Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Informatics

Remote Configuration of (Meshed) Sensor Nodes in Home and Building Automation

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Mirza Hodović, BSc.
Registration Number 11839430

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Dipl.-Ing. Dr. techn. Wolfgang Kastner
Assistance: Dipl.-Ing. Patrick Denzler, MSc.
Dipl.-Ing. Daniel Ramsauer

Vienna, 6th December, 2022

Mirza Hodović   Wolfgang Kastner

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Mirza Hodović, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 6. Dezember 2022

Mirza Hodović

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

I want to thank my advisors, Dr. Wolfgang Kastner, Dipl.-Ing. Patrick Denzler and Dipl.-Ing. Daniel Ramsauer, for their support and guidance throughout the thesis.

In addition, I want to thank my sister, my father and the rest of my family, for their continuous support and motivation until the very end. I also want to thank Marina, for being there for me and helping me during the tough times that accompanied this thesis.

I dedicate this thesis to my mother. You always pushed me to do my best and reminded me to persist on the right path until the end. Without you, I wouldn't be the person I am today. You have always been an inspiration to me and I hope you will win this battle, like you won all the battles until now.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Sensorknoten sind ein integraler Bestandteil der Heim- und Gebäudeautomation. Diese Geräte können über drahtlose Sensornetzwerke verbunden werden. Aufgrund der großen Anzahl von eingesetzten Sensorknoten und der Reichweite, die sie abdecken müssen, