INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelor's Thesis

IoT meets HPC: Securely transferring wireless sensor data to a supercomputer utilizing the LRZ Cloud

Kevin Owen Edmonds

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelor's Thesis

IoT meets HPC: Securely transferring wireless sensor data to a supercomputer utilizing the LRZ Cloud

Kevin Owen Edmonds

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Dr. Nils gentschen Felde Tobias Guggemos Dr. Helmut Heller (LRZ)
Abgabetermin:	9. Oktober 2017

I assure the single handed composition of this Bachelor's Thesis only supported by declared resources.

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 9. Oktober 2017

. . . .

(Unterschrift des Kandidaten)

Abstract

Small smart devices communicating together via the Internet may achieve significantly more than single devices would be able to: this is the Internet of Things. There is a plenitude of devices and services available for customers. These devices are often very short lived and are only supported by the developer for a short period of time. The lack of upgrades and difficulty to patch the devices often lead to serious security issues. The protocols used in IoT are designed to be lightweight, as the devices usually have limited processing capacity. The heavy lifting is therefore done by the server the IoT devices send their data to. This Bachelor's thesis aims to design and create an end to end solution for sending data collected by wireless sensors to a supercomputer in a secure way, utilizing a central platform in the cloud. The customer requirements were gathered through interviews leading to user stories and raw requirements. Those are analyzed to determine functional and non functional requirements, which are used to develop a system design. Three implementations of the system design are created, with in the end two working prototypes. The prototypes are evaluated against the customer requirements, their power consumption measured, and the prototype satisfying all customer requirements is tested for reliability over a longer period of time. Finally, the results are discussed with one prototype proving to fulfill all customer requirements.

Contents

1. Introduction

The Internet of Things (IoT) is the vision where large number of physical things are connected to the Internet [Kop11]. One of the first definition of IoT defines ”Things” as simple Radio-Frequency Identification (RFID) tags [Lab12].

With miniaturization and cost-reduction these simple things have evolved into smart-objects such as wireless smart-sensors that collect environmental data and send it back to the user. For example wireless sensors positioned within a strawberry plot gathering humidity and temperature data from the soil and sending it to the cloud. The central platform could automatically activate a watering system in areas where the soil is too dry, allowing for optimal growing conditions and water savings. When the temperature goes towards freezing point, an alert could be sent to the farmer so that he may cover up the plants just in time to avoid the frost.

Those wireless sensors make up what is called a Wireless Sensor Network (WSN). These sensor nodes are designed to be small, use little energy, have limited processing capacity and are cheap to produce. This enables WSNs to scale up to thousands of sensor nodes with different network topologies such as star or mesh depending on their implemented communication protocols [YMG08].

These readily available and affordable smart-objects give rise to multiple usage scenarios of IoT technologies in science. An application of WSNs are Environmental Sensor Networks (ESN). ESN are made out of sensor nodes that gather environmental data autonomously and deliver their data to a base station. The base station then sends the bundled data on to a server, often in a cloud, where it is further processed and made accessible to users. These networks provide valuable data to scientists for further analysis [HM06]. Other domains such as health care can largely profit from the rise of smart-objects by tracking people and objects within a hospital or by sensing patients physiological data [AIM10].

The projections for the growth of the IoT market are very positive [Lue14], leading to many companies having a growing interest in the field. The IoT ecosystem consists of many players that focus on one or more areas of the business model. This results in a complex value chain *(see figure 1.1)* where multiple stakeholders need to work together to deliver a working product [Agr17].

The lack of an end to end solution from one provider sets the basis for this thesis. It has the aim to create a system design and implement a working prototype to bring data from wireless sensors into a cloud platform. The prototype is intended to be open source allowing for the implementation to be available for further projects and easily modifiable to suit other requirements.

1.1. Motivation

The Leibnitz Supercomputing Center of the Bavarian Academy of Sciences and Humanities (LRZ) has interest for research purposes in a solution to access and aggregate wireless sensor

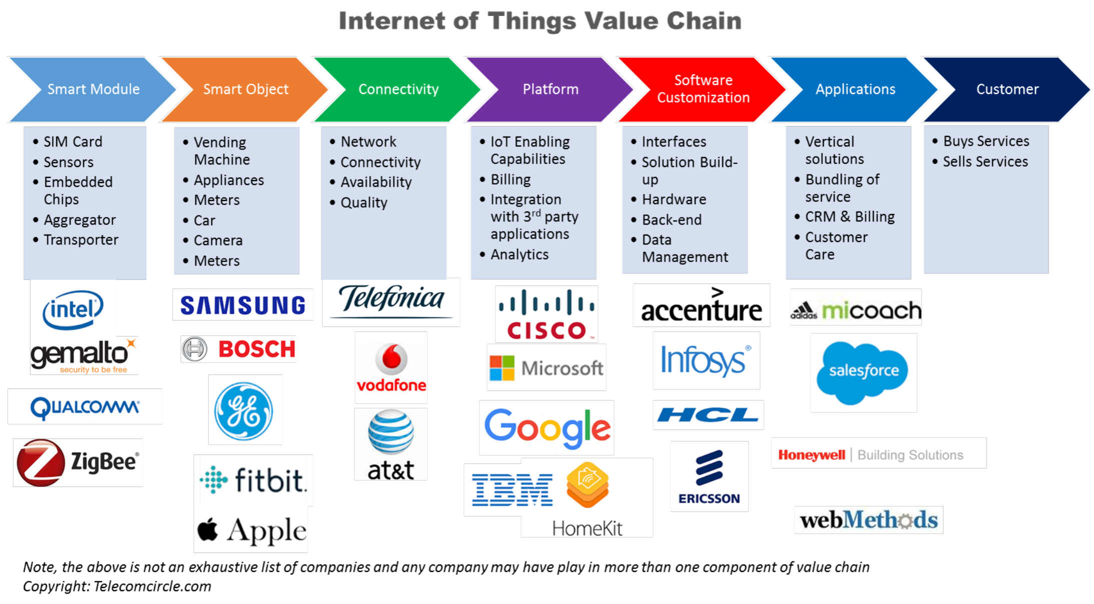


Figure 1.1.: IoT value chain, Source: [Agr17]

data into a central service running in the supercomputing center and allowing processing of the data by a supercomputer. A current project that could make use of this research requires environmental data gathered in the field. This project has sensors that are physically connected to a Raspberry Pi with the data stored locally on an SD card and powered by car batteries. Each Raspberry Pi has it's own Universal Mobile Telecommunications System (UMTS) modem to upload sensor data, but the devices often run out of power. Having data near real time allows the scientist to monitor the experimentation site for any problems or failures, and to decide if someone has to go on site to fix things. These locations can often be far away from where the scientist works. Therefore the reduction of maintenance trips is of great interest. In addition, the collected data needs to be stored in a reliable way to be compared or used as input for computer simulations.

There are solutions available for each aspect of the ESN such as Waspmotte for wireless sensors [Lib17b], Carriots as an IoT platform [Car17] and all-in one business solutions [Lib17a]. Where there is a research gap is an end to end solution built with readily available hardware and open-source software.

Wunderbar One of the first IoT kits was the Wunderbar IoT StarterKit conceived in October 2013 and released in October 2014 by the IoT company "relayr" [rel17c]. The Wunderbar is a completely wireless product with Bluetooth Low Energy (BLE) sensor modules and an Internet connection over WiFi. The device connects to a central platform on the relayr cloud where it uploads sensor data and provides a graphical user interface (GUI) for users to visualize the data and configure events. An interesting application developed with the Wunderbar is BabyBiCO 1.0, a smart monitor that tracks babies sleeping environment, and relays information from the cloud to a smart device [Hac17].

IoT Security Early 2017, a practical attack against the SHA-1 hashing algorithm was unveiled by CWI Amsterdam and Google [al.17]. The Wunderbar master module firmware had a hardcoded SSL certificate using that hashing algorithm, which lead to relayr dropping support for the Wunderbar. The hardware schematics and the firmware were open-sourced and made available on github [rel17a], but the forums and documentation website were taken offline [rel17c].

IoT has a bad security track record with hardcoded credentials for SSH/Telnet protocols and the inability to easily update firmware. With more and more "Things" exposed to the Internet there is the potential for large scale infection.

In August 2016, the Mirai malware was discovered by the white-hat security research workgroup MalwareMustDie!. This trojan backdoor exploited default hardcoded SSH or Telnet login credentials on Linux-based IoT devices. Once shell access is obtained, a malicious payload is downloaded and executed on the device [Mal16]. On the 20th September, the Mirai botnet was unleashed on the website of security researcher Brian Krebs, generating traffic on the webservers estimated at 620 Gbps in size. The Distributed Denial of Service attack was attributed to an IoT botnet that was comprised of hacked devices from all over the world such as routers, digital video recorders and IP cameras [Kre16].

With a lot of the Things from different manufacturers using the same libraries, one single bug may affect millions of devices. On the 18th June, the IoT security company Senrio revealed a vulnerability in the gSOAP library from Genivia allowing remote code execution on an Axis security camera [Sen17a]. Research was made by Senrio to estimate how many devices actually used the library and proposed that it was likely be run on tens of millions of products. After being informed about the bug, a patch for the bug was quickly provided by the library's developer and Axis released a new firmware version for their cameras with customers required to update their devices[Sen17b].

Problem Statement The issue of how to keep all of the deployed devices secure is raised. The above example of the Wunderbar shows how when a security issue with the hardcoded SSL certificate in the firmware was found, the developer did not provide an update for their product. Instead the company dropped support for the device by cutting its ability to connect to the relayr cloud, thus making the device non functional. With the current patching process being per device/manufacturer relying on manual firmware updates, it is very likely that during the lifecycle of the device, patches will not be available or be too late to prevent exploits [YSS+15].

Approach This lead to the idea of developing a system design and implementation of an end to end IoT solution with security and transparency in mind. The starting point of this thesis is reviving the Wunderbar by modifying its firmware. The main change would be that it transmits the module data to a different server with a new secure SSL certificate. Since the relayr cloud service code wasn't released, a separate central platform would need to be used for the data aggregation, user management and data visualization.

Outline of this Thesis Chapter 2 provides an introduction to the technologies used in IoT such as IoT middleware, networking, messaging protocols and how these can be secured. In Chapter 3 the customer concept is defined and functional/non-function requirements for the hardware and software of the prototype are derived to implement the concept. From those requirements a system design is proposed in Chapter 4. Chapter 5 describes the multiple prototype implementations that were explored to fulfill the customer requirements. An evaluation of the prototypes is performed in Chapter 6 and discussed in Chapter 7.

2. Background

This chapter aims to provide some background knowledge required for this thesis. The first section describes different technologies such as IoT devices and network protocols used in IoT. The second section focuses on services an IT service provider can provide for IoT.

2.1. Internet of Things Technologies

The IoT ecosystem is comprised of a large array of devices collecting and exchanging data. With the deployment of ever more "things", comes the need of a software platform, also named middleware, to provide an abstraction layer to applications and users from those "things". The platform provides an Application Programming Interface (API) for the physical layer communication and services that use the data received from the "things" [BSMD11].

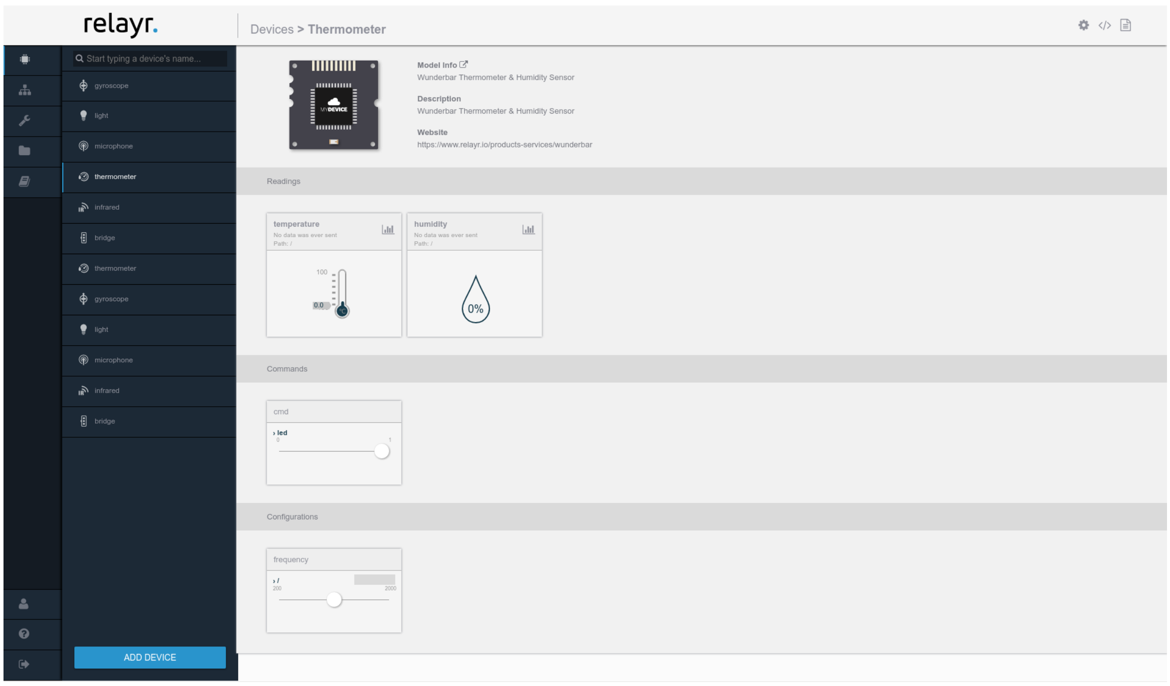
2.1.1. Wunderbar Internet of Things Starter Kit for App Developers

The Wunderbar Internet of Things Starter Kit for App Developers was conceived in October 2013 and released in October 2014 [rel17c]. The kit consists of six sensor modules that connected wirelessly over Bluetooth Low Energy (BLE) to a master module that is connected to a local WiFi. On the other side of the local WLAN was the relayr cloud service which collected the bundled sensor module data from the master module and displayed it in a GUI *(see figure 2.1)* [rel14]. The hardware device firmware was released on Github under an open source license, the cloud service was not.

2.1.2. IoT Network Protocols

The network protocols in IoT should be lightweight, minimize power consumption, network overhead and processing power requirements due to their use on constrained devices.

Bluetooth Low Energy (BLE) BLE is an enhancement to the Bluetooth wireless standard that focuses on minimizing power consumption and providing ad hoc networking capabilities [(SI09]. Controlling the connections and advertising in Bluetooth is the Generic Access Profile (GAP). Two main roles are defined in GAP; a central device which accepts and manages connections from peripheral devices, and peripheral devices that connect to a central device [Tow15]. Once a connection has been established, the peripheral devices expose a hierarchical data structure defined by the Generic Attribute Profile (GATT) to the central device *(see figure 2.2)*. This structure built on top of the Attribute Protocol (ATT) is comprised of a profile which describes the purpose of the BLE device, and one or more services which are collections of characteristics that define the functionality of the device. A characteristic is the lowest level of the GATT hierarchy consisting of a single data point and properties. Each service and characteristic have a Universally Unique IDentifier (UUID). GATT also



Model Info
Wunderbar Thermometer & Humidity Sensor

Description
Wunderbar Thermometer & Humidity Sensor

Website
https://www.relayr.io/products-services/wunderbar

temperature
No data was ever sent
Path: /

humidity
No data was ever sent
Path: /

Figure 2.1.: Relayr cloud platform

defines operations to read, write, notify and indicate characteristics to communicate in both directions [Gro17].

IoT Transport Layer Protocol In IoT the network bandwidth available to the devices can often be limited. It therefore makes sense to minimize the overhead produced by the chosen transport layer protocol. Transmission Control Protocol (TCP) is a widely used connection oriented protocol. It uses a handshake to establish the connection and sequence numbers to identify each byte of data. Acknowledgments are sent telling the sender up to which specified byte the data has been received so that if a transmission error occurs, the data can be resent [SC81]. This level of reliability adds overhead to the protocol.

User datagram protocol (UDP) is a connectionless protocol which does not set up a connection and just transmits information to the receiver. It does not make use of acknowledgments so if transmission errors occur, the bytes are not retransmitted. The advantage of the protocol is the reduced overhead, with the drawback of having no guarantee that the sent data has been received [Pos80].

Data sent by IoT devices often doesn't suffer too much from individual lost packets. With a video stream loosing a frame still being viewable, or a plotted sensor data graph missing one value probably still showing a clear picture of the environment, the trade off of using the UDP protocol instead of TCP can often be made.

2.1.3. IoT Messaging Transport Protocols

Above the Transport Layer the data is often encapsulated within a messaging transport protocol.

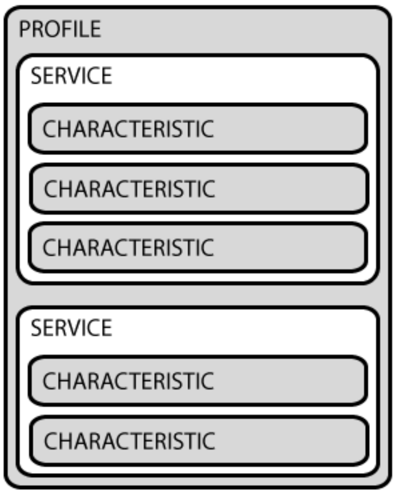


Figure 2.2.: GATT structure Overview, Source: [Tow17]

MQTT Protocol The Message Queue Telemetry Transport (MQTT) protocol is a lightweight Client Server publish/subscribe messaging transport protocol running over TCP/IP. A central server named the MQTT broker accepts incoming connections from MQTT clients, and manages the topics and subscription each client has made. When a message is published on the topic, the MQTT broker sends the message to all clients that have subscribed to the topic [Ope14]. In WSNs, running over TCP/IP can be a disadvantage due to the limited capabilities of the low-cost devices.

For this purpose, the UDP based MQTT-SN protocol based on the original MQTT specification was designed with the underlying idea of being optimized for low-cost wireless devices with limited processing power and storage capabilities. MQTT-SN clients connect to the MQTT broker via an MQTT-SN gateway which uses the MQTT protocol to communicate with the MQTT broker [SCT13] *(see figure 2.3)*.

CoAP Protocol Another messaging protocol used in IoT is the Constrained Application Protocol (CoAP). The protocol uses a Client Service request/response model designed to easily interface with HTTP by implementing a subset of Representational State Transfer (REST) with a very low overhead running over UDP [(IE14].

