INTERNET OF THINGS

INTERNET OF THINGS

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE
Bc. DAVID PIŠKULA

SUPERVISOR
VEDOUCÍ PRÁCE
Ing. PETR MUSIL

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání diplomové práce

22167

Student:	**Piškula David, Bc.**	
Program:	Informační technologie	Obor: Počítačové a vestavěné systémy
Název:	**Internet of Things**	
	Internet of Things	
Kategorie:	Vestavěné systémy	

Zadání:

1. Prostudujte dostupnou literaturu týkající se Internet of Things (IoT) s důrazem na problémy inicializace zařízení, autonomnosti při výpadku spojení, zabezpečení přenosu a zpracování výsledků na cloudu, včetně provedení rešerše dostupných řešení.
2. Seznamte se s vhodnými platformami pro IoT
3. Navrhněte metody řešící problémy bodu 1 zadání s důrazem na jednoduchost nastavení pro uživatele s zajištěním spolehlivosti a bezpečnosti.
4. Implementujte a otestujte tyto metody na zvolené IoT platformě.
5. Vhodným způsobem demonstrujte funkčnost Vámi navrženého řešení.
6. Zhodnoťte dosažené výsledky z pohledu funkčnosti a bezpečnosti a zvažte možností dalšího rozšíření či vylepšení

Literatura:

- Dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce:	**Musil Petr, Ing.**
Vedoucí ústavu:	Černocký Jan, doc. Dr. Ing.
Datum zadání:	1. listopadu 2018
Datum odevzdání:	22. května 2019
Datum schválení:	1. listopadu 2018

Abstract

This thesis focuses on the Internet of Things and some of the most important problems it faces today. Among these are the overdependence on the Cloud and lack of autonomy, poor security and privacy, complicated initialization and power consumption. The work aims to implement a complex IoT solution that solves the discussed problems. The project is part of a collaboration with NXP Semicondutors and will be used to showcase the company's technologies.

Abstrakt

Tato práce se zabývá Internetem věcí a některými z jeho nejdůležitějších problémů. Mezi ně patří příliš velká závislost na Cloudu a chybějící autonomie, slabé zabezpečení a soukromí, komplikovaná inicializace a spotřeba energie. Práce má za cíl implementovat komplexní IoT systém, který řeší tyto diskutované problémy. Projekt vznikl ve spolupráci s firmou NXP Semiconductors, a bude využit k prezentaci jejich technologií.

Keywords

Internet of Things, autonomy, initialization, security, Thread, Zigbee, Bluetooth Low Energy, CoAP, MQTT, cloud computing, Google Cloud, edge computing, Android Things, smart home

Klíčová slova

Internet věcí, autonomnost, inicializace, zabepečení, Thread, Zigbee, Bluetooth Low Energy, CoAP, MQTT, cloud computing, Google Cloud, edge computing, Android Things, chytrý dům

Reference

PIŠKULA, David. *Internet of Things.* Brno, 2019. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Petr Musil

Internet of Things

Declaration

Hereby I declare that this master's thesis was prepared as an original author's work under the supervision of Ing. Petr Musil. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

. . . .
David Piškula
May 20, 2019

Acknowledgements

I would like to thank Ing. Petr Musil for supervising this thesis and providing important feedback and technical advice, my family and friends for their support, my manager Ing. Pavel Šádek for his leadership and goodwill and my colleagues for all their help, especially Ing. Karel Povalač, Phd., Ing. Rastislav Pavlanin, Phd. and Ing. Richard Heger.

Contents

Chapter 1

Introduction

The Internet of Things is a phenomenon that has been quickly gaining popularity in the last decade. As devices keep getting smarter every year, a future where everything is connected to an automated network of cooperating sensors and actuators that make human lives more convenient and efficient is very close. However, in order to make this concept a reality, several important problems still need solving. In its current state, the Internet of Things suffers from issues that can lead to dangerous security or privacy breaches, is often too reliant on services that are out of their consumer's control and there are too many incompatible technologies that create a fragmented market. The goal of this thesis is to research some of these problems, design a solution that solves them and demonstrate the design's functionality with a physical model of a smart home.

This thesis builds on the knowledge and experience the author gained by working on a bachelor's thesis that focused on the Internet of Things and created a simple demonstration with a Thread network connected to the Microsoft Azure Cloud platform. The scope of this thesis however, goes beyond the previous one and intends to provide a more complex Internet of Things solution. The master's thesis studies several problems of IoT, like its overdependence on the Cloud and lack of autonomy, poor security and privacy, complicated initialization and power consumption. The aim of this work is to create a system, which is autonomous and does not rely on the Cloud for its main functionality, is properly secured and whose communication is kept private, provides a means of both remote and local device control through a mobile application, uses power efficient devices and offers an easy to use initialization technique. The result will be a physical model of a smart home that presents these solutions. The thesis is being developed under the patronage of NXP Semiconductors and will use their technologies.

The first part of this document describes what the Internet of Things is, what technologies make it possible, what problems it faces and how current solutions try or fail to solve them and lastly talks about the possible future of the Internet of Things. The next chapter goes through the design goals and the design itself, the chosen technologies and the proposed topology. The following chapter focuses on the implementation of the complex system and the problems encountered during its development. The second to last chapter presents the experiments carried out on the system in order to assess its functionality. Finally, the thesis is summarized and future goals and possibilities are discussed in the last chapter.

Chapter 2

Important Problems of IoT

This chapter offers an overview of the Internet of Things, it's history, the technologies that make this phenomenon possible and the problems that hold it back. It also briefly discusses the future of IoT and talks about both the good and the bad that IoT might bring.

2.1 What Is the Internet of Things

Internet of Things [1, 2] is a buzzword that has been gaining a lot of popularity in the last few years. The name was coined by Kevin Ashton[1] in 1999 during a presentation for Procter and Gamble (P&G) about linking RFID technology to the internet.

Internet of Things describes a system where items in the physical world, and sensors within or attached to these items, are connected to the Internet via wireless and wired Internet connections. These sensors can use different communication technologies such as RFID[2], NFC[3], Wi-Fi, Bluetooth, Zigbee or Thread. The goal of this system is to connect inanimate objects and living things, use sensors to collect and analyze data to provide new information or improve efficiency of existing systems and to create autonomous environments where things and beings cooperate seamlessly.

Thanks to rapidly improving technologies for low-power wireless communication, powerful, yet relatively cheap, microcontrollers, machine learning and Cloud technologies, it is possible for IoT to spread into more and more areas. Smart homes enable power savings through more efficient use of electricity and temperature control systems and provide their owners with increased comfort in the way of automatic lights, home control through their phones and better security. In health care, nursing systems in hospitals can become more advanced by equipping patients with devices that report their health or location directly to their doctors. Industry 4.0 [3] takes advantage of IoT to upgrade automation within factories, enable machine to machine and machine to product communication and monitor equipment use and servicing needs. Agriculture and farming industries may utilize sensor data to better and more effectively take care of animals and plants.

[1] ASHTON, Kevin. That 'Internet of Things' Thing. *RFID Journal* [online]. [cit. 2019-01-11]. Retrieved from: https://www.rfidjournal.com/articles/view?4986

[2] VIOLINO, Bob. What is RFID? *RFID Journal*. January 2005. [online]. [cit. 2019-05-17]. Retrieved from: https://www.rfidjournal.com/articles/view?1339/

[3] About the Technology. *NFC Forum*. [online]. [cit. 2019-05-17]. Retrieved from: https://nfc-forum.org/what-is-nfc/about-the-technology/

2.2 Technologies Connected With IoT

The success of the Internet of Things depends on a wide range of technologies. Among these are the hardware components like sensors and actuators or MCUs, that need to be highly energy efficient when used for end devices or powerful enough to be able to maintain a large IoT network or even process the data output of the network. Moreover, the software technologies that provide the functionality and communication capabilities to these MCUs have to also comply with these requirements, which is why so many new low-power consuming IoT focused communication technologies and protocols have been emerging in the last two decades. Additionally, simplified IoT versions of common operating systems, that can be used on smart gateways, are being introduced.

Furthermore, Cloud Computing gained a lot of popularity, as it provides the ability to process and store the huge amounts of data generated by IoT networks and to provide automation by analysing the data and manipulating the end devices without the need for companies to maintain their own servers. Lately, however, Edge Computing has started to become the new norm, as MCUs keep getting more powerful and can handle some of the complexity of data processing on their own. Artificial intelligence technologies will also play a large role in the possible mainstream adoption of IoT by the world, as it can bring efficiency and autonomy to the IoT networks.

Another crucial area is the privacy and security of IoT networks which involves both software, in the form of, for example, cryptographic algorithms and hardware, that can be used to securely store the keys and secrets used to secure network access and communication.

This chapter will provide an overview of current technologies and describe the ones that were chosen for this thesis in more detail.

Connectivity and Communication Technologies

A wide variety of connectivity technologies useful for IoT has been developed over the last three decades. Some of the earlier ones include RFID, Bluetooth and Wi-Fi, among the more modern are NFC, Zigbee, Thread and Bluetooth Low Energy. Zigbee and Thread are both based on the IEEE 802.15.4 standard [4], which focuses on low-rate wireless personal area networks with low-power devices. Bluetooth Low Energy is a new version of Bluetooth that was also created with low energy consumption in mind.

Communication protocols with large overhead like TCP/IP are not suitable for low-power MCUs. Due to this, more efficient IoT-focused communication protocols have been invented, like CoAP, MQTT, AMQP or XMPP-IoT,

Zigbee Zigbee technologies are developed by the Zigbee Alliance members. They focus on creating a standard for the Internet of Things. Zigbee 3.0[4] is their latest unified solution that builds on IEEE 802.15.4, Zigbee PRO and the Zigbee Cluster Library. Zigbee supports large low-power mesh networks operating in the sub-GHz band locally and 2.4GHz globally. Zigbee PRO supports four device types - end devices, routers that maintain the network, coordinators that create, maintain and secure the network and gateways providing Internet connection. The devices can support Zigbee Green Energy which is an energy harvesting technology allowing the use of small sensors or devices without a battery or wired power

[4]Zigbee for Developers. *Zigbee* [online]. [cit. 2019-01-11]. Retrieved from: https://www.zigbee.org/zigbee-for-developers/zigbee-3-0/

source. Zigbee devices communicate through an application layer protocol called Dotdot[5] and in collaboration with the Thread Group, Dotdot can be used with Thread devices to integrate IPv6 connectivity to Zigbee networks.

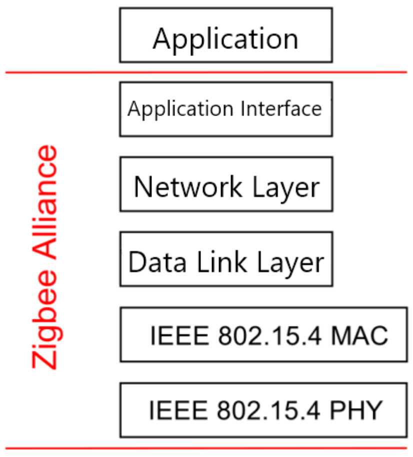


Figure 2.1: Zigbee Architecture[6]

Bluetooth Bluetooth is a technology originally created in 1994 that provides wireless data streaming between devices, like music to headphones or a mobile phone-to-car communication. In 2011 Bluetooth Low Energy[7] was released with focus on low-power devices and the Internet of Things. Devices using BLE spend most of their operating time in sleep modes, allowing low energy consumption. In 2017 BLE was extended with mesh capabilities through the Bluetooth Mesh Profile specification [5].

Thread Thread[8] [6] is a wireless mesh communication protocol invented and maintained by the Thread Group. The main areas of focus for this technology are smart homes and smart commercial buildings. It is designed to be low-power and secure and was based on IEEE 802.15.4. Thread enables IPv6 communication to devices through 6LoWPAN [7], an open standard adaptation layer for IPv6 packet transportation in IEEE 802.15.4 networks. Thread supports several device types in its networks - a border router, a router-eligible end device, an end device and a sleepy end device.

Border router is a specific type of router, which facilitates connectivity from the 802.15.4 network to networks on other physical layers, like Wi-Fi or Bluetooth. They also maintain the 802.15.4 network and provide various services like routing for off-network operations. A single Thread network can have several border routers. When that is the case, the network then has no single point of failure for the connectivity outside of Thread.

[5]The Dotdot Story. *Speak Dotdot* [online]. [cit. 2019-01-11]. Retrieved from: https://www.speakdotdot.com/dotdotstory/

[6]PIŠKULA, David. Internet of Things zařízení s podporou Thread a 6LoWPAN. Brno, 2017. Bachelor's Thesis. Brno University of Technology, Faculty of Information Technology.

[7]Bluetooth Vs. Bluetooth Low Energy: What's The Difference?. *Link Labs* [online]. [cit. 2019-01-11]. Retrieved from: https://www.link-labs.com/blog/bluetooth-vs-bluetooth-low-energy

[8]What is Thread. *Thread Group* [online]. [cit. 2019-01-11]. Retrieved from: https://www.threadgroup.org/What-is-Thread

Routers take care of routing services, joining and security services for network devices and devices that are trying to join the network. A router eligible end device can become a router when needed or can be demoted to a regular end device. When demoted, they do not forward messages or provide the mentioned services. The promoting and demoting of router eligible end devices is managed by the Thread network without the need for user interaction. These devices cannot go to sleep to save energy.

The sleepy end devices never provide any network services and need a parent router to be able to communicate with the rest of the network. However, they spend most of their life time asleep, thus conserving energy and ensuring that they can operate for years using a coin-cell battery.

When at least two routers are in the network, the Thread stack forms a mesh topology. If only one router or border router is present, the network works with a basic star topology instead. The Thread mesh is self-healing and has no single point of failure, which means that when one of the routers is lost, the devices that needed them automatically find a substitute and discover alternate routes to maintain full functionality.

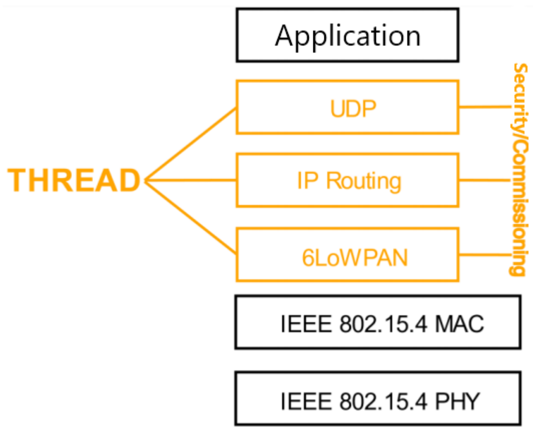


Figure 2.2: Thread Architecture[6]

The Tread stack describes three phases for joining of new devices to the network. First, the device enters the discovery phase, during which it discovers the Thread network and establishes contact with a router for commissioning. It does this by scanning all available channels, sending beacon requests on each and waiting for potential responses. Once a router is discovered, the device starts communicating with it so that the commissioning phase can begin. If the device is already commissioned, it does not need to enter the discovery phase at all.

In the commissioning phase, the device either uses information provided to it by some out-of-band method to attach to he Thread network or a commissioning session is established between the device and some commissioning application on a smartphone, tablet or the web. Through this session the device is provided with the necessary information needed to attach to the network. During the commissioning phase, the device needs a correct Pre-Shared Key for the Device (PSKd) and a network Master Key to be allowed to join.

Lasty, the device attaches itself to the network, exchanges configuration messages with its parent router and is allocated a network address by the router.

Table 2.1: Technology comparison

	Thread	Zigbee 3.0	Bluetooth Low Energy
Standard	IEEE 802.15.4	IEEE 802.15.4	IEEE 802.15.1
Frequency	2.4 GHz	2.4 GHz, sub-GHz	2.4 GHz
Range	20-30 m	10-20 m	200 m
Data Rate	250 Kb/s	250 Kb/s	up to 2Mb/s
IPv6	Yes, 6LoWPAN	No	No
Max no. Devices	250	65000	Not Limited
Topology	Mesh, Star	Mesh, Star, Tree	Mesh, Point-to-Point, Broadcast
Cloud Connectivity	Border Router, Gateway	Gateway	Smartphone, Gateway

CoAP Constrained Application Protocol is a specialized web transfer protocol defined in the RFC 7252 [8]. It was created for constrained nodes and networks, such as low-power devices or lossy networks. CoAP provides a request/response interaction model between application endpoints, often machine-to-machine communications, that includes key concepts of the Web such as URIs. It can easily interface with HTTP while maintaining specialized features like multicast support and very low overhead. Unlike HTTP though, CoAP communicates asynchronously over UDP. IoT technologies like Thread are built on CoAP and utilize it for network communications.

MQTT MQTT is defined in [9] as a client-server publish/subscribe messaging transport protocol that is light weight, open and ideal for Machine to Machine communications and the Internet of Things because of its small footprint. It runs over TCP/IP, communicates through a broker application that provides one-to-many message distribution and offers three qualities of service. These are the QoS 0: At most once, that delivers messages to the best efforts of the operating environment, QoS 1: At least once, where messages are assured to arrive but duplicates can occur and QoS 2: Exactly once. MQTT can be secured through TLS with SSL Certificate exchanges between the broker and subscribers/publishers or with WebSockets.

Cloud computing

Cloud computing [10, 11] is an on-demand computing model composed of autonomous, networked IT (hardware and/or software) resources. The main advantage of this model is that organizations no longer need to own and maintain their own datacenters and can get resources on-demand from the Cloud computing provider instead. These Clouds are rapidly scalable, allow resource pooling, where several consumers use the same pool of resources. They are also easy to use, as their providers usually offer GUI access and development guides, and subscription based, allowing consumers to only pay to the time and resources they need.

Cloud platforms are often divided into three categories, Sotfware as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). Infrastructure as a Service provides consumers with access to processing, storage, networks and other resources where the consumer can deploy and run arbitrary software. Platform as a Service allows consumers to use programming languages, libraries, services and tools supported by the provider to deploy onto the Cloud infrastructure. Software as a Service gives users the use of applications running on a Cloud infrastructure.

Cloud computing is a very important part of the Internet of Things. It enables sensors to collect very large amounts of data and store them in highly scalable databases. It provides the means to run deep analytics on such data, use machine learning to make IoT systems more efficient. In the previous years, the trend was to utilize the Cloud's computing capabilities to enhance low-power low-performance devices and run most of the processing on Cloud platforms. However, with growing concerns about data privacy and with improvements to microcontrollers, Edge computing is becoming the new popular phrase. Edge computing leverages the strengths of gateways or even end devices equipped with more powerful microcontrollers to make a lot of the computation local again, lowering latency, improving privacy and decreasing the network's dependence on the Cloud.

Edge Computing

In [12], Edge Computing's advantages are listed as the option to process the massive data generated by devices at the network edge instead of transmitting them to the centralized Cloud. It can provide services with faster responses and greater quality and can be considered the future the Internet of Things infrastructure. They also describe the various approaches to Edge Computing that are being implemented today.

Cloudlets, are small-scale data centers located at the edge of the internet. They aim to improve end-to-end responsiveness between a mobile device and the Cloud. They consist of resource rich computers providing powerful computing resources to nearby mobile devices with lower latency. They can serve as a middle point between the mobile devices and the Cloud. The paper also lists several important differences between the Cloud and the Cloudlet. Cloudlets need to be much more agile in its provisioning, because mobile devices connect to them dynamically due to user mobility. A virtual machine handoff technology needs to be utilized to pass the needed services between Cloudlets as a user moves. The mobile devices need to discover the Cloudlets, as they are small, geographically distributed data centers. Cloudlets can be for example be used for wearable cognitive assistance. They can improve the robustness of the whole IoT system by adding a layer between potential cyberattacks and the Cloud and encapsulating each user's environment in a separate virtual machine.

Mobile edge networking, on the other hand, is defined as a technology that provides an IT service environment and Cloud Computing capabilities at the edge of a mobile network. This can be represented by applications running as a virtual machine on a powerful mobile edge platform. It can be used for handling video streaming services in smart cities, augmented reality or smart vehicles. It can collect, classify and analyze IoT data streams, manage protocols, distribution of messages and processing of analytics.

Finally, Fog Computing aims to distribute resources and services along the continuum from the Cloud to things, using more powerful end nodes where possible and small Clouds at the edge. It can enable interoperability in IoT, 5G, virtual reality and other applications. It creates a hierarchical network of Fogs that can communicate among each other and with the Cloud. Each Fog can collect and process data on its own or pass them along the hierarchy as needed.

Operating System

With the emergence of devices like the Raspberry Pi and with an increasing number of powerful low-energy microcontrollers on the market, OS providers began developing simpler or less resource intensive versions of their systems that can be used for IoT devices. Among these are Ubuntu Core, Windows 10 IoT and Android Things. These operating systems share some attributes important for IoT devices - they are built to be highly secure, allow over-the-air updates and promise fast development. Furthermore, Windows 10 IoT can be easily integrated with Microsoft's Cloud platform, Azure while Android Things can similarly integrate with Google Cloud.

Developers then have a choice between running these OS on powerful end devices and having them connect directly to the Cloud or to use a gateway that runs the OS and communicates with the Cloud on behalf of the end devices.

Android Things Android Things is an operating system made by Google and originally inteded for Internet of Things devices. It is based on Project Brillo, which was another operating system from Google and while both share the same core, Brillo was focused on C++ development, while Android Things supports a subset of the Android SDK. Android Things extends the core Android framework[9] and offers developers the ability to develop using the Android SDK and Android Studio, natively access hardware through the Android framework, integrate additional peripherals through Peripheral I/O APIs and use the Android Things Console to push over-the-air feature and security updates. It is supported on several development kits[10], including the NXP Pico i.MX7D and Raspberry Pi 3 Model B.

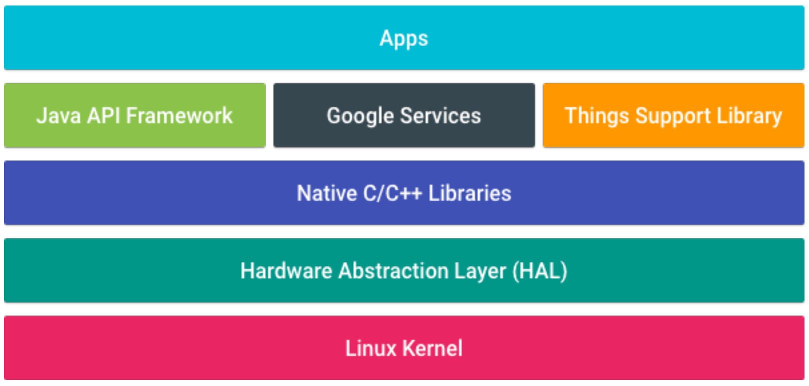


Figure 2.3: Android Things Platform Architecture[11]

Hardware

The devices used for IoT can be generally divided into two categories - low-power devices with smaller computation power but better energy savings, that use technologies like Zigbee

[9]Overview. *Android Developers*. [online]. [cit. 2019-02-05]. Retrieved from: https://developer.android.com/things/get-started

[10]Supported Hardware. *Android Developers*. [online]. [cit. 2019-02-05]. Retrieved from: https://developer.android.com/things/hardware/

[11]Android Things Platform Architecture. *Android Developers*. [online]. [cit. 2019-02-05]. Retrieved from: https://developer.android.com/things/images/platform-architecture.png

or Thread, and more powerful devices with IoT operatings systems like the Windows 10 IoT or Android Things, and usually connect to the Internet directly.

Several companies focus on creating low-power microcontrollers suitable for IoT development. Among these are Texas Instruments with their CC2650 wireless MCU capable of communicating over Bluetooth Low Energy, Zigbee and 6LoWPAN. STMicroelectronics offer the STM32WB55xx series of multiprotocol devices that can communicate over Bluetooth Low Energy and IEEE 802.15.4 based protocols as well. NXP Semiconductors make the ultra-low-power KW41Z devices likewise using Bluetooth Low Energy and IEEE 802.15.4 connectivity.

Development boards and small computers like the Raspberry Pi, Qualcomm DragonBoard or NXP Pico are powerful enough to fully utilize the aforementioned operating systems and can therefore act as powerful but more energy consuming end devices or network gateways, connecting the low-power networks to the Internet.

Cryptography

The basic concepts of cryptography, as described by [13], are privacy/confidentiality, authentication, integrity, non-repudiation and key exchange. Privacy ensures that only the intended receiver can read the data of the message, authentication is the proces of proving one's identity, integrity is for making sure the transmitted message is not tampered with, non-repudiation is a mechanism of checking that the sender really sent the message and key exchange is a method of sharing cryptographic keys between the sender and the receiver. The paper classifies cryptographic algorithms into three categories, the Secret Key Cryptography, Public Key Cryptography and Hash Functions.

Secret Key Cryptography uses a single key for both encryption and decryption of messages. The sender takes a plaintext message, encrypts it with the secret key and sends it as ciphertext to the receiver. The receiver then uses the same key to decrypt the ciphertext back to readable plaintext. Since both sides use the same key, this method is also called Symmetric Encryption. The method works either in the mode of stream ciphers or block ciphers. Stream ciphers operate on single bits, bytes or computer words at a time and utilize a feedback mechanism to ensure the key is constantly changing. Block ciphers, on the other hand, operate on blocks of data using the same key every time. In general, stream cipher encrypted plaintext will encrypt to a different ciphertext every time whereas a block cipher plaintext will always encrypt to the exact same ciphertext.

Public Key Cryptography depends on mathematical functions called one-way functions, which are easy to compute but whose inverse functions are difficult to compute. The scheme employs two keys that are mathematically related but one cannot be used to easily determine the other. One of the keys can be used to encrypt plaintext and the other to decrypt the resulting ciphertext and it does not matter with key is applied first. Since the sender and receiver use a different key each, the approach is called Asymmetrical Encryption. One of the most commonly used Public Key algorithms is the RSA algorithm, which can be used for key exchange, digital signatures or encryption of small blocks of data.

Hash Functions, or otherwise called message digests and one-way encryption are algorithms that do not use any keys. Instead, they use a fixed-length hash value computed upon the plaintext that makes it so that it is not possible to recover the contents nor the length of the original plaintext. They are often used to encrypt passwords, for digital fingerprinting or to ensure the integrity of a computer file. Since they are one-way encryption algorithms, the ciphertext cannot be decrypted. Commonly used Hash algorithms today inlcude the Message Digest (MD) algorithms which are a series of byte oriented algorithms that produce a 128-bit hash value from an arbitrary-length message and the Secure Hash Algoritms (SHA).

The Secure Sockets Layer (SSL) and Transport Layer Security (TLS) are protocols that ensure secure transactions over communication protocols like HTTP. SSL uses RSA certificates for authentication and the initial handshake, after which both communicating endpoints agree upon an encryption scheme. After the SSL was found to be breakable, the TLS protocol became the new popular protocol. It extends and is backwards compatible with SSL, supports additional cryptographic schemes and is used for HTTPS communication. The communication between a client and a server is initiated after a TLS protocol handshake which has three phases. First, an unencrypted key exchange is performed. Afterwards, all communication is encrypted, starting with the second phase of the handshake, where the server sends the server parameters, which specify additional handshake parameters. Lastly, the server and optionally the client are authenticated, keys are confirmed and the integrity of the handshake is assured.

2.3 The Evolution of Autonomous Systems

One of the biggest selling points of the Internet of Things are autonomous systems that control whole networks of devices based either on their own decisions through machine learning or on preprogrammed rules, in order to bring efficiency and larger quality of life everywhere, from factories and industrial plants to agriculture, cities, houses and more. The original approach to this concept was to completely integrate the Internet of Things with Cloud Computing and leave most of the processing of the data generated by IoT networks to the Cloud. In [14] and [15], the researchers focused on the difficulties of implementing advanced features like data processing on computationally weak devices and analyzed the trends and technologies used back then. They introduced a new paradigm that they called CloudIoT, which would merge the Cloud and IoT to enhance them both. With their proposal, the Cloud would bring better storage resources to IoT, as the huge amounts of unstructured or semi-structured data produced by IoT devices could be stored in the virtually limitless Cloud servers. Additionally, it would bring much larger and more scalable computational resources and allow the processing of the data, which was not very feasible to do on-site. Lastly, the Cloud would serve as a sort of communication hub, where all the devices and applications could connect and talk to each other. The papers also suggest the possible applications of the CloudIoT, for example in health care, smart cities or smart homes. At the end, they also briefly mention Fog Computing, which is a subset of Edge Computing, showing how quickly IoT keeps evolving and changing.

The trend to integrate these two areas was also evident from the was the Cloud platform providers advertised their solutions and what their suggested ways of using them were. Google Cloud, for example, used to recommend sending all data to their MQTT broker and use their Cloud-based services to analyze and process the data and respond to the devices with new commands, as can be seen in figure 2.4.

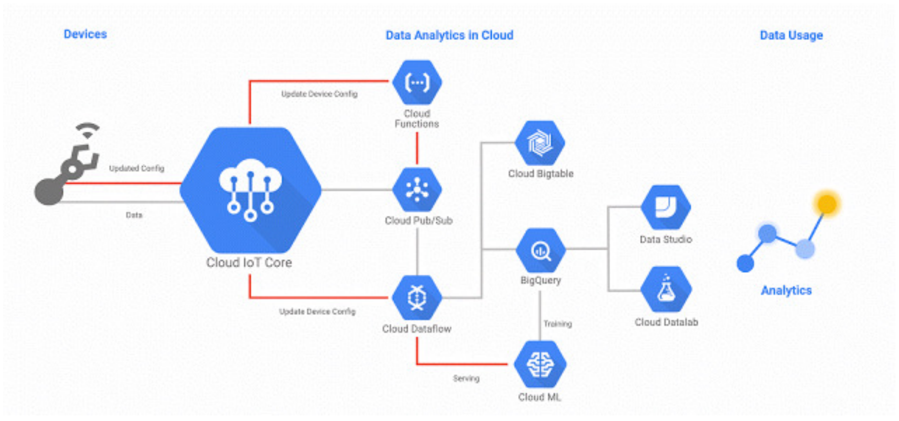


Figure 2.4: Old Cloud IoT Core approach[12]

However, this approach brings large latency to communication both between the devices and between the data processing algorithm and the devices it controls. Furthermore, the whole system is then completely dependent on an internet connection and the connection to the utilized Cloud platform, which makes it vulnerable and potentially dangerous.

As an example, a smart home that uses thermostats to control the heating and cooling of air inside and performs data processing purely in the Cloud can, in the best case scenario, stop heating and cooling entirely without an internet connection and in the worst case might get stuck in one of these operations and endanger the lives of its inhabitants.

In order to find a solution to these issues the research and development of IoT have shifted towards a new paradigm, the Edge Computing. There are many studies and papers that focus on Edge Computing, among them [16], where the author compares both Cloud computing with Edge Computing, talks about the history of Edge Computing and explains why the decentralized ideology of Edge Computing is gaining popularity. Some of the most important advantages they mention are highly responsive Cloud services provided by Cloudlets, better enforcement of privacy and the masking of Cloud outages by keeping the system functional even without a Cloud connection. In [17], the team of researchers present the problems of Cloud Computing solutions like the high communication latency and the substantial stress put on network links to the Cloud by all the data generated by IoT devices. They introduce Edge Computing and its usefulness in several different areas of technologies, including the Internet of Things and lastly they describe their experimental evaluation of mobile gaming with the use of Edge Computing.

The problems of Cloud-dependence have also been mentioned in various media outlets, like the Medium[13] that published an article exploring the large disadvantages of systems completely reliant on the internet and Cloud for their functionality and offering a list of steps that could be taken in order to get rid of the overdependence on Cloud. Similarly,

[12]CHAKRABORTY, Indranil. Announcing Cloud IoT Core public beta. *Google Cloud Platform*. September 2017. [online]. [cit. 2019-02-05]. Retrieved from: https://storage.googleapis.com/gweb-Cloudblog-publish/original_images/Cloud2BIoT2BFinal2BV6s64u.GIF

[13]BURNS, Patrick. Why The Internet of Things and the Cloud Should Break Up. *Medium* [online]. [cit. 2019-01-12]. Retrieved from: https://medium.com/@patburns/why-the-internet-of-things-and-the-Cloud-should-break-up-8aa6ec563a81

networkworld[14] has an article talking about how Edge Computing is critical for the evolution of IoT and about the important improvements Edge Computing can bring to IoT.

Finally, the change in philosophy can also be noticed in the way the guides from Cloud platform providers have changed in the last two years. Contrary to the previous mentality of pushing everything to the Cloud, as previously mentioned and shown in figure 2.4, nowadays they offer various types of Edge Computing platforms that can be used on-site on powerful enough devices and that can take care of the network-to-Cloud connection and data and information exchange. Even Cloud IoT Core's documentation has been altered, as can be seen in figure 2.5.

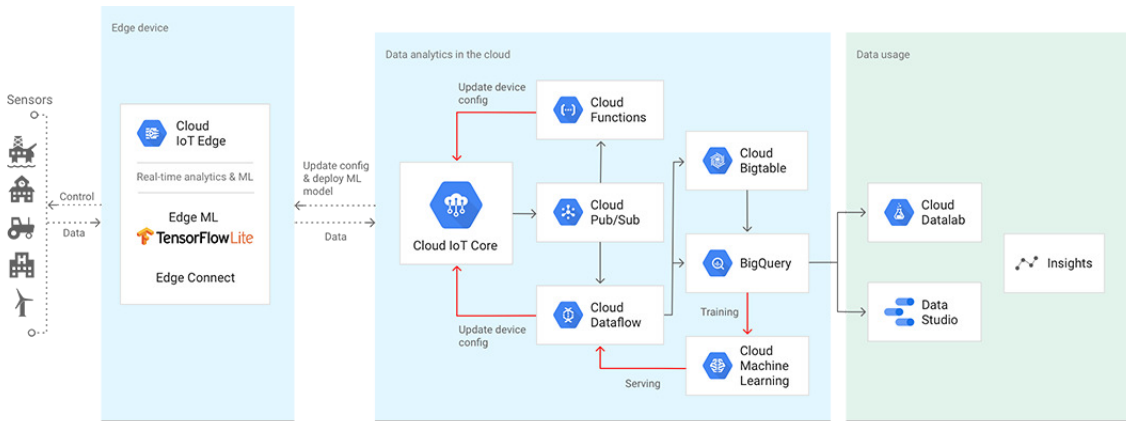


Figure 2.5: New Cloud IoT Core approach[15]

Google now proposes to use an Edge Device with their Cloud IoT Edge services in order to perform real time analysis and machine learning while using the Cloud to store data, retrain machine learning models and provide online analytics in the form of data graphs.

2.4 Current Autonomous Solutions and Their Problems

As mentioned previously, the original Cloud Computing solutions have many issues and nowadays even the providers of Cloud platforms are changing their suggestions for IoT networks, like Google with their Cloud IoT Edge or Amazon and their Amazon IoT Edge.

Among home automation solutions, there have been several failing products on the market. Revolv[16] was a smart home hub with the purpose of controlling a wide range of different gadgets via a smartphone app. However, the hub was completely reliant on the developer company's Cloud-based service. Revolv was eventually bought by Nest and subsequently, the Cloud service was shut down, rendering all Revolv hubs inoperational. Another product

[14]TALLURI, Raj. Why edge computing is critical for the IoT. *Network World* [online]. [cit. 2019-01-12]. Retrieved from: https://www.networkworld.com/article/3234708/internet-of-things/why-edge-computing-is-critical-for-the-iot.html

[15]Cloud IoT Core. *Google Cloud Platform*. [online]. [cit. 2019-02-05]. Retrieved from: https://Cloud.google.com/iot-core/images/benefits-diagram.png

[16]FINLEY, Klint. Nest's Hub Shutdown Proves You're Crazy to Buy into the Internet of Things. *Wired* [online]. [cit. 2019-01-14]. Retrieved from: https://www.wired.com/2016/04/nests-hub-shutdown-proves-youre-crazy-buy-internet-things/

with a similar fate was Emberlight[17], a smart light socket designed to work with ordinary light bulbs. Its aim was to enable a mobile application to control regular light bulbs instead of having to buy specialized and expensive ones. The issue was that all controls were done though a Cloud service which was, again, shut down.

Another examples are home controlled through smart assistants like Alexa in Amazon's Echo or Google Assistant in Google Home. These products use voice recognition in the Cloud in order to understand commands and communicate with humans. Older versions did not support offline speech recognition which could result in complete loss of control over smart homes that relied on them. One experience with this critical fault was described in an article by Mashable[18], where the author could not control their Alexa device which resulted in them not being able to even turn on the lights in the house.

The idea of Edge Computing was explored in [18] with the goal of implementing a smart e-Health gateway to bring the Internet of Things to healthcare. The paper focuses on inventing a system architecture that would allow for a secure network with local storage, data filtering and analytics. The result of their work was UT-GATE, a functional gateway with a WebSocket server providing local servicing while communicating with a remote Cloud platform to receive improved processing rules gained through deeper analytics.

Another study concerning this area was described in [19]. This team of researchers tried to tackle the issues by proposing a hierarchical fog computing architecture that is flexible, scalable and brings computing resources close to end devices. The architecture consists of a network of computationally powerful fog nodes that can each connect to a Cloud server to offload their work if needed. The resulting design can substantially reduce traffic loads in networks and the communication delay that can be a problem in purely Cloud Computing based systems.

There are also products on the market already that offer partial or even full functionality to connected networks even without an internet connection. The latest Amazon Echo Plus, for example, allows limited usage of their voice controlled home control hub offline. Phillips Hue Bridge is another example, as it only needs an internet connection for remote control but locally can work completely offline, as explained in an article by howtogeek[19]. Hubitat[20] is a home automation platform, that was built with offline functionality in mind. Despite the ability to use an internet connection for updates and Cloud communication, it does not need the internet connection for any of its major functions. This makes the whole system more secure, private and removes latency.

2.5 Security and Privacy

When IoT became popular, many manufacturers focused on performance and usability of IoT devices and ignored security measures and encryption mechanisms, leaving their devices

[17]PAUL, Fredric. What happens when an IoT implementation goes bad?. *Network World* [online]. [cit. 2019-01-14]. Retrieved from: https://www.networkworld.com/article/3238004/internet-of-things/what-happens-when-an-iot-implementation-goes-bad.html

[18]WONG, Raymond. TFW your internet goes down and takes your smart home with it. *Mashable* [online]. [cit. 2019-04-13]. Retrieved from: https://mashable.com/2016/07/05/smart-home-useless-internet-down/?europe=true#B2SqpRDVRkqa

[19]LLOYD, Craig. What Happens If My Philips Hue Lights Go Offline?. *How-To Geek* [online]. [cit. 2019-01-14]. Retrieved from: https://www.howtogeek.com/293341/what-happens-if-my-philips-hue-lights-go-offline/

[20]Home Automation Features. *Hubitat* [online]. [cit. 2019-01-14]. Retrieved from: https://hubitat.com/pages/home-automation-features

vulnerable. A Forbes report predicts more than 80 billion smart devices to be connected to the Internet by 2025. Such a huge amount of potentially insecure devices would be very dangerous not only to their owners, but to the Internet and the whole world.

There have already been many cases proving the scale of this issue, from a casino being hacked into through their fish tank[21], to cardiac devices that hackers could misuse to cause incorrect pacing or shocks, a connected vehicle vulnerable to being taken over and allowing hackers to shut down the car's engine or cut the breaks[22], to a massive DDoS attack through Mirai botnets[23] [20] that infected tens of millions of IoT devices and left much of the Internet inaccessible on the U.S. east coast.

Apart from taking control over vulnerable devices, hackers can also listen to their communications and steal or misuse their data. This can lead to privacy issues like smart cameras being used to remotely spy on homes or important personal information being leaked, like medical records or time periods when a house or a building is often empty and easier to break into.

Due to the importance of these issues, the world's governments and research teams have been focusing on figuring out proper precautions and safety and security principles to battle the increasing risk an unsecured IoT presents. In Japan for example, in order to strengthen the security of their country before the 2020 Summer Olympic Games, the government has been testing and breaking into hundreds of millions of devices like routers and web cameras by using their default passwords which users often leave unchanged. They plan to compile a list of potentially compromised devices and report them to the authorities.[24]

In a paper [21] published at an international conference in 2012, a group of Chinese researchers examined and reviewed the state of security in the Internet of Things and proposed the most important challenges that still needed to be overcome. They divided IoT into four layers described in figure 2.6.

[21]WEI, Wang. Casino Gets Hacked Through Its Internet-Connected Fish Tank Thermometer. *The Hacker News* [online]. [cit. 2019-01-12]. Retrieved from: https://thehackernews.com/2018/04/iot-hacking-thermometer.html

[22]From Connected Cars, Healthcare to Uranium Enrichment Facilities, 5 IoT Security Hacking Instances to Take Note of!. *Embitel* [online]. [cit. 2019-01-12]. Retrieved from: https://www.embitel.com/blog/embedded-blog/security-challenges-faced-by-iot-based-industries

[23]FRUHLINGER, Josh. The Mirai botnet explained. *CSO* [online]. [cit. 2019-01-12]. Retrieved from: https://www.csoonline.com/article/3258748/security/the-mirai-botnet-explained-how-teen-scammers-and-cctv-cameras-almost-brought-down-the-internet.html

[24]THUBRON, Rob. Japan's government will start hacking its citizens' IoT devices next month. *Techspot*. January 2019. [online]. [cit. 2019-04-18]. Retrieved from: https://www.techspot.com/news/78456-japan-government-start-hacking-citizens-iot-devices-next.html

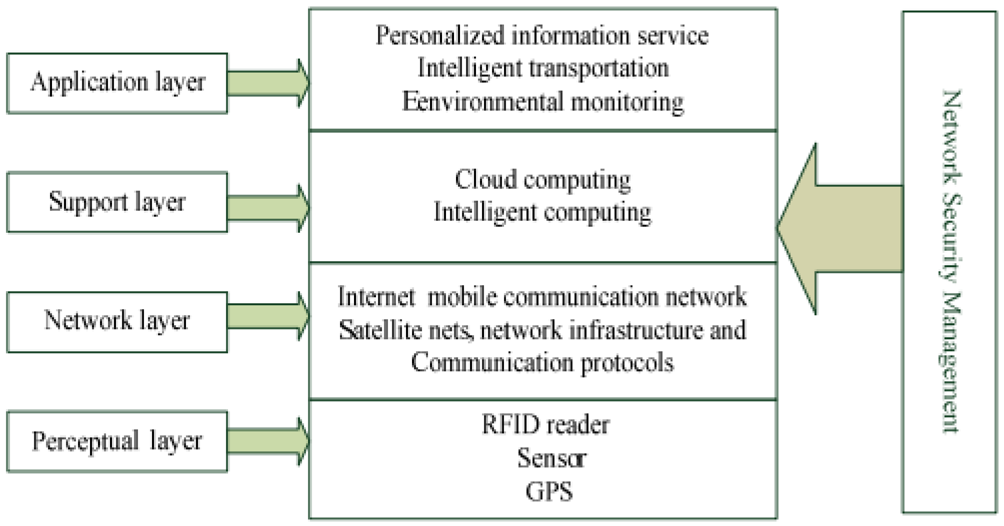

Application layer
Personalized information service
Intelligent transportation
Eenvironmental monitoring
Support layer
Cloud computing
Intelligent computing
Network layer
Internet mobile communication network
Satellite nets, network infrastructure and
Communication protocols
Perceptual layer
RFID reader
Sensor
GPS
Network Security Management

Figure 2.6: Security Architecture [21]

The nodes in the perceptual layers have limited computing power and storage but at the same time need to support authentication to prevent illegal access. In the network layer, man-in-the-middle attacks are a danger and DDoS attack prevention is also necessary. The support layer has to be protected from malicious information and should use stronger system security technologies and anti-virus programs. For the application layer, the security needs depend on the type of application but mostly concern data privacy and access control. The paper also briefly talks about some of the available encryption and cryptographic methods and algorithms and finally summarizes some common challenges for the security of IoT. Among these are cryptogrpahic key management and security laws and regulations.

Another paper [22] that studied the current status and open issues of security and privacy in IoT in 2014 highlights how there might be over thirty billion connected things in the world by 2020 but warns about the absence of solid security in place. It summarizes the overall vision and technologies of IoT and identifies some of the attacker models and threats and security challenges. Finally it summarizes the state-of-the-art of 2014 IoT. Among the attacker models and threats they mention are the intruder model, which can intercept all transmitted messages within a network, the denial-of-service attacks that attempt to make machines or even whole networks unavailable, physical attacks performed through physical tampering with devices and attacks on privacy, like eavesdropping, traffic analysis and data mining.

The Internet of Things is spreading into the industry and is giving birth to the latest potential industrial revolution. Industrial IoT is often being called the Industry 4.0. The security in this area of IoT was explored in [23]. The paper goes through several cyber attacks targetted towards industrial control systems, like the Slammer worm which infected a U.S.A. nuclear power plant, a virus that infected the control system of a major transportation network in the U.S.A. and Stuxnet which was used to make centrifuges at an Iranian nuclear facility to fail. The paper then talks about the difficulty of adapting existing information security concepts for the cyberphysical production systems (CPPS). For example, CPPS cannot be temporarily disabled and restarded like typical IT systems, have strict real-time requirements and constrained computational, memory and energy resources. Finally, the work surveys existing literature and research regarding these issues and offers an overview of necessary future research directions.

A more recent review of IoT's security from 2017 [24] shows that while all the IoT connected technologies are evolving, there are still too many security issues even five years after the paper cited in previous paragraphs. This survey paper explores the IoT security and privacy issues, presents the most relevant limitations of IoT devices and their solutions, discusses the classification of existing IoT attacks, examines IoT authentication and access control schemes and architectures proposed in recent literature and analyzes the security issues in all four of the layers from figure 2.6. Among the device limitations they list short battery life of IoT devices and lightweight computation capabilities, due to which conventional cryptography cannot work on IoT systems. The paper concludes with the opinion that the safety of IoT is dependant on the technologies, protocols and security mechanism implemented by manufacturers and in specific cases the resulting devices could be vulnerable to certain types of attacks. They suggest that there is an urgent need of developing a general security policy and standards for IoT products.

There are also mechanisms that can be used to ensure that when the software on a device or the cryptographic keys are compromised, there is still a way to recover the affected devices. Among them are the Over-The-Air (OTA) updates and automatic cryptographic key rotation. OTA updates are software updates that can be configured and sent to devices over the internet without the need for user interaction. As mentioned in some articles[25] [26], OTA will be critical for IoT, as either not updating the devices at all or having users update them manually poses security risks and gives harmful parties the chance to overtake a device forever or intercept the physical medium which is used to transport the updated software and tamper with it before it reaches its intended users. Android Things supports OTA through Google's console[27] which can be similarly used for all Android devices.

The cryptographic key rotation has a very similar goal of OTA. When a device's key is stolen, if the device is set up to automatically discards its current keys and register new ones in some time period, then then attacker can only access the network through the device for the length of this period. This is why Google Cloud IoT Core and other Cloud platforms support switching of keys or even using several at a time to smooth out the rotation and keep access outages at the minimum[28].

2.6 Device Initialization

Initialization is an important part or the lifetime of a device. I needs to be performed when connecting it to a network for the first time or after a factory reset was triggered on the device. There are several layers to initialization. First, the device needs to use some initial values and state which can either be pre-programmed or configured dynamically through some kind of communication. Next, the device needs to establish a connection to

[25]LEE, Jeffrey. Over-The-Air Firmware: The Critical Driver of IoT Success. *Hackernoon.* Dec, 2017. [online]. [cit. 2019-03-05]. Retrieved from: https://hackernoon.com/over-the-air-firmware-the-critical-driver-of-iot-success-f4604bd0b881

[26]GONZÁLEZ, Ana Rosa.The Importance Of OTA Updates For IoT Devices. *barbaraiot.* Jan, 2019. [online]. [cit. 2019-03-05]. Retrieved from: https://barbaraiot.com/articles/importance-ota-updates-iot-devices/

[27]Push an update for an Android Things product. *Android Developers* [online]. [cit. 2019-03-05]. Retrieved from: https://developer.android.com/things/console/update

[28]Device security. *Google Cloud* [online]. [cit. 2019-03-05]. Retrieved from: https://Cloud.google.com/iot/docs/concepts/device-security#key_rotation

whatever network it is connecting into and resources for the device need to be allocated and prepared in the rest of the system, which in the case of this thesis would be the gateway and the Cloud.

Among the most commonly used technologies for this purpose today are, for example, QR codes, RFID tags and NFC tags[29]. QR codes are two-dimensional codes that can be printed on a sticker and later scanned by a mobile application or some other kind of reader. It is a very simple and low-cost one-way communication solution. RFID is a technology using electromagnetic fields instead of image scanning and its effective range can be from a few centimeters to a kilometer. RFID can also be used as a one-way communication only. NFC evolved from RFID and is designed to only work at close range to improve security. Unlike RFID and QR codes, NFC supports information exchange between devices.

