.pdf [Accessed 21 July 2019].

Sigfox SA, 2017. *Make things come alive in a secure way.* [Online] Available at: https://www.sigfox.com/sites/default/files/1701-SIGFOX-White_Paper_Security.pdf [Accessed 25 July 2019].

Sigfox, 2019. *Coverage | Sigfox.* [Online] Available at: https://www.sigfox.com/en/coverage [Accessed 22 July 2019].

Silva, J. d. C. et al., 2017. *LoRaWAN — A low power WAN protocol for Internet of Things: A review and opportunities.* Split, IEEE.

SIMCom Wireless Solutions Co., Ltd, 2017. *SIM7000E Product Description.* [Online] Available at: https://simcom.ee/documents/SIM7000E/SIM7000E_SPEC_2017-9-21.pdf [Accessed 24 July 2019].

Sinha, R. S., Wei, Y. & Hwang, S.-H., 2017. *A survey on LPWA technology: LoRa and NB-IoT.* Seoul, Elsevier B.V..

ST Microelectronics, 2018. *STM32L4x5 and STM32L4x6 advanced ARM-based 32-bit MCUs.* [Online] Available at: https://www.st.com/content/ccc/resource/technical/document/reference_manual/02/35/09/0c/4f/f7/40/03/DM00083560.pdf/files/DM00083560.pdf/jcr:content/translations/en.DM00083560.pdf [Accessed 11 July 2019].

Stanford-Clark, A. & Truong, H. L., 2013. *MQTT For Sensor Networks Protocol Specification.* [Online] Available at: http://www.mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf [Accessed 11 July 2019].

Sultania, A. K., Zand, P., Blondia, C. & Famaey, J., 2018. *Energy Modeling and Evaluation of NB-IoT with PSM and eDRX.* Abu Dhabi, IEEE.

Telia Finland Oyj, 2019. *Telian verkkokartta.* [Online] Available at: https://www.telia.fi/asiakastuki/verkko/verkko/verkkokartta [Accessed 22 July 2019].

Texas Instruments, 2016. *The RS-485 Design Guide (Rev. C).* [Online] Available at: http://www.ti.com/lit/an/slla272c/slla272c.pdf [Accessed 24 July 2019].

The Internet Engineering Task Force, 1980. *RFC 768 - User Datagram Protocol.* [Online] Available at: https://tools.ietf.org/html/rfc768 [Accessed 22 July 2019].

The Internet Engineering Task Force, 1981. *RFC 793 - Transmission Control Protocol.* [Online] Available at: https://tools.ietf.org/html/rfc793 [Accessed 22 July 2019].

The Internet Engineering Task Force, 2014. *RFC7252 - The Constrained Application Protocol (CoAP).* [Online] Available at: https://tools.ietf.org/html/rfc7252 [Accessed 24 July 2019].

ThrowTheSwitch.org, 2019. *CMock.* [Online] Available at: http://www.throwtheswitch.org/cmock [Accessed 24 July 2019].

ThrowTheSwitch.org, 2019. *Unity.* [Online] Available at: http://www.throwtheswitch.org/unity [Accessed 24 July 2019].

Wang, E. Y.-P. et al., 2017. *A Primer on 3GPP Narrowband Internet of Things.* New Jersey, IEEE Communications Magazine.