Securing the Messaging Transport Protocols Both MQTT-SN and CoAP messaging protocols do not have security out of the box, but rely on the encryption of the underlying transport layer. In the case of MQTT-SN and CoAP the Datagram Transport Layer Security (DTLS) protocol is used. DTLS provides communication privacy for datagram protocols and is based on the Transport Layer Security (TLS) protocol [For12].

2.2. IoT to the LRZ Super Computing Center

The Leibnitz Supercomputing Center of the Bavarian Academy of Sciences and Humanities (LRZ) was founded in 1962 in Munich and is the IT service provider for the Munich univer-

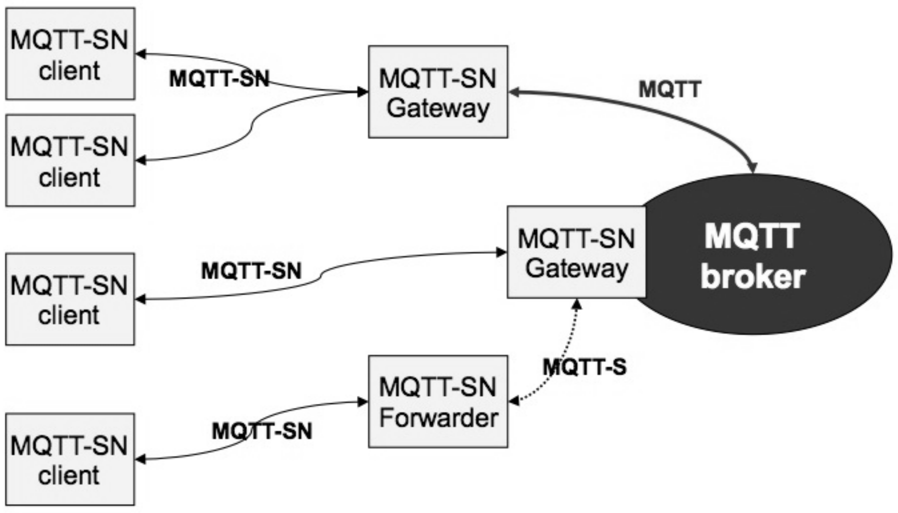


Figure 2.3.: MQTT-SN architecture, Source: [SCT13]

sities and colleges. Alongside providing communication infrastructures, e-learning platforms and a large range of services, it houses the high-end supercomputer "SuperMuc" [LRZ17b].

2.2.1. LRZ Compute Cloud

The LRZ compute cloud is a high performance computing (HPC) service allowing customers to upload and use their own virtual machine (VM) with root access. The Cloud service is based on the cloud middleware OpenNebula. The middleware allows users to manage and configure their VMs over a GUI *(see figure 2.4)* [LR16].

2.2.2. Big Data Storage

Data is often stored locally in science departments, raising the problem of how to make it available to HPC services. This is often done through a Wide Area Network (WAN) which can be hard to scale due to physical bandwidth limitations and the ever increasing amount of data needing to be transfered.

The LRZ provides the Data Science Storage (DSS) solution allowing to consolidate Big Data and the LRZ compute resources under one roof. This service allows the use of a single filesystem that can be mounted across many LRZ computer systems [LRZ17a].

2.3. Summary

This chapter introduced the reader to the IoT concept with the example of the Wunderbar Starter Kit. The network protocol BLE was explained along with the IoT transport layer protocol of choice UDP. The messaging transport protocols CoAP and MQTT were described along with a protocol used to secure the datagram protocol, DTLS. The LRZ was introduced along with some of the services it provides to their customers.

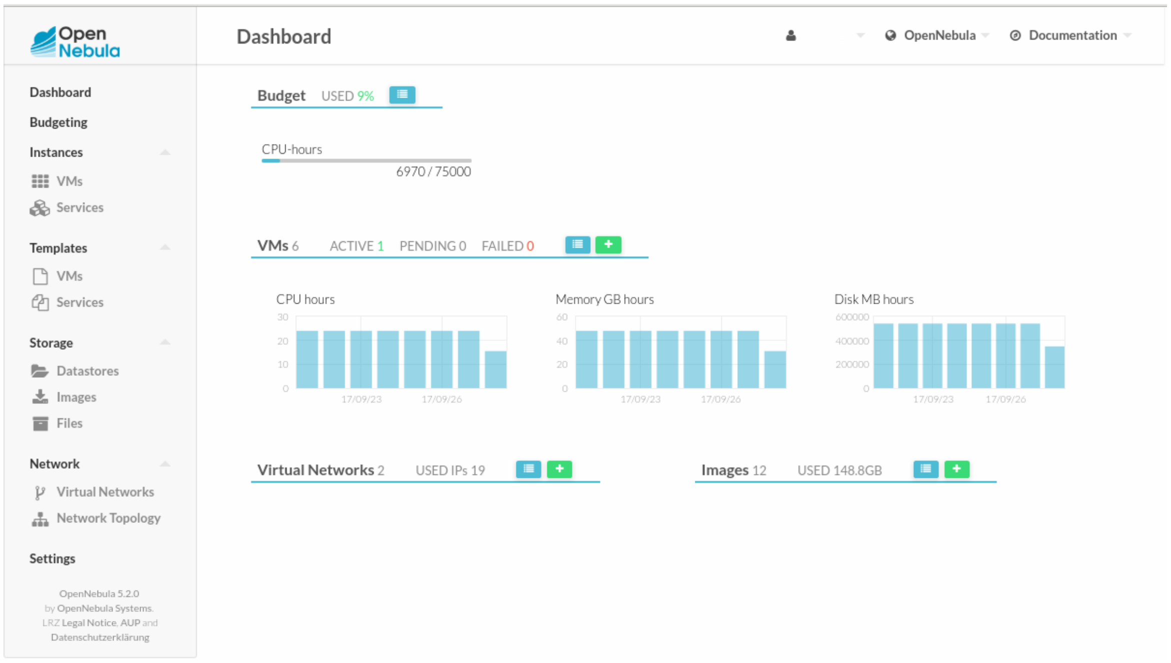


Figure 2.4.: LRZ Compute Cloud Interface

3. Requirement Analysis

The requirement analysis aims to determine the expectations of the customer and with those define a set of requirements for the end product. The initial concept and user stories were gathered during interviews with the customer and are refined into functional and non-functional requirements. Those are then used to design an appropriate system.

3.1. Raw Requirements

The initial requirements were gathered by taking notes during the conversations with the customer.

3.1.1. Customer Concept

The customer formulated the desire for an end to end solution that sends data from remote wireless sensors to a central platform in a secure manner. The central platform should be run in the cloud, allowing for multi-user access to the sensor data through a GUI. The data should be available for further processing by the SuperMUC.

3.1.2. User Stories

During the interviews with the customer use cases for the concept were discussed. The following user stores were then developed.

> As an environmental scientist, I want to access and view sensor data from remote locations while going as seldom as possible on site to collect data or change batteries.

The environmental scientist wants live access to the data from different locations she is monitoring. With the measurement site being 100km away from her work site, fewer maintenance trips amount to more locations she can monitor and time she can spend on actual work.

> As a medical practitioner, I want to gather physiological data without having to wire the patient up with invasive cables.

The medical practitioner wants to minimize the use of invasive equipment that may cause discomfort and falsify physiological readings.

> As a scientist, I want to analyze the data collected with a supercomputer instead of my workstation.

A supercomputer allows the scientist to use complex models and spend less time waiting for results.

As a system administrator, I want to have software that I can fix and patch.

The system administrator is interested in a product that can be supported for a long time and is secure.

Preliminary list of requirements

- The sensors need to be wireless and use as little energy as possible to be able to be powered by small battery packs.
- The central platform needs to be available over the Internet, display sensor data in a meaningful manner and allow multi-user access.
- The sensor data needs to be sent over the Internet in a secure way.
- The overhead of the data sent over Internet should be minimized as the Internet connection might be made on a metered dataplan.
- Handling data from multiple wireless sensors is required.
- The data should be accessible by a supercomputer.
- The software and hardware should be possible to be fixed and patched during the lifetime of the product.

3.2. Customer Requirements

The customer concept defines two physically distinct locations with different requirements put on each package residing there. A data collection device, and a central platform.

3.2.1. Functional Requirements

The features that the system must have are described by the following functional requirements.

Data Collection Device Requirements

- Wireless data transmission.
- Handle multiple wireless sensors.
- Send sensor data over the Internet.

Central Platform Requirements

- Multi user/client access to the sensor data.
- Collect data from multiple sensor sources.
- Data accessible from a supercomputer.

3.2.2. Non Functional Requirements

The non functional requirements of the system are defined by the following characteristics.

Data Collection Device Requirements

- Open-source or readily available.
- Energy consumption small enough to be powered by a small battery.
- Secure protocol for receiving/sending sensor data over the Internet.
- Minimize the overhead of data sent over the Internet.

Central Platform Requirements

- Open-source or readily available.
- Display sensor data in a meaningful manner.
- Secure transmission protocol for user access to sensor data.

3.3. Summary

In this chapter the customer concept was established along with user stories defining what expectations the customer has for the system. A set of functional and non functional requirements were derived from those stories as shown in Table 3.1.

Functional Req.	
Data Collection Device	Wireless data transmission Handle multiple wireless sensors Send sensor data over the Internet
Central Platform	Multi user/client access to the sensor data Collect data from multiple sensor sources Data accessible from a supercomputer
Non Functional Req.	
Data Collection Device	Open-source or readily available Energy consumption small enough to be powered by a small battery Secure protocol for receiving/sending sensor data over the Internet Minimize the overhead of data sent over the Internet
Central Platform	Open-source or readily available Display sensor data in a meaningful manner Secure transmission protocol for user access to sensor data

Table 3.1.: Summary of functional and non functional requirements

4. System Design

This chapter describes the system design that aims to satisfy all of the customer requirements specified in Chapter 3. The requirements define two distinct packages that differ in functionality: the data collection device, from now on named the sensor package, and the central platform *(see figure 4.1)*.

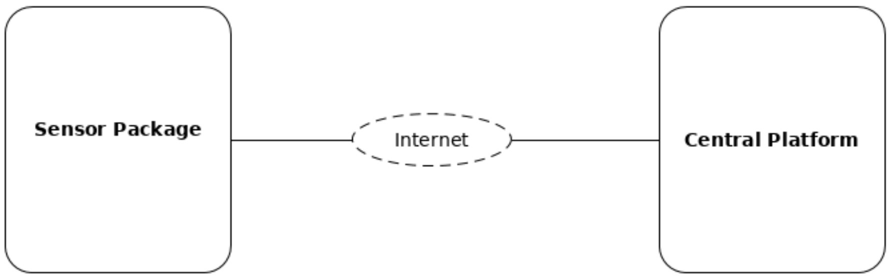


Figure 4.1.: System Design Overview

- The sensor package that gathers environmental data and sends it to the central platform over the Internet.
- The central platform that receives the data gathered from the sensor package, stores the data, provides a graphical user interface for clients to access and allows the data to be accessed from a supercomputer.

The following system was designed taking into account the customer requirements with Figure 4.2 illustrating the system design for the sensor package and central platform described in this chapter.

4.1. Sensor Package

The sensor package is the part of the system which resides at the customer location. It is made up of a bunch of wireless sensors which send their data wirelessly to a low power sensor gateway. The sensor gateway manages the connected sensors, processes the received data and is connected to the Internet. The data is sent from the sensor gateway over to the central platform using a secure transmission protocol.

The system design takes into account the need for multiple wireless sensors by making use of a wireless sensor gateway. Having the individual sensors connecting to a gateway allows to minimize the hardware required on each individual sensor module helping to reduce the power usage of each sensor. The centralization also means that only one connection to the Internet has to be made with only one device needing to implement and run the secure

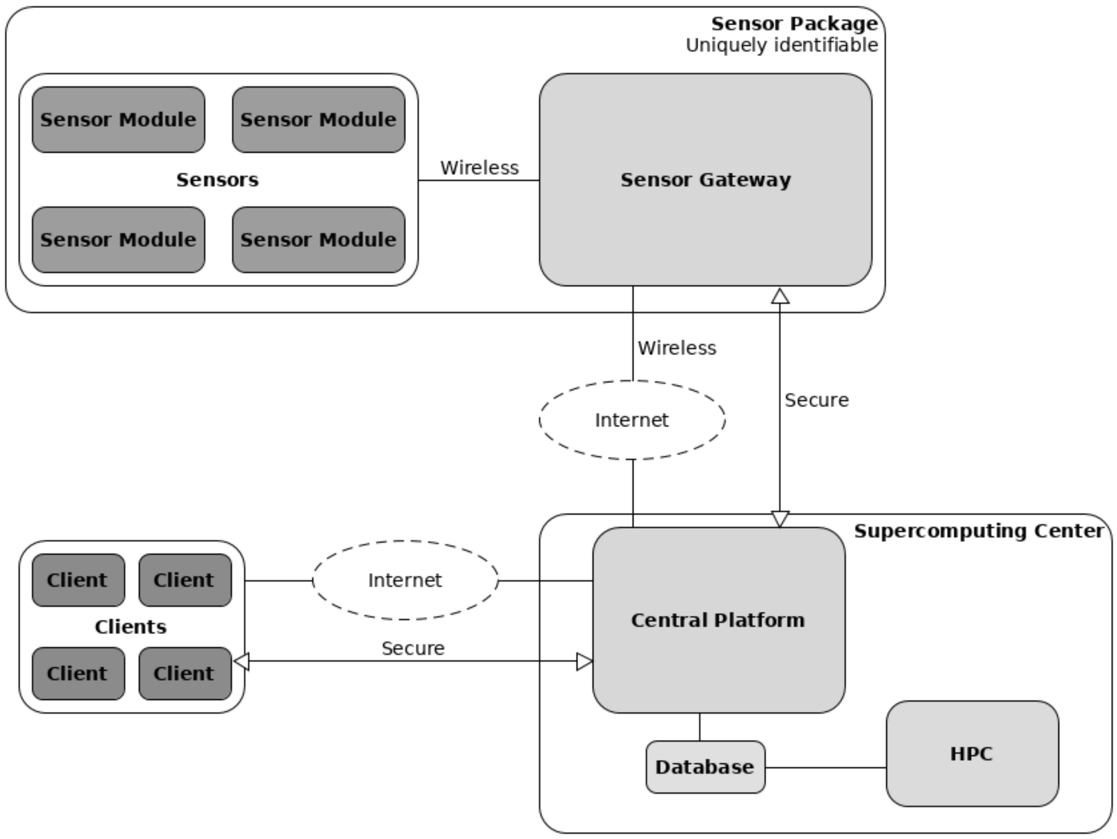


Figure 4.2.: System Design

transmission protocol to the central platform. Both the sensor gateway and the sensor module connect wirelessly. Table 4.1 summarizes the compliance of the system design with the defined customer requirements.

Requirement	
Wireless data transmission	OK
Handle multiple wireless sensors	OK
Send sensor data over the Internet	OK
Secure protocol for receiving/sending sensor data over the Internet	OK

Table 4.1.: Sensor package system design requirement compliance

4.2. Central Platform

The Central Platform part of the system design resides in a supercomputing center. It is connected to the Internet and provides the endpoint of the secure connection from the sensor gateway. The data from the sensor modules is stored in a database on a filesystem that can be accessed by a supercomputer for further computations. The central platform also uses the data stored in that database to provide a visualization of the data accessible by multiple clients over the Internet. The access of the data from the clients is done over a secure transmission protocol. The system design takes into account that the sensor data needs to be accessible by a supercomputer, but also provides visualizations for users by storing the data on a database. A unique identifier for each sensor package is used to differentiate data coming from different packages allowing data from multiple sensor sources to be collected.

Table 4.2 summarizes the compliance of the system design with the defined customer requirements.

Requirement	
Multi user/client access to the sensor data	OK
Collect data from multiple sensor sources	OK
Data accessible from a supercomputer	OK
Secure transmission protocol for user access to sensor data	OK
Display sensor data in a meaningful manner	OK

Table 4.2.: Supercomputing center system design requirement compliance

4.3. Summary

This chapter describes the system design developed to fulfill the customer requirements in Chapter 3. Two distinct parts of the system are defined; the sensor package and central platform. The function of both parts is described while making sure that all customer requirements are taken into account.

5. Implementation

The implementation of the system design went through multiple iterations to find suitable hardware and software solutions to fulfill the customer requirements. The first prototype is based on the initial idea of this thesis, making use of the available Wunderbar software and hardware to implement the customer requirements.

Section 5.1 describes the analysis of the design, architecture, hardware and software of the Wunderbar IoT Kit followed by the method attempted to reprogram the Wunderbar master module.

In Section 5.2 an implementation of the system design is developed for the sensor package with the Wunderbar master module replaced by off the shelf hardware, along with a central platform implementation. The messaging protocol used for this design is MQTT-SN.

The messaging protocols investigated do not provide builtin security and rely on the underlying transport layer to take care of the encryption. The prototype described in Section 5.3 builds on the previous implementation and uses DTLS to secure the communication between the sensor package and the central platform using the TinyDTLS RIOT library.

5.1. First Prototype - Wunderbar

Being a major source of inspiration for the customer concept, the Wunderbar Internet of Things Starter Kit for App Developers system architecture was looked into first *(see figure 5.1)*. The company relayr had a working implementation of the customer concept described in Chapter 3, but the hardware and software support was discontinued due to the security issue described in Section 1.1. The software running on the Wunderbar IoT Kit has been open-sourced and is available on github[1]. The central platform software is not available publicly.

5.1.1. Sensor Package Hardware

The hardware contained in the Wunderbar Internet of Things Starter Kit for App Developers was used for this prototype.

The package consists of six smart-modules and a master module. The smart-modules are using the nordic nRF51822 BLE System on a Chip (SoC), each of them connecting to the master module over BLE and are powered (except the bridge module) with a Li-ion coin cell battery.