The research in this area is looking into even more ways to perform efficient initialization. One such concept, called BlinkComm, involves visible light communication [25]. The paper talks about the issues of initialization schemes used in today's products. In some cases, the task requires very complicated procedures or even a cable connection which is not very user friendly, especially with the large amounts of devices that are supposed to be part of the Internet of Things in the near future. They also discuss the security of different initialization methods and talk about several vulnerabilities in systems, which, for example, use preinstalled secret keys for their initialization phase, which is dangerous, because compromising one such device can endanger the whole network. The goal of the research was to present a solution that offers simple and secure initial configurations of IoT devices without the need to use any specialized hardware. They evaluate related solutions and calculate the theoretical capacity of the visible light communication channel and implement coding techniques that achieve more that three times faster speeds than the competition with minimal hardware requirements, such as one LED and one photodiode.

The same researchers who wrote the paper mentioned in the previous paragraph continued their visible light-based initialization research and invented two new multichannel key deployment schemes called LISA and LISAT [26], that make use of a light source device, like a smartphone, tablet or a multi-touch screen to securely interface with resource constrained devices. LISA stands for Light based Initialization and SMS based Authentication, LISAT for Light based Initialization and SMS based Authentication involving Trusted third party. The difference between them is that with LISAT, a user can initiate two wireless devices, one in their vicinity and the other in a geographically remote location, using their smartphone as a Trusted Third Party. The protocols use the visible light generated by, for example, a smartphone's screen to communicate information from the smartphone to the device that is being initialized and the device responds to the smartphone through encrypted sms messages. In LISAT the smartphone communicates with the remote device over SMS as well. The researchers claim that, while the protocols were implemented to use SMS, they can be adopted to support other technologies, such as LoRaWAN, Sigfox, NB-IoT or even BLE, Wi-Fi and NFC. The protocols were implemented on a commercially available platform and tested by 34 users who provided feedback the researchers plan to use to improve their future versions of the protocol.

[29] What are the differences between QR Code/RFID/NFC?. *Medium* [online]. [cit. 2019-01-13]. Retrieved from: https://medium.com/ucot/what-are-the-differences-between-qr-code-rfid-nfc-ucot-at-the-cutting-edge-of-nb-iot-communications-693769926dee

2.7 Power Consumption

Devices in the Internet of Things can generally be divided into two categories, when it comes to power consumption. There are the gateways and other computationally powerful units that consume to much energy to be sustained by batteries and thus require a constant power source and the more efficient and less computationally powerful end devices, which need to be wireless but at the same time have to last for months or even years without getting recharged. Therefore, there have been many studies and several different approaches [27, 28, 29] to how such end devices might be powered and how they can be made as energy efficient as possible while still providing the required services of IoT.

To make devices energy efficient, new communication standards and protocols have been invented, like the IEEE 802.15.4 and 6LoWPAN. Devices that communicate through technologies based on such standards and protocols conserve energy by transmitting limited amounts of data, decreasing the overhead of packets as much as possible, using a small transfer rate at small distances and lower bandwidth. They generally operate in peer-to-peer, mesh or star topologies.

Another important way of conserving energy is by using various sleep modes that turn off hardware components, so that the device draws energy only when necessary. The KW41Z MCU sold by NXP Semiconductors [27] for example, has four main STOP modes, which range from a stopped CPU with all I/O, logic and memory states retained and certain asynchronous mode peripherals operating to a powered down CPU with only I/O and a small register file retained and very few asynchronous mode peripherals operating, while the remainder of the MCU is powered down. The four main modes are Normal Stop, Very-Low Power Stop, Low-Leakage Stop and Very-Low-Leakage Stop.

By making devices energy efficient, it became possible to also focus on different strategies when it comes to powering them. One of these is energy harvesting [28], which harnesses energy from the environment or other energy sources and converts it to electrical energy. Some techniques can convert solar energy, while others use wind or mechanical energy, like when a mechanical stress is applied to piezo-electric materials or when a rotating arm is connected to a generator. Even humans themselves can serve as the power source through body movements, blood pressure, finger motion and other means. According to the cited paper, energy harvesting can be divided into two architectures - the Harvest-Use or the Harvest-Store-Use. The Harvest-Use based devices are powered directly by the harvesting system and can operate only while some form of energy is being converted. The Harvest-Store-Use devices can conserve the harvested energy and use it later. The paper discusses various aspects of energy harvesting systems, presents the basics of energy harvesting and existing nodes that use this technology and offer insights into the opportunities and potential applications of energy harvesting.

Wireless energy harvesting for IoT devices is explored in [29]. They call this approach the Wireless Energy Harvesting IoT or WEH-IoT systems. WEH-enabled sensor devices consist of an antenna, a transceiver, a WEH unit, a power management unit (PMU), a sensor/processor unit and optionally a battery. For energy harvesting, the two essential ones are the WEH-unit, which harvests RF energy and PMU that controls the device and manages energy consumption. The article also explores the enabling technologies. They classify energy sources into two categories, the dedicated sources, which are RF sources deployed to provide predictable energy supply to devices and ambient sources, among which are broadcast radios, TVs, Wi-Fi access points and others. Finally, they present and evaluate possible implementations of this system.

There are products on the market already that use different energy harvesting schemes. Among them are the Williot tags[30], which use Bluetooth and harvest ambient RF energy from cellular, Bluetooth, Wi-Fi and other 2.4GHz connectivity technologies, even Zigbee and Thread. These tags connect to Williot's Cloud service to provide secure authentication and sensor processing. Another energy harvesting IoT technology is the Zigbee Green Power[31], which allows battery-less Zigbee PRO devices by harvesting energy from light switches, piezo-electric elements and dynamo or electro-mechanic converters.

2.8 The Future of IoT

While the future of the Internet of Things might seem bright, Cisco estimates 50 billion connected devices by 2020 [30] and IDC predicts 80 billion devices by 2025[32], there are still many concerns they need to be solved. As microcontrollers get more energy efficient, machine learning becomes more advanced, communication protocols and connectivity technologies get more unified and standardized, IoT will quickly expand and most likely become an important part of every day life. However, while this future promises greater comfort and better lives, there is also the danger of large scale hacks and attacks on vulnerable and poorly secured devices and networks. If these dangers aren't solved in the near future, IoT can open the doors to a lot of harm, from privacy issues in people's homes, to health concerns with connected medical equipment and extremely large DDoS attacks endangering the whole Internet.

In its current state, the Internet of Things is far from ready to be adopted and incorporated into the lives of billions of people. However, as can be seen in the articles and publications cited in previous paragraphs, the research of its problems and the search for valid solutions is very intensive. Even MCU and device manufacturers[33] [34] are now realizing the mistakes of the last few years and are working on solutions to all of the aforementioned issues, especially edge computing and security of IoT networks.

It seems like the next few years will be crucial in convincing humanity that IoT will make the world a better, safer and more comfortable place to live in instead of a fully controlled environment that takes away their choice and is in danger of being overtaken and misused by harmful parties on a daily basis. After all, living in a house that saves energy, takes care of its inhabitant's needs and makes their lives more convenient in various ways is an attractive prospect but living in a house that is remotely controlled by someone who means to harm its inhabitants sounds more like a terrifying nightmare.

[30]HAZELRIGG, Jessie. FAQ. *Williot* [online]. [cit. 2019-05-01]. Retrieved from: `https://support.wiliot.com/hc/en-us/articles/360021588534-FAQ`

[31]The Green Power feature of Zigbee PRO allows battery-less devices to securely join Zigbee PRO networks, making it the most eco-friendly way to power a range of Zigbee products. *Zigbee Alliance* [online]. [cit. 2019-05-01]. Retrieved from: `https://www.zigbee.org/zigbee-for-developers/greenpower/`

[32]KANELLOS, Michael. 152,000 Smart Devices Every Minute In 2025: IDC Outlines The Future of Smart Things. *Forbes* [online]. [cit. 2019-01-13]. Retrieved from: `https://www.forbes.com/sites/michaelkanellos/2016/03/03/152000-smart-devices-every-minute-in-2025-idc-outlines-the-future-of-smart-things/#1cece5074b63`

[33]INTERNET OF THINGS Secure connections for a smarter world. *NXP* [online]. [cit. 2019-02-05]. Retrieved from: `https://www.nxp.com/applications/solutions/internet-of-things:Internet-of-Things-IoT`

[34]Internet of Things. *Qualcomm* [online]. [cit. 2019-02-05]. Retrieved from: `https://www.qualcomm.com/solutions/internet-of-things`

Chapter 3

Design

The goal of this thesis is to research existing literature and solutions and, based on the gathered knowledge, to create a functioning demo representing a smart home that solves the problems mentioned in the previous chapters. The thesis is made in collaboration with NXP Semiconductors and will therefore use NXP's technologies described below. The final product will consist of a model of a house with several battery powered wireless end devices and a gateway. The house will be connected to the Cloud for long term telemetry storage and remote control purposes. The design will aim to provide users with an easy means of device initialization, secure communication both locally and over the Internet, automation at the edge performed by the gateway and a mobile application allowing users to display data from the sensors and control the entire house remotely.

The individual devices and technologies were all chosen by NXP Semiconductors in order to show a complete solution made possible through their products. For this purpose, all software used for the end devices will be built on available NXP SDK bundles. However, the software for the gateway, the Cloud and the mobile application will have to be designed from scratch.

All hardware will be supplied by NXP, including any additional custom designs needed for the thesis' purposes. The model of the house will also be created in-house and all financial responsibilities regarding end device peripherals and Google Cloud services will be taken care of by NXP.

Since the hardware chosen for the gateway supports Android Things and can be used to promote the development of this operating system and because Android Things is made by Google, the solutions presented in the bachelor's thesis which this thesis is partially based on, that used Microsoft Azure, can no longer be applied here. Additionally, the bachelor's thesis' design was simpler. It consisted of fewer devices, used HTTP to send all data to the Cloud, didn't provide automation, security and easy initialization and did all data processing in the Cloud. This means that most of the practical solutions presented in the bachelor's thesis could not be repurposed here.

From an academic perspective, the research itself is of the utmost importance. However, most of the knowledge gained by this research can also be applied in writing application notes for NXP to supply their customers with in order to offer suggestions and advice on how they could develop their own Internet of Things systems also using NXP's technologies. And since a practical demonstration of what a student learned through their thesis is also important, NXP's and the university's goals go hand in hand.

This chapter will be separated into several sections corresponding to the different areas the thesis is trying to solve. Each section will propose a suitable design that will be used for the implementation.

3.1 Design Goals

The overall goal of the design was to create a complex system that solves many of the issues discussed in previous chapters and provides users with a functional and extensible IoT solution. It was important that no specialized hardware or internet services were required other than what is necessary for the chosen technologies.

Several main goals were set for the design of the autonomous system. It was important that it provides automation through a simple and easy to understand but at the same time effective and extensible protocol. While being as independent of the Cloud as possible, it was also necessary that it takes advantage of the Cloud's benefits like remote communication and long term data storage. To achieve smooth operation of such a system even when a connection loss occurs, it was also essential for the system to be able to automatically reconnect to the Cloud and resynchronize all of the device state information in a deterministic way that ensures that, as long as the Cloud connection is functional, both the Cloud and the gateway's data are up to date and consistent.

The main requirements for the Cloud were the means of both-way communication between the Cloud and the gateway and the Cloud and the mobile application, a way to process and store incoming data if needed, to have a way of maintaining device states and to enable the mobile application with remote device and gateway control. I was also important to have a way of remotely manipulating the utilized Cloud services through the mobile application without the need to use the online console.

For the sake of making the whole system secure and able to protect data privacy, a few goals were set for separate parts of the topology. There are four main connection routes that needed to be protected - the Thread communication, local communication between the gateway and the mobile application, the remote communication between the gateway and the Cloud and the remote communication between the mobile application and the Cloud.

The goals for the mobile application included a way of displaying current device states, latest reported telemetry, providing device control and a means of adding new automation conditions, an authentication method to connect to the Cloud and the gateway, remote communication with the Cloud and local communication with the gateway and device initialization capabilities both in the way of letting the gateway know about the new device and creating the needed resources in the Cloud.

There were three goals for device initialization. The first goal concerns connecting new devices to the Thread network. The next goal was to design a way of initiating device states and all the needed resources like MQTT clients and cryptographic keys at the gateway. The last goal was to create all needed resources in the Cloud and to store the MQTT client public keys so that the gateway can use its clients to securely connect to Cloud IoT Core.

Apart from being made by NXP, the end devices also needed to support the IEEE 802.15.4 communication technology or more specifically, Thread. They had to allow low power consumption and include an easily extensible SDK. Furthermore, several different sensors and actuators were needed to create the model of the smart home. The goal was to support a temperature sensor, a humidity sensor, a PIR sensor, a ventilator, an RGB LED light or matrix, a button and a buzzer.

Proposed Topology

The proposed topology for the whole demo includes a small network of end devices acting as a smart home, a single gateway that processes their data and takes care of both automation of the network and of connectivity to the Cloud. Google Cloud will receive and store data from the gateway and enable an Android based mobile application to connect to the system from anywhere in the world. The mobile application will be able to control the network, initiate devices and display their data and information.

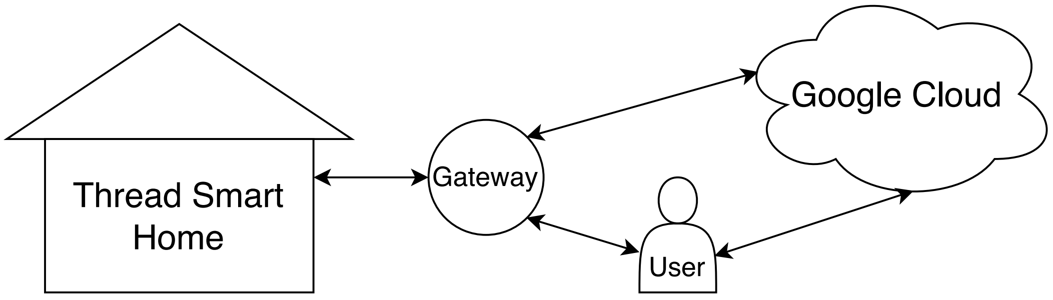


Figure 3.1: Proposed topology

3.2 Autonomy

In order to fulfill the goals set for the autonomy of the system, the design was inspired by the Edge Computing paradigm. All of the data processing and the computation of all device automation is therefore performed at the edge of the network. To make that work, an automation scheme was devised, which uses adjustable conditions stored inside device states in the gateway's permanent storage and based on them analyses all incoming data before passing it to the Cloud. Thanks to this approach the gateway is able to successfully keep controlling the network as needed even if the connection to the Cloud is severed.

Another aspect that needed to be looked into was a mechanism that would prevent important data from being lost. The created mechanism is based on message buffering and was designed with customizability in mind, in order to give users control over their data. It was necessary to provide not only short term solutions for networks with frequent disconnections but also to take care of prolonged disconnected states in cases where data history is of great significance and the local storage is insufficiently large. The gateway handles these cases by reducing the amount of stored data through several different schemes that can be chosen separately for each device.

Lastly, to eliminate the inconvenience of not being able to manually control devices while disconnected from the Cloud, offline communication with a nearby mobile phone was developed as well. For this purpose, the gateway makes use of its WiFi and Bluetooth capabilities to communicate with and receive commands from similarly enabled mobile phones. The result of this design choice is that the only cases where internet connection is absolutely necessary are the initialization of the gateway itself and the adding and removing of devices to and from the network.

To incorporate remote control, remote storage and pave the way for possible extensions to the design, the gateway application was envisioned with Cloud connectivity from the get go. This was approached by analyzing available Cloud platforms and making sure the design is as portable between them as possible. That's why the standard MQTT protocol was chosen for gateway-to-Cloud communication and why the gateway does not use any Cloud platform-specific APIs.

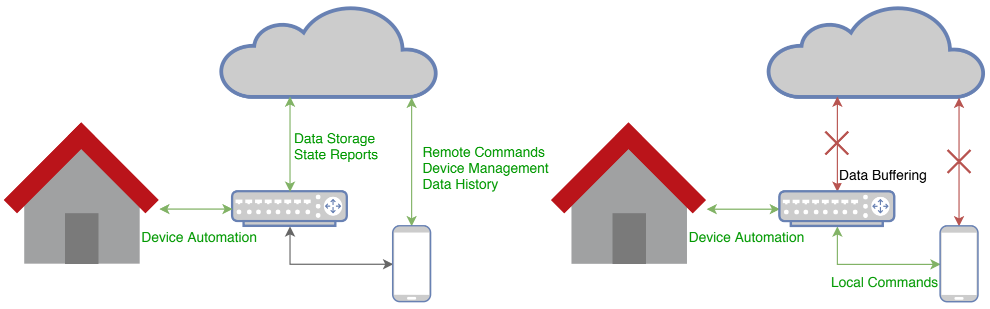


Figure 3.2: Functionality while connected and disconnected

In order to maintain an autonomous network, this thesis will focus on a few key elements - buffers, reconnection, data and device state consistency, a deterministic communication flow and offline data processing.

Reconnection Since the system will consist of a Thread network on one side of a gateway and an MQTT connection to the Cloud on the other, there are two separate connection types that need to be maintained. On the Thread side, when a known end device loses connection, it can simply try to reconnect back and when successful, send any initializing information required, like the last configuration state of the device.

The gateway to Cloud connection will have to be more complicated than that. The gateway has to maintain an MQTT client for every end device. Any data that comes from an end device and needs to be sent to the Cloud is published through these clients and any configuration or state change requests from the Cloud are subscribed to by the clients. When one of the clients loses connection, the way it reconnects has to be deterministic in order to keep all states and data consistent.

There are two main ways a client can disconnect - either by being closed from the Cloud side or by the gateway losing connection to the Internet. In both cases, the first action before attempting reconnection is checking if the device represented by the client is still alive by sending it a CoAP request. The result of this check is stored in the device's state before the reconnection process begins and the MQTT client is reconnected regardless of the result.

If the connection was closed from the Cloud side but the gateway is still connected to the Internet, the MQTT client's credentials are refreshed and it attempts to connect to the Cloud's MQTT broker again. If successful, it can then resubscribe to it's configuration channel as well. After establishing the connection and subscription, the client sends its device's latest state to the Cloud, thus preserving consistency. Should the client fail to reconnect however, it will stay offline until its device either needs to send new data to the Cloud or a state change has to be reported. When that happens, another reconnection will be attempted, if the gateway is connected to the Internet, and the process will be the same as it was immediately after the disconnection.

If the reason was an Internet connection loss, the client will neither attempt to reconnect right after the disconnection, nor when a new message needs to be sent to the Cloud. Instead, the gateway waits for its Internet connection to resume and when that happens, it goes through all of the known MQTT clients, reconnects them, applies any configuration requests from the Cloud and synchronizes device states with the Cloud, sends any buffered data and then resumes regular operation.

With this design, all reconnection handling is deterministic and will not cause any inconsistencies between the Cloud and the gateway's data.

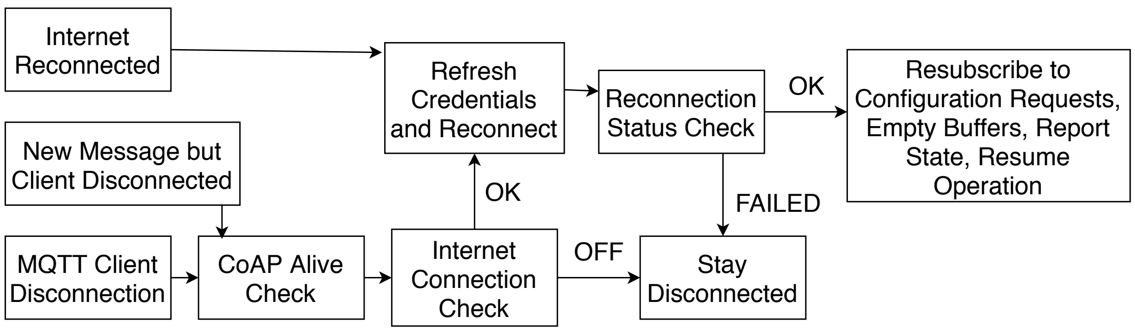

Internet Reconnected
New Message but Client Disconnected
MQTT Client Disconnection
CoAP Alive Check
Internet Connection Check
OK
OFF
Refresh Credentials and Reconnect
Reconnection Status Check
OK
FAILED
Stay Disconnected
Resubscribe to Configuration Requests, Empty Buffers, Report State, Resume Operation

Figure 3.3: Reconnection flow

Data buffering Even though most data processing will be done at the edge, some devices, like telemetry sensors, still need their measurements to be stored in the Cloud for further analysis. All of the data can be published to the Cloud through the device's MQTT client, but in the event of disconnection, the data can either be discarded or buffered and resent at a later time.

This thesis proposes three main types and one subtype of buffers that could be used for such purposes. The simplest is of type HOLD. Once full, this buffer will stop accepting any new data and will simply keep the data it has gathered so far. Another type is FIFO, which, when filled, will start acting as a FIFO pipe that discards the oldest message and accepts the newest one. The third type is DYNAMIC. Once DYNAMIC buffers are filled up, they will first clear up space by only keeping messages with a specified time interval between them. After this initial reduction, the buffer will only accept messages that maintain this time interval spacing. After filling up for the second and every next time, the time interval will be increased by a specified amount of time and the process will repeat itself. The last buffer type is VARIANT. VARIANT is a subtype of all HOLD, FIFO and DYNAMIC. When used, the buffer will only store data with telemetry values that are larger or smaller, by a specified amount, than the last stored value. Once a VARIANT buffer is full, it will either keep or start removing messages based on its main type.

In order to keep data consistency between the gateway and the Cloud, buffers need to be checked and cleared out every time an MQTT client connects to the Cloud. Once the client connects, it goes into a syncing state, during which it publishes all buffered data before it starts publishing any new incoming data from the Thread network. This way everything will be sent in the correct order, keeping consistency.

Communication flow The communication flow will be mostly dictated by the CoAP and MQTT communcation protocols. After establishing connection of a new device into the network, its main function can be initiated by a CoAP POST command. Similarly, whenever a new configuration request is either received from the Cloud or generated by some data processing rule at the gateway, a CoAP POST command will be sent to the affected device. The end devices themselves will also use CoAP POST messages during initialization and when reporting their data, if there is data that needs to be periodically reported, like telemetry sensor data. The gateway might also use CoAP GET requests during device initialization and when checking if a device is alive.

Communication between the gateway and the Cloud will all be done via a Cloud based MQTT broker. The gateway will publish telemetry data and state changes resulting from configuration requests and data processing. The broker will then pass on the messages to the rest of the Cloud system for further processing or storage. Configuration requests generated from the Cloud, or through the Cloud by a mobile application, will be subscribed to by the gateway. When a gateway receives a new configuration request, it will have to check its time stamp to make sure states are changed consistently and afterwards can perform any operations that are required. Additionally, every connected MQTT client must publish periodic heartbeat messages to the broker.

Offline data processing For the offline data processing, a rule based system was devised. This system will use JSON strings encoded into messages sent and received through the MQTT broker. There will be three types of JSON strings - a data string, a state string and a configuration string.

The data strings will be very simple - they will contain the ID of the device that measured the data, the time stamp of when the data was received at the gateway and the value of the data.

The state strings will contain information about their device, like its ID, its type, what configurable states the device has, when the last configuration request for this device was received, whether it uses a buffer and what type the buffer is, whether it's alive or dead and what data processing rules are assigned to it.

The configuration strings will be requests received from the Cloud and will contain the time stamp of the request, the type of request and the requested actions. One of these types will be the data processing rules.

Every device will be able to have a set of data processing rules. Each rule will consist of a list of affected devices, a control condition and an action. The conditions will define circumstances under which the incoming data will cause the actions to be taken. For example, one such condition might be a value limit for a temperature sensor. The actions will then be configuration requests that, if the condition is met, will change the states of the devices included in the list. Even the device itself that reported the new data could be included in the list.

3.3 Security

The Thread stack comes with several security mechanisms that are handled automatically. The Commissioning White Paper [31] describes them in a basic overview. The fundamental security employed by the Thread networks is based on an elliptic curve variant of J-PAKE [32]. J-PAKE is a password-authenticated key exchange and essentially uses an elliptic curve Diffie-Hellmann for key agreement and Schnorr [33] signatures to authenticate two nodes and create a shared secret between them based on the passphrase. Additionally, DTLS is in development, which is a version of TLS suitable for UDP networks. Thread is also protected with a network-wide key used at the MAC layer, which protects the 802.15.4 data frames. It serves as a basic form of security used to prevent eavesdropping. However, since it is network-shared, compromising a single device also compromises the whole network, which is why it typically is not used as the main security feature of Thread networks.

Lastly, new joiners need to authenticate through a commissioner in a controlled manner to allow the DTLS handshake authentication protocol to be performed.