- Light/Proximity sensor
- Accelerometer/Gyro sensor
- Temperature/Humidity sensor
- IR transmitter
- Microphone
- Bridge module

[1]https://github.com/relayr/WunderBar-firmware-release

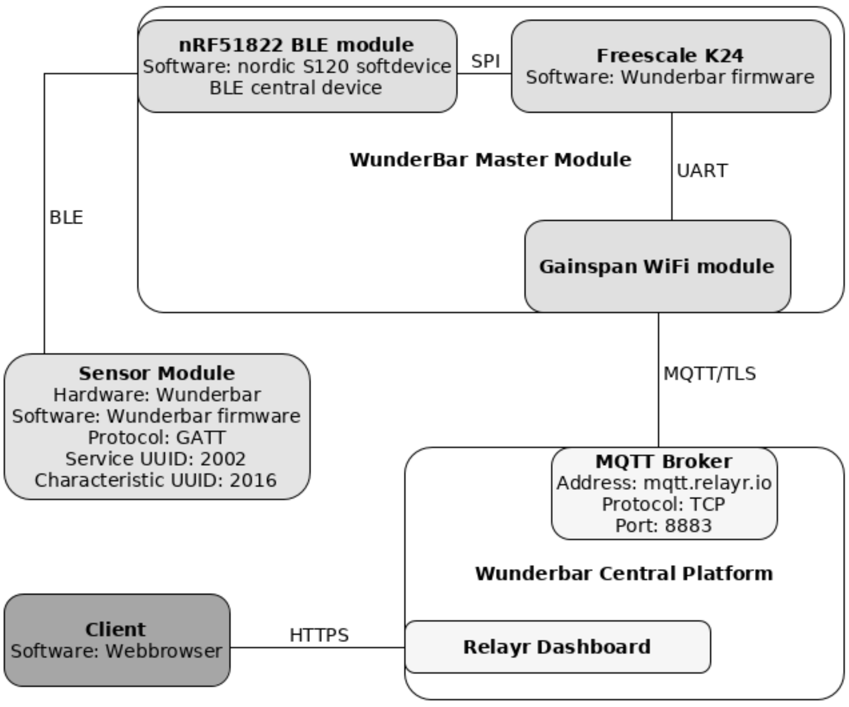


Figure 5.1.: Wunderbar prototype implementation

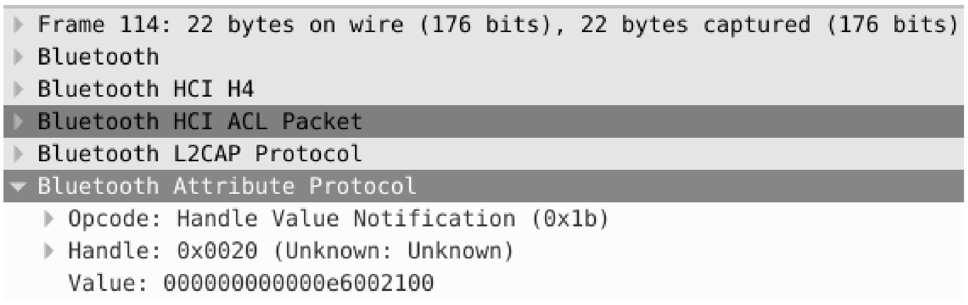
The master module has three chips onboard and is powered by a Li-ion rechargeable battery. As central processor, it uses a Freescale K24 chip powered by a 120MHz ARM Cortex-M4 Microcontroller with 256kB RAM. For the BLE connectivity it uses a Nordic nRF51822 chip and for the WiFi connectivity it uses a Gainspan GS1500M WiFi module. The three modules use Serial Peripheral Interface Bus (SPI) and Universal Asynchronous Receiver-Transmitter (UART) respectively to communicate with each other [rel17b].

5.1.2. Sensor Package Software

The sensor package has specific software applications running on each of the different parts of the system.

Sensor Module The firmware of the sensor modules is based on the Nordic Semiconductor Softdevice v110 which implements the Bluetooth low energy (BLE) Peripheral protocol stack [Sem13].

Each sensor exposes after connecting to a central device a GATT Service with the UUID of 0x2002. This service contains a characteristic named SensorData with the UUID of 0x2016. Reading this characteristic does not retrieve any live sensor data but a default value. On the other hand, requesting notifications for the SensorData characteristic triggers the sensor module to send data within a notification packet every time the characteristic value changes *(see figure 5.2)*. Enabling notifications for a specific characteristic requires writing 0x0001 to its Client Characteristic Configuration Descriptor (CCCD) [(SI17].



```
▶ Frame 114: 22 bytes on wire (176 bits), 22 bytes captured (176 bits)
▶ Bluetooth
▶ Bluetooth HCI H4
▶ Bluetooth HCI ACL Packet
▶ Bluetooth L2CAP Protocol
▼ Bluetooth Attribute Protocol
  ▶ Opcode: Handle Value Notification (0x1b)
  ▶ Handle: 0x0020 (Unknown: Unknown)
    Value: 000000000000e6002100
```

Figure 5.2.: SensorData characteristic notification packet

The data sent within the SensorData characteristic notification is a string of bytes that represents all measurements taken by the sensor at a point of time. This means that if the sensor measures temperature and humidity, the SensorData characteristic notification value needs to be split and translated accordingly to get the individual values. Clues for the conversion algorithms were found within the relayr Wunderbar firmware release.

Master Module The firmware of the master module is divided into two applications. On the nRF51822 chip runs a program based on the Nordic Semiconductor Softdevice v120 which implements the BLE Central protocol stack [Sem14b]. This application manages connections and collects data from the sensor modules. The sensor data is then sent via SPI to the firmware running on the Freescale K24 module. This program creates an MQTT payload from the data it has received via SPI which is sent to an MQTT server it has connected to through the onboard WiFi module. The data sent over the Internet is encrypted with the TLS protocol.

Wunderbar Central Platform The Wunderbar central platform is made out of an MQTT server listening on the address ”mqtt.relayr.io”. How the sensor data is stored and processed is unknown due to the software not being available. Users can log in to the relayr dashboard where the data is displayed per sensor in the dashboard web application.

5.1.3. Reviving the Wunderbar Master Module

Connecting to the sensor modules with a BLE application proved non problematic with data being received after connecting and registering for notifications to the SensorData characteristic. Therefore the firmware running on the nRF51822 chips of the sensor modules is used as is for all prototypes.

The firmware running on the master module K24 microcontroller needs to be modified to connect to a different MQTT server. The SSL certificate also needs to be changed as it uses the obsolete hashing algorithm SHA-1. The investigation of the firmware source code showed that the MQTT server address is defined as a macro in the header file Common_Defaults.h, in the folder WunderBar_WiFi/Sources/ *(see listing 5.1).* The SSL certificate is defined within the file GS_Certificate.c in Distinguished Encoding Rules (DER) format, line 109 in the folder Wunderbar_WiFi/Sources/GS/GS_User/.

```
#define DEFAULT_MQTT_SERVER_URL "mqtt.relayr.io"
```

Listing 5.1: Common_Defaults.h file line 78

Looking at the project files in the WunderBar_WiFi/ folder allowed to determine that the Software can be built using Kinetis Design Studio (KDS), available on the NXP website [NXP17]. Adjusting the paths within the KDS project file allowed to compile the firmware successfully.

The next challenge was to load the compiled firmware onto the master module. The master module has a firmware update mode that is accessed by a long-press of a button on the right side of the module. Upon entering firmware update mode while connected over Universal Serial Bus (USB) to a computer, the module connects to the USB bus, but no driver for the device is found. A clue was found with the "lsusb" tool showing that the device uses the Device Firmware Upgrade (DFU) mechanism for firmware upgrades *(see listing 5.2)*.

```
Bus 001 Device 003: ID 15a2:1000 Freescale Semiconductor,
    Inc.
Device Descriptor:
[...]
      Device Firmware Upgrade Interface Descriptor:
        bLength 9
        bDescriptorType 33
        bmAttributes 13
          Will Detach
          Manifestation Tolerant
          Upload Unsupported
          Download Supported
        wDetachTimeout 0 milliseconds
        wTransferSize 4096 bytes
        bcdDFUVersion 1.10
[...]
```

Listing 5.2: lsusb output for the Wunderbar master module

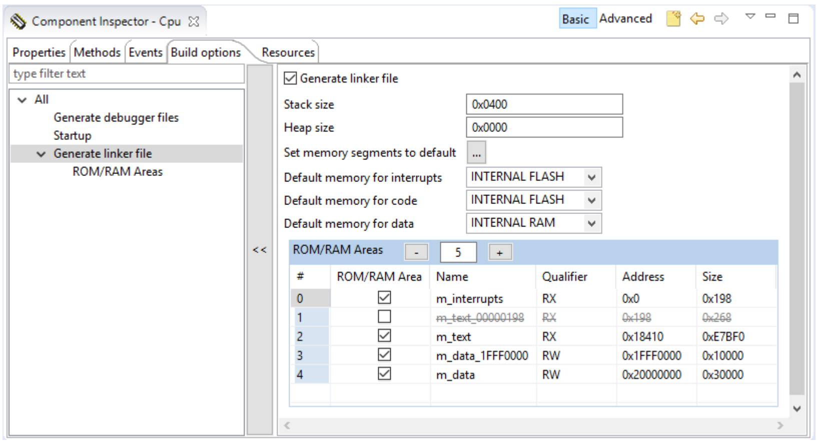
Using the "dfu-utils" utility[2], the previously compiled firmware for the K24 microcontroller was downloaded successfully onto the master module but leaving it unresponsive. Suspected for this behavior is incorrect linker memory values within the master module KDS processor expert file *(see figure 5.3)*. With no values found to work, the Wunderbar module was abandoned.

Although the Wunderbar starter kit could have been able to fulfill the customer requirements, it wasn't possible in this thesis, to program the master module with a modified relayr Wunderbar firmware. The Wunderbar sensor modules were kept for further prototypes as they are wireless sensors and the firmware running on them worked as intended.

5.2. Second Prototype - Replacing the Master Module

Due to the unsuccessful programing of the Wunderbar master module, a solution to replace it was sought after. Figure 5.4 provides an overview of the hardware and software used in this prototype.

[2] http://dfu-util.sourceforge.net/



☑ Generate linker file

Stack size	0x0400
Heap size	0x0000
Set memory segments to default	...
Default memory for interrupts	INTERNAL FLASH
Default memory for code	INTERNAL FLASH
Default memory for data	INTERNAL RAM

ROM/RAM Areas 5

#	ROM/RAM Area	Name	Qualifier	Address	Size
0	☑	m_interrupts	RX	0x0	0x198
1	☐	~~m_text_00000198~~	~~RX~~	~~0x198~~	~~0x268~~
2	☑	m_text	RX	0x18410	0xE7BF0
3	☑	m_data_1FFF0000	RW	0x1FFF0000	0x10000
4	☑	m_data	RW	0x20000000	0x30000

Figure 5.3.: KDS K24 linker file configuration

The relayr Wunderbar sensor package satisfied all the customer requirements except minimizing the size of data sent over the Internet by using the TCP protocol. To reduce the overhead of the transmission protocol, the MQTT protocol was replaced with the MQTT-SN protocol. MQTT-SN uses the transaction-oriented UDP protocol instead of the connection-oriented TCP protocol reducing the amount of packet overhead.

5.2.1. Sensor Gateway Hardware

The sensor gateway acts as a bridge between the sensor modules and the wireless gateway. Therefore it needs to have BLE and WiFi capability. Interest was turned towards Arduino, specifically the Arduino M0 Pro board that is powered by an ATSAMD21G18 MCU running at 48MHz with 32Kb RAM and has a low power consumption of 44mA [Ard17b]. In comparison the most lightweight Raspberry Pi Zero bare-board consumes 100mA [Fou17b]. Although the Arduino M0-Pro has no networking capabilities onboard, it has multiple Digital I/O Pins which can be used to add the missing connectivity.

WiFi Module The Espressif esp8266 WiFi module [Sys17] was chosen to enable the Arduino to connect to the wireless gateway. The module implements it's own TCP/UDP stack and the interaction with the WiFi module is done via UART by sending AT commands [Inc17]. It has been shown that the module can successfully provide TCP and UDP connectivity to an Arduino M0-pro board [Hei17].

BLE Module The connection to the wireless sensors over BLE required investigating possible BLE modules for the Arduino platform.

The first BLE module looked into was a module purchased as a HC-08 BLE Cc2541. Following the documentation for HC-08 modules [wav14], sending AT commands without a newline to the module did not incite any response. Sending a newline and carriage return after a command did and most of the commands from the above documentation did not work. The article from Martyn Currey [Cur16], helped identify the module as an AT-09 with a firmware from bolutek. With the module identified, it was possible to run the supported AT commands as described in the documentation for the bolutek firmware [Bol16]. Using

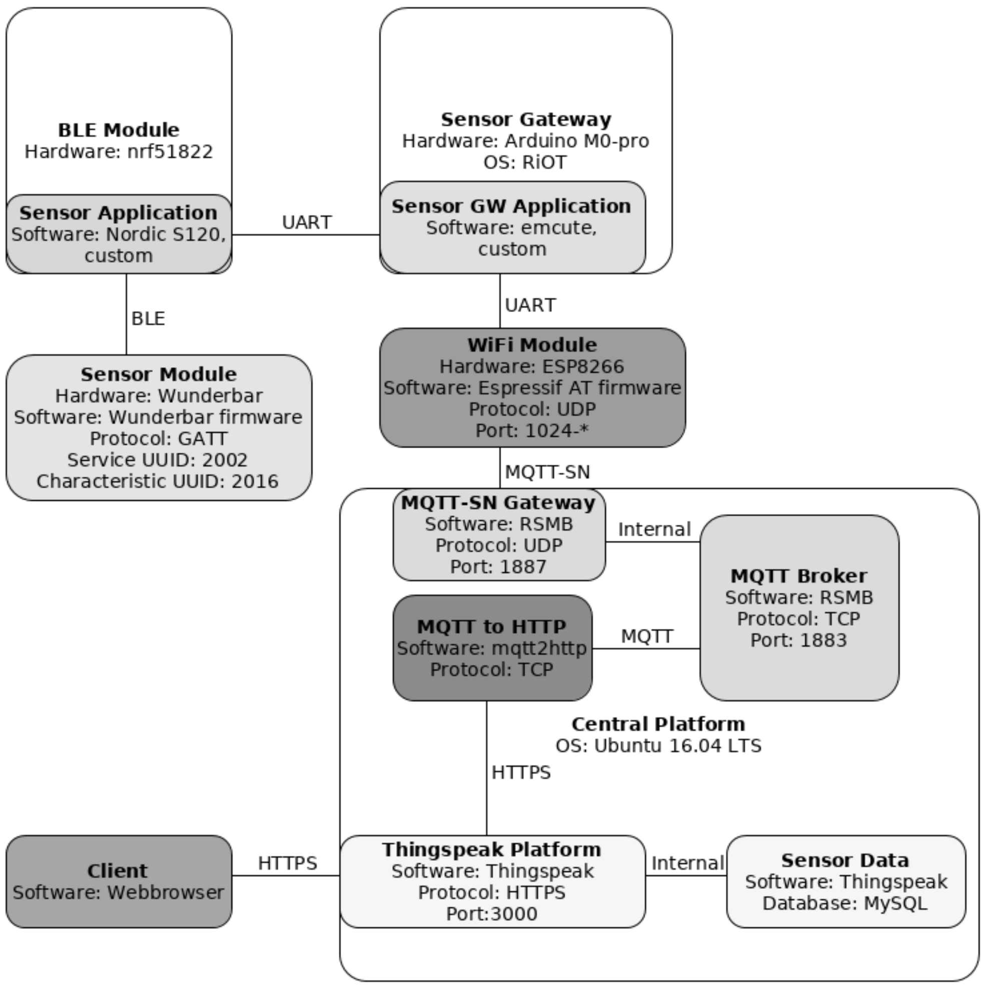


Figure 5.4.: Second prototype implementation

the available functionality of the BLE module it was possible to connect to the Wunderbar SensorData characteristic, but the firmware does not have an AT command to write to the CCCD. Therefore no data is received from the Wunderbar sensor. The lack of this feature made the use of the AT-09 BLE module alongside the Wunderbar sensors not possible without modifying its firmware. With the development of software for the Texas Instruments CC2541 requiring a proprietary non-readily available platform [Inc15], the software would not be able to be developed with readily available tools.

The next BLE module investigated was the nordic nRF51822 [Sem17]. As shown by the Wunderbar master module, this MCU can connect and gather data from the Wunderbar sensor modules. Importantly the software for this BLE module can be developed with readily available tools such as GCC or Keil MDK-ARM [Sem17].

5.2.2. Sensor Gateway Software

The software for this prototype has the Wunderbar sensor modules connected to the nRF51822 device via BLE. The device sends sensor data over UART to the Arduino M0-pro board, which in turn sends the data via UART to the esp8266 module. The data is then transmitted over UDP to the central platform.

BLE Module The development of the software running on the nRF51822 chip was done in Keil MDK V5.23.0.0 and is based on the S120 Softdevice BLE Multi-link Central application. This implements the connection/management of multiple bluetooth peripheral devices running the BLE Multi-link Peripheral application from the nRF51 SDK and displays the data received on an LCD display [Sem14a].

The BLE Multi-link Central application was modified so that only BLE peripheral devices with the names matching Wunderbar sensor modules may connect to it with UART used to output the received data.

Upon connecting to a device with the Wunderbar sensor module name, the software was programmed to determine whether it contains a service with the UUID of 0x2002 and a characteristic with the UUID of 0x2016 . If those requirements are met, notifications are enabled for that characteristic. The amount of characteristics discovered per service needed to be increased to at least seven due the UUIDs being discovered in increasing UUID order.

To differentiate between sensor modules, the client context structure was extended to contain the client device name and BLE address *(see listing 5.3)*.

```
typedef struct
{
        ble_db_discovery_t srv_db; /**< The DB Discovery
            module instance associated with this client. */
        dm_handle_t handle; /**< Device manager identifier
            for the device. */
        uint8_t char_index; /**< Client characteristics index
            in discovered service information. */
        uint8_t state; /**< Client state. */
        const uint8_t * device_name; /**< Client Device
            Name. */
        ble_gap_addr_t peer_addr; /**< Bluetooth Low
            Energy address. */
} client_t;
```

Listing 5.3: BLE client context information

Upon receiving a BLE notification from a connected device, the software was programmed to print the received data, alongside the device name and BLE address to the nRF51822 UART pins *(see listing 5.4)*.

```
device_name:peer_address:data\n
```

Listing 5.4: nRF51822 BLE application UART data packet

Sensor Gateway Module The RIOT operating system was chosen to run the sensor gateway application on the Arduino board. RIOT is a lightweight real-time operating system using very little RAM, supports a large amount of boards and has full C support [RO17a]. The task of the software running on the Arduino M0-pro is to act as a gateway between the sensor modules and the central platform.

So that both the nRF51822 and esp8266 module can connect to the Arduino M0-pro an extra UART interface was required. Per default the Arduino M0-pro only has one UART interface, but three out of six available Serial Communication Modules (SERCOM) can be customized. Research shows that SERCOM5 is unused [ada17] and was therefore configured as an extra UART interface for the Arduino M0-pro board *(see listing 5.5)*.

```
static const uart_conf_t uart_config[] = {
        {
                .dev = &SERCOM5->USART,
                .rx_pin = GPIO_PIN(PB,23),
                .tx_pin = GPIO_PIN(PB,22),
                .mux = GPIO_MUX_D,
                .rx_pad = UART_PAD_RX_3,
                .tx_pad = UART_PAD_TX_2
        },
        {
                .dev = &SERCOM0->USART,
                .rx_pin = GPIO_PIN(PA,11),
                .tx_pin = GPIO_PIN(PA,10),
                .mux = GPIO_MUX_C,
                .rx_pad = UART_PAD_RX_3,
                .tx_pad = UART_PAD_TX_2
        },
        {
                .dev = &SERCOM1->USART,
                .rx_pin = GPIO_PIN(PA,18),
                .tx_pin = GPIO_PIN(PA,16),
                .mux = GPIO_MUX_C,
                .rx_pad = UART_PAD_RX_2,
                .tx_pad = UART_PAD_TX_0
        }
};

/* interrupt function name mapping */
#define UART_0_ISR isr_sercom5
#define UART_1_ISR isr_sercom0
#define UART_2_ISR isr_sercom1
```

Listing 5.5: arduino-zero periph_conf.h SERCOM5 configuration

The interaction between the RiOT and the esp8266 module was made based on the esp8266 driver developed by Tobias Heider and Florian Eich [Hei17]. The driver sends AT commands as strings over the UART interface, with the esp8266 responding to those with preset responses such as "OK" or "SEND OK". The data received by the WiFi module is sent over UART and stored in a ringbuffer within the driver. The esp8266 driver receive function parses the ringbuffer for the AT command signaling the beginning of a packet, then stores the data that follows into a string. The driver was extended to enable it to be used for multiple UART devices. Additionally, timeouts were implemented for receiving data functions to allow non blocking functionality. Methods to close sockets and get the MAC address of the WiFi module were also implemented.

The sensor gateway application consists of two threads. A controller thread that handles incoming UART sensor packets, creating MQTT-SN packets and sending them, and a listener thread that receives incoming packets and handles them appropriately. The program uses the RIOT MQTT-SN client library "emcute", reprogrammed for the esp8266 driver instead of the RIOT modular default IP network stack (GNRC) [RO17b].

1. Controller Thread: The thread communicates over UART with the esp8266 WiFi controller, keeping a UDP socket open to the server running the MQTT-SN gateway and sends data to it. Using the emcute MQTT-SN client library, it manages the connection to the MQTT-SN gateway.

   Data received over the UART controller to whom the nRF51822 MCU is connected to, is saved into a string until the newline command is received. This string represents one sensor data packet and is parsed using the colon character as a separator saving the sensor device name, BLE address and data into separate strings. The data string is further processed with different algorithms depending on which sensor device type the data comes from converting the data bytes into integer values representing individual data points such as temperature or humidity. The sensor device type is derived from the device name received in the sensor packet. The individual sensor values are then sent over the Internet to the MQTT-SN gateway. The JavaScript Object Notation (JSON) data-interchange format [jso17] is used allowing for simple parsing of the message using existing JSON libraries *(see listing 5.6)*. The field names and write API Key in this implementation are hardcoded to a specific sensor module and sensor gateway respectively.

2. Listener Thread: Handles incoming packets by implementing the MQTT-SN protocol. If a disconnect packet has been received or a timeout has occurred, it signals the controller thread via inter process communication (IPC) using the RIOT Messaging API [RO17c]. This allows the controller thread to reconnect to the MQTT-SN gateway.

```
{"key": WRITE_API_KEY, "field1": DATA_VALUE-2, "field2
    ": DATA_VALUE-1}"
```

Listing 5.6: MQTT-SN message JSON format

5.2.3. Central Platform Hardware

The central platform software is running within a virtual machine on the LRZ Compute Cloud, decoupling the operating system from the physical hardware [Edu07]. This allows flexible deployment of the platform without worrying about the underlying hardware and access to the data from the SuperMUC with the use of DSS filesystems.

5.2.4. Central Platform Software

Many IoT platforms have been developed to connect smart objects with different sets of features and architectures enabling users to interact with those objects [MMST16]. Research has been done to map the IoT platform landscape with detailed analysis of their technologies and objectives. Thingspeak was chosen from the recent IoT platform gap-analysis of Mineraud et al. [MMST16] to be used in this thesis as it is:

- Open source.
- Cloud-based.
- Supports heterogeneous devices.

Thingspeak Platform Thingspeak is an open source Internet of Things application that allows to store and retrieve data sent to it using a REST API [iob14]. The information is organized in channels that can contain up to eight individual fields of data. Each channel has a write API key, that allows reading from or writing data to that specific channel.

The data is uploaded to the channel via a GET/POST request using the write API key with all eight fields being able to be updated at once *(see listing 5.7)*. The user can access his data via a webbrowser with a username and password. This data is represented in a time series graph for each field in the channel view.

```
https://IP_ADDRESS:PORT/update?key=
    WRITE_API_KEY&field1=DATA_VALUE2&
    field3=DATA_VALUE1
```

Listing 5.7: Thingspeak channel update request format

The operating system on the VM was chosen with the purpose to run the Thingspeak application. The Thingspeak installation documentation was written for Ubuntu 12.04 LTS [iob14], but since that version has reached end of life [Ltd17], the current Ubuntu LTS version 16.04 was chosen.

MQTT-SN Server The Really Small Message Broker (RSMB)[3] is used to provide the MQTT-SN gateway functionality. The program compiled without any complications following the instructions found on the RSMB github page [Ecl16]. It is configured to listen on the Internet facing network interface on port 1887 for the MQTT-SN listener, and the normal MQTT listener bound to localhost on port 1883 with IPv6 disabled *(see listing 5.8)*.

```
# Normal MQTT listener
listener 1883 127.0.0.1
# MQTT-SN listener
listener 1887 INADDR_ANY mqtts
```

Listing 5.8: RSMB configuration options

MQTT2HTTP The MQTT2HTTP[4] application is used to create the REST requests from the MQTT-SN messages received from the sensor application package. Originally it used the insecure HTTP protocol but the application is modified to use HTTPS. The application is configured to connect to the MQTT broker over localhost create HTTPS requests from each message received Appendix A.3. The program extracts the field and data from the MQTT-SN message described in Listing 5.6 and sends an HTTPS request with the sensor data to the Thingspeak server. As the MQTT-SN message from different sensors contains variable amounts of fields, the extraction method of the MQTT2HTTP software was reprogrammed.

[3]https://github.com/eclipse/mosquitto.rsmb
[4]https://github.com/FokkeZB/mqtt2http

Thingspeak The Thingspeak installation on the server required some extra steps compared to the original iobridge documentation due to the newer operating system. In particular the ruby environment 2.1.0 and specific versions of gem files are required. The software uses HTTP by default and is modified to use HTTPS with the installation steps documented in Appendix A.1. Thingspeak stores sensor data within a mysql database. The database can be on a DSS filesystem which can be accessed by the SuperMUC for further computations.

5.3. Third Prototype - Secure Data Transfer

With the central platform requiring a major rework to use DTLS, an implementation was sought after that would reduce the amount of resources required on the client side. This was done by eliminating the messaging protocol, and building the applications around the DTLS protocol.

The sensor gateway in this prototype directly sends the sensor data in the DTLS payload to the central platform. The application running on the Arduino thus only has to handle DTLS related events and forward sensor data. The parsing and conversion of sensor data was moved to the central platform application.

Not using the messaging protocol allows to reduce the complexity and memory usage of the application. It also cuts some overhead off the data sent over the Internet between the client and the server. The major drawback is the loss of the publish/subscribe functionality, but this feature was not used specifically by the prototypes. Client session management is handled by the DTLS protocol instead of the MQTT-SN protocol.

The prototype was designed to take advantage of the DTLS protocol to provide confidentiality for the data sent between the sensors and the IoT platform. The applications running on the nRF51822, esp8266 and Wunderbar sensors were taken from the previous prototype, with new applications being developed for the sensor gateway application and the central platform. This approach allows to move computations away from the sensor gateway which has little hardware resources, to the central platform which has a lot of hardware resources along with consolidating the three central platform applications into one *(see figure 5.5)*.

The RIOT tinydtls DTLS package [RO17d] provides support for the latest DTLS version 1.2 and is open source [Fou17a]. A customized version of the RIOT dtls-echo example application using the esp8266 driver on the client and Linux sockets on the server was used for initial testing of the RIOT TinyDTLS library[5]. After changing the TinyDTLS package version to the current TinyDTLS release, encrypted messages were exchanged successfully.

5.3.1. Sensor Gateway Hardware

The sensor gateway hardware needed to be changed as running the TinyDTLS library with 32Kb RAM of the Arduino M0-pro did not work. Each time the TinyDTLS library tried to create a new dtls_peer_t structure using the malloc function call, the program failed. This forced changing the sensor gateway hardware to an Arduino due with 96Kb RAM [Ard17a]. Due to the nature of the RIOT operating system supporting multiple boards and architectures, the hardware change did not require any modifications to the application.

[5]https://github.com/rfuentess/TinyDTLS

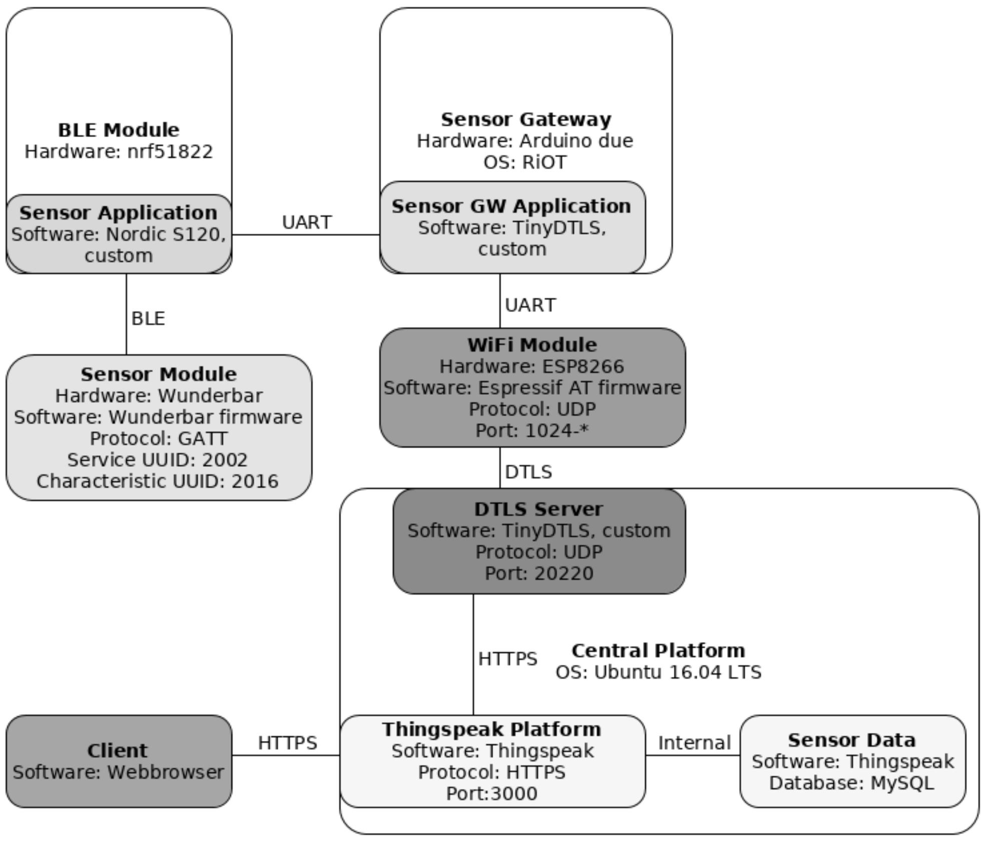


Figure 5.5.: Third prototype implementation

5.3.2. Sensor Gateway Software

The TinyDTLS library internally handles the DTLS packets and provides the dtls_handler_t structure holding function pointers allowing to define a function to be run when specific events occur. Those pointers are:

- Read: Called from dtls_handle_message() and delivers application data that was received. The data is provided only after decryption and verification has succeeded. In this implementation received data packets are not further processed except when a specific keepalive string is received. This string triggers the keepalive timer to be reset as described further in this section.
- Write: Called from dtls_handle_message() to send DTLS packets over the network. This function is programmed to use the esp8266 driver to send the data.
- Event: The state of the DTLS session has changed or an alert has been raised. In this program the DTLS session state signaling that the connection with the dtls-server has been established, triggers the start of the listener thread.

The application logic of the client was built upon the previous prototype, with a main thread handling the UDP connection and DTLS session with the dtls-server via the esp8266

WiFi module. A second thread is used to listen for incoming data on the UART interface from the nRF51822 module and send it to the dtls-server. Errors within the threads are signaled to each other via IPC using the RIOT Messaging API [RO17c], allowing them to terminate before reconnecting to the dtls-server.

The MAC address of the esp8266 WiFi module is added to the sensor data packet allowing to determine from which sensor gateway the packet has come from *(see listing 5.9).*

```
gateway_address:device_name:peer_address:data\n
```

Listing 5.9: Sensor gateway data packet

When a sensor packet is received, the listener thread passes the data on to the DTLS library for encryption which when completed triggers the write function defined in the dtls_handler_t structure.

Due to the performance-constrained environment, the choice of using DTLS with pre-shared keys was made [Gro05]. Using pre-shared keys also circumvents the problem of expiring or insecure certificates as seen with the Wunderbar.

Handling Broken Connections By using the datagram protocol the client does not know for sure if it is still connected to the dtls-server at any point of time. For example the peer could have been invalidated by the server but the alert message from the server has not been received. The client could be sending data that is just being dropped by the dtls-server. For this purpose this implementation uses a keepalive mechanism. The client sends at regular intervals a specific message to the server that when decrypted by the server triggers the message to be echoed back to the client. On the client side receiving the keepalive string resets the timeout timer. If the timeout is reached on the client, the DTLS session is closed and reinitiated.

5.3.3. Central Platform Software

Not using the MQTT-SN protocol on the central platform allows cutting the MQTT-SN/MQTT gateway/broker and the mqtt2http publisher replacing those with one application that implements the dtls-server, parsing and conversion of the sensor messages and sending HTTPS requests to the Thingspeak application. The Thingspeak IoT platform remains the same as described in Section 5.2.4

DTLS Server The dtls server is built around the TinyDTLS library and therefore works with the same dtls_handler_t structure function pointers as the dtls-client (see section 5.3.2). The read function parses the received sensor data with the functions that were developed for the previous prototype. The application directly creates the Thingspeak HTTPS request to update the channel data using the cURL library[6]. A framework to build the request with the write API key depending on the MAC address of the sensor gateway and the fields depending on the sensor device name parsed from the data received is present *(see listing 5.9).* The write function sends data over an open socket to the specific connected peer. The event function doesn't have any logic in this server implementation.

[6]https://curl.haxx.se/libcurl/

5.4. Summary

This chapter describes the different prototypes that were developed to implement the system design. In Section 5.1 the software and hardware of the Wunderbar IoT Kit is analyzed, followed by the attempt to successfully reprogram the Wunderbar master module with a modified firmware. This did not succeed. The prototype in Section 5.2 uses an off the shelf replacement for the Wunderbar master module that is programmed to act as a bridge between the BLE sensor modules and the central platform using WiFi to connect to the Internet. The MQTT-SN messaging protocol was chosen instead of the MQTT protocol used by the Wunderbar IoT Kit. With the second prototype not having any security mechanisms a third prototype was developed in Section 5.3 using the DTLS protocol. Instead of building DTLS into the previous prototype, a leaner implementation was chosen that sends the sensor data directly within the DTLS payload. This allows the overhead of the data sent over the Internet to be further reduced, cut down computations required by the sensor gateway and remove the need to forward MQTT messages within the central platform to the Thingspeak REST API.

6. Evaluation

Each prototype was evaluated whether it satisfies the customer requirements set in Chapter 3. The central platform and sensor package was setup in a testbed environment as described in Section 6.1. Prototype two and three were run to see whether sensor data is sent successfully to the central platform and their power consumption was measured. Prototype three was left to run for a longer period of time to test the reliability of the implementation.

6.1. Test Setup

A full evaluation of the first implementation was not done as the modified Wunderbar master module firmware was not able to be programmed successfully onto the device.

The implementation of the second and third prototype was setup as shown in Figure 6.1. The Wunderbar temperature/humidity (HTU) and accelerometer/Gyroscope (GYRO) sensors were used to gather environmental data. The HTU sensor was attached outside the building, and the GYRO sensor on a window within range of the BLE central device. A new lithium ion coin cell battery (CR2032) was inserted into the GYRO sensor. The sensor gateway was placed on the windowsill and powered by a USB charger. A wireless router was setup within WiFi range of the sensor gateway with a WPA2 secured WiFI network with access to the Internet. The power consumption of the devices were measured using a multimeter.

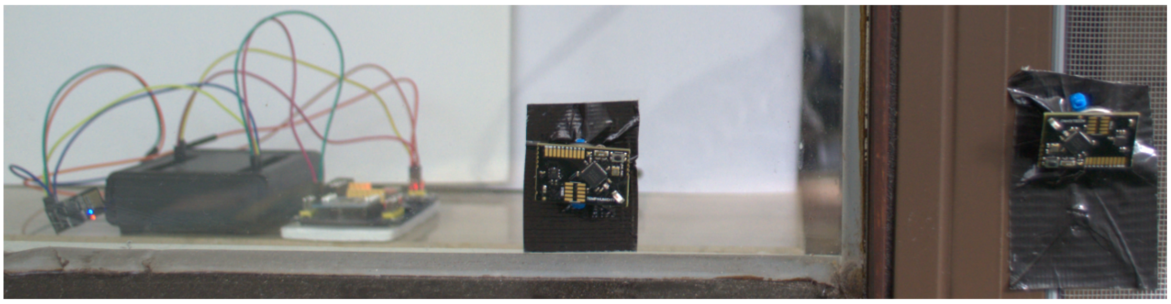


Figure 6.1.: Test setup

6.1.1. Central Platform Installation

The central platform was installed on a VM on the LRZ Compute Cloud as described in Appendix A.1 along with the RSMB and dtls-server. The MQTT2HTTP connector was installed and configured on the same VM following the instructions compiled in Appendix A.3. A startup script was created to start all the services.