In the case of the local communication between the gateway and mobile applications, a scheme using cryptographic keys for authentication and encryption was devised. The gateway and the mobile applications each maintain their own public-private key pair. The gateway shares its public key with every connected mobile phone and the phones share their keys with the gateway. Both the owned private and public key pair and the shared public keys are then permanently stored in a secured storage. The keys then can be used to authenticate both sides and to encrypt any messages sent between them which means that only registered phones can control the gateway locally and any messages that get listened to by malicious parties are kept private.

Since the MQTT protocol was chosen for the communication between the gateway and the Cloud, their messages can be secured and kept private through a very similar mechanism. The Google Cloud IoT core requires authentication through cryptographic keys and stores a public key for each connected client. The MQTT communication additionally uses the TLS 1.2 protocol to keep the communication private and secure.

In order to also maintain a secure access of the mobile application to the Cloud, a slightly different approach was taken with the use of Google Firebase. Firebase offers user authentication through e-mail accounts and an Android API that can be used to for logging in and out of the system. By using this service and API, the security and privacy is taken care of by Google's own algorithms.

Furthermore, Google Cloud keeps special API service keys that can be used to protect access to the individual services inside the platform. Thanks to these, even the communication within the platform is secure and kept private to the specific project that contains the used services.

3.4 Cloud Processing

Google Cloud offers Cloud IoT Core for the purposes of either HTTP or MQTT device communication. Since the initial stages of this thesis, Cloud IoT Core has undergone several changes and has evolved into a more flexible solution. It can maintain a history of state changes and configuration requests, pass incoming data into other services through Cloud Pub/Sub, send one-time commands, keep track of each connected device with a separate MQTT client and can even use a client for the gateway itself. It can also be controlled through HTTP commands, that can be used to create or upload resources into the service, create and remove device clients or get information about the clients.

For the purpose of data processing, several different services can be used. Among those are Cloud Functions and Cloud Dataflow. While Dataflow offers an easier-to-use and potentially more efficient way of controlling the flow of and processing data, it is also far more expensive than Cloud Functions. Cloud Functions on the other hand require manual writing of code but can be used at little to no charge for a project the size of this thesis. Moreover, since the developer is in complete control over the code, the Functions can also offer more flexibility.

Google Cloud can facilitate data storage through several services that all serve slighty different purposes. The requirements for this project were the means to have the data accessible through some remote API, the ability to store large amounts of data and for the storage to be free or very cheap. Among the major choices of storage services Google offers are Bigtable[1], BigQuery[2] and Firestore. Bigtable is a fast and powerful fully managed NoSQL database. It offers seemless scaling and replication of hundreds of petabytes of data with millions of operations per second. It can integrate with other services like Cloud Dataflow or tools like Hadoop and is the very database that powers a lot of Google's most popular services like Gmail, Maps or Search. BigQuery is a serverless, highly scalable and cost-effective data warehouse. It supports SQL queries and thus provides powerful analytic tools. BigQuery can also be directly connected to Google's DataStudio, which is an online data visualization tool. Firestore is an evolution of Cloud Datastore. It is another NoSQL database that is organized into collections of documents, integrates with Firebase, is very cost effective and supports efficient and flexible queries. While Bigtable would be unnecessarily powerful and expensive for this project, BigQuery and Firestore both satisfy the needs of the thesis with different pros and cons. However, Firestore can be integrated with Firebase which was chosen for user authentication and management and Google offers an Android API for Firestore access which is why Firestore was the final choice for the storage.

Firebase Functions are an extension of Cloud Functions and offer authenticated HTTP callbacks that can be called through an Android API. This allows the mobile application to communicate with the Cloud in a secure and private way while also protecting the Cloud and the network from unauthorized user access. The functions otherwise work the same as Cloud Functions and can be used to manipulate and talk to other Cloud services.

3.5 Mobile Application and Operating System

The application design was divided into several sections. The first one is the main overview of all connected devices, their latest reported telemetry or most important feature, like the current color of an RGB LED light. It also includes buttons to easily turn the devices on or off and a button add a new device to the network. Next is the detail section, which offers a detailed look at a single device with all of the state information and controls available for it. Another section is the visualization section for telemetric devices that displays a graph of past data reports. The next section is the overview of device groups that the application can keep to make creating conditions more user-friendly. An overview of currently used conditions is among the sections as well. Both the group and condition overviews also allow adding new groups and conditions to the system. The last section is the gateway control section, which allows to send commands to the gateway like adding a new user or initiating the gateway.

It was also important to allow automated refreshing of data but to also have an option to do it manually and limit data consumption. Therefore two background tasks were also design to take care of both Cloud and gateway communication.

Android was selected for the mobile application. Android is currently the most popular mobile operating system and NXP provides SDK Bundles for NFC communication with their NFC tags. Android can also easily interface with Google Cloud and can communicate with Android Things over Bluetooth and Wi-Fi through Android specific APIs.

Android Things was chosen as the operating system for the physical demo. It provides the strengths of Android development and was designed with security and scalability in

[1]Cloud Bigtable. *Google Cloud* [online]. [cit. 2019-04-21]. Retrieved from: https://Cloud.google.com/bigtable/

[2]Cloud BigQuery. *Google Cloud* [online]. [cit. 2019-04-21]. Retrieved from: https://Cloud.google.com/bigquery/

mind. While there are some issues with missing drivers that complicate development, overall it is a suitable operating system.

3.6 Device Initialization

The Thread stack supports two ways of commissioning a new device to allow it to join the network [31]. These are the external and the native commissioning. When using an external commissioner, the commissioner cannot directly connect to the Thread network and thus has to communicate with it through a border router. It first needs to register with one of the network's border routers to ensure that only one commissioner is present in the network. After establishing the connection through Commissioner Credentials and a DTLS handshake, the commissioner can start adding devices. In the case of native commissioning, the commissioner can be a part of the Thread network and can register itself in the network through a commissioner router. The Thread Group offers an open source mobile application that can be used for Thread commissioning through BLE or NFC communication. However, adding BLE/Thread cooperation would be outside the scope of this thesis' design goals and NFC requires additional hardware. Additionally, as described later in this document, there were issues with USBNET drivers in Android Things which could further complicate the border router communication. Instead, a different solution was designed by using the already secure communication between the mobile application and the gateway, QR codes and adding of device credentials through UART communication between the gateway and the border router dongle.

After allowing the device to connect to the Thread network, the gateway creates a public-private cryptographic key pair, pass the public key to the mobile application and start creating the device state and MQTT client resources. After the device connects and the state for it is created, the gateway then waits for an incoming confirmation of device initialization from the Cloud through a dummy MQTT client and then connects the device's MQTT client. After all that is complete, the device has been initialized at the gateway.

When initiating a device, the mobile application first needs to be used to let the gateway know that a new device is being connected and then has to wait for the device's public key that will be sent from the gateway. Once the public key is received, the application can pass the key and device information to the Cloud, where all the needed resources in Cloud IoT Core will be created so that the gateway's MQTT clients can connect. Once the Cloud is done initializing the resources, it sends a confirmation message to the gateway over the dummy client.

When deleting a device, a similar approach is taken, however instead of the gateway waiting for the Cloud's confirmation, the Cloud waits for the gateway to let it know that all device information was removed successfully. First, the mobile application sends a remote request over the Cloud to the gateway to delete a specific device, then the gateway clears all known records of the device includíng the buffers, keys and the MQTT client and then it sends a confirmation message over the dummy client to the Cloud. After this confirmation is received, it triggers a function that clears all the records of the device in the Cloud as well, including the data stored in Firestore.

3.7 Hardware

The requirements for hardware were a little more specific then the other parts of the thesis. As mentioned in previous paragraphs, the goal was to use NXP's technologies for the end devices and the gateway.

End devices As it satisfied all the requirements, NXP's KW41Z microcontroller was chosen for the end devices and a dongle that provides Thread connectivity to the network's gateway. NXP offers free SDK bundles for the KW41Z that include demo applications and documentation for devices using Thread, Zigbee 3.0 and Bluetooth Low Energy. The dongle will be a standard USB-KW41Z dongle sold by NXP, the end devices will consist of custom modules integrating sensors or actuators and rechargable batteries.

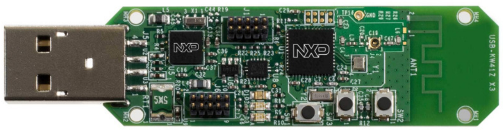


Figure 3.4: NXP's USB KW41Z Dongle[3]

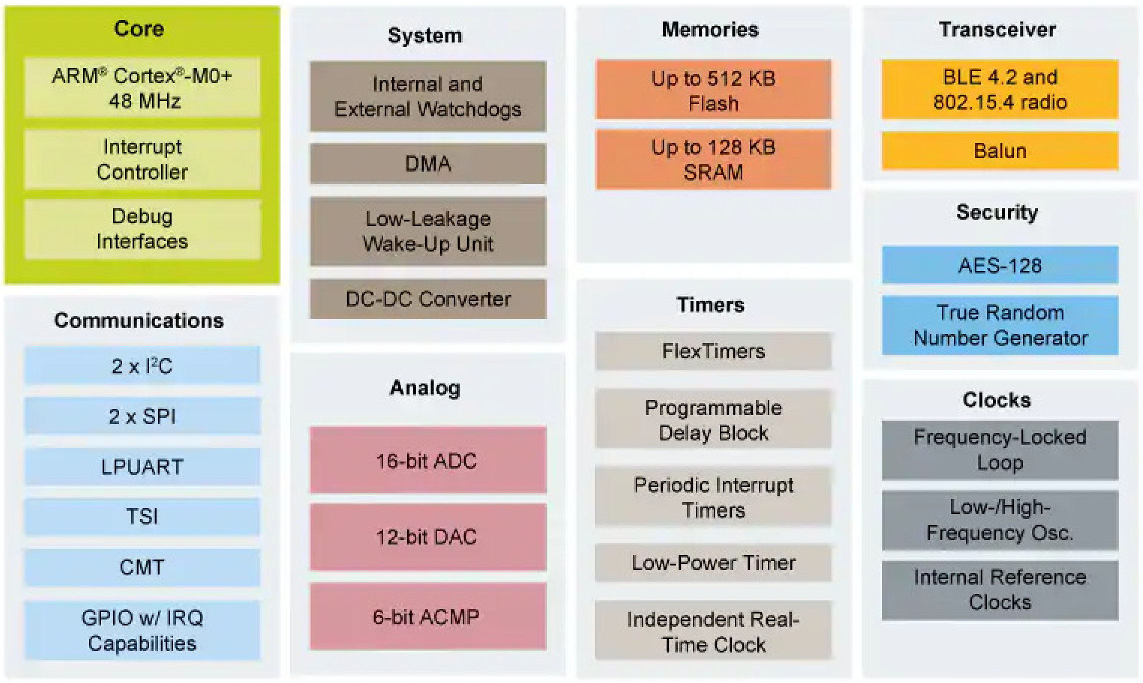


Figure 3.5: Kinetis® W Series KW41Z MCUs Block Diagram[4]

[3]USB-KW41Z. *NXP* [online]. [cit. 2019-01-14]. Retrieved from: https://www.nxp.com/assets/images/en/dev-board-image/USB-KW41Z-GS-BOARD.JPG

[4]KW41Z: Kinetis® KW41Z-2.4 GHz Dual Mode: Bluetooth® Low Energy and 802.15.4 Wireless Radio Microcontroller (MCU) based on Arm® Cortex®-M0+ Core. *NXP* [online]. [cit. 2019-01-14]. Retrieved from: https://www.nxp.com/assets/images/en/block-diagrams/KINETIS-KW41Z-BD.jpg

Gateway The platform for the gateway had to be sufficiently powerful in order to be able to run an operating system and handle all the necessary computation and data processing. Therefore, the NXP Pico i.MX7D was chosen for the gateway implementation. It can connect to the Internet both through a Wi-Fi module and an Ethernet port, has a Bluetooth module and is powerful enough to be a suitable gateway device. It is also one of the support development boards for the Android Things operating sytem.

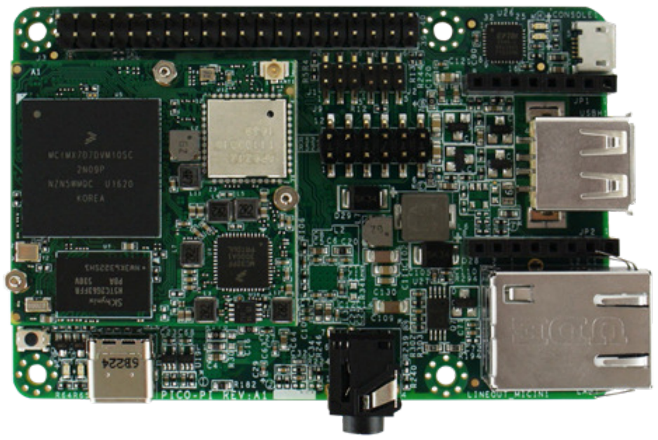


Figure 3.6: NXP Pico i.MX7D[5]

Power Consumption

From the available deep sleep modes [34] for the KW41Z MCU, the Low Leakage Stop 3 (LLS3) mode was chosen. It is a state retention power mode, in which most peripherals are in state retention, cannot operate and their clocks are stopped but several low power timers can still be used. A low leakage wake up is used to wake the sleeping MCU up. All SRAM contents, the Register File, I/O and oscillator states are retained as well.

The power efficiency was focused mainly on the temperature, humidity and PIR sensor devices. The buzzer device cannot go to sleep while it's buzzing, but does enter the sleep cycles while idle, and the button never sleeps at all. The LED devices were based on the FRDM-KW41Z development kits instead of having custom modules built for them and are used as Router Eligible End Devices and powered directly through a USB cable instead of the battery used for the other device types, which means they never go to sleep either.

[5]NXP Pico i.MX7D Board. *Android Developer* [online]. [cit. 2019-01-14]. Retrieved from: `https://developer.android.com/things/images/nxp-pico7-board.png`

Chapter 4

Implementation

4.1 End Devices

The software for the end devices was based on the C-based NXP SDK for KW41Z and developed in the IAR Embedded Workbench. Originally, the Thread wireless end device example was used, later the Thread wireless low power end device. The SDK provides a very complex but extensible application and takes care of all the underlying Thread communication, hardware access and control and offers a simple example of how an end device could work. It includes various configuration files that were edited so that the devices behave the way this thesis needed them to, like setting the unique device passwords and MAC extended addresses, making the polling interval for incoming messages as well as the sleep length be 3000 ms long, changing the default network ID, channel mask, turning various features on or off etc.

Several CoAP callbacks, processing functions and a device configuration file were created for the functionalities of the device types. The device configuration file includes the choices of which type the device is, what its ID is and what its configurable states are. The device ID must be based on the MAC address of the device, for example, when a device is assigned a MAC address of 0x146E0A0000000001, its ID must be the string 146E0A0000000001. This is because of the initiation protocol that will be described later in this document. The CoAP callbacks and their related functions can be sorted into the categories described in the following paragraphs.

Temperature (Temp), Humidity (Hum) and PIR Sensors These three sensor types share the same scheme of functions with the difference between them being the way of reading the desired value and formatting it into a CoAP message. Each type has a APP_Report[Type] function, that is used to start the automated reporting cycle. It calls APP_SendDelayed[Type]Handle handler, which first calls the APP_Send[Type]Handle immediately and then starts a loop with the configured reporting period that calls APP_Send[Type]Handle repeatedly until StopReport[Type] is called. The handlers are required, so that the actual processing functions are not called directly through function calls and their requests can be inserted into a message queue used by the SDK, which takes care of the calls instead. The APP_Send[Type]Handle simply places the APP_Send[Type]Cb callback into the message queue. When APP_Send[Type]Cb is called, it reads the current sensor value, compares it to the value that had been read the last time this callback was called and if the values differ or if the maximum amount of time between reports has passed, is calls the final APP_SendCoap[Type] function, which takes care of sending the value through a CoAP message to the gateway. Additionally, CoAP callbacks for each type are registered, so that a CoAP GET request for the desired type can be received and responded to, however, it was decided not to use this feature in the final implementation.

RGB For the RGB type, a CoAP callback is registered that reacts to bot GET and POST requests. It responds to GET requests with the current RGB state. The POST request messages are used to change the current color of the available LED.

Buzzer The buzzer also uses the handlers described in the paragraph with the temperature, humidity and pir sensors. These handlers are called from the function APP_BuzzerControl, which is used to either start or stop the periodic calls to the Buzz_toggle function, through the handlers, which toggles the value in the buzzer's input pin between one and zero, which causes the buzzer to make noise.

Button When the button is pressed, the APP_CoapButtonPressed function is triggered, which sends a CoAP message to the gateway telling it that the button was pressed.

ID There are two functions connected to the device ID. The first one is called AnnounceID and is always called when the device connects to a Thread network. It announces the device's ID to the gateway. The second one is a CoAP callback APP_CoAPGetIDCb, which can be called with a GET request to get the device's ID.

Type, Alive and Configurable States Three simple CoAP callbacks take care of responding with either the device's type, configurable states or just a confirmation message saying that the device is alive and works fine.

OnOff An OnOff CoAP callback is also registered and it calls functions that either start or stop the device's main function, like reporting telemetry or buzzing.

Update Interval The CoAP callback for the interval can be used for both GET messages, to get the currently configured report interval value and for POST messages to set a new one.

Furthermore, certain changes were required to be made to the pin configurations so that they reflect the wiring of the sensors and actuators. Most of the pins on the module are also turned off for the whole lifetime of the device. Depending on the device's type, some pins are only turned off while the device is asleep. The same goes for the ADC, which is only initialized if the device uses a temperature or humidity sensor and then is deinitialized every time the device goes to sleep.

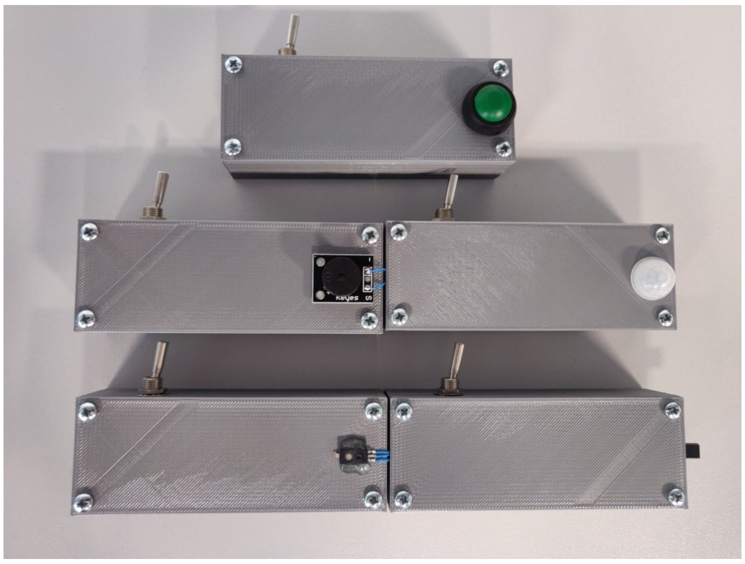


Figure 4.1: Custom devices - Button, Buzzer, PIR Sensor, Humidity Sensor, Temperature Sensor

4.2 Gateway

The gateway application was developed for Android in the Android version of Java. The Android Studio IDE was used for development.

Device states, buffers and conditions

The gateway maintains device states for all the connected devices in the network. These states are permanently stored on the gateway as well as uploaded to the Cloud. The states are stored in the form of JSON strings, which provide a human-readable and easily extensible format that can be parsed by many different libraries in commonly used programming languages.

The states contain several items: the device ID, device alias, device type, different configurables, syncing, life, initialized, buffer and controls. The device ID is a static unique identifier preprogrammed into the device. The alias is a name for the device assigned by users through the mobile application. The device type is also static and preprogrammed into the device and specifies the capabilities of the device. The syncing state is used for synchronization purposes, life contains information on whether the device is responsive or not and initialized is used during the device initialization to prevent uninitialized devices from being connected to the Cloud.

Among the available configurables are onoff, which in most cases turns the device on or off, except for the garage type, which uses it to open and close the door. Others include interval, which sets the report interval for telemetry devices and rgb that sets the rgb value of an rgb led device.

The configurable states are dependent on the device itself and its capabilities. Temperature, humidity and PIR devices use the interval and onoff. The ventilator, buzzer and garage use only onoff, button has no configurable states at all and RGB has rgb and onoff.

The buffers can either be initialized to specific values and modes or use the preconfigured default ones. Additionally, these configurations can later be changed through corresponding methods. The values include the maximum number of messages, the length of the spacing interval required for the messages, the value by which the spacing interval gets incremented when the buffer is filled and the variance threshold value. As described in earlier chapters, the buffers operate in 3 main modes, HOLD, FIFO and DYNAMIC with one submode, VARIANT.

When storing a new message, the application first checks if the buffer is full or not. If not, then depending on the buffer's mode, it stores the message at the end of the buffer or simply drops it. When space is available, in the HOLD and FIFO modes, the message is always stored. In the DYNAMIC mode, if the buffer is completely empty, the message is always stored, otherwise the timestamps of the last stored message and of the new message are compared and if the difference is at least as long as the specified spacing interval, the new message is stored, otherwise it is discarded. For the VARIANT submode, if the buffer is being filled for the first time, the messages are stored normally according to the main modes. However, if the buffer was full at least once already, the values of the last stored message and the new message are compared first, and if the difference between them is at least as large as the variance threshold, then the application proceeds according to the main modes.

If the buffer was full when the new message was requested to be added, the application behaves differently. In the HOLD mode, the new message is discarded. In the FIFO mode, the oldest message is discarded and the new one is stored. In the DYNAMIC mode, the spacing interval is incremented by the configured increment value and the application goes through the whole buffer, starting from the oldest message, and discards messages so that the buffer complies with the new spacing interval again. It repeats this process until at least one message was discarded, then stores the new message at the end, if the spacing interval allows. In the VARIANT submode, the application examines the buffer from the oldest messages to the newest and keeps only the messages that are different from both their direct neighbors inside the buffer by at least the specified variance value. Afterwards, the application proceeds with the main mode again.

Device Management (CoAP Clients)

For the purposes of receiving messages, a CoAP server with several resources was created at the gateway, one for each supported device type that reports data and one for initialization. For sending messages, the gateway holds one CoAP client for each resource, one for the MQTT request processor and one for the offline communication request processor.

The CoAP resources are the following: ID, Temperature, Humidity, PIR and Button. Each resource, except ID, is based on a similar algorithm and is separated into two stages. The first stage accepts the CoAP message, checks whether the device's MQTT client has an up-to-date authorization token and refreshes it if needed, marks the sending device as alive (and if it was dead, reports this change to the Cloud) and starts the second stage. In the case of the ID resource, the gateway tries to load the devices stored state and if that fails, starts a communication chain with the device to gather all the needed information and create a new state record. If there was a state stored from a previous session, the device is configured to the loaded values.

For all the resources other than ID, the second stage focuses on processing the received data according to all control conditions configured for the sending device. Each device type supports different condition types, which is why resources have differences in the second stage. Conditions of incompatible types are ignored. The Temperature and Humidity resources support the limit control type, which checks whether the reported temperature is above, below or equal to a desired value. The Button type supports the button control type which can operate in three different modes - an onoff mode, which turns the device affected either on or off, based on the condition's configuration, a toggle mode that turns off active devices and turns on inactive devices and an rgb mode, which sets the color of a RGB device to the configured value. Finally, the PIR type supports the detect control type, which can work in three modes as well. These are the security, regular and combined modes. The security mode generates a configured alert action when movement is detected. The regular mode generates separate configured actions for when an activity is detected and when not and the combined mode can do both, depending on whether the reporting device is in the secured or unsecured state.

After deciding the required actions, the function cycles through all the target devices of the condition and sends them the needed configurations in CoAP messages. Any changes to the device states are then reported to the Cloud through MQTT messages.

Cloud Communication (MQTT Clients)

Cloud IoT Core is a Cloud-based MQTT broker provided by Google. The gateway maintains an MQTT client for every device in the network and one additional dummy client for special Cloud messages and connects them all to Cloud IoT Core. The connected clients post their telemetry to the event topic and state changes to the state topic. The dummy client uses the event topic to send its special messages as well. All clients except the dummy subscribe to the config topic, the dummy subscribes to the command topic. The config topic's messages are retained in the Cloud and whenever a client connects to Cloud IoT Core and subscribes to config, it receives the last stored config message. The command topic does not retain any messages.