6.1.2. Programming the Sensor Gateway Modules

The nRF51822 device was programmed with an external programmer as the NRF51822 Eval Kit did not have one.

Figure 6.2 shows on the right a STM32F412 Nucleo-144 board with its top section being a ST-Link programmer/debugger. The two CN4 jumpers were removed so that the programmer/debugger can be used to program an external Cortex-M chip. On the left is the nRF51822 Eval Kit. The Serial Wire Debug (SWD) ports of both devices are connected with male/female wires that are passed through a breadboard as seen in the middle of fig. 6.2. The nRF51822 device was first fully erased, then programmed with the nordic Softdevice S120 v1.0.1, and finally with the custom application as described in Appendix A.2 using "openocd" v0.10.0[1].

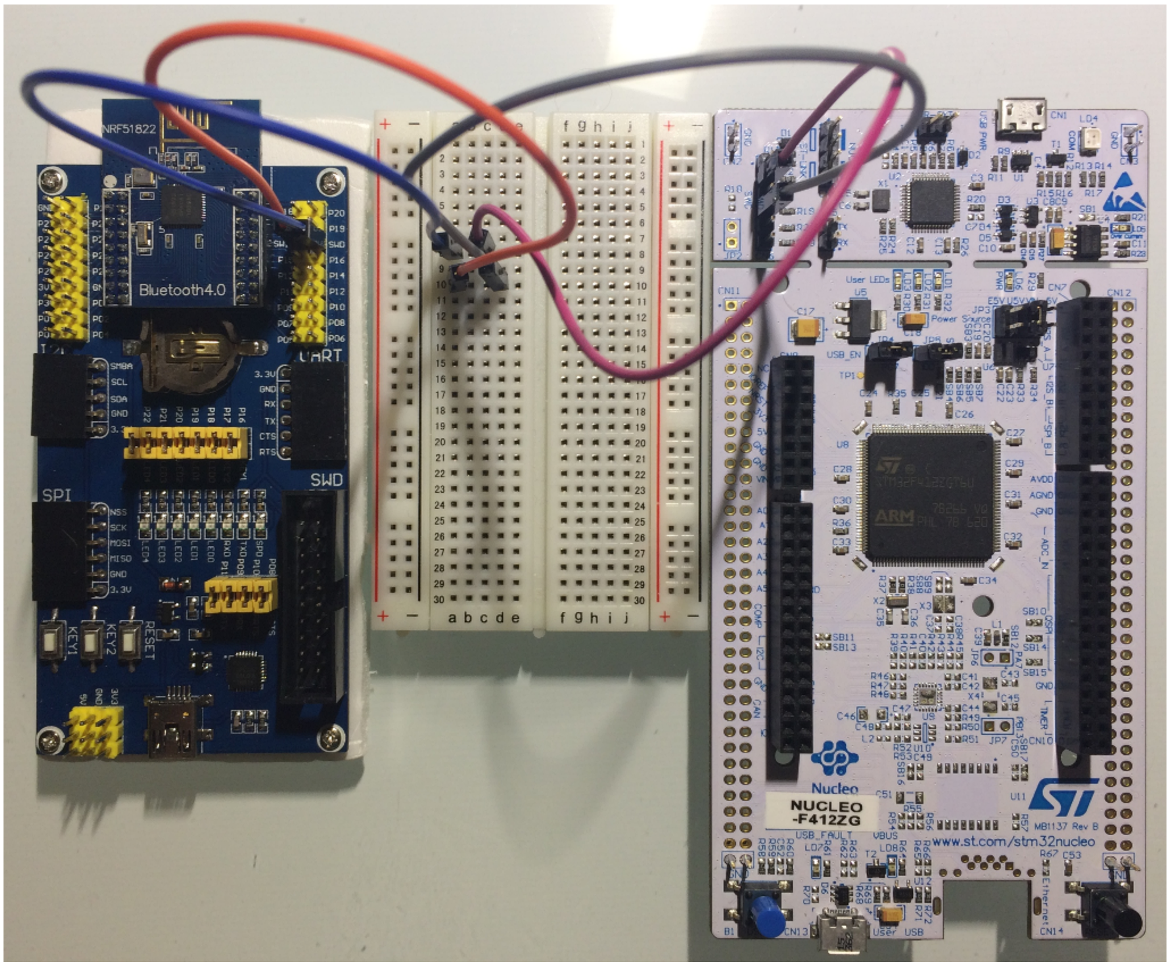


Figure 6.2.: nRF51822 programming setup

The Arduino devices were programmed using the builtin USB programming port using the tools provided by the RIOT distribution. The RIOT flashing tool uses openocd for the Arduino-zero and bossac for the Arduino-due.

[1]http://openocd.org/

6.2. First Prototype Results

Sensor Package Hardware The Wunderbar hardware albeit not being readily available anymore after being discontinued has its schematics open sourced. These documents have not been reviewed in this paper but with the assumption that those are complete, anyone with the appropriate skills and facilities could reproduce the hardware. The sensor package is fully wireless with the sensor modules being connected to the master module via BLE and the master module via WiFi to a wireless gateway. Both are powered by small battery packs. This meets all the customer requirements made for the sensor package hardware *(see table 6.1)*.

Requirement	
Open-source or at least readily available	OK
Wireless data transmission	OK
Powered by a small battery	OK

Table 6.1.: Wunderbar IoT hit hardware customer requirement analysis

Sensor Package Software The sensor package software can handle the six sensor modules provided by the kit, sends the data over the Internet with a secure transmission protocol and is open source. The overhead of the data sent over the Internet is not minimized as the device uses the TCP Protocol to transmit the data from the sensor gateway to the central platform *(see table 6.2)*.

Requirement	
Open-source or at least readily available	OK
Handle multiple sensors	OK
Send sensor data over the Internet	OK
Secure transmission protocol for receiving/sending sensor data	OK
Minimize the overhead of data sent over the Internet	X

Table 6.2.: Wunderbar IoT kit software customer requirement analysis

First Prototype Test Run A test run for this prototype was not possible as the modified Wunderbar firmware wasn't successfully flashed onto the Wunderbar master module.

Summary This prototype seemed at first very promising with only minor modifications needing to be done but with the failure to reprogram the master module it had to be abandoned.

6.3. Second Prototype Results

Sensor Package Hardware The final hardware setup used for this prototype consists of the Wunderbar sensor modules, a nRF51822 Eval Kit from Waveshare [Wav16] and an esp8266 both connected via UART to an Arduino M0-pro *(see figure 6.3)*. The prototype hardware therefore has full wireless data transmission. The Arduino microcontroller is open source and the WiFi and BLE modules are readily available. The WiFi and BLE module are powered over the Arduino M0-pro 3.3V output.

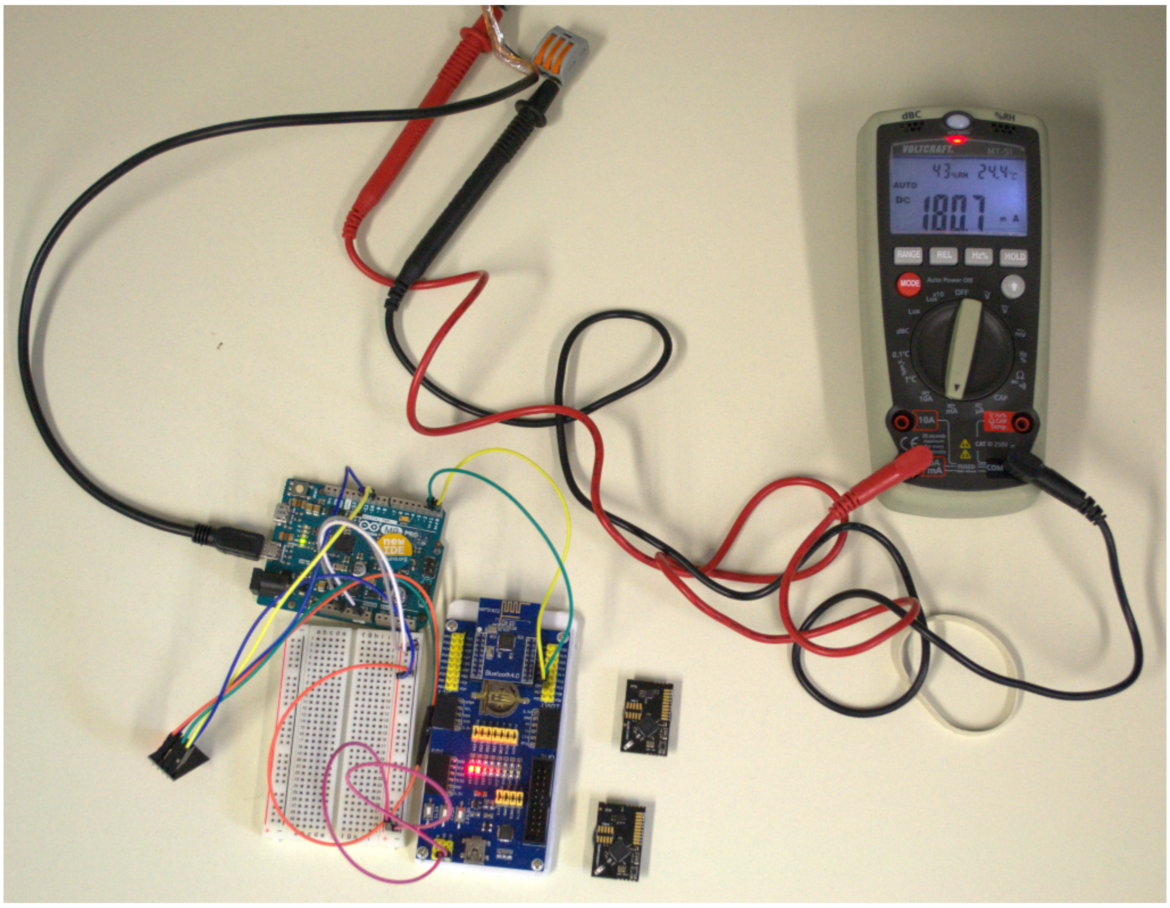


Figure 6.3.: Second prototype hardware

The power consumption of the Arduino zero was measured using the MT-51 multitester from "Voltcraft". A USB cable was cut open and the power consumption of the device was measured on the +5V line. The power consumption of the setup was measured in miliampere (mA) with the device running an endless loop, and while running the application. Using a standard 20000mAh USB powerbank would translate to a battery life of approximately 111 hours under full load *(see table 6.3).*

Power consumption	mA
Endless loop	180.2 +-0.2
Second prototype implementation	126 - 170

Table 6.3.: Second prototype hardware power consumption

The sensor package hardware therefore satisfies all customer requirements *(see table 6.4).*

Requirement	
Open-source or at least readily available	OK
Wireless data transmission	OK
Powered by a small battery	OK

Table 6.4.: Second prototype hardware customer requirement analysis

Sensor Package Software The software running on the sensor gateway is open source and the nordic SDK is readily available. The sensor gateway can handle multiple sensors and sends data over the Internet using the UDP protocol minimizing the overhead sent. The sensor gateway software in this prototype does not support a secure transmission protocol *(see table 6.5)*.

Requirement	
Open-source or at least readily available	OK
Handle multiple sensors	OK
Send sensor data over the Internet	OK
Secure transmission protocol for receiving/sending sensor data	X
Minimize the overhead of sensor data sent over the Internet	OK

Table 6.5.: Second prototype sensor package software customer requirement analysis

Central Platform Software All of the software used on the central platform is open source with the data being accessible from a supercomputer due to the mysql database being able to be on a DSS filesystem on the LRZ Compute Cloud. The Thingspeak platform can collect data from multiple sensors with each channel being able to hold eight data points and from multiple gateways with multiple channels being possible. The data is displayed in a timeseries graph which allows trends and outliers to be spotted easily on a per sensor basis. With the user interface being accessible through a browser it allows multiple clients to access the data and the platform has a user management system. Access to the data is secure with the use of HTTPS *(see table 6.6)*.

Requirement	
Open-source or readily available	OK
Multi user/client access to the sensor data	OK
Collect data from multiple sensor sources.	OK
Data accessible from a supercomputer	OK
Secure transmission protocol for user access to sensor data	OK
Display sensor data in a meaningful manner	OK

Table 6.6.: Second prototype central platform customer requirement analysis

Summary This prototype meets all requirements except for the secure transmission from the sensor package to the central platform. Since the MQTT-SN provides no built-in communication security mechanism, the MQTT-SN packet would need to be encapsulated within a secure transport layer. This could be done with the DTLS protocol as it provides communication privacy for datagram protocols.

The RIOT operating system supports the Tinydtls DTLS v1.2 implementation as a package, but the emcute MQTT-SN client doesn't have builtin DTLS support. Research for an MQTT-SN gateway with DTLS support didn't return any results at the time of writing. Therefore to use a message transport protocol, DTLS support would need to be built in to both the MQTT-SN client and server implementations.

Second Prototype Test Run Environmental data collected by the sensor modules was successfully sent to the central platform.

6.4. Third Prototype Results

Sensor Package Hardware The hardware hasn't changed from the previous prototype except for the replacement of the Arduino M0-pro with a Arduino due *(see figure 6.4)*.

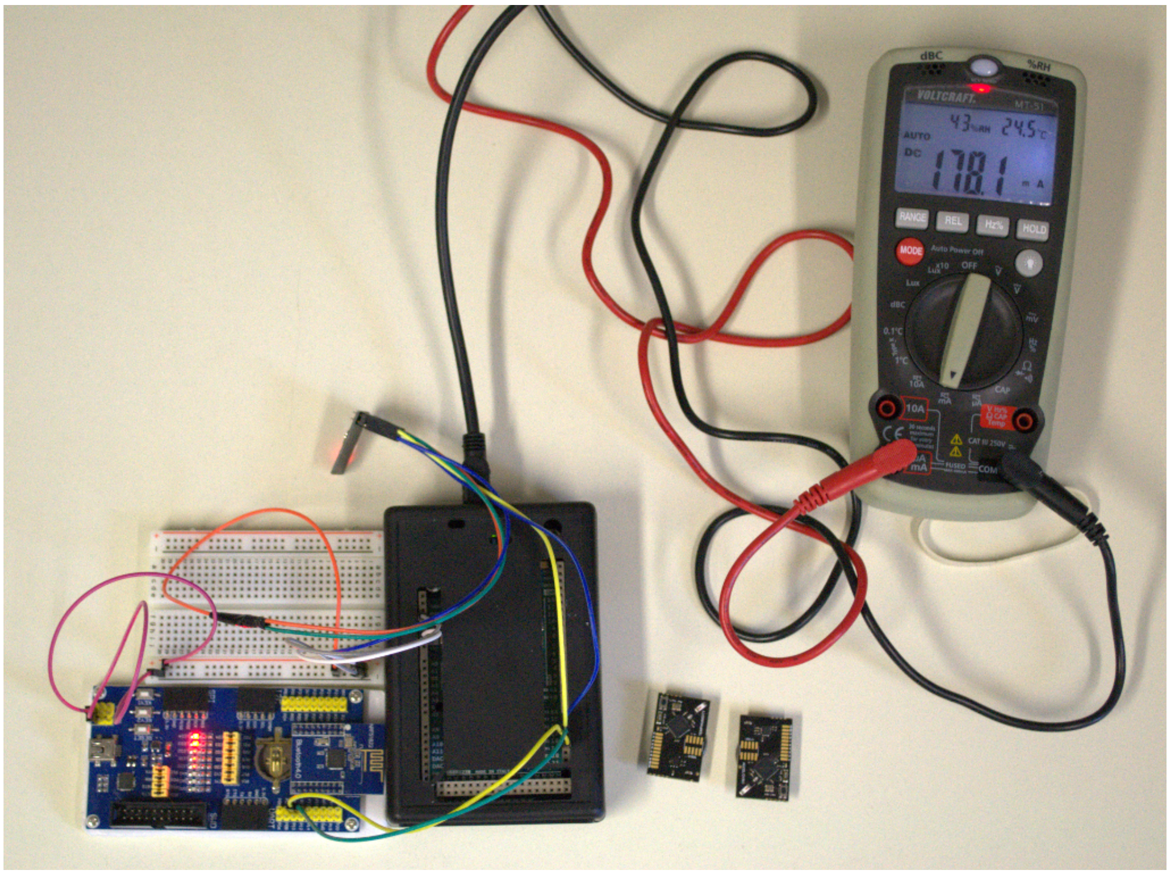


Figure 6.4.: Third prototype hardware

The power consumption of this sensor package was measured in the same way as for the second prototype. Using a standard 20000mAh USB powerbank would translate to a battery life of 109 hours under full load *(see table 6.7)*.

Power consumption	mA
Endless loop	183.5 +-0.2
Third prototype implementation	134 - 185

Table 6.7.: Third prototype hardware power consumption

This setup still satisfies all customer requirements as the Arduino due still is a constrained low power device with the third prototype sensor package only consuming a few more mA than the second prototype *(see table 6.8)*.

Requirement	
Open-source or at least readily available	OK
Wireless data transmission	OK
Powered by a small battery	OK

Table 6.8.: Third prototype hardware customer requirement analysis

Sensor Package Software Sending the sensor data directly within the DTLS payload satisfies the customer requirement of using a secure transmission protocol. All the other requirements are still met with the only change being the extra MAC address of the sensor gateway within the sensor packet *(see table 6.9)*.

Requirement	
Open-source or at least readily available	OK
Handle multiple sensors	OK
Send sensor data over the Internet	OK
Secure transmission protocol for receiving/sending sensor data	OK
Minimize the overhead of sensor data sent over the Internet	OK

Table 6.9.: Third prototype sensor package software customer requirement analysis

Central Platform Software The central platform software has remained the same except for replacing the MQTT connector and MQTT-SN gateway with the dtls-server. The DTLS server is open source and allows to distinguish data from multiple sensor gateways due to the addition of the MAC address of the esp8266 to the sensor data packet *(see table 6.10)*.

Summary This prototype sends data from wireless sensor modules into the cloud and displays it in a meaningful manner, and by using the DTLS protocol sends the data securely to the central platform. This implementation satisfies all customer requirements.

Third Prototype Test Run Environmental data collected by the sensor modules was successfully sent in a secure manner over the Internet to the central platform and displayed in a graph within the Thingspeak platform *(see figure 6.5)*.

The DTLS handshake can take more than one try to complete successfully due to decryption or packet sequence errors. When the handshake fails, the client clears the DTLS

Requirement	
Open-source or readily available	OK
Multi user/client access to the sensor data	OK
Collect data from multiple sensor sources.	OK
Data accessible from a supercomputer	OK
Secure transmission protocol for user access to sensor data	OK
Display sensor data in a meaningful manner	OK

Table 6.10.: Third prototype central platform customer requirement analysis

session and attempts to connect again. This involves waiting for timeouts from the keepalive mechanism described in Section 5.3.2 and can therefore be time intensive.

Long Test Run This prototype was left to run for a longer period of time and kept sending data to the central platform for approximately four and a half days amounting to thousands of datapoints. At this point the central platform VM ran out of memory with the Thingspeak ruby environment using up most of it. Investigations found that the VMs 2GB of memory was insufficient for the high resolution graphs configured for the temperature and humidity fields. Out of the box Thingspeak has a hard limit of 8000 datapoints per graph but this limit was increased for the purpose of showing multi-day temperature and humidity graphs. As the sensor gateway in this test setup sends data at intervals between one and five seconds, the datapoints required for a one day graph can be up to 86400.

To allow for these specific graphs, the VM was shutdown and its available RAM increased to four gigabytes. The maximum amount of datapoints per graph should be adjusted to the allocated memory of the VM or vice versa to not run out of memory on the server running Thingspeak.

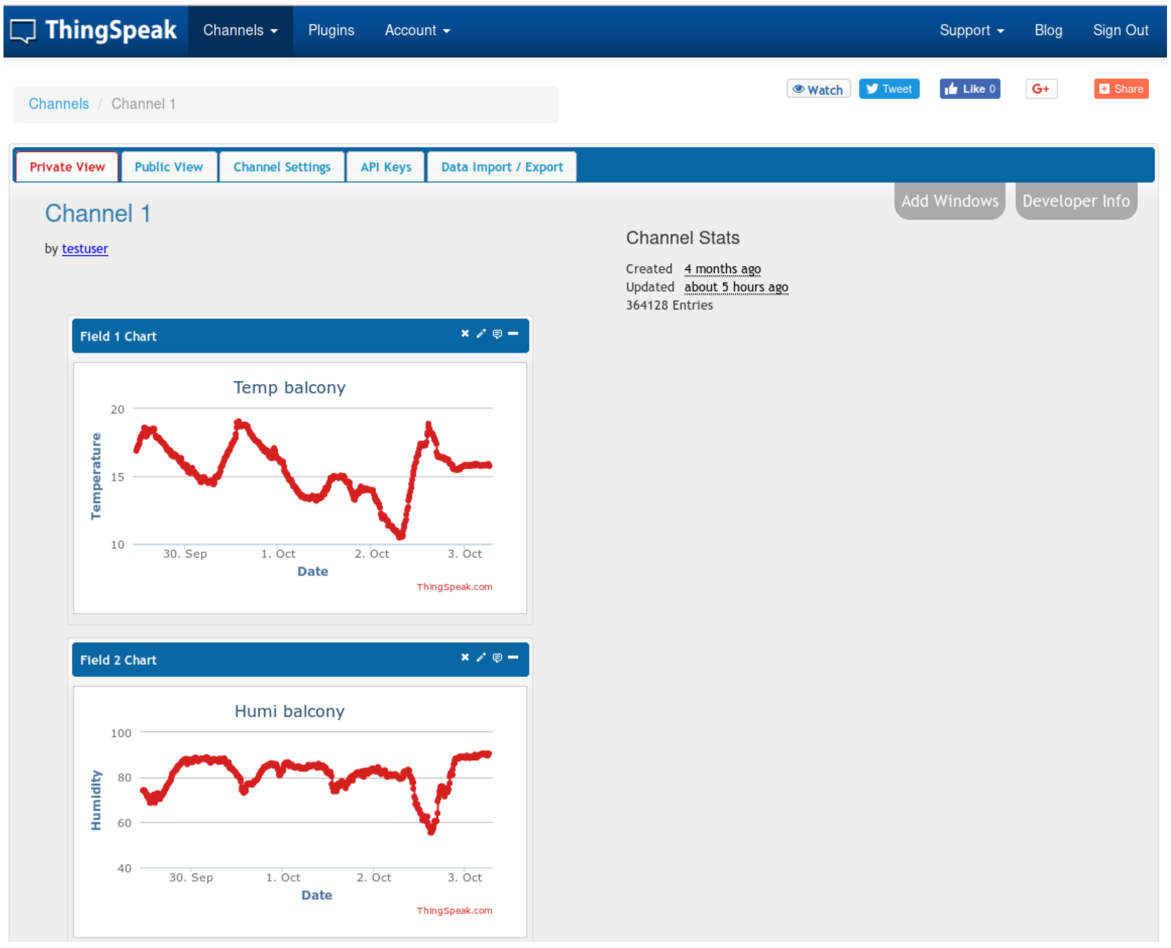
Issues Identified Two issues were identified during the long term test run:

- The esp8266 WiFi module used is problematic as the received UDP packets need to be parsed from the UART buffer on the sensor gateway. The UART buffer is implemented as a ringbuffer were all the data received over UART is stored. The receive function of the driver pops data from the ringbuffer until a string matches the AT command signaling that a packet was received *(see listing 6.1)*. It is possible that the UART buffer fills up rapidly due to very fast incoming packets, or just from interference on the UART interface. Such interferences causes the RIOT UART driver to write 0xff in the UART buffer, and therefore may cause the device overwriting valid received data.

```
        +IPD,<len>:<data>
```

  Listing 6.1: esp8266 AT command signaling received data

  This can cause the DTLS handshake to take more than one try to complete successfully due to decryption or packet sequence errors. When the handshake fails, the client clears the DTLS session and attempts to connect again. This involves waiting for timeouts from the keepalive mechanism described in Section 5.3.2 and can therefore be time intensive.



Channel Stats

Created 4 months ago
Updated about 5 hours ago
364128 Entries

Figure 6.5.: Thingspeak sensor data

In this setup the keepalive interval was set to 30 seconds and the keepalive timeout to three times the keepalive interval. This interval can be reduced, with the disadvantage of increased network overhead and computation time spent sending/receiving and encrypting/decrypting keepalive packets. The advantage of a shorter interval is that a disconnection from the DTLS server can be noticed sooner, amounting to a shorter time until a reconnection attempt.

- The initialization of many DTLS sessions over time may cause the sensor gateway to become unresponsive. The TinyDTLS library uses dynamic memory for its data structures which goes against the RIOT programming best practices [Hah16]. The RIOT implementation of the free() function doesn't free up the heap memory allocated which leads to the device at some point running out of memory. At the time of writing work is being done to fix this upstream issue in the TinyDTLS package[2].

[2]https://github.com/RIOT-OS/RIOT/pull/7615

6.5. Summary

In this chapter the three prototypes were evaluated against the customer requirements described in Chapter 3. The first prototype could not be tested as the customized Wunderbar firmware could not be successfully programed onto the Wunderbar master module. The second prototype was tested and showed that the sensor data could be sent from the sensor modules to the central platform and stored in a database. The central platform application Thingspeak provided the functions that were required by the customer such as multi-user access and displaying the sensor data in a meaningful way. The third prototype using the DTLS secure transmission protocol was evaluated and found to successfully implement all of the customer requirements. A long term test run was made with the third prototype and found to run stably through multiple reconnects.

7. Conclusion

The goal of this thesis was to implement the customer concept of ”sending data from remote wireless sensors to a central platform in a secure manner”. This concept was gathered during interviews with multiple use cases found to be fueling the interest to develop the idea. User stories were built from the use cases and were analyzed to form a set of functional and non functional requirements. A system was designed from the requirements derived in Chapter 3 that satisfy all customer requirements.

The next step was to build a prototype that implements the system design. The Wunderbar IoT kit was investigated as relayr had already implemented the customer concept with that hardware. The open source code was studied to find out how the system works and what would need to be changed to use the hardware with a different central platform and a secure certificate. A new firmware for the Wunderbar master module was compiled, but programming the module successfully wasn't achieved.

With the Wunderbar master module falling out of the race, a new device had to be found to replace it. This meant having to assemble a constrained device with BLE and WiFi connectivity. Finding BLE hardware that was fit for the purpose proved to be a challenge with devices having alternate firmware such as the first BLE device tested in Chapter 5. The final hardware of the sensor gateway was composed of the nRF51822 BLE and esp8266 WiFi modules connected to an Arduino board. The BLE module was programmed to allow the Wunderbar BLE sensors to connect to it and send the sensor data over UART. The Arduino was programed to receive sensor data from the BLE module, package it in MQTT-SN packets and send it over the Internet to a central platform using the WiFi module. Sending the data over the Internet was first done using an MQTT-SN client.

The central platform was built around the open source IoT middleware Thingspeak which allowed multi-user access to the sensor data with a web based GUI, and a REST API for uploading data to it. Due to Thingspeak API only using REST, a connector was used to forward the data received by the MQTT-SN gateway to the REST API.

The implementation using the MQTT-SN protocol to transmit data from the sensor to the central platform worked, but had no security mechanisms. Securing the link between the sensor gateway and the central platform was a task in itself. Although the DTLS library was included in the RIOT OS, the COaP and the MQTT-SN clients have no support for a secure transmission protocol at the time of writing this thesis. As the use of a messaging protocol was not required to fulfill the customer requirements, it was dropped in favor of directly sending sensor data within the DTLS payload. On the central platform this allowed cutting the MQTT/HTTPS connector and sending the Thingspeak REST API calls within the DTLS server application.

The prototypes were evaluated against the requirements defined in Chapter 3 and the power consumption of the working prototypes was measured. The first prototype fulfilled all requirements except minimizing the overhead of the data sent over the Internet. Without being able to flash firmware onto the Wunderbar master module, it failed to be a working prototype. The second prototype with the Arduino based sensor gateway satisfied all customer requirements except the secure transmission of data over the Internet. The third prototype implemented the system design successfully, showing that it is possible to design and implement an end to end solution for sending data from remote wireless sensors securely over the Internet to a central platform with the ability to process the data by a supercomputer.

Part of the implementation developed for this thesis is already in productive use. Sonja Teschemacher, an environmental engineer at the Technical University of Munich (TUM) uses the Thingspeak central platform and MQTT2HTTP connector as an endpoint for meteorological and hydrological data data gathered in the Glonn catchment [TUM17]. Further developments are being made such as an improved graphing functionality *(see figure 7.1)* and the possibility to replace certain Raspberry Pi devices with Arduino hardware and software.

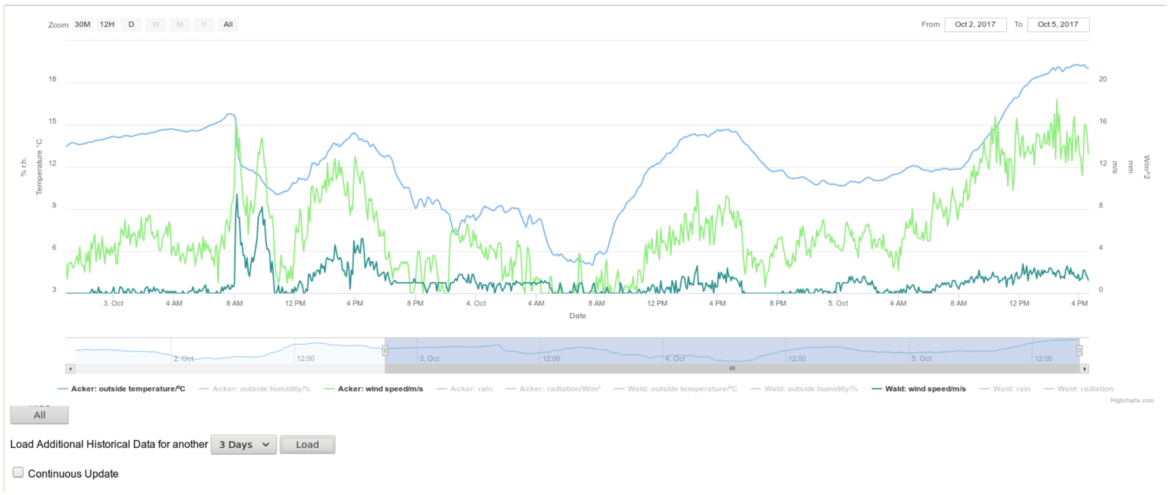


Figure 7.1.: Multi variable sensor data graph: Source[1]

[1] `http://tunnel.dyn.mwn.de/~heller/`

Future Work While the third prototype implementation of the customer concept satisfies all the requirements, further improvements would need to be done for productive use.

The esp8266 WiFi driver/firmware used is not entirely satisfactory due to the limitations of having to parse received packets within the UART ringbuffer. This method caused unforeseen deadlocks due to noise on the UART Interface or problems with data being overwritten in the ringbuffer. The deadlocks were resolved by implementing timeouts, but the loss of data issue still remains. Developing a new driver/firmware for the WiFi module that works directly with the RIOT GNRC would be of interest.

In this thesis, the focus was put on securing the data sent over the Internet. Albeit the attack surface of the BLE link between the wireless sensors and the sensor gateway is smaller due to the short range and ad hoc connection style of BLE, the BLE packets could be passively collected by an appropriate device within range. BLE provides a security mechanism in the form of pairing, which enables devices to exchange encryption keys to secure the connection between devices [Gro16]. This would have to be implemented on both the wireless sensors and the BLE module of the sensor gateway, for the BLE connection to be secure.

The Thingspeak IoT platform could be further customized to allow for more sensor fields or different types of graphs. Collecting more information from sensor gateways such as battery levels could also be of interest to not miss out on readings due to power outages.

Another feature that would be required is the management of the sensor gateways and modules. The third implementation provides the framework for differentiating the devices by the means of MAC addresses, but the Thingspeak write API key and field names for the REST API request are set within the DTLS server program. Adding additional sensor gateways to the setup requires defining MAC_address/write_API_key pairs within a function that returns a write API key on the DTLS server. Additional sensor types require defining new REST API request strings also in the DTLS server source code. A database assigning write API keys to sensor gateway MAC addresses, and sensor module MAC addresses to fields could be setup and queried dynamically by the DTLS server to create the HTTPS request sent to the Thingspeak API. At first this could be done manually by adding MAC addresses to the database. This could be improved by collecting unknown MAC addresses sending data to the DTLS server into a separate table for further assignment to channels and fields.

A. Software Documentation

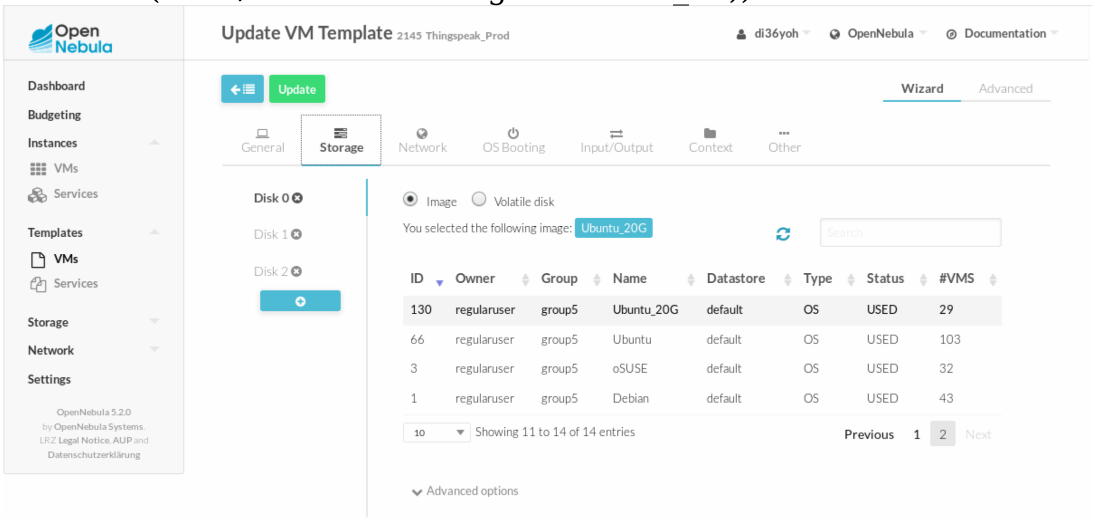
This appendix contains installation, configuration and programming documentation compiled on the project wiki. Appendix A.1 documents the installation process of a VM on the LRZ Compute Cloud with Thingspeak. Appendix A.2 describes how to program the Nordic nRF51822 MCU using an ST-Link programmer and Appendix A.3 provides a detailed documentation of how the MQTT2HTTP connector was configured to forward messages received from the MQTT-SN client on to the Thingspeak REST API.