A callback for connection changes and incoming messages is registered for every connected MQTT client. When a client loses its connection to the Cloud, the gateway first checks if the client's device is still alive through a CoAP message and if it doesn't receive a response, marks the device as dead by setting the appropriate value in the device's state. Next, the client checks the gateway's internet connection. If the gateway is still connected, the client tries to reconnect to the Cloud regardless of whether its device is alive or dead. Once successfully reconnected, the client sends its current state to the Cloud. If the gateway is offline or if the reconnection failed, no further actions are taken.

When a new MQTT message arrives, it is first acknowledged and then processed. Since the messages are used to change device states, except with the dummy client, they are ignored when a device's state is uninitialized. Otherwise, the client checks the message's configuration timestamp and discards all messages that are older than the last configuration performed for the device. Next, the client devices what to based on the configuration's type. Eleven types are supported. The types used for device control interval, onoff and rgb for direct controls of the end devices, opmode and secure for different operational modes for the PIR sensors, controls to manipulate the control conditions of the device, buffer to change its buffer settings, alias to set the device's display name, button_press to allow the mobile application to virtually press the connected buttons. The types used for the dummy communication are add for adding new devices and remove for removing them.

When a direct control message is received, the client tries sending a CoAP configuration message to its device. If the change is acknowledged, it is marked in the device's state at the gateway. The opmode, secure, controls, buffer and alias only cause state changes at the gateway but do not generate any CoAP messages. All state changes resulting from these configuration messages are reported to the Cloud.

Connection management

Apart from the connection callbacks registered for each MQTT client, there are two more ways the gateway manages connections. While the gateway is online, a background thread is active that periodically tries to reconnect all disconnected MQTT clients. Furthermore, an internet connection callback that catches all connection state changes is also registered. When the connection is lost, the background reconnection thread is stopped. When the connection is regained, all MQTT clients are reconnected, all data buffers are emptied, the last configurations are received from the Cloud, all current states are reported and the background reconnection thread is reactivated.

Offline Communication

The offline communication can be divided into three sections - user management. gateway management and device management. The user management is important to keep the system secure and private whereas the device management enables the whole system to remain functional even when offline. The gateway management allows the user to initiate the gateway.

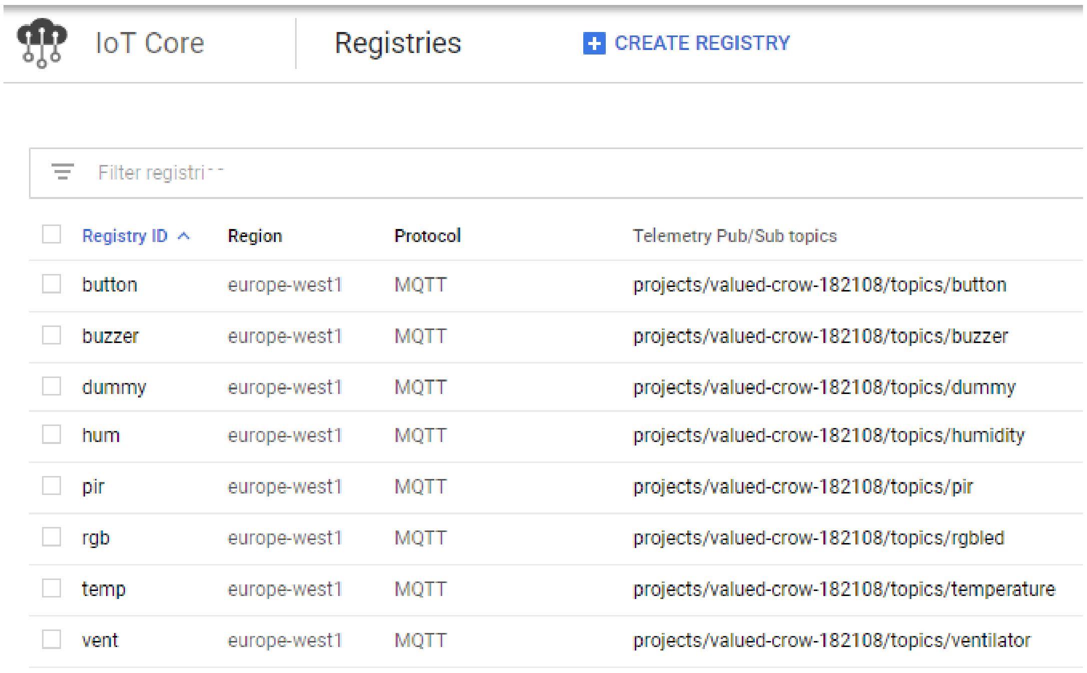
When the gateway has no registered users, anyone can send their credentials and register their phone. This first registration is neither secure nor encrypted, the gateway simply saves the phone's public key and user ID and replies with its own public key. All subsequent communication is both secure and encrypted by using these keys. When a new phone wants to register but the gateway already has at least one user, the registration is rejected and has to be performed through one of the already registered user's phones instead.

When a message is received, the gateway checks if it is encrypted and if not, rejects it by replying with UNAUTHORIZED. Afterwards, the message's signature is examined and if wrong, it's rejected as well. There are three message types used for user management - these allow to add a user, remove a user and request a list of currently registered users. Device management offers two types - the offline message, which is then processed just like an MQTT message would be, and the update request, which is replied to with all current device states. The gateway management can be used to configure the Google Cloud project ID on the gateway and to start the gateway's main functionality.

4.3 Google Cloud

Various services in the Google Cloud were used to ingest, process and store data and device states and to pass this information along to the mobile application and to allow remote commands sent from the mobile application over the Cloud. Among these services are Cloud IoT Core, Storage, Cloud Functions, Pub/Sub and Firestore. Additionally, Firebase was used for user authentication and login and to allow some of the Cloud Functions to be called from the mobile application.

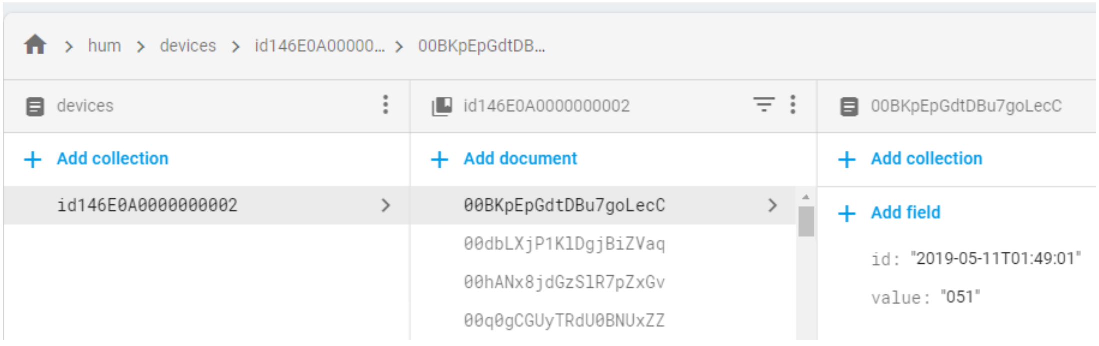
A device registry for each supported device type plus a registry for the dummy device are maintained in Cloud IoT Core. When a new device is added, it is added to its respective registry. The structure of Cloud IoT Core is similar to a file system, as can be seen in the figure 4.2.



Registry ID	Region	Protocol	Telemetry Pub/Sub topics
button	europe-west1	MQTT	projects/valued-crow-182108/topics/button
buzzer	europe-west1	MQTT	projects/valued-crow-182108/topics/buzzer
dummy	europe-west1	MQTT	projects/valued-crow-182108/topics/dummy
hum	europe-west1	MQTT	projects/valued-crow-182108/topics/humidity
pir	europe-west1	MQTT	projects/valued-crow-182108/topics/pir
rgb	europe-west1	MQTT	projects/valued-crow-182108/topics/rgbled
temp	europe-west1	MQTT	projects/valued-crow-182108/topics/temperature
vent	europe-west1	MQTT	projects/valued-crow-182108/topics/ventilator

Figure 4.2: IoT Core Structure

Firestore stores collections of documents and can be structured by chaining them as collection > document > collection > document and so on. Using this feature, Firestore keeps a collection for humidity and a collection for temperature. Inside each collection is a document that holds collections of devices and these collections hold individual reported values and timestamps inside documents.



id146E0A0000000002

00BKpEpGdtDBu7goLecC
00dbLXjP1KlDgjBiZVaq
00hANx8jdGzSlR7pZxGv
00q0gCGUyTRdU0BNUxZZ

id: "2019-05-11T01:49:01"
value: "051"

Figure 4.3: Firestore Structure

Firebase authentication allows adding and removing of users through the Firebase console and setting the supported sign-in methods, like a Google account, an e-mail, a mobile phone number or various other accounts. The methods chosen for this project were the Google account and e-mail accounts.

Several node.js functions were created for the purposes of processing the incoming data to the Cloud, communicating with the mobile application and manipulating of Cloud resources.

There are three telemetry trigger functions, one for temperature, one for humidity and one for PIR data. All three of them receive the incoming data and store it into a bucket in Cloud Storage as the last telemetry receieved from the particular device. Moreover, telemetry and humidity values are stored in Firestore documents.

Another trigger function that is called when a message arrives at Cloud IoT Core is the dummy trigger. This function receives the ID of a removed device as confirmation that all data about this device had been deleted at the gateway. Upon receiving the confirmation, the function takes care of removing all allocated resources in Cloud IoT Core and Cloud Storage and documents from Firestore for the particular device.

Finally, four HTTP Firebase functions were created as well. These can be called from the mobile application, if the user is authenticated and has access to the Cloud project. The functions are for adding and removing of devices, changing a device's configuration and getting a device's current state and last reported telemetry. All four functions start with loading the service key that is used to authenticate communication between services in Google Cloud. Afterwards, the key is used to generate an access token that can be used in signed JSON Web Token messages. Next the JSON message is constructed, the token is included and the message is sent over HTTP to Cloud IoT Core.

Adding devices is performed by sending a message with the device ID and its RSA public key to the /devices topic of the desired registry. If the device already exists, the function tries to instead update the existing device with the newly received RSA public key by sending the same message again to the ?updateMask=credentials topic of the specific device. After setting up the IoT Core resources is finished, the function sends a command to the dummy device, which is passed to the gateway's dummy MQTT client, to let the gateway know, that the added device's MQTT client is free to connect to the Cloud.

The function of removing devices does not actually take care of the removals, as that is handled by the dummy trigger described in an earlier paragraph. Instead, it relays the request for a device removal through the dummy device to the gateway in the form of a device command.

To change a devices configuration remotely, the setConfig function can be called and it will relay the configuration request to the device by sending the message to its :modifyCloudToDeviceConfig topic. The gateway will then receive this message on the device's MQTT client.

Getting current states and last reported telemetry is done by sending a few HTTP GET request to Cloud IoT Core. The first request is used to get a list of all available registries, then the devices inside the registries. The list of devices is compared to the list of known devices included by the requesting mobile application and if the application included a device ID that does not exist in the Cloud anymore, the function lets the application know in its response. Next, the function gets the current state and optionally last reported telemetry for each device, checks if it should report any changes and then sends a response to the mobile application.

4.4 Mobile Application

Like the gateway application, the mobile application was developed for Android in the Android version of Java. Likewise, the Android Studio IDE was used for development. The application requires its user to be signed into an email or gmail account registered in the associated Firebase project. As described in the design chapter, the application was divided into several sections. These are represented by several activities.

Navigation Menu

The navigation menu shows the currently signed-in user, has a button to switch between offline and online device control, buttons to navigate between the activities for managing devices, groups and conditions, the gateway control activity and buttons to sign in and sing out.

When offline control is turned on, the application looks for available gateways and offers a list of them for the user to choose from. After one is clicked, the phone connects to it and all communication becomes offline.

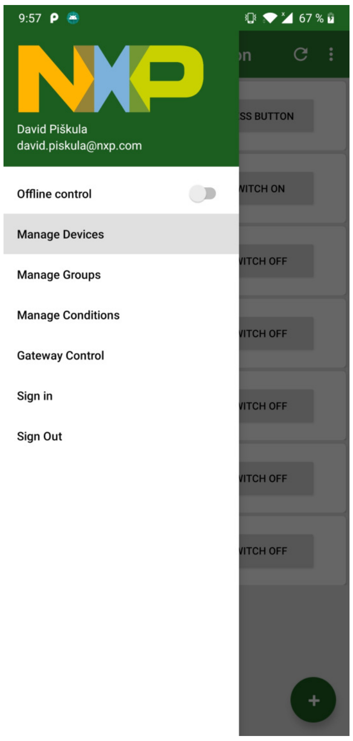
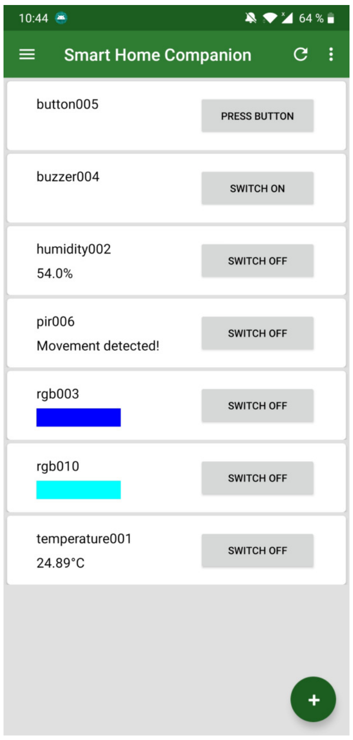


Figure 4.4: Navigation Menu

Home Activity (Manage Devices)

The home activity serves as a basic overview of all connected devices with the most essential device controls made available through buttons. The individual devices are represented by cards set inside a recycler view, which is a scrollable container. Each card displays the device's alias, its latest reported telemetry or current color in the case of telemetry and RGB devices respectively and a button to switch the device on or off. In the case of the button device type, the on off button is instead used for virtual button presses. By touching the current color the user also opens a color picket dialog that can be used to change the color. If a device's MQTT client becomes unresponsive, the card changes into an orange color. If the device becomes dead, it turns red. Furthermore the activity features a toolbar with a button to open the navigation menu, the refresh button and the settings button for making the Cloud data updates automated or manual. Lastly, a floating button for adding new devices is present at the bottom of the activity.


10:44 64 %

Smart Home Companion

button005	PRESS BUTTON
buzzer004	SWITCH ON
humidity002 54.0%	SWITCH OFF
pir006 Movement detected!	SWITCH OFF
rgb003 🟦	SWITCH OFF
rgb010 🟦	SWITCH OFF
temperature001 24.89°C	SWITCH OFF

Figure 4.5: Devices Overview

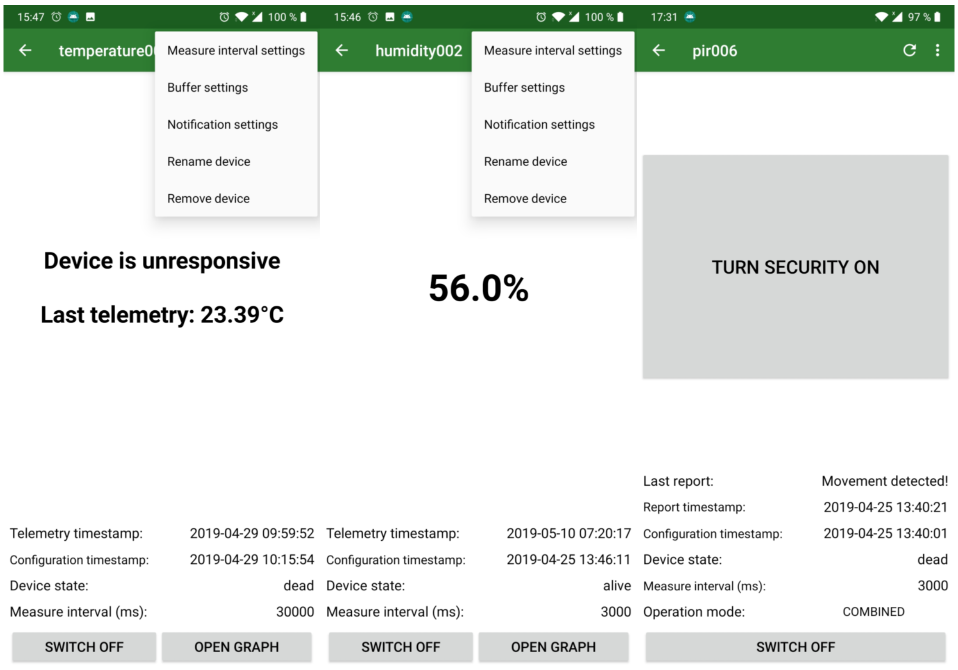
When adding a new device, the user is first asked to choose the device's alias and then a QR reader activity is started using the ZXing barcode reader API. After a QR code is scanned, the application searches for nearby gateways and displays a list of the ones it found. After the user chooses the desired gateway, the application sends the device's ID and PSKd to the gateway and waits for a response. When it receives a response with the device's public key, it sends the device's ID, type and public key to Google Cloud to register the device in Cloud IoT Core.

```
{
    "dev_id":"146E0A0000000002",
    "pskd":"222222",
    "type":"hum"
}
```

Figure 4.6: Example of a Device QR Code

Detail Activities

Touching one of the device cards in the home activity navigates the user to a detail activity for that device. This activity is different based on the type of the chosen device. The buzzer, button and ventilator have a simplified version, which only contains a large button for turning them on or off or triggering a virtual button press, the last configuration timestamp and information about whether the device is alive or dead. The humidity and temperature devices display the latest reported telemetry, latest telemetry and configuration timestamp, whether the device is alive or dead, the configured measurement interval, a button for switching the device on or off and a button that navigates to a graph activity. The PIR details activity showcases a button that turns the security functionality on or off. It also shows the last report, the timestamps, life, measurement interval, a button for switching between the security, regular and combined modes and a button to switch the device on or off. The RGB details activity shows the current color and allows to change it through a color picker, the last configuration timestamp, life and on/off button. Additionally, all the detail activities include a toolbar with a navigation button that takes the user back to the home activity, a refresh button and a settings button for renaming and removing devices, changing measure interval settings and changing notification settings. The notifications, when turned on, can be set up for the telemetry and PIR devices to alert a user when the reported value exceeds a threshold.



Device is unresponsive

Last telemetry: 23.39°C

Telemetry timestamp:	2019-04-29 09:59:52
Configuration timestamp:	2019-04-29 10:15:54
Device state:	dead
Measure interval (ms):	30000

56.0%

Telemetry timestamp:	2019-05-10 07:20:17
Configuration timestamp:	2019-04-25 13:46:11
Device state:	alive
Measure interval (ms):	3000

Last report:	Movement detected!
Report timestamp:	2019-04-25 13:40:21
Configuration timestamp:	2019-04-25 13:40:01
Device state:	dead
Measure interval (ms):	3000
Operation mode:	COMBINED

Figure 4.7: Telemetry Detail Activities

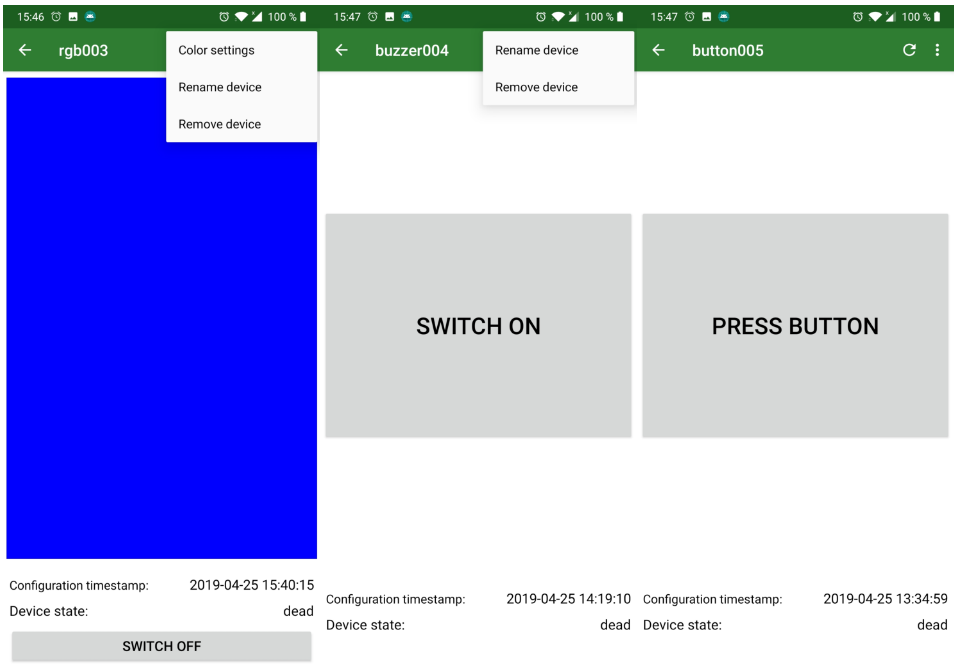


Figure 4.8: Other Detail Activities

Graph Activity

The graph activity displays a graph of the last 50 reports. It was built with the GraphView API, which allows scrolling and zooming into the graph and is simple to use and configure to achieve a desired look. The activity also includes a toolbar with a refresh button and a back navigation button to return to the details activity.

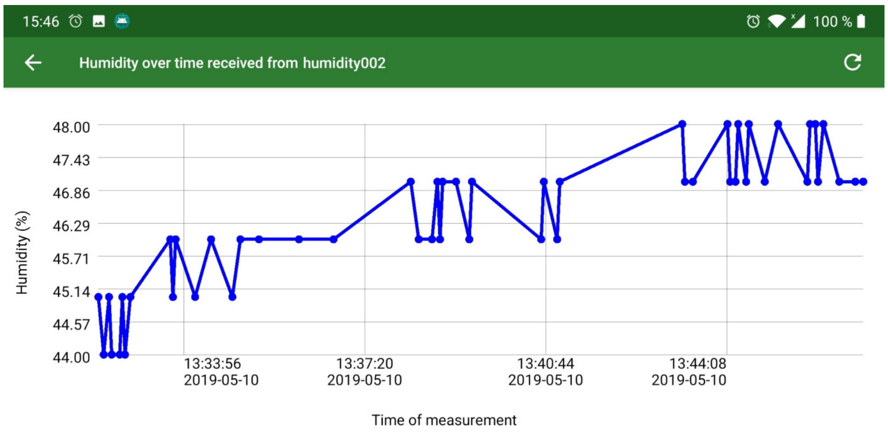
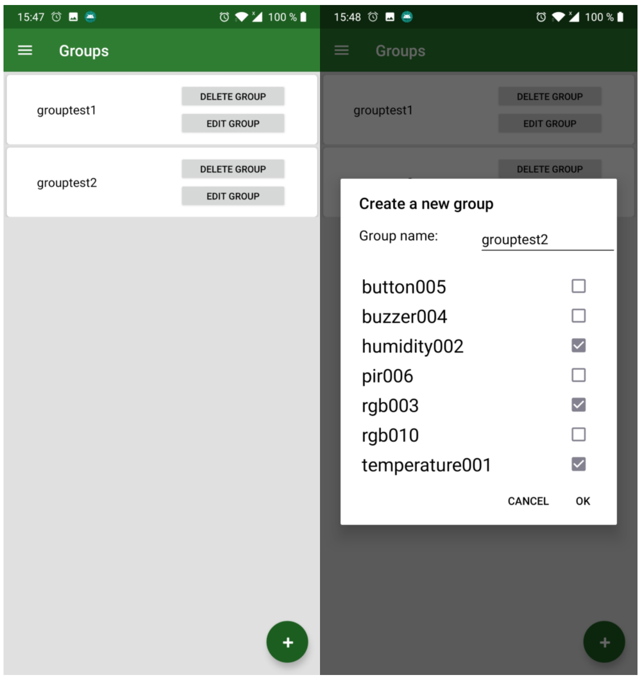


Figure 4.9: Graph Activity

Groups Activity (Manage Groups)

The application allows creating of groups of devices as a quality of life feature. When a group is chosen for a condition, the application inserts the individual devices into the condition's list of affected devices. When displaying which group was chosen for a condition, the application actually looks through the known groups and checks if a device belongs to any of them or not. Thanks to this, the groups are local to the mobile phone they were created at and have no effect on other user's applications, which means that each user can create their own personal groups. The group can be given a name and the user chooses which devices it includes from a list of available devices.

The activity shows a list of existing groups in the form of cards with the group's name and buttons to delete and edit the group. A floating button for adding new groups and a toolbar that provides access to the navigation menu are also included.



Group name: grouptest2

button005	☐
buzzer004	☐
humidity002	☑
pir006	☐
rgb003	☑
rgb010	☐
temperature001	☑

Figure 4.10: Groups Activity

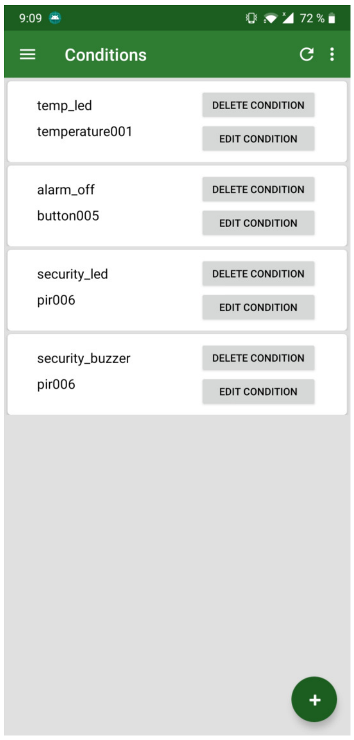
Conditions Activity (Manage Conditions)

The activity contains a list of existing conditions. The condition cards show the condition's name, owned and buttons to delete and edit the condition. A floating button for adding new conditions and a toolbar that provides access to the navigation menu, manual refresh button and automated refresh options are also included.

When adding a new condition, a dialog is presented to the user. The conditions need to be assigned a name and an owner. The owner can be chosen from a drop down list of all known devices. There are three condition types, the limit, detector and button type. When the limit type is chosen, the user can also choose the limit value. Three command types are also included, they can be RGB, switch or toggle. RGB allows to change the device's LED color.

When the limit condition type was chosen, there are three color configurations to choose - the color for when the reported values are below the limit, equal to the limit and above the limit. When the detector type was chosen, the color configurations are for the alert when a secured PIR detects movement, activity when an unsecured PIR detects movement and idle for when no movement was detected. Finally, the button condition type simply sends a command to change the device's LED to the specified single color when the owner button is pressed. The switch command type options are chosen the same way as the RGB options. Instead of color howver, the user chooses whether the command should turn the device on or off. The toggle option simply causes to turn off an active device and turn on an inactive one.

Lastly, the user can choose which devices are affected by the condition. For this purpose, the devices can either be chosen directly from a list of devices or through a list of device groups. In the list of groups, when a no device from a group was chosen, the group's card is grey colored. When at least one was chosen, it turns blue and when all the devices inside a group were chosen, it turns green.



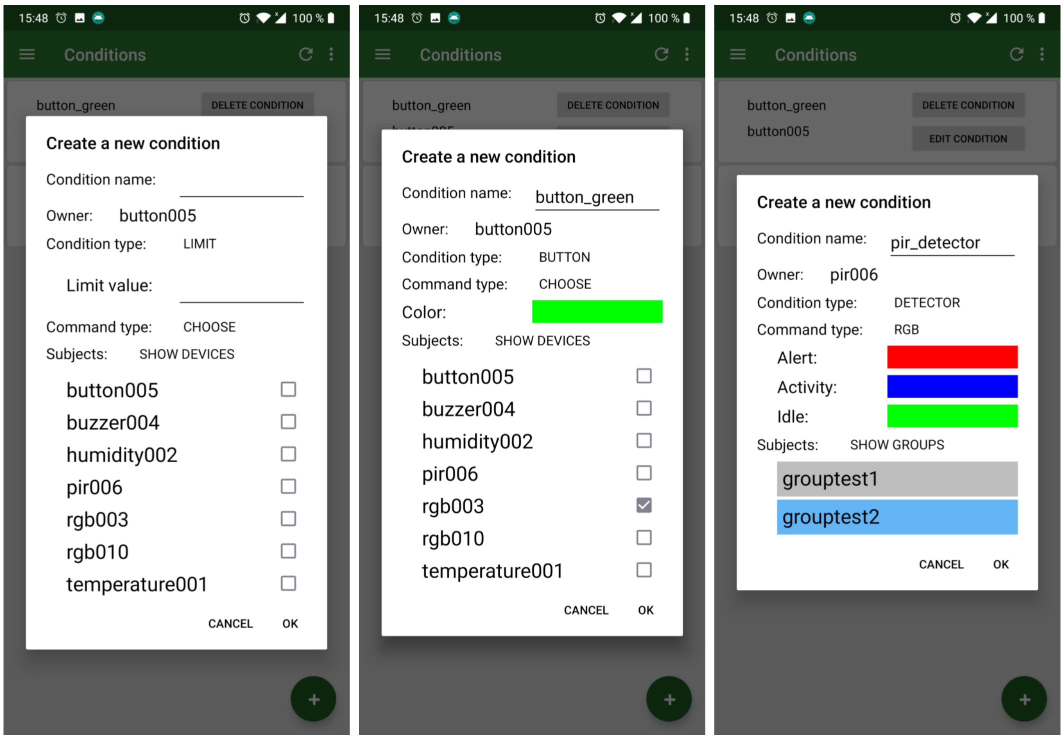
temp_led
temperature001

alarm_off
button005

security_led
pir006

security_buzzer
pir006

Figure 4.11: Conditions Overview


15:48 100 %

Conditions

button_green DELETE CONDITION

Create a new condition

Condition name:

Owner: button005

Condition type: LIMIT

Limit value:

Command type: CHOOSE

Subjects: SHOW DEVICES

- button005 ☐
- buzzer004 ☐
- humidity002 ☐
- pir006 ☐
- rgb003 ☐
- rgb010 ☐
- temperature001 ☐

CANCEL OK

15:48 100 %

Conditions

button_green DELETE CONDITION

Create a new condition

Condition name: button_green

Owner: button005

Condition type: BUTTON

Command type: CHOOSE

Color: 🟩

Subjects: SHOW DEVICES

- button005 ☐
- buzzer004 ☐
- humidity002 ☐
- pir006 ☐
- rgb003 ☑
- rgb010 ☐
- temperature001 ☐

CANCEL OK

15:48 100 %

Conditions

button_green DELETE CONDITION

button005 EDIT CONDITION

Create a new condition

Condition name: pir_detector

Owner: pir006

Condition type: DETECTOR

Command type: RGB

Alert: 🟥

Activity: 🟦

Idle: 🟩

Subjects: SHOW GROUPS

- grouptest1
- grouptest2

CANCEL OK

Figure 4.12: Condition Creation

Gateway Control Activity

The gateway control activity offers seven buttons in total, one of them used to exit the activity and the rest for controlling the gateway. All communication between a registered user and the gateway is signed and encrypted through RSA key pairs and JWT token messages to ensure secure and private communication. When connecting to a gateway for the first time or after resetting their credentials, the user can only use the set up access functionality. When this button is pressed, the application initializes a new key pair and asks the user to choose a user name. Afterwards, it attempts to send the username and the generated public key to the gateway in an unsecured and unencrypted message. If the gateway had no users registered at the time, it accepts the new registration. Otherwise, it replies with its own public key and a message telling the application that it tried an unauthorized access, as described in the gateway implementation section. If the registration was rejected, the mobile application instead generates a QR code from the user name and the public key and displays it on the mobile phones screen.

If the user is already registered at the gateway, they can use the remaining functions, which include initializing the gateway, adding a user, removing a user, removing all devices and resetting their own credentials. When adding a user, the application turns on a QR scanner that can be used to scan the QR code generated on the phone that needs to be added. When scanned, the information is signed and encrypted and sent to the gateway. When removing a user, the application sends a request for the list of currently registered users to the gateway. After decrypting the response and verifying its signature, the application presents this list to the user. The user can they choose whom they want to remove and the request to do so is sent to the gateway. Removing all devices is more of a debug functionality than a feature that would be included in a production application. The preferred way of removing devices is individually through the detail activities, but in the case of a failure or if the Cloud is reset or some other way of inconsistency happens, this functionality can be used to remove all known devices and their data at the gateway. However, it does not generate any requests at the Cloud, which means that the Cloud will have to be cleaned up manually. A user can choose to reset their credentials at any time. This will cause their public key to be removed at the gateway and for the mobile application to remove its existing public-private key pair and the gateway's public key.

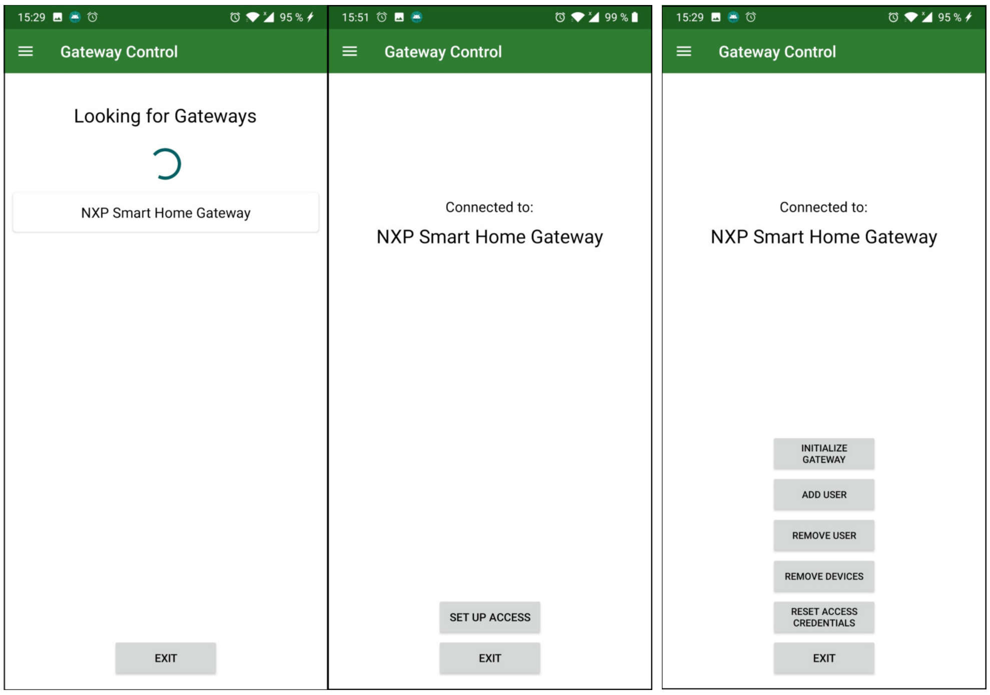
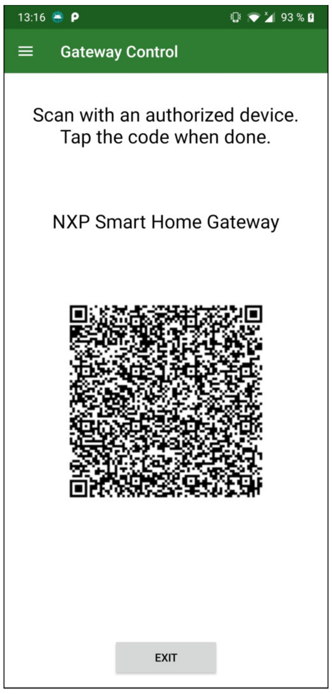


Figure 4.13: Gateway Control Activity



Scan with an authorized device.
Tap the code when done.

NXP Smart Home Gateway

Figure 4.14: Adding a New User

Communication With the Cloud and the Gateway

The application uses JSON strings to communicate with both the Cloud platform and the gateway. During online communication with the Cloud, these JSONs are sent inside HTTP messages through the firebase functions API. When offline, they are first inserted into a singed and encrypted JWT object which is then sent as a message to the gateway through the Nearby Connections API.

4.5 Design Flow and the Problems Encountered During Development

The initial intention of the project was to use an already existing demonstration system and connect it to the Cloud. However, this plan quickly changed into a far more complex set of goals ranging from the local Thread side to the remote Cloud part and everything in between.

It was decided from the start that Android Things, Google Cloud, Thread, KW41Z and the Pico i.MX7D would be used as the technologies for this thesis. The development of the system therefore began with figuring out a way of connecting the Pico board to Thread. Typically this would not be a problem, as demonstrated in the Bachelor's thesis[6] which connected a Raspberry Pi board to Thread by using the KW41Z dongle. However, in the previous thesis, the operating system used on the Raspberry Pi was Ubuntu MATE which supports a lot more drivers than the AT and allows the option of installing new driver modules. Android Things does not allow this at all and unlike typical UNIX based OS', it does not support the USBNET[1] drivers which are required for the dongle to be used as an ethernet-over-usb adapter that can connect a computer to a network. At first the plan was to try and find a method of enabling the drivers in the OS[2]. Since Android Things also does not support many of the commonly used functions of the UNIX terminal, the Busy-Box[3] application was used to try to extend it but after adding the driver modules was determined impossible, another solution had to be found. The NXP SDK for the KW41Z dongle comes with a software package called Host SDK, which among other things allows controlling a Thread border router though UART commands and passing dataframes between the dongle and the host computer over a TUN/TAP tunnel. However, this SDK is only implemented in C and Python, while Android Things only supports the Android version of Java. Thankfully, Google provides a Native Development Kit[4] (NDK), which can be used to compile C or C++ based applications to use most commonly as libraries for Android applications but also, though not recommended, can be utilized to compile the C or C++ source code into an executable that can be run on Android. After slightly adjusting the C-implemented Host SDK, it was possible to cross-compile the source code by using the NDK, the Ubuntu OS and a custom CMAKE file, into the executable file that takes care of relaying data between the dongle and the Pico board over UART. The problem with this solution is that it requires console access to the Pico, as the software has to be run before the main application on the Pico is started. A better solution would be to implement a custom driver using the LoWPAN Androd Things API[5], however that was beyond the scope of this thesis and the API became available over six months after the development of this project started.

Furthermore, a few other issues, although not as impactful, also had to be dealt with. The OS had to be reinstalled a few times when it became impossible to either connect to any Wi-Fi endpoint at all or when it was only possible to connect to the first one used after installation. One of the USB-C cables used to connect the board to the development notebook also stopped working fully at some point. This resulted in a problem where even though it was possible to program the board with the gateway application, the reinstallation of the whole OS kept failing with unknown errors. After a few failed installations, the board could not even boot up and had to be reflashed with a default OS image through specialized software. The issue was resolved by randomly trying a new USB-C cable.

At one point, the nCoAP library used for CoAP communication was rebased and became unavailable through Gradle. This caused the whole gateway code to become uncompilable and had to be solved by temporarily including the library from a custom location until its owners resolved the issues with accessibility.

The next part was connecting the Pico the Google Cloud. At the time, Cloud IoT Core was in an early open beta phase and so over the whole development of this thesis, it has changed quite a lot. Originally it only allowed configuration messages that are retained and

[1]BROWNELL, David. The GNU/Linux „usbnet“ Driver Framework. *Linux-USB*. September 2015. [online]. [cit. 2019-02-05]. Retrieved from: http://www.linux-usb.org/usbnet/

[2]PISKULA, David. How do I enable the ethernet over USB drivers on Android Things?. *StackOverflow*. August 2017. [online]. [cit. 2019-02-05]. Retrieved from: https://stackoverflow.com/questions/45671128/how-do-i-enable-the-ethernet-over-usb-drivers-on-android-things

[3]ANDERSEN, Erik. BusyBox: The Swiss Army Knife of Embedded Linux. *BusyBox*. [online]. [cit. 2019-05-05]. Retrieved from: https://busybox.net/

[4]Android NDK. *Android Developers*. [online]. [cit. 2019-02-05]. Retrieved from: https://developer.android.com/ndk

[5]LoWPAN. *Android Developers*. [online]. [cit. 2019-02-05]. Retrieved from: https://developer.android.com/things/sdk/apis/lowpan

resent every time an MQTT client connects, while later an option of command messages that are only delivered once was added. The documentation and API for the service was also much simpler. Instead of relying on Google's custom API, the MQTT communication was implemented through the use of a popular Java library PAHO MQTT. This way, the Cloud platform can even be changed if needed and the system is not dependent on Google Cloud specifically. Additionally, the API for Node.js that can be used to communicate with IoT Core over HTTP did not exist at the time, so the messages that are sent from Cloud Functions had to be created using available HTTP libraries as well.

When choosing which services to use inside Google Cloud, Cloud Dataflow was recommended by Google as a data processing service that is powerful and easy to use. However, while testing this service, it consumed several euros from the free credit provided by Google within a few days of having a single sensor sending data to the Cloud every five minutes. Due to this, the service was deemed unfit for a project of this size and instead all data processing functions were written manually in Cloud Funtions.

When the essential CoAP and MQTT communication was ready and it was possible to store data generated by the Thread network in the Cloud, the thesis was presented as part of a project in a subject about Intelligent Sensors. Back then, the BigQuery service was used to store data and visualization was performed online in DataStudio.

For simplicity's sake, all data processing was initially performed in a simplified way in the Cloud, As expected, this brought a latency of up to several seconds between the data report and the action that was triggered by it. Eventually, development of the edge processing began which brought dramatic improvements to system responsibility and latency. The automatic reconnection and buffers were developed alongside the automation. There were several issues with the PAHO implementation of the MQTT clients, for example, the buffers the library provides were not very flexible or adjustable at all and sometimes when connecting or reconnecting clients the process failed. This required more thorough and redundant connectivity checks.

While working on making the devices into low-power sleepy end devices the initial difficult part was getting oriented in the extensive SDK. Even though the work was mostly centered around correctly setting pins and initializing or deinitializing clocks and peripherals, it was required to debug the application and step through the assembly code to learn the details of when certain processes happen. However, debugging a sleepy end device was problematic because when the device goes to sleep, it stops responding to the connected debugger who then thinks the device disconnected. This was partially avoided by setting special debug values in the configuration files of the IAR IDE that was used for development. Part of the need to debug was caused by some functions in the SDK having a hidden implementation which sometimes complicates development. During the work, a bug was discovered, when the devices would randomly stop going to sleep properly. At first it was not apparent whether this bug was caused by the SDK or by the additional code created for this thesis, so further ASM debugging was needed, as well as various observational tests. Finally, it was discovered through Wireshark that the devices would sometimes enter a so-called fast polling mode, which is generally used for joining the network, during which the devices poll for incoming messages by sending a hello message every 100 ms, instead of the 3000 or more that were configured. This fact was confirmed through debugging and watching the states of the registers that control the configurations of sleep modes. Not only did this prevent the devices from properly using the desired sleep mode, it also caused the whole network to be flooded with at least 10 messages per second by every affected device. This bug was reported and quickly fixed by the development team at NXP.

The biggest issues encountered while developing the mobile application were connected with inexperience with Android development and various compatibility issues for different Android versions or even libraries. For example, in order to be able to use firebase authentication API, several libraries could not be included in their most up to date versions because the API requires specific versions and will not work with the latest ones. Originally, the online based visualization tool DataStudio was intended to be used for the mobile application to offer a customizable data report that is accessible both online and through the smartphone, however, the embedding feature of the serviece is imperfect and has issues with authentication, which results in the graph often not being accessible. The only solution that worked was to make the graph public, which was not desirable. The final implementation present in the mobile application solved these issues through Firebase and by drawing the graph offline on the smartphone itself. One more issue concerning user registration at Firebase also needed solving. Firebase has no way of requiring administrator confirmation or automatically blocking registrations of new users, which means that anyone could connect their account to the Cloud project. To prevent this, a function that is triggered every time someone tries to register was put in Google Cloud. This function disables the newly registered user so that he cannot access the project.

It was planned to include a ventilator as an end device in the smart home model, however, one of the ventilator device's oscillators was broken and the device did not work at all. As the amount modules made for the project was limited and the device was assembled at the end of April, there was not enough time to resolve this issue before finishing.

The last and most major problem encountered near the very end of development was the cancellation of Android Things development for IoT. The initial Android Things developer preview was released in December 2016[6] and was meant to provide a solution to the poor state of IoT security and to enable Android developers to make smart devices for IoT. The developer preview was released with turn key solutions for Intel Edison, NXP Pico, and Raspberry Pi 3. Over the next year and a half, six more developer previews were released. In May 2018, the Android Things 1.0 release was introduced[7] with four hardware modules certified for production based on the available platfroms NXP i.MX8M, Qualcomm SDA212, Qualcomm SDA624 and MediaTek MT8516. In the end however, Google posted a final update on Android Things in February 2019[8] where they announced that they would stop developing Android Things for IoT and would instead refocus on speakers and smart displays. This fast paced process, which took barely nine months to go from release 1.0 to development cancellation points to another unfortunately common problem with IoT, which are short-lived products and sudden cancellations without warning. Since development on this thesis began before the release 1.0 and the gateway software is built for Android Things, it was too late to switch to a different operating system after the final announcement.

[6]PIEKARSKI, Wayne. Announcing updates to Google's Internet of Things platform: Android Things and Weave. *Google Developers Blog.* December 2018. [online]. [cit. 2019-02-05]. Retrieved from: https://developers.googleblog.com/2016/12/announcing-googles-new-internet-of-things-platform-with-weave-and-android-things.html

[7]SMITH, Dave. Say Hello to Android Things 1.0. *Android Developers Blog.* May 2018. [online]. [cit. 2019-02-05]. Retrieved from: https://android-developers.googleblog.com/2018/05/say-hello-to-android-things-10.html

[8]SMITH, Dave. An Update on Android Things. *Android Developers Blog.* February 2019. [online]. [cit. 2019-02-05]. Retrieved from: https://android-developers.googleblog.com/2019/02/an-update-on-android-things.html

4.6 Project Installation

A Windows PowerShell script was created, which uses gCloud commands available in the Google Cloud SDK to manipulate the Cloud project, for the sake of making the installation of the whole system as simple as possible. To use the script, a user first needs to create a Google Cloud account and project, add a billing account to it, link the project with Firebase, set up a sign in method and add desired users, create an empty Firestore instance, add the mobile application to the firebase project and generate a google-services.json file for it. Lastly, the json file needs to be put into the mobile application's source code's app directory. None of these steps can be automated and therefore must be performed manually. After all of them are complete however, the user can run the script.

The script first asks to choose the preferred Cloud region between europe-west1 and us-central1. Next, it checks whether NPM is present on the computer and if not, asks the user to install it. Afterwards, the script downloads the Google Cloud SDK and installs it. Subsequently, it runs the gCloud initialization, which is interactive and will ask the user for some more information. Finally, it starts creating all the required resources in the desired project. First, Cloud IoT Core registries are created, then the Cloud and Firebase Functions are deployed along with additional configuration files and lastly the Firestore rules are set up. The script also inserts information about the chosen region into the source code of the mobile application.

Chapter 5

Experiments

A set of experiments was performed to confirm correct behavior and functionality of the implemented system. The tests performed can be separated into four categories based on the issues that were explored in this thesis.

5.1 Autonomy

The experiments carried out for assessing the autonomy of the system involved running the network with connected functionality, disconnecting it and finally having it reconnect automatically. Furthermore, the behavior of the conditions along with the correct assignment of groups was examined. Lastly, a comparison of latency was made between sending configuration requests to the devices over the Cloud and over Nearby Connections.

Connected Functionality

The first experiment was done with a fully connected network of seven end devices. Among them were a humidity sensor, a temperature sensor, a PIR sensor, a buzzer, a button and two RGB LED devices. All three sensors had their measurement interval set to 3000 ms. Most of the devices, except for the RGB LEDs, were used as the custom modules created for this thesis. The RGB LED devices were implemented on FRDM KW41Z development boards. Four conditions and one group were set up. The conditions included a limit condition on the temperature sensor that would turn one of the RGB LEDs red when the temperature rose above 24.0°C, two PIR detector conditions and a button press condition. One of the PIR detectors would turn the second RGB LED red when a movement was detected while the sensor was in the secured mode, blue while it was in the normal mode and green while it was not detecting any movement. The other detector would turn on the buzzer when movement was detected while in the secured mode. The button press condition was set to turn off the buzzer and the RGB LED affected by the PIR detector conditions. Both devices were placed inside a group, which was used by the button press condition. Additionally, a notification was configured for the humidity reports, which would alert the user when humidity reached over 38%.