A.1. ThingSpeak Installation

Thingspeak installation on the LRZ Compute Cloud

Creating the VM

- Log in to the LRZ OpenNebula https://www.cloud.mwn.de/
- Under Templates, clone the Ubuntu Template
- Update the Ubuntu template to contain as storage image disk0 the "Ubuntu_20G" image (At the time of this writing it is Ubuntu 16.04 LTS (GNU/Linux 4.4.0-21-generic x86_64)):



Update VM Template 2145 Thingspeak_Prod

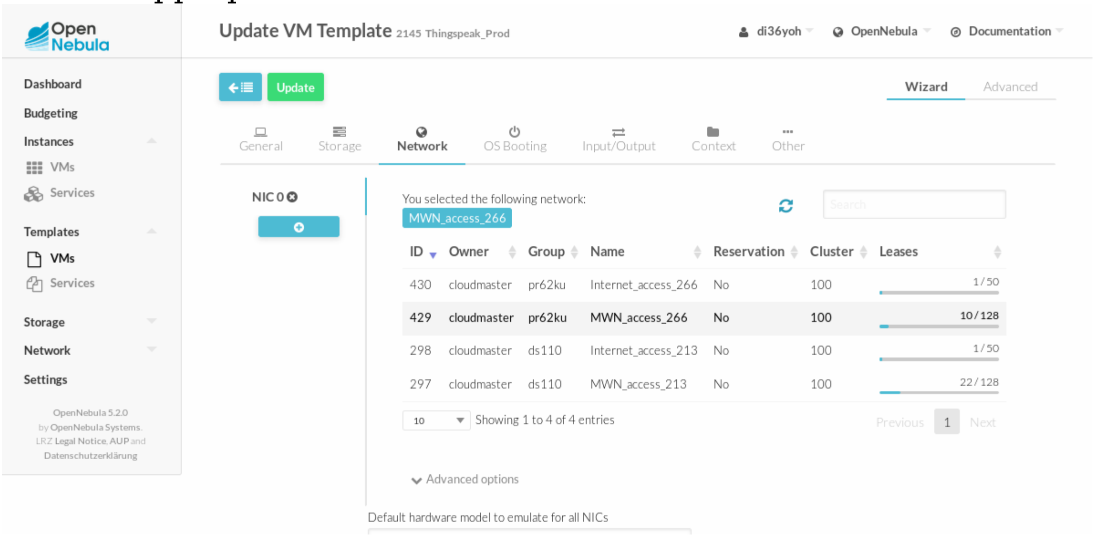
● Image ○ Volatile disk

You selected the following image: Ubuntu_20G

ID	Owner	Group	Name	Datastore	Type	Status	#VMS
130	regularuser	group5	Ubuntu_20G	default	OS	USED	29
66	regularuser	group5	Ubuntu	default	OS	USED	103
3	regularuser	group5	oSUSE	default	OS	USED	32
1	regularuser	group5	Debian	default	OS	USED	43

10 Showing 11 to 14 of 14 entries

- Select an appropriate network interface. In our case MWN:



Update VM Template 2145 Thingspeak_Prod

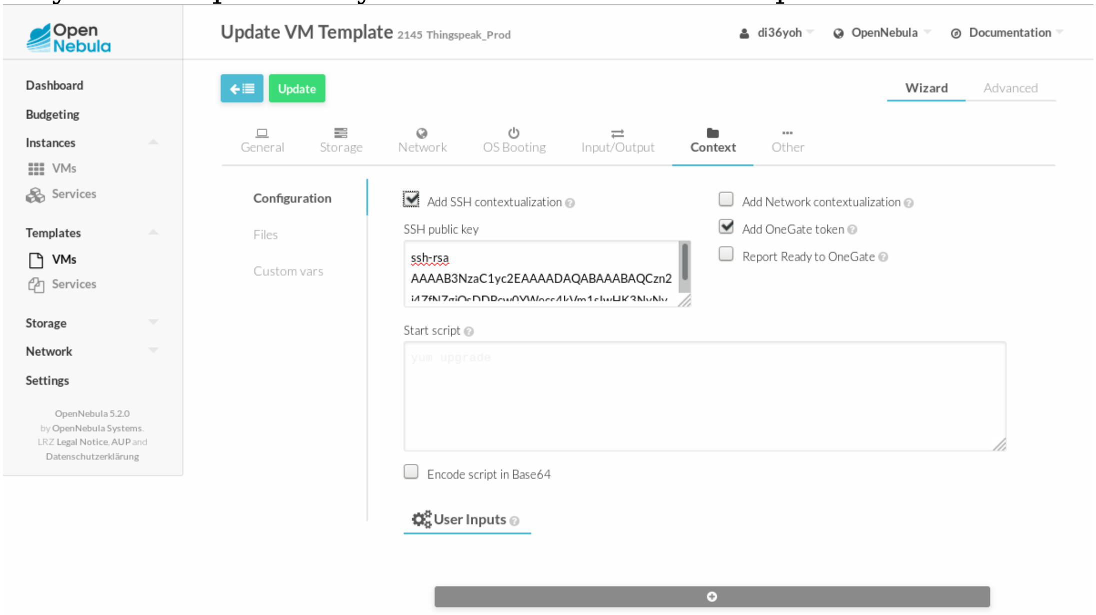
You selected the following network: MWN_access_266

ID	Owner	Group	Name	Reservation	Cluster	Leases
430	cloudmaster	pr62ku	Internet_access_266	No	100	1 / 50
429	cloudmaster	pr62ku	MWN_access_266	No	100	10 / 128
298	cloudmaster	ds110	Internet_access_213	No	100	1 / 50
297	cloudmaster	ds110	MWN_access_213	No	100	22 / 128

10 Showing 1 to 4 of 4 entries

Default hardware model to emulate for all NICs

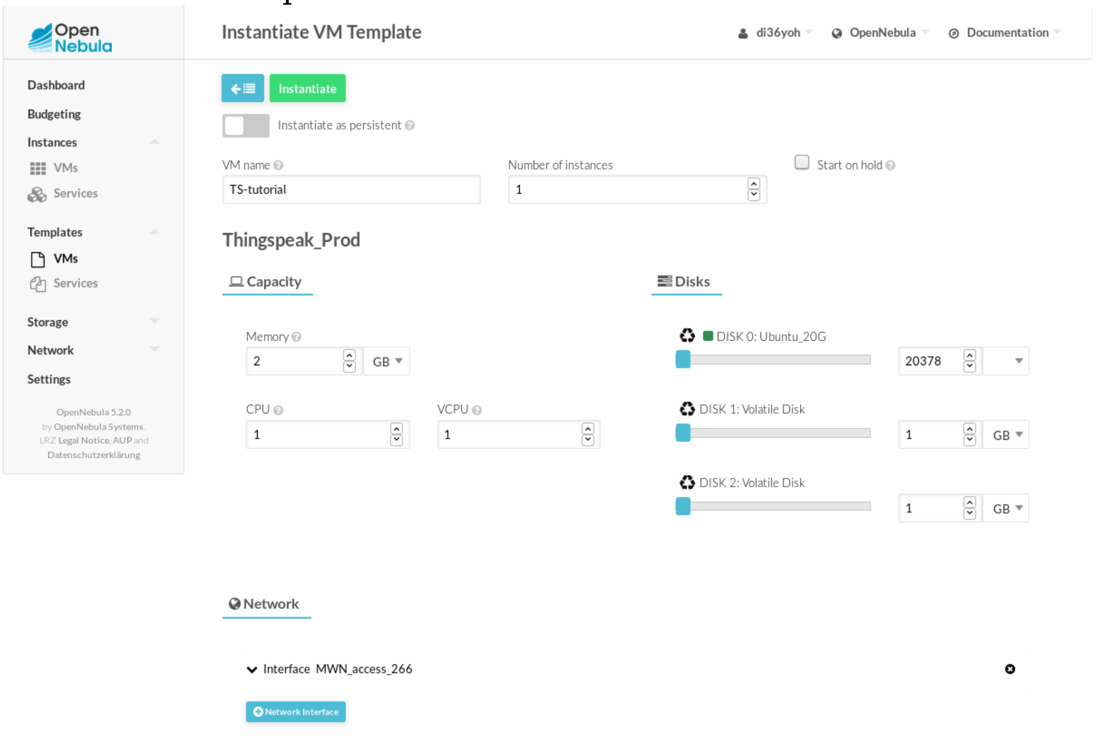
- Add your SSH public key to the context of the template:



Update VM Template 2145 Thingspeak_Prod

[✓] Add SSH contextualization
SSH public key:
```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCzn2
```
[ ] Add Network contextualization
[✓] Add OneGate token
[ ] Report Ready to OneGate
Start script: yum upgrade
[ ] Encode script in Base64

- Instantiate the template:



Instantiate VM Template

Instantiate as persistent
VM name: TS-tutorial
Number of instances: 1
[ ] Start on hold

Thingspeak_Prod

Memory: 2 GB
CPU: 1
VCPU: 1
DISK 0: Ubuntu_20G — 20378
DISK 1: Volatile Disk — 1 GB
DISK 2: Volatile Disk — 1 GB
Interface MWN_access_266

First steps in the VM

- Make note of the IP Address of the VM under the instances tab and ssh to it

- Make sure to change the root password and follow the security consideration under https://www.lrz.de/services/compute/cloud_en/security/
- Create a new user that will be installing/running the software

  ```
  adduser USERNAME
  ```

- update the /etc/sudoers file to allow the new user to install packages

  ```
  USERNAME    ALL=(ALL:ALL) ALL
  ```

- Log out the root user and relog in with the new user.

Installing Thingspeak

- Make sure Ubuntu is up to date

  ```
  sudo apt-get update
  sudo apt-get upgrade
  sudo apt-get dist-upgrade
  sudo apt-get autoremove
  sudo apt-get clean
  ```

- install some packages to build the application

  ```
  sudo apt-get -y install build-essential g++ mysql-server
  mysql-client libmysqlclient-dev libxml2-dev libxslt-dev git-
  core curl rubygems
  ```

- Install Ruby Environment

  ```
  gpg --keyserver hkp://keys.gnupg.net --recv-keys D39DC0E3
  curl -sSL https://get.rvm.io | bash -s stable
  source ~/.rvm/scripts/rvm
  rvm install 2.1.0
  rvm --default use 2.1.0
  ```

- Install Thingspeak

  ```
  git clone https://github.com/iobridge/thingspeak.git
  cd thingspeak
  gem install bundler -v 1.14.6
  bundle install
  ```

- Move database configuration and define the username and password for the databases within the config/database.yml file

  ```
  cp config/database.yml.example config/database.yml
  ```

- Edit the Gemfile in the Thingspeak root directory to use a newer version of the mysql2 gem

```
gem 'mysql2', '~> 0.3.18'
```

- Update the mysql2 gem

```
bundle update mysql2
```

- Create the mysql databases with the correct permissions for the USERNAME and PASSWORD specified in the config/database.yml file:

```
mysql -u root -p
CREATE DATABASE thingspeak_test DEFAULT CHARACTER SET utf8
COLLATE utf8_unicode_ci;
CREATE DATABASE thingspeak_development DEFAULT CHARACTER SET
utf8 COLLATE utf8_unicode_ci;
CREATE DATABASE thingspeak_production DEFAULT CHARACTER SET
utf8 COLLATE utf8_unicode_ci;
GRANT ALL PRIVILEGES ON thingspeak_test.* TO
USERNAME@localhost IDENTIFIED BY 'PASSWORD';
GRANT ALL PRIVILEGES ON thingspeak_development.* TO
USERNAME@localhost;
GRANT ALL PRIVILEGES ON thingspeak_production.* TO
USERNAME@localhost;
FLUSH PRIVILEGES;
exit;
mysqladmin -u root -p reload
```

- Create the databases for Thingspeak

```
rake db:create
```

- MySQL 5.7 doesn't allow NULL Key Rails Github. Create config/initializers/abstract_mysql2_adapter.rb file with:

```
class ActiveRecord::ConnectionAdapters::Mysql2Adapter
  NATIVE_DATABASE_TYPES[:primary_key] = "int(11)
auto_increment PRIMARY KEY"
end
```

- Add to the **end** of config/environment.db

```
require
File.expand_path('../../config/initializers/abstract_mysql2_a
dapter', __FILE__)
```

- Load the Thingspeak database schema

```
rake db:schema:load
```

Setup iptables firewall to accept incoming traffic to the Thingspeak webserver

- Add to the /etc/iptables/rules.v4 file **before** COMMIT:

  ```
  -A INPUT -p tcp -m tcp --dport 3000 -m state --state NEW -j
  ACCEPT
  ```

- Reload the firewall rules

  ```
  sudo iptables-restore < /etc/iptables/rules.v4
  ```

Create an SSL Certificate

- If you already have an SSL certificate you may skip creating the certificate.

  ```
  cd thingspeak
  mkdir certs
  cd cert
  openssl req -new > new.ssl.csr
  openssl rsa -in privkey.pem -out new.cert.key
  openssl x509 -in new.cert.csr -out server.crt -req -signkey
  server.key -days 730
  ```

Add SSL to webrick

- replace *bin/rails* script within the Thingspeak folder with following code. If you already have an SSL certificate, adjust the paths in *:SSLPrivateKey* and *:SSLCertificate*.

```
#!/usr/bin/env ruby

require 'rails/commands/server'
require 'rack'
require 'webrick'
require 'webrick/https'

if ENV['SSL'] == "true"
  module Rails
      class Server < ::Rack::Server
          def default_options
              super.merge({
                  :Port => 3000,
                  :environment => (ENV['RAILS_ENV'] ||
"development").dup,
                  :daemonize => false,
                  :debugger => false,
                  :pid =>
File.expand_path("tmp/pids/server.pid"),
                  :config => File.expand_path("config.ru"),
                  :SSLEnable => true,
```

```
                  :SSLVerifyClient => OpenSSL::SSL::VERIFY_NONE,
                  :SSLPrivateKey => OpenSSL::PKey::RSA.new(

File.open("certs/server.key").read),
                  :SSLCertificate =>
OpenSSL::X509::Certificate.new(

File.open("certs/server.crt").read),
                  :SSLCertName => [["CN",
WEBrick::Utils::getservername]],
              })
          end
      end
  end
end

APP_PATH = File.expand_path('../../config/application',
__FILE__)
require_relative '../config/boot'
require 'rails/commands'
```

Start the server

```
cd thingspeak
SSL=true rails server webrick -b `hostname --ip-address` -d
```

Resources

- https://www.lrz.de/services/compute/cloud_en/
- http://www.cnx-software.com/2016/12/07/how-to-install-thingspeak-in-ubuntu-16-04/
- https://diyprojects.io/install-thingspeak-ubuntu-16-04-lts/#.WU-PIkBpzmi
- https://stackoverflow.com/questions/3640993/how-do-you-configure-webrick-to-use-ssl-in-rails#23141638

A.2. Nordic nRF51822 Programming

Nordic nRF51822

Hardware used

- Waveshare nRF51822 eval kit
- STM32F412 Nucleo-144

Software used on windows 10-64bit

- Keil MDK V5.23.0.0 with legacy support for Cortex-M devices
- Nordic nRF51822 softdevice S120 v1.0.1
- openocd v0.10.0
- nRF51 SDK v5.2.0.39364

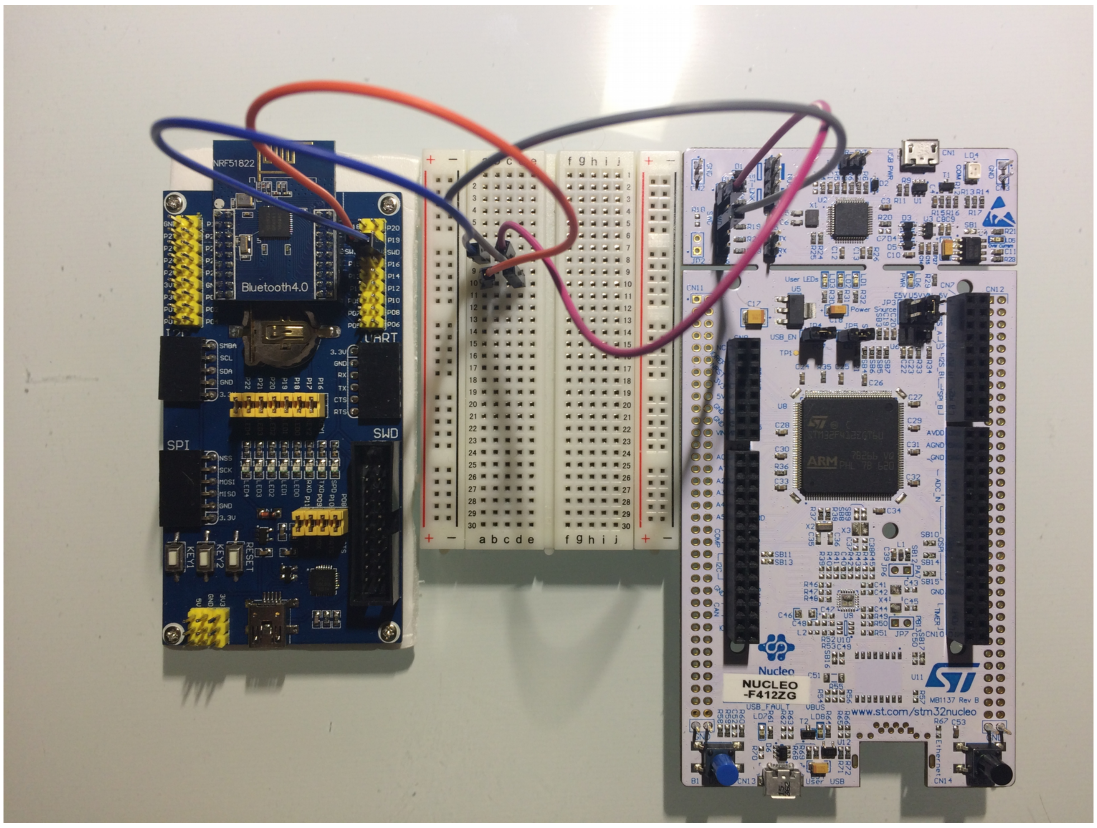
Keil installed packages

- Important to install softdevice S120 v1.0.1-3

Connect the BLE400 board to the ST-Link interface of the nucleo board

- Remove the two CN4 Jumpers from the Nucleo board
- Connect the SWDIO Pin from the BLE400 to pin4 of the CN6 connector

- Connect the SWD Pin from the BLE400 to pin2 of the CN6 connector



How-to flash device with s120 softdevice and application code

In the openocd folder add following to the beginning of the scripts51.cfg file. This slows down the flashing, but is less prone to errors.

```
set WORKAREASIZE 0
```

In the command line start the openocd server

```
.\bin-x64\openocd.exe -f scripts\interface\stlink-v2-1.cfg -f
scripts\target\nrf51.cfg -c "init"
```

In another console connect to the openocd instance

```
telnet 127.0.0.1 4444
```

in the terminal flash the device the softdevice and compiled application:

```
halt
nrf51 mass_erase
```

```
reset halt
program softdevice.hex
program app_s120.hex
reset
```

After flashing for the first time the nrf51822 chip you do not need to mass_erase and program the softdevice each time. Just halt the device, program the application and reset. ## Hints * Make sure the nordic softdevice version matches the one that Keil MDK is using (Check in the project code the version of the component). * make sure the nordic softdevice is supported by the nRF51822 version

Useful links

- https://devzone.nordicsemi.com/blogs/485/programming-nrf51-with-st-link-uvision/
- https://devzone.nordicsemi.com/question/70314/program-bluetooth-for-nrf51822-yunjia-board-with-stlink-v2/
- https://devzone.nordicsemi.com/question/50069/nrf51822-ev-board-ble400/
- http://www.rogerclark.net/arduino-on-the-nrf51822-bluetooth-low-energy-microcontroller/
- https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.s120.api.v2.1.0%2Fmodules.html

A.3. MQTT2HTTP Connector Installation

MQTT2HTTP

We use this application to connect the MQTT broker with the REST API of Thingspeak. ## Installation * Install nodejs (version >6.x!)

```
sudo apt-get install nodejs npm
```

To get a newer version, use

```
curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
sudo apt-get install -y nodejs
```

- Clone the repository

  ```
  git clone https://github.com/FokkeZB/mqtt2http.git
  ```