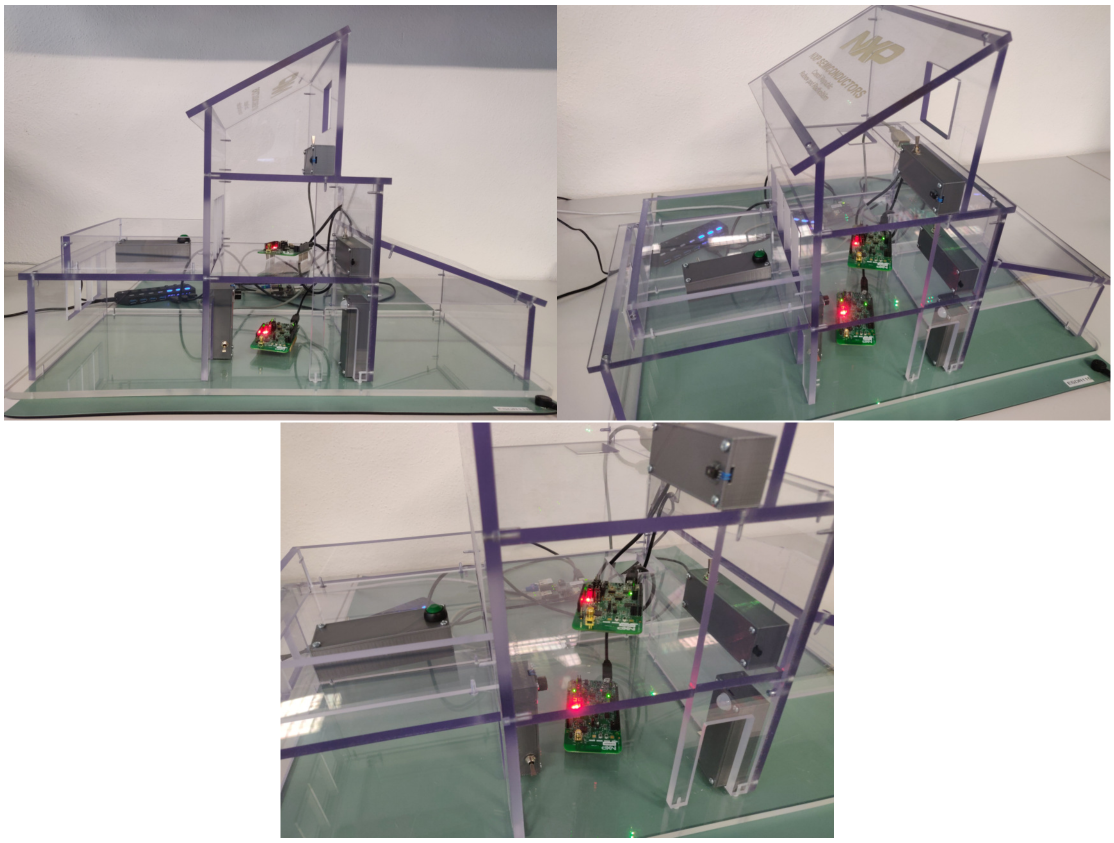
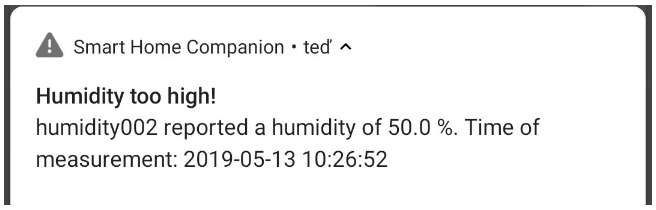


Figure 5.1: Model of the Smart Home

When left alone, the network behaved as expected, reacting to the changes of temperature in the room, storing temperature and humidity data in the Cloud and monitoring movement in front of the PIR sensor. In order to get controlled results, several manual and observational tests were performed on the sensors. First, the temperature measured by the temperature sensor was raised artificially by either breathing on or holding the sensor between two fingers. This caused a faster spike in temperature and thus could be used to quickly determine whether the temperature condition works properly. As was expected, by holding the sensor and pushing the temperature above the limit, the gateway processed the incoming data and changed the desired RGB LED to red. After leaving the sensor alone for a short while and allowing it to cool down again, the temperature dropped back bellow the limit and the gateway changed the LED's color to green.



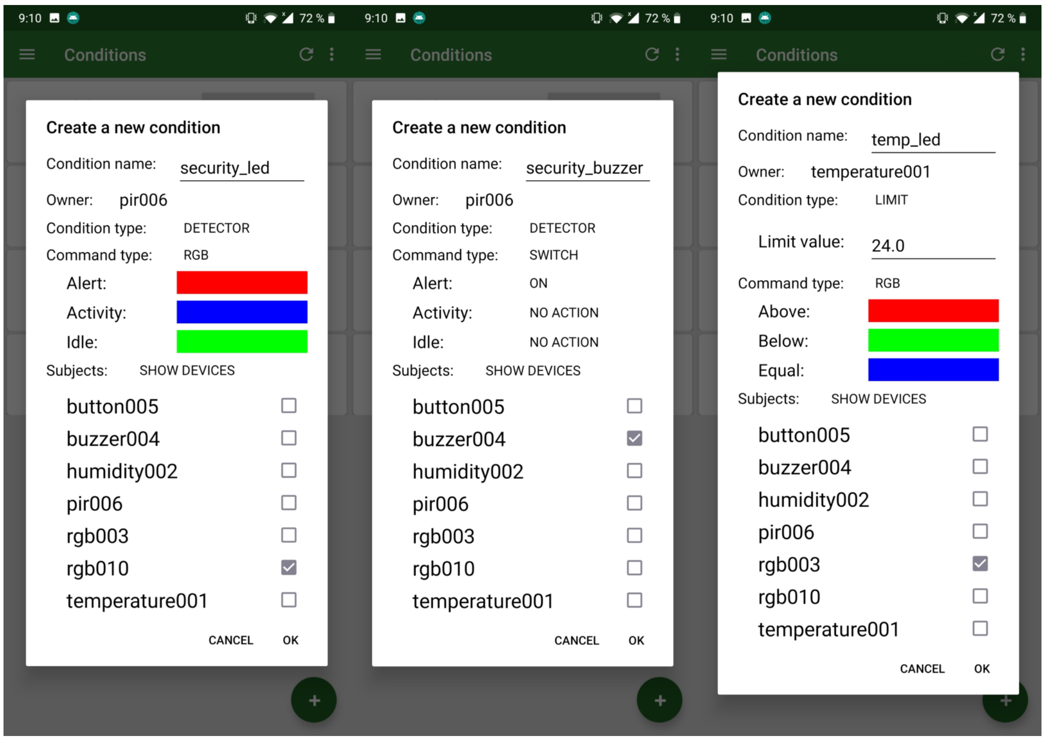
Smart Home Companion • teď

Humidity too high!
humidity002 reported a humidity of 50.0 %. Time of measurement: 2019-05-13 10:26:52

Figure 5.2: Notification

The PIR sensor could be influenced in a similar fashion by waving a hand in front of it. The affected RGB LED remained green for as long as no movement was detected. While the PIR sensor was set to the unsecured mode, waving a hand in front of it would cause the LED's color to be changed to blue. After stopping the wave and moving the hand away, the color would go back to green again. However, while using the secured mode instead, waving the hand caused the LED to turn red and the buzzer to start making a noise. As designed, even after stopping the movement, the buzzing did not stop and the LED stayed red.

Pressing the button when the buzzer was silent did not have any effect on the buzzer. The RGB LED was turned off regardless of what color it was set to at the time, but if it was already turned off, pressing the button did not affect it either. The alarm represented by the red LED and active buzzer was turned off by pressing the button.



Create a new condition

Condition name: security_led
Owner: pir006
Condition type: DETECTOR
Command type: RGB
Alert: 🟥
Activity: 🟦
Idle: 🟩
Subjects: SHOW DEVICES

button005	☐
buzzer004	☐
humidity002	☐
pir006	☐
rgb003	☐
rgb010	☑
temperature001	☐

CANCEL OK

Create a new condition

Condition name: security_buzzer
Owner: pir006
Condition type: DETECTOR
Command type: SWITCH
Alert: ON
Activity: NO ACTION
Idle: NO ACTION
Subjects: SHOW DEVICES

button005	☐
buzzer004	☑
humidity002	☐
pir006	☐
rgb003	☐
rgb010	☐
temperature001	☐

CANCEL OK

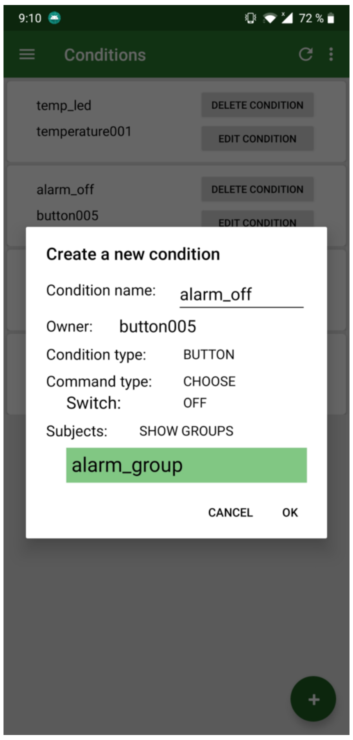
Create a new condition

Condition name: temp_led
Owner: temperature001
Condition type: LIMIT
Limit value: 24.0
Command type: RGB
Above: 🟥
Below: 🟩
Equal: 🟦
Subjects: SHOW DEVICES

button005	☐
buzzer004	☐
humidity002	☐
pir006	☐
rgb003	☑
rgb010	☐
temperature001	☐

CANCEL OK

Figure 5.3: PIR and Temperature Sensor Condition Settings



temp_led

temperature001

alarm_off

button005

Create a new condition

Condition name: alarm_off

Owner: button005

Condition type: BUTTON

Command type: CHOOSE

Switch: OFF

Subjects: SHOW GROUPS

alarm_group

Figure 5.4: Button Condition Settings

Apart from controlling the devices through the conditions it was also possible to change their values through the mobile application. As this was an experiment focused on the connected functionality, the commands for the devices were all sent over the Cloud. It was possible to remove all of the conditions and make the network static, as well as to change every device's settings like the measurement interval or the color of the LED and to turn their main functionalities on or off.

Lastly, some of the devices were manually turned off using their power switches. The gateway noticed this either when a configuration change was necessary for the device, regardless of whether it was generated by a user or a condition or when the device's MQTT client's credentials needed to be refreshed. After noticing it, the gateway marked the device as dead and reported this state to the Cloud, which in turn could be used to retrieve that information in the mobile application as well. Once these states were downloaded to the application, the corresponding devices were marked red in the main overview. Furthermore, the detail activities of these devices also displayed to the user that the devices were dead and unresponsive.

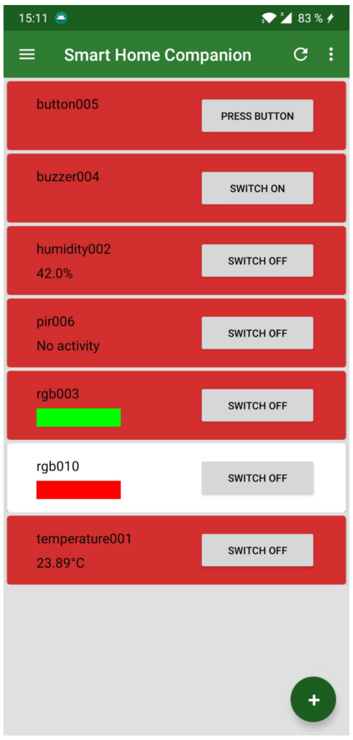


Figure 5.5: Dead Devices

Disconnected Functionality

The next portion of tests consisted of controlling the devices while either the mobile application or the gateway or both were offline. First, only the gateway was disconnected from the internet. Once the gateway noticed this, it started discarding state updates and buffering humidity and telemetry reports. However, the condition that had been set up before the disconnection were still functional exactly as described in the previous experiment.

As the mobile application was still connected to the internet, it was able to contact the Cloud and send new configuration requests and get the latest state updates. However, the Cloud eventually noticed that the MQTT clients are unresponsive, because their heartbeats stopped being sent, and let the application know in state update responses. When this information was received by the application, it marked the devices as orange in the main overview. Furthermore, the configuration requests were stored in Cloud IoT Core at each corresponding device but could not be propagated all the way to the gateway.

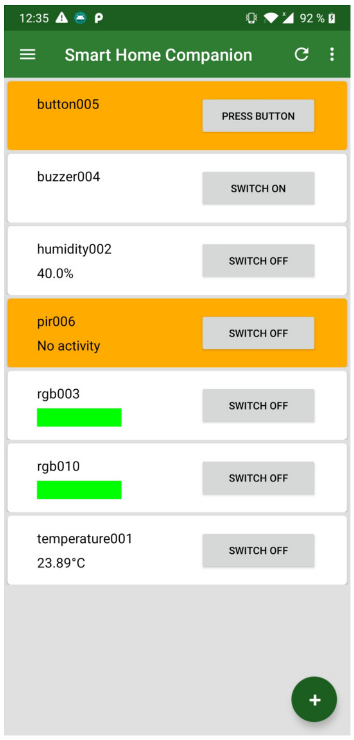


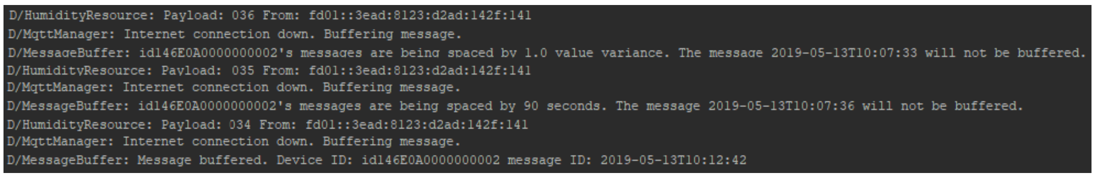
Figure 5.6: Unresponsive MQTT Clients

After disconnecting the mobile application as well, it could not receive any more updates or send new requests to the Cloud. However, it was possible to switch to the offline mode, locate the gateway and communicate with it directly. By doing so, hitting the refresh button could actually provide up to date state information and even properly represent all the changes that were performed automatically after the gateway had disconnected. Even so, the temperature and humidity values could not be updated and the application was only able to show the last values it retrieved from the Cloud, which is the expected behavior.

All of the available offline functionality also worked while either the gateway or the smart phone or both were connected to the internet. When the gateway was connected, it was able to report the offline requested changes to the Cloud without any issues.

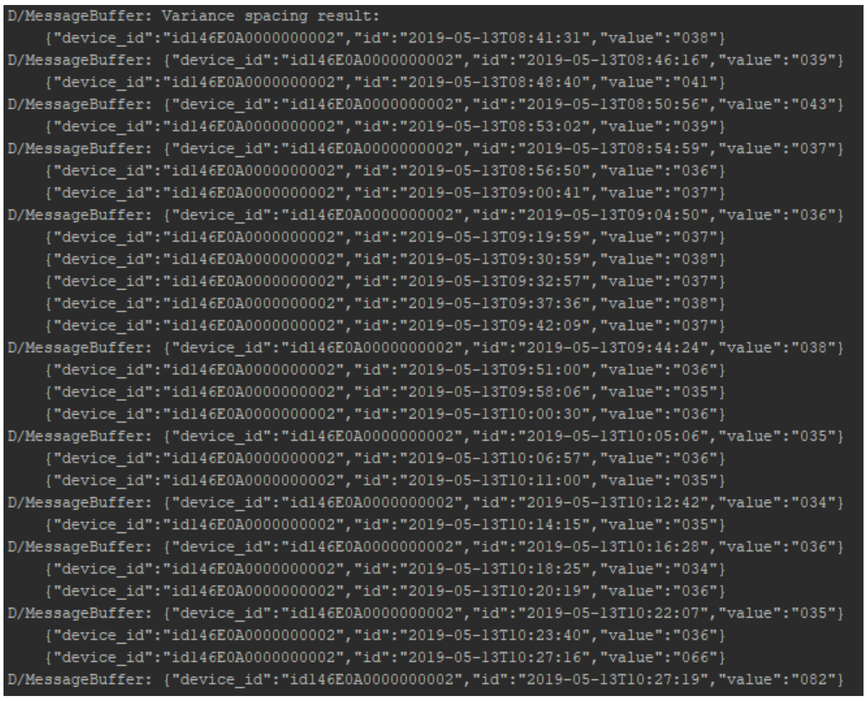
It was impossible to add or remove any devices to and form the network while either of them were offline. However, this is the expected behavior as well.

During the gateway's offline operation, humidity and temperature reports were buffered according the their buffer settings. The buffer for humidity was set to variant dynamic, with the variance threshold of 1.0, maximum messages limit of 30, 0 seconds between messages and 30 second increment for the seconds between messages. The buffer for temperature was set to variant FIFO, with the variance threshold of 1.0 and maximum messages limit of 10.



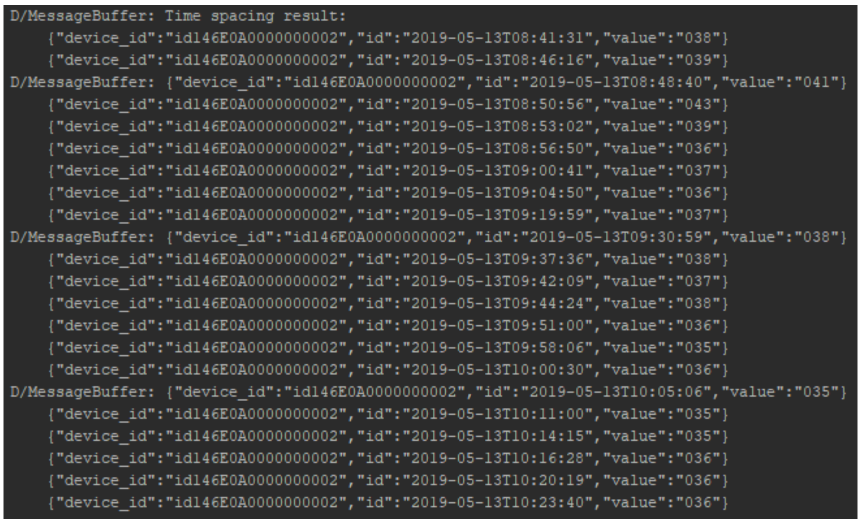
```
D/HumidityResource: Payload: 036 From: fd01::3ead:8123:d2ad:142f:141
D/MqttManager: Internet connection down. Buffering message.
D/MessageBuffer: id146E0A0000000002's messages are being spaced by 1.0 value variance. The message 2019-05-13T10:07:33 will not be buffered.
D/HumidityResource: Payload: 035 From: fd01::3ead:8123:d2ad:142f:141
D/MqttManager: Internet connection down. Buffering message.
D/MessageBuffer: id146E0A0000000002's messages are being spaced by 90 seconds. The message 2019-05-13T10:07:36 will not be buffered.
D/HumidityResource: Payload: 034 From: fd01::3ead:8123:d2ad:142f:141
D/MqttManager: Internet connection down. Buffering message.
D/MessageBuffer: Message buffered. Device ID: id146E0A0000000002 message ID: 2019-05-13T10:12:42
```

Figure 5.7: Humidity Data Buffering



```
D/MessageBuffer: Variance spacing result:
    {"device_id":"id146E0A0000000002","id":"2019-05-13T08:41:31","value":"038"}
D/MessageBuffer: {"device_id":"id146E0A0000000002","id":"2019-05-13T08:46:16","value":"039"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T08:48:40","value":"041"}
D/MessageBuffer: {"device_id":"id146E0A0000000002","id":"2019-05-13T08:50:56","value":"043"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T08:53:02","value":"039"}
D/MessageBuffer: {"device_id":"id146E0A0000000002","id":"2019-05-13T08:54:59","value":"037"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T08:56:50","value":"036"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T09:00:41","value":"037"}
D/MessageBuffer: {"device_id":"id146E0A0000000002","id":"2019-05-13T09:04:50","value":"036"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T09:19:59","value":"037"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T09:30:59","value":"038"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T09:32:57","value":"037"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T09:37:36","value":"038"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T09:42:09","value":"037"}
D/MessageBuffer: {"device_id":"id146E0A0000000002","id":"2019-05-13T09:44:24","value":"038"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T09:51:00","value":"036"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T09:58:06","value":"035"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T10:00:30","value":"036"}
D/MessageBuffer: {"device_id":"id146E0A0000000002","id":"2019-05-13T10:05:06","value":"035"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T10:06:57","value":"036"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T10:11:00","value":"035"}
D/MessageBuffer: {"device_id":"id146E0A0000000002","id":"2019-05-13T10:12:42","value":"034"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T10:14:15","value":"035"}
D/MessageBuffer: {"device_id":"id146E0A0000000002","id":"2019-05-13T10:16:28","value":"036"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T10:18:25","value":"034"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T10:20:19","value":"036"}
D/MessageBuffer: {"device_id":"id146E0A0000000002","id":"2019-05-13T10:22:07","value":"035"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T10:23:40","value":"036"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T10:27:16","value":"066"}
D/MessageBuffer: {"device_id":"id146E0A0000000002","id":"2019-05-13T10:27:19","value":"082"}
```

Figure 5.8: Variance Spacing

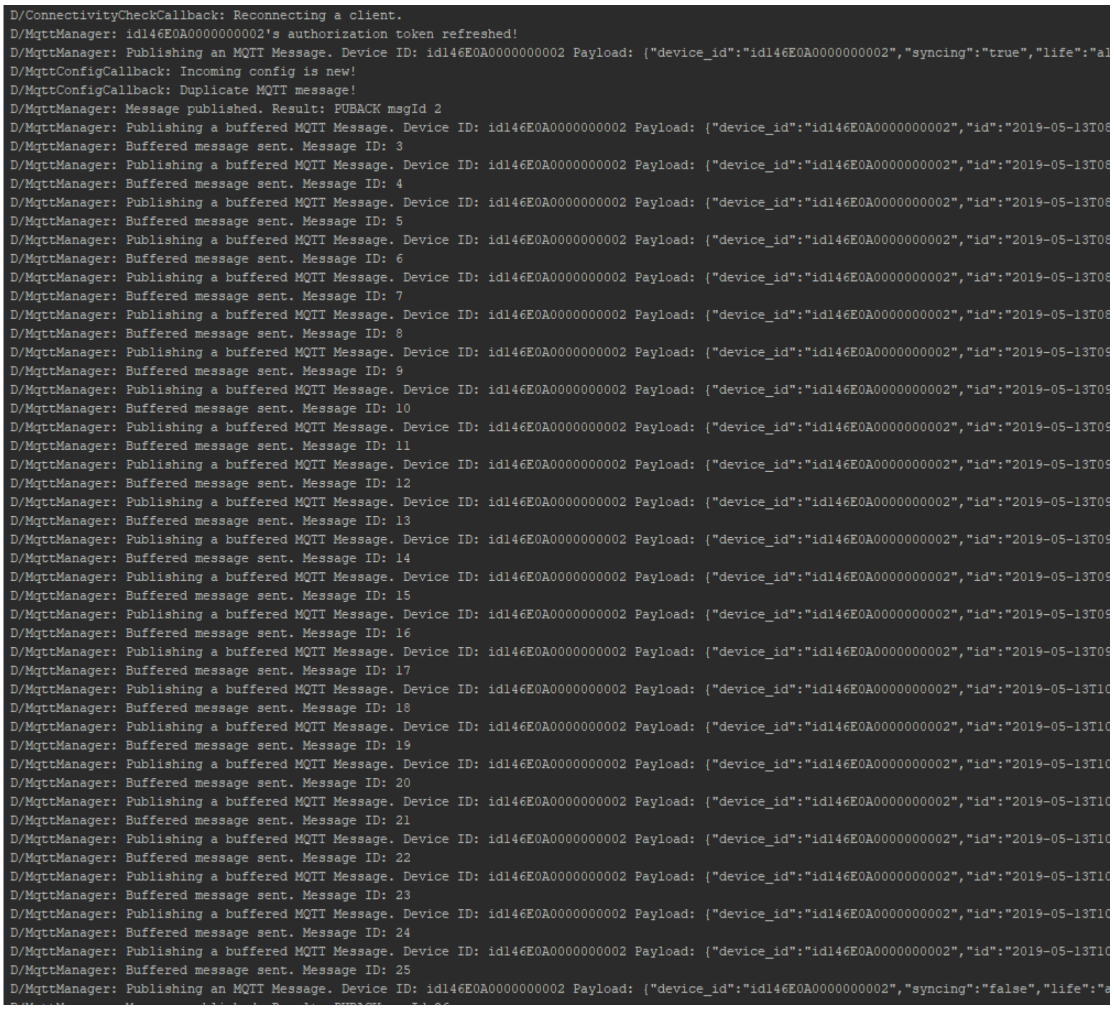


```
D/MessageBuffer: Time spacing result:
    {"device_id":"id146E0A0000000002","id":"2019-05-13T08:41:31","value":"038"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T08:46:16","value":"039"}
D/MessageBuffer: {"device_id":"id146E0A0000000002","id":"2019-05-13T08:48:40","value":"041"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T08:50:56","value":"043"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T08:53:02","value":"039"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T08:56:50","value":"036"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T09:00:41","value":"037"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T09:04:50","value":"036"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T09:19:59","value":"037"}
D/MessageBuffer: {"device_id":"id146E0A0000000002","id":"2019-05-13T09:30:59","value":"038"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T09:37:36","value":"038"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T09:42:09","value":"037"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T09:44:24","value":"038"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T09:51:00","value":"036"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T09:58:06","value":"035"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T10:00:30","value":"036"}
D/MessageBuffer: {"device_id":"id146E0A0000000002","id":"2019-05-13T10:05:06","value":"035"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T10:11:00","value":"035"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T10:14:15","value":"035"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T10:16:28","value":"036"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T10:20:19","value":"036"}
    {"device_id":"id146E0A0000000002","id":"2019-05-13T10:23:40","value":"036"}
```

Figure 5.9: Time Spacing

Automatic Reconnection

Once the gateway's connection to the internet was reestablished, it quickly began going through all connected devices, including the dummy, and reconnecting them. It marked all devices with the syncing state and reported the current states to the Cloud. Then it went through the humidity and temperature buffers and reported all the stored data as well. When new reports came from the devices during this time, they were not reported immediately, but the application waited until the buffers were completely empty and sent the new data to the Cloud afterwards. The gateway also received the latest requested remote configuration from Cloud IoT Core as well, but only used them if they were newer than the last change performed either automatically through a condition or manually through offline controls. After the synchronization was over, the states were reported again, this time marked with syncing set to false and regular operation of the gateway was resumed.



```
D/ConnectivityCheckCallback: Reconnecting a client.
D/MqttManager: id146E0A0000000002's authorization token refreshed!
D/MqttManager: Publishing an MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","syncing":"true","life":"al
D/MqttConfigCallback: Incoming config is new!
D/MqttConfigCallback: Duplicate MQTT message!
D/MqttManager: Message published. Result: PUBACK msgId 2
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T08
D/MqttManager: Buffered message sent. Message ID: 3
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T08
D/MqttManager: Buffered message sent. Message ID: 4
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T08
D/MqttManager: Buffered message sent. Message ID: 5
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T08
D/MqttManager: Buffered message sent. Message ID: 6
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T08
D/MqttManager: Buffered message sent. Message ID: 7
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T08
D/MqttManager: Buffered message sent. Message ID: 8
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T09
D/MqttManager: Buffered message sent. Message ID: 9
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T09
D/MqttManager: Buffered message sent. Message ID: 10
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T09
D/MqttManager: Buffered message sent. Message ID: 11
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T09
D/MqttManager: Buffered message sent. Message ID: 12
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T09
D/MqttManager: Buffered message sent. Message ID: 13
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T09
D/MqttManager: Buffered message sent. Message ID: 14
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T09
D/MqttManager: Buffered message sent. Message ID: 15
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T09
D/MqttManager: Buffered message sent. Message ID: 16
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T09
D/MqttManager: Buffered message sent. Message ID: 17
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T10
D/MqttManager: Buffered message sent. Message ID: 18
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T10
D/MqttManager: Buffered message sent. Message ID: 19
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T10
D/MqttManager: Buffered message sent. Message ID: 20
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T10
D/MqttManager: Buffered message sent. Message ID: 21
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T10
D/MqttManager: Buffered message sent. Message ID: 22
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T10
D/MqttManager: Buffered message sent. Message ID: 23
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T10
D/MqttManager: Buffered message sent. Message ID: 24
D/MqttManager: Publishing a buffered MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","id":"2019-05-13T10
D/MqttManager: Buffered message sent. Message ID: 25
D/MqttManager: Publishing an MQTT Message. Device ID: id146E0A0000000002 Payload: {"device_id":"id146E0A0000000002","syncing":"false","life":"a
```

Figure 5.10: Resynchronization

Apart from the internet loss caused disconnections, the MQTT clients also periodically closed due to their credentials expiring. The application then had to refresh the credentials from the device's private key and reconnect, after which it received the latest stored configuration requests as well. This mostly happened so fast that no telemetry had to be buffered during the process.

Latency Comparison

A simple test to compare the latency difference between local and remote communication was performer. It was done on one of the RGB LED devices and the travel times were measured by counting the seconds between submitting the request and the LED changing its color. As the LED devices were router eligible devices, they did not enter any sleep cycles and could react to CoAP messages instanlty. Additional latency of up to the length of the polling interval would be added for the low power end devices.

Five measurements were performer for both communication types. In the case of local communication, the changes were reflected more or less instantly, taking less than a second. The remote communication however, ranged from 6 to 8 s. The average time it took from remotely submitting the command to the LED changing its color was 7.2 s.

5.2 Security

The experiments testing the security were divided into four parts. One was to test the mobile application's access to the Cloud, another to test its access to the gateway, the third one to test the gateway's access to the Cloud and the last one to test the device's access to the network.

Two smart phones were required to test the access to the gateway. One of them was set as the first registered device and could be used to manage the gateway, while the other was not registered and was used to skip the registration to get to the management screen without having its QR scanned and public key registered beforehand. While the user with this smart phone was able to see and use the buttons that send the gateway control messages, the gateway always replied with the Unauthorized response. This unregistered smart phone could not be used to alter the gateway's settings or add or remove any devices or even to get the current states or send configuration requests through the local communication channel. After resetting the keys of this smart phone and actually scanning the QR code to register it at the gateway, both phones could be used to control the gateway and the network locally.

When trying to use the application without signing in, the application kept popping up a window requiring the user to choose their account. This could be skipped and ignored to a certain extent but trying to alter the device configurations or retrieve the latest state updates resulted in rejections from the Cloud.

When an MQTT client's public key was removed from the Cloud IoT Core through the Google Cloud console, the client cold no longer communicate with or reconnect to the broker. The gateway kept trying to reconnect it anyway, first right after it disconnected for the Cloud and then through the automated reconnection thread but it never succeeded until the public key was put back into IoT Core.

5.3 Initialization

The initialization experiments comprised of initializing the Cloud project and installing applications, initializing the gateway and adding and removing devices and observing if the correct allocations and deletions were performed both in the Cloud and on the gateway.

When the prerequisites for running the initialization script were correctly set up, the script was able to initiate both Google Cloud and Firebase and alter the project ID and region values for the gateway application and copy the service_key generated by the Cloud for the mobile application to the application's source code folder. Afterwards, both applications could be safely installed to their corresponding devices.

After the gateway was turned on for the very first time, it remained in an uninitialized state, without starting the CoAP server endpoints and without connecting any MQTT clients. By using an authorized smart phone to trigger the initialization process, the gateway generated a key pair for the dummy device and sent the public key to the mobile application, which in turn put it in the Cloud, where the dummy device was created in IoT Core. After that was done, the mobile application told the gateway to start its functions and the gateway began creating the CoAP endpoints and connected the dummy MQTT client.

When a device was turned on without its QR code being scanned first, it kept trying to connect to the network to no avail. However, as soon as it was added as an expected joiner by the gateway, the device connected and started announcing its ID. Scanning the device's QR code with an authorized smart phone correctly passed all the needed information to the gateway, which in turn created the correct key pair and added the device as an expected joiner to the network. It then responded to the smart phone with the device's public key, which the mobile application delivered to the Cloud. Inside the Cloud all the required resources were properly set up and the gateway was informed through the dummy client that the device can connect to the Cloud.

When using a signed in smart phone, it was possible to send the command to remove a device to the Cloud. When the command was generated while the gateway was offline, it had no effect, as the request did not, by design, get retained at IoT Core. Sending it to an online gateway however, caused all of the information the gateway had about the device to be removed. Subsequently, the gateway sent a message through the dummy client letting the Cloud know that it can safely remove all resources previously allocated for the device.

5.4 Power Consumption

Power consumption measurements were performed for the temperature sensor with different lengths of sleep and measurement periods. The Keysight N6705C DC Power Analyzer was used to measure the consumption and export the results into the graphs presented in this section.

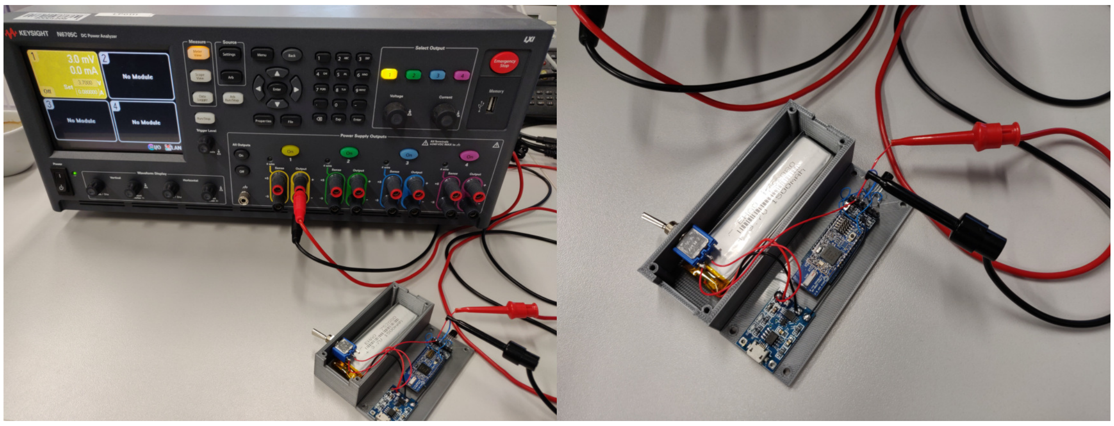


Figure 5.11: Consumption Measurement Set Up

Three measurements were performed in total, each for five minutes but with different settings. The deep sleep duration for the devices was set to 3000 ms in all three measurements. In the first measurement, the measurement interval of the sensor was set to 3000 ms and the polling interval to 1000 ms. The results showed a maximum consumption of 25.585 mA, a total average of 179.513 μA and the average consumption while asleep was approximately 80 μA.

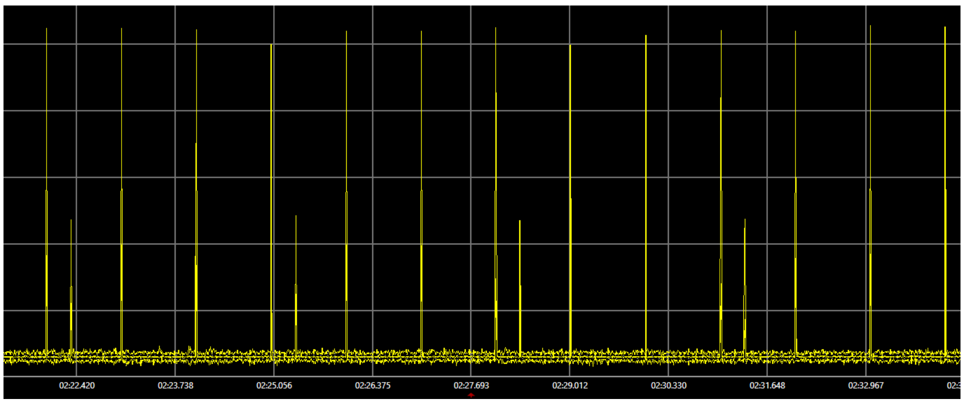


Figure 5.12: First Measurement

In the second measurement, the measurement interval was set to 30000 ms and the polling interval to 10000 ms. This time the results had a maximum consumption of 25.996 mA, a total average of 98.735 μA and the average consumption while asleep was approximately 80 μA.

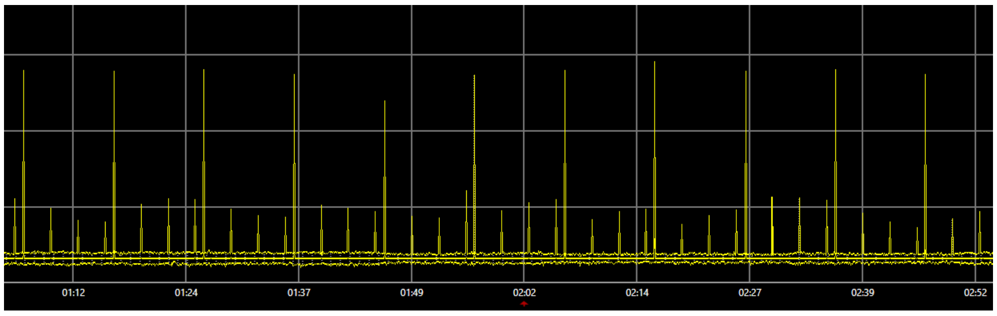


Figure 5.13: Second Measurement

In the last measurement, the measurement interval was set to 30000 ms and the polling interval to 9000 ms. The final measured results had a maximum consumption of 25.665 mA, a total average of 63.24 μA and the average consumption while asleep was approximately 50 μA.

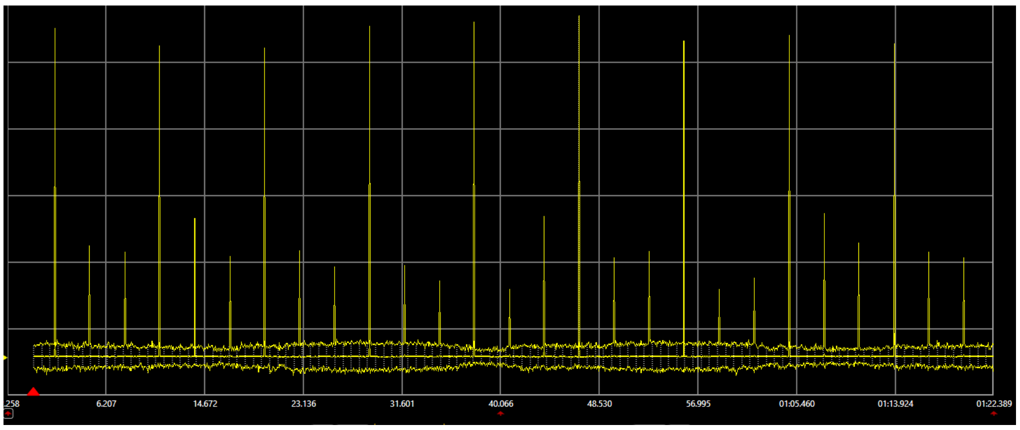


Figure 5.14: Third Measurement

The reason for the large difference between the last two measurements and between the average consumption in deep sleep between the first two and the last one was the settings for the polling interval. When the interval was set to 1000 ms, it woke the device twice over the duration of the 3000 ms long deep sleep. When the interval was set to 10000 ms instead, the intervals could not properly overlap, as 10 is not divisible by 3. In the last measurements however, the intervals completely overlapped, which led to a dramatic improvement in consumption.

The batteries used for the devices have a capacity of 1500 mAh, which means that with the settings of the third measurement, the temperature sensor device could last for up to 23719 hours, or 988 days before needing to recharge. The trade-off for this longevity is response latency, as the device can take up to 9 seconds to notice a new message. The predicted time can be greatly impacted by the device having to reconnect, especially if the border router is unresponsive and the end device remains in a reconnection cycle.

Chapter 6

Conclusion

This goal of this thesis was to research the current state of the Internet of Things, assess its problems and describe the state of the art solutions and based on this research to then design and implement a complex IoT solution that fixes several of the problems under the patronage of NXP Semiconductors. The design had to be autonomous, independent of the Cloud while utilizing its advantages, secure, energy efficient, extensible and needed to include a mobile application that could be used to control it both remotely and locally and display the states and reported data of the devices in the network. The intended result was a model of a smart home based on NXP's Thread communication protocol and the KW41Z and i.MX7D MCUs. All of the above goals were successfully accomplished, implemented and tested in the solution presented in this work.

The final product includes several types of devices, among which are both sensors and actuators. The devices were built for low power consumption and utilize sleep modes which let them operate for very long periods of time without recharging. The automation of the network is performed through a simple and easily extensible but flexible and effective scheme and all major data processing is done on the gateway at the edge of the network. Thanks to this, the system is autonomous and independent of the Cloud and remains functional even when it loses its internet connection. Furthermore, all communication is secured and kept private through various cryptographic schemes. The mobile application provides a user friendly interface for interaction with the network and an initiation scheme which automatically allocates all the needed resources in the Cloud. The Cloud is used for long term data storage and authenticated remote communication between the gateway and the mobile application. The autonomous automation part of this thesis was submitted to and featured at the Excel@FIT 2019 conference together with the model of the smart home.

In the future this project can be extended with more complex device types and new automation conditions. Further technologies could also be added, like an over-the-air updating scheme which would allow security patches and an automated key-pair rotation to further improve security or other initialization techniques that use Bluetooth or NFC to connect to the border router of gateway. Moreover, the Cloud's rich computational capabilities could also be utilized for machine learning and the resulting models could be included in the automation scheme on the gateway. Finally, communication protocol coexistence between Thread and other technologies could also be explored, as there are already standards like DotDot that focus on this goal.

Bibliography

[1] An Introduction to the Internet of Things (IoT). Lopez Research LLC, November 2013, [Online; visited 2019.01.11.].
URL https://www.cisco.com/c/dam/en_us/solutions/trends/iot/introduction_to_IoT_november.pdf

[2] Vermesan, O.; Friess, P.: *Internet of things: converging technologies for smart environments and integrated ecosystems*. Aalborg, Denmark: River Publishers, 2013, ISBN 978-879-2982-735.

[3] Ballantyne, B.: *INDUSTRY 4.0*. London, UK: World Market Intelligence, December 2016, 34-36,38,40-41 pp.
URL https://search.proquest.com/docview/1850307322

[4] IEEE Standard for Low-Rate Wireless Networks. *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, April 2016: pp. 1–709, doi:10.1109/IEEESTD.2016.7460875.

[5] Bluetooth Mesh Profile Specification 1.0. Bluetooth SIG, July 2017, [Online; visited 2019.01.11.].
URL
https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=429633

[6] Thread Stack Fundamentals. Thread Group, July 2015, [Online; visited 2019.04.30.].
URL https://www.threadgroup.org/Portals/0/documents/support/ThreadOverview_633_2.pdf

[7] Olsson, J.: *6LoWPAN demystified*. Dalas, Texas, USA: Texas Instruments Incorporated, October 2014.
URL http://www.ti.com/lit/wp/swry013/swry013.pdf

[8] Shelby, Z.; Hartke, K.; Bormann, C.: *The Constrained Application Protocol (CoAP)*. Internet Engineering Task Force (IETF), June 2014, iSSN: 2070-1721.

[9] MQTT Version 3.1.1. OASIS, October 2014, [Online; visited 2019.01.11.].
URL http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf

[10] Hassan, Q. F.: Demystifying Cloud Computing. CrossTalk: The Journal of Defense Software Engineering, 2011, [Online; visited 2019.01.11.].
URL http://static1.1.sqspcdn.com/static/f/702523/10181434/1294788395300/201101-Hassan.pdf?token=KuQ711b4oxK2IWlsogkRvalQSfc%3D

[11] Mell, P.; Grance, T.: *The NIST Definition of Cloud Computing*. Gaithersburg, MD, USA: NIST National Institute of Standards and Technology, September 2011, [Online; visited 2019.01.11.].
URL http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf

[12] Ai, Y.; Peng, M.; Zhang, K.: Edge computing technologies for Internet of Things: a primer. *Digital Communications and Networks*, vol. 4, no. 2, 2018: pp. 77 – 86, ISSN 2352-8648, doi:https://doi.org/10.1016/j.dcan.2017.07.001.
URL http://www.sciencedirect.com/science/article/pii/S2352864817301335

[13] Kessler, G. C.: An overview of cryptography. *the Handbook on Local Area Networks, Auerbach*, 1998, revision from 2019.04.11. [Online; visited 2019.04.28.].
URL https://www.garykessler.net/library/crypto.html

[14] Babu, S. M.; Lakshmi, A. J.; Rao, B. T.: A study on cloud based Internet of Things: CloudIoT. In *2015 Global Conference on Communication Technologies (GCCT)*, April 2015, pp. 60–65, doi:10.1109/GCCT.2015.7342624.

[15] Botta, A.; de Donato, W.; Persico, V.; et al.: On the Integration of Cloud Computing and Internet of Things. In *2014 International Conference on Future Internet of Things and Cloud*, Aug 2014, pp. 23–30, doi:10.1109/FiCloud.2014.14.

[16] Satyanarayanan, M.: The Emergence of Edge Computing. IEEE Computer Society, January 2017, [Online; visited 2019.01.11.].
URL
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7807196&tag=1

[17] Premsankar, G.; Di Francesco, M.; Taleb, T.: Edge Computing for the Internet of Things: A Case Study. *IEEE Internet of Things Journal*, vol. 5, no. 2, April 2018: pp. 1275–1284, ISSN 2327-4662, doi:10.1109/JIOT.2018.2805263.

[18] Rahmani, A.-M.; Thanigaivelan, N. K.; Gia, T. N.; et al.: Smart e-Health Gateway: Bringing intelligence to Internet-of-Things based ubiquitous healthcare systems. In *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, IEEE, 2015, ISBN 9781479963904, ISSN 2331-9852, pp. 826–834.

[19] Sun, X.; Ansari, N.: EdgeIoT: Mobile Edge Computing for the Internet of Things. *IEEE Communications Magazine*, vol. 54, no. 12, 2016: pp. 22–29, ISSN 0163-6804.

[20] Lindqvist, U.; Neumann, P. G.: *Inside Risks The Future of the Internet of Things*. San Francisco, CA, USA: Communications of the ACM, February 2017, doi:10.1145/3029589.
URL http://www.csl.sri.com/users/neumann/cacm240.pdf

[21] Suo, H.; Wan, J.; Zou, C.; et al.: Security in the Internet of Things: A Review. In *2012 International Conference on Computer Science and Electronics Engineering*, vol. 3, March 2012, pp. 648–651, doi:10.1109/ICCSEE.2012.373.

[22] Abomhara, M.; Køien, G. M.: Security and privacy in the Internet of Things: Current status and open issues. In *2014 International Conference on Privacy and Security in Mobile Systems (PRISMS)*, May 2014, pp. 1–8, doi:10.1109/PRISMS.2014.6970594.

[23] Sadeghi, A.; Wachsmann, C.; Waidner, M.: Security and privacy challenges in industrial Internet of Things. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2015, ISSN 0738-100X, pp. 1–6, doi:10.1145/2744769.2747942.

[24] Yang, Y.; Wu, L.; Yin, G.; et al.: A Survey on Security and Privacy Issues in Internet-of-Things. *IEEE Internet of Things Journal*, vol. 4, no. 5, Oct 2017: pp. 1250–1258, ISSN 2327-4662, doi:10.1109/JIOT.2017.2694844.

[25] Perković, T.; Kovacevic, T.; Cagalj, M.: BlinkComm: Initialization of IoT Devices Using Visible Light Communication. *Wireless Communications and Mobile Computing*, vol. 2018, 06 2018: pp. 1–16, doi:10.1155/2018/8523078.

[26] Perković, T.; Čagalj, M.; Kovačević, T.: LISA: Visible light based initialization and SMS based authentication of constrained IoT devices. *Future Generation Computer Systems*, vol. 97, 2019: pp. 105 – 118, ISSN 0167-739X, doi:https://doi.org/10.1016/j.future.2019.02.052.
URL `http://www.sciencedirect.com/science/article/pii/S0167739X18321083`

[27] MKW41Z/31Z/21Z Reference Manual. NXP Semiconductors, October 2016, [Online; visited 2019.02.05.].
URL
`https://www.nxp.com/files-static/32bit/doc/ref_manual/MKW41Z512RM.pdf`

[28] Sudevalayam, S.; Kulkarni, P.: Energy Harvesting Sensor Nodes: Survey and Implications. *IEEE Communications Surveys Tutorials*, vol. 13, no. 3, Third 2011: pp. 443–461, ISSN 1553-877X, doi:10.1109/SURV.2011.060710.00094.

[29] Kamalinejad, P.; Mahapatra, C.; Sheng, Z.; et al.: Wireless energy harvesting for the Internet of Things. *IEEE Communications Magazine*, vol. 53, no. 6, June 2015: pp. 102–108, ISSN 0163-6804, doi:10.1109/MCOM.2015.7120024.

[30] Evans, D.: The Internet of Things, How the Next Evolution of the Internet Is Changing Everything. Cisco IBSG, April 2011, [Online; visited 2019.01.13.].
URL `https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf`

[31] Thread Commissioning. Thread Group, July 2015, [Online; visited 2019.02.05.].
URL `https://www.threadgroup.org/Portals/0/documents/support/CommissioningWhitePaper_658_2.pdf`

[32] F. Hao, E.: *J-PAKE: Password-Authenticated Key Exchange by Juggling*. Internet Engineering Task Force (IETF), September 2017, iSSN: 2070-1721.

[33] F. Hao, E.: *Schnorr Non-interactive Zero-Knowledge Proof*. Internet Engineering Task Force (IETF), September 2017, iSSN: 2070-1721.

[34] Power Management for Kinetis MCUs. NXP Semiconductors, April 2015, [Online; visited 2019.02.05.].
URL `https://www.nxp.com/docs/en/application-note/AN4503.pdf`