- Patch the application to send HTTP request to the IP address the Thingspeak server is listening on. If you are running Thingspeak on a different server than the MQTT broker or have multiple External interfaces, do not use this patch and make sure that the URL MQTT2HTTP is calling is correct .http_url.patch

```
diff -crB mqtt2http/index.js mqtt2http.new/index.js
*** mqtt2http/index.js  2017-07-06 15:04:27.810416573 +0200
--- mqtt2http.new/index.js  2017-07-06 15:07:38.026469074 +0200
***************
*** 1,3 ****
--- 1,4 ----
+ "use strict";
  require('dotenv').config({
    silent: true
  });
***************
*** 5,10 ****
--- 6,16 ----
  const mqtt = require('mqtt');
  const request = require('request');
  const Handlebars = require('handlebars');
+ const os = require('os');
+
+ // Define IP address and port to call
+ var ipAddress = os.networkInterfaces().eth0[0].address;
+ var port = 3000;

  Handlebars.registerHelper('stringify', object => {
    return new Handlebars.SafeString(JSON.stringify(object));
***************
*** 24,29 ****
--- 30,36 ----
```

```
  const MQTT_TOPIC = IS_JSON.test(process.env.MQTT_TOPIC) ?
JSON.parse(process.env.MQTT_TOPIC) : process.env.MQTT_TOPIC;
  const MQTT_SUBSCRIBE = process.env.MQTT_SUBSCRIBE ?
JSON.parse(process.env.MQTT_SUBSCRIBE) : undefined;

+ process.env.HTTP_URL = "http://" + ipAddress + ":" + port +
process.env.HTTP_URL;
  const HTTP_URL = process.env.HTTP_URL ?
Handlebars.compile(process.env.HTTP_URL) : undefined;
  const HTTP_JSON = !!process.env.HTTP_JSON;
  const HTTP_REQUEST = process.env.HTTP_REQUEST ?
Handlebars.compile(process.env.HTTP_REQUEST) : undefined;
```

```
patch -p0 -i path_to/http_url.patch
```

- Install node packages

  ```
  cd mqtt2http
  npm install
  ```

Configuration

- Create the configuration file

  ```
  touch .env
  ```

- Edit the .env file to contain. This will connect to the MQTT server on localhost and subscribes to the sensors topic.

  ```
  MQTT_URL=mqtt://127.0.0.1
  MQTT_TOPIC=sensors
  HTTP_URL=/update?
  key={{ message.channel }}&{{ message.field }}={{ message.data
  }}
  HTTP_JSON=1
  ```

- The HTTP_URL variable will extract the value of "channel", "field" and "data" and insert it into {message.channel}, {message.field} and {message.data}. For this configuration the MQTT-SN message from the sensor gateway needs to be in the format:

  ```
  {"channel": "foobar", "field": "foo", "data": "barfoo"}
  ```

- If you have a fixed IP, you do not have to patch the URL as described above. However, you can **not use the DNS name in the URL**! Apparently, mqtt2http does not do a DNS lookup. You must use the numerical IP! And the localhost/127.0.0.1 does **not** work, as Thingspeak is not listening on it! An example for a proper entry in the .env file would be:

```
HTTP_URL=http://141.40.254.22:3000/update?
key={{ message.channel }}&{{ message.field }}={{ message.data
}}
```

- On the sending side (Arduino, Raspberry, etc. The IoT thing in general) you have to compose your MQTT message like this:

  ```
  mosquitto_pub -h mqtt_broker_host.somewhere.de -p 1883 -t
  "sensors" -m '{"channel": "BLOS73UPTRHOU0L0", "field":
  "field1", "data": "20.5"}'
  ```

  "sensors" is the MQTT topic that you submit data to. It is just any name, but it has to match the topic name you specify in the .env file: MQTT_TOPIC=sensors
  The value for "channel" (here "BLOS73UPTRHOU0L0") is a Thingspeak API key. You get this key once you generate a channel on Thingspeak and then request the API key for it. If you have multiple channels, you can use multiple API keys.
  You can have up to 8 fields (called field1 to field8) in one channel.
  And finally you have the data for this item.

Start the service

To start the service run:

```
node index.js
```

Run the service

It can deal with an arbitrary number of fields in an MQTT message. You define the field names in the MQTT message. All you have to do is to expand the HTTP_URI that is then passed to Thingspeak. A more complete HTTP_URL would look like this:

```
HTTP_URL=http://141.40.254.140:3000/update?key={{ message.channel
}}&{{ message.field1 }}={{ message.data1 }}&{{ message.field2 }}=
{{ message.data2 }}&{{ message.field3 }}={{ message.data3 }}&{{ m
essage.field4 }}={{ message.data4 }}&{{ message.field5 }}={{ mess
age.data5 }}&{{ message.field6 }}={{ message.data6 }}&{{ message.
field7 }}={{ message.data7 }}&{{ message.field8 }}={{ message.dat
a8 }}&{{ message.field9 }}={{ message.data9 }}
```

The 9 fields allow also for passing of a time stamp (use "created_at" as value of the field name and an ISO8601 date as data value). You can select the field names arbitrarily, but you have to use the same field names in the MQTT message.

List of Figures

Bibliography

[ada17] ADA lady: *Using ATSAMD21 SERCOM for more SPI, I2C and Serial ports*. https://cdn-learn.adafruit.com/downloads/pdf/using-atsamd21-sercom-to-add-more-spi-i2c-serial-ports.pdf. Version: 2017. – accessed, 12. September 2017

[Agr17] AGRAWAL, Mohit: *INTERNET OF THINGS ? BUSINESS MODELS*. https://www.linkedin.com/pulse/internet-things-business-models-mohit-agrawal. Version: 2017. – accessed, 24. September 2017

[AIM10] ATZORI, Luigi ; IERA, Antonio ; MORABITO, Giacomo: The internet of things: A survey. In: *Computer networks* 54 (2010), Nr. 15, S. 2787–2805

[al.17] AL., Marc S.: *Announcing the first SHA1 collision*. https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html. Version: 2017. – accesed, 6. July 2017

[Ard17a] ARDUINO: *Arduino Due*. https://store.arduino.cc/arduino-due. Version: 2017. – accessed, 16. September 2017

[Ard17b] ARDUINO: *Arduino M0 Pro*. https://store.arduino.cc/arduino-m0-pro. Version: 2017. – accessed, 10. September 2017

[Bol16] BOLUTEK: *DX-BT05-A 4.0*. http://www.martyncurrey.com/?wpdmdl=3123. Version: 2016. – accessed, 10. September 2017

[BSMD11] BANDYOPADHYAY, Soma ; SENGUPTA, Munmun ; MAITI, Souvik ; DUTTA, Subhajit: Role of middleware for internet of things: A study. In: *International Journal of Computer Science and Engineering Survey* 2 (2011), Nr. 3, S. 94–105

[Car17] CARRIOTS: *Carriots - Internet of things platform*. https://carriots.com/. Version: 2017. – accessed, 21. September 2017

[Cur16] CURREY, Martyn: *Bluetooth Modules*. http://www.martyncurrey.com/bluetooth-modules/#AT-09. Version: 2016. – accessed, 10. September 2017

[Ecl16] ECLIPSE: *Mosquitto rsmb - github*. https://github.com/eclipse/mosquitto.rsmb. Version: 2016. – accessed, 13. September 2017

[Edu07] EDUCATION, IG: Virtualization in education. In: *IBM Corporation, Whitepaper* (2007)

[For12] FORCE, Internet Engineering T.: *Datagram Transport Layer Security Version 1.2*. https://tools.ietf.org/html/rfc6347. Version: 2012. – accessed, 14. September 2017

[Fou17a] Foundation, Eclipse: *Eclipse tinydtls*. https://projects.eclipse.org/projects/iot.tinydtls. Version: 2017. – accessed, 14. September 2017

[Fou17b] Foundation, Raspberry P.: *Raspberry Pi FAQs*. https://www.raspberrypi.org/help/faqs/#topPower. Version: 2017. – accessed, 25. September 2017

[Gro05] Group, Network W.: *Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)*. https://tools.ietf.org/html/rfc4279. Version: 2005. – accessed, 16. September 2017

[Gro16] Group, Bluetooth Special I.: *Security, Bluetooth Low Energy*. https://www.bluetooth.com/~/media/files/specification/bluetooth-low-energy-security.ashx?la=en. Version: 2016. – accessed, 06. October 2017

[Gro17] Group, Bluetooth Special I.: *GATT Overview*. https://www.bluetooth.com/specifications/gatt/generic-attributes-overview. Version: 2017. – accessed, 09. September 2017

[Hac17] Hackster.io: *Wunderbar's Community Hub*. https://www.hackster.io/wunderbar. Version: 2017. – accessed, 28. September 2017

[Hah16] Hahm, Oleg: *Best Practice for RIOT Programming*. https://github.com/RIOT-OS/RIOT/wiki/Best-Practice-for-RIOT-Programming. Version: 2016. – accessed, 26. September 2017

[Hei17] Heider, Tobias: *Minimal GIKEv2 implementation for RIOT OS*. http://www.nm.ifi.lmu.de/pub/Fopras/heid17/PDF-Version/heid17.pdf. Version: 2017. – accessed, 10. September 2017

[HM06] Hart, Jane K. ; Martinez, Kirk: Environmental sensor networks: A revolution in the earth system science? In: *Earth-Science Reviews* 78 (2006), Nr. 3, S. 177–191

[(IE14] (IETF), Internet Engineering Task F.: *The Constrained Application Protocol (CoAP)*. https://tools.ietf.org/html/rfc7252. Version: 2014. – accessed, 24. September 2017

[Inc15] Incorporated, Texas I.: *CC2540 and CC2541 Bluetooth low energy Software Developer's Reference Guide*. http://www.ti.com/lit/pdf/swru271. Version: 2015. – accessed, 10. September 2017

[Inc17] Inc., Espressif: *ESP8266 AT Instruction Set*. https://espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf. Version: 2017. – accessed, 10. September 2017

[iob14] iobridge: *Thingspeak github*. https://github.com/iobridge/ThingSpeak. Version: 2014. – accessed, 13. September 2017

[jso17] json.org: *Introducing JSON*. http://www.json.org/. Version: 2017. – accessed, 13. September 2017

[Kop11] Kopetz, Hermann: Internet of things. In: *Real-time systems*. Springer, 2011, S. 307–323

[Kre16] KrebsOnSecurity: *KrebsOnSecurity Hit With Record DDoS.* `https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/`. Version: 2016. – accessed, 22. September 2017

[Lab12] Labs, Auto-ID: *Auto-ID Labs.* `http://www.autoidlabs.org/`. Version: 2012. – accesed, 12. July 2017

[Lib17a] Libelium: *The IoT Marketplace.* `https://www.the-iot-marketplace.com/`. Version: 2017. – accessed, 21. September 2017

[Lib17b] Libelium: *Libelium - Connecting sensors to the cloud.* `http://www.libelium.com/`. Version: 2017. – accessed, 21. September 2017

[LR16] Leibniz-Rechenzentrum: *LRZ: Cloud Manual.* `https://www.lrz.de/services/compute/cloud_en/cloud-manual_en/`. Version: 2016. – accessed, 10. September 2017

[LRZ17a] LRZ: *LRZ: About Big Data Storage.* `https://www.lrz.de/services/datenhaltung/bigdata/aboutbigdata/`. Version: 2017. – accessed, 23. September 2017

[LRZ17b] LRZ: *LRZ: Leibnitz Supercomputing Center.* `https://www.lrz.de/english/`. Version: 2017. – accessed, 23. September 2017

[Ltd17] Ltd, Canonical: *Ubuntu release end of life.* `https://www.ubuntu.com/info/release-end-of-life`. Version: 2017. – accessed, 13. September 2017

[Lue14] Lueth, Knud L.: *IoT Market ? Forecasts at a glance.* `https://iot-analytics.com/iot-market-forecasts-overview/`. Version: 2014. – accessed, 24. September 2017

[Mal16] MalwareMustDie!: *Linux/Mirai, how an old ELF malcode is recycled..* `http://blog.malwaremustdie.org/2016/08/mmd-0056-2016-linuxmirai-just.html`. Version: 2016. – accessed, 22. September 2017

[MMST16] Mineraud, Julien ; Mazhelis, Oleksiy ; Su, Xiang ; Tarkoma, Sasu: A gap analysis of Internet-of-Things platforms. In: *Computer Communications* 89 (2016), S. 5–16

[NXP17] NXP: *Kinetis® Design Studio Integrated Development Environment (IDE).* `http://www.nxp.com/products/wireless-connectivity/bluetooth-low-energy-ble/kinetis-design-studio-integrated-development-environment-ide:KDS_IDE`. Version: 2017. – accessed, 09. September 2017

[Ope14] Open, OASIS: *MQTT Version 3.1.1.* `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html`. Version: 2014. – accessed, 24. September 2017

[Pos80] Postel, J.: *User Datagram Protocol.* http://www.rfc-base.org/txt/rfc-768.txt. Version: 1980. – accessed, 24. September 2017

[rel14] relayr: *WunderBar by relayr.* https://www.dragoninnovation.com/projects/35-wunderbar-by-relayr. Version: 2014. – accessed, 17. September 2017

[rel17a] relayr: *relayr · GitHub.* https://github.com/relayr/. Version: 2017. – accesed, 10. July 2017

[rel17b] relayr: *wunderbar-hardware · GitHub.* https://github.com/relayr/wunderbar-hardware/tree/master/Datasheets/Master_Module/. Version: 2017. – accesed, 10. July 2017

[rel17c] relayr: *The WunderBar has gone away, but we would like to offer a replacement.* https://campaign.relayr.io/wunderbar-eol-iot-sensor-kit-offer. Version: 2017. – accesed, 6. July 2017

[RO17a] RIOT-OS: *The friendly Operating System for the Internet of Things.* http://riot-os.org. Version: 2017. – accessed, 29. September 2017

[RO17b] RIOT-OS: *Generic (GNRC) network stack.* http://riot-os.org/api/group__net__gnrc.html. Version: 2017. – accessed, 12. September 2017

[RO17c] RIOT-OS: *Messaging / IPC.* http://riot-os.org/api/group__core__msg.html. Version: 2017. – accessed, 12. September 2017

[RO17d] RIOT-OS: *Packages.* https://www.riot-os.org/api/group__pkg.html. Version: 2017. – accessed, 14. September 2017

[SC81] Southern California, Information Sciences Institute U.: *RFC 793 - Transmission Control Protocol.* https://tools.ietf.org/html/rfc793. Version: 1981. – accessed, 24. September 2017

[SCT13] Stanford-Clark, Andy ; Truong, Hong L.: *MQTT For Sensor Networks (MQTT-SN).* http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf. Version: 2013. – accessed, 24. September 2017

[Sem13] Semiconductor, Nordic: *S110 SoftDevice Specification.* http://infocenter.nordicsemi.com/pdf/S110_SDS_v1.2.pdf. Version: 2013. – accessed, 07. September 2017

[Sem14a] Semiconductor, Nordic: *BLE Multi-link Example.* http://developer.nordicsemi.com/nRF51_SDK/nRF51_SDK_v7.x.x/doc/7.0.1/s120/html/a00045.html. Version: 2014. – accessed, 10. September 2017

[Sem14b] Semiconductor, Nordic: *S120 SoftDevice Specification.* http://infocenter.nordicsemi.com/pdf/S120_SDS_v1.1.pdf. Version: 2014. – accessed, 07. September 2017

[Sem17] Semiconductor, Nordic: *nRF51822.* https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF51822. Version: 2017. – accessed, 10. September 2017

[Sen17a] Senrio: *Devil's Ivy - Senrio.* `http://blog.senr.io/devilsivy.html`. Version: 2017. – accessed, 22. September 2017

[Sen17b] Senrio: *Sensio Blog - Senrio.* `http://blog.senr.io/blog/devils-ivy-flaw-in-widely-used-third-party-code-impacts-millions`. Version: 2017. – accessed, 22. September 2017

[(SI09] (SIG), Bluetooth Special Interest G.: *SIG INTRODUCES BLUETOOTH LOW ENERGY WIRELESS TECHNOLOGY, THE NEXT GENERATION OF BLUETOOTH WIRELESS TECHNOLOGY.* `https://www.bluetooth.com/news/pressreleases/2009/12/17/sig-introduces-bluetooth-low-energy-wireless-technologythe-next-generation-of-bluetooth-wireless-technology`. Version: 2009. – accessed, 07. September 2017

[(SI17] (SIG), Bluetooth Special Interest G.: *Client Characteristic Configuration.* `https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.descriptor.gatt.client_characteristic_configuration.xml`. Version: 2017. – accessed, 09. September 2017

[Sys17] Systems, Espressif: *ESP8266 Overview.* `https://espressif.com/en/products/hardware/esp8266ex/overview`. Version: 2017. – accessed, 10. September 2017

[Tow15] Townsend, Kevin: *Introduction to Bluetooth Low Energy.* `https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap`. Version: 2015. – accessed, 09. September 2017

[Tow17] Townsend, Kevin: *microcontrollers GattStructure.* `https://learn.adafruit.com/assets/13828`. Version: 2017. – accessed, 09. September 2017

[TUM17] TUM: *Lehrstuhl fuer Hydrologie und Flussgebietsmanagement, Glonn.* `http://www.hydrologie.bgu.tum.de/index.php?id=265`. Version: 2017. – accessed, 2. October 2017

[wav14] wavesen: *HC-08 BLUETOOTH UART COMMUNICATION MODULE V2.0 USER MANUAL.* `http://www.wavesen.com/mysys/db_picture/news3/2015121885146101.pdf`. Version: 2014. – accessed, 10. September 2017

[Wav16] Waveshare: *NRF51822 Eval Kit.* `http://www.waveshare.com/wiki/NRF51822_Eval_Kit#How_to_compile_and_program_ble_app_hrs`. Version: 2016. – accessed, 10. September 2017

[YMG08] Yick, Jennifer ; Mukherjee, Biswanath ; Ghosal, Dipak: Wireless sensor network survey. In: *Computer networks* 52 (2008), Nr. 12, S. 2292–2330

[YSS$^+$15] Yu, Tianlong ; Sekar, Vyas ; Seshan, Srinivasan ; Agarwal, Yuvraj ; Xu, Chenren: Handling a Trillion (Unfixable) Flaws on a Billion Devices: Rethinking Network Security for the Internet-of-Things. In: *Proceedings of the 14th ACM Workshop on Hot Topics in Networks.* New York, NY, USA : ACM, 2015 (HotNets-XIV). – ISBN 978–1–4503–4047–2, 5:1–5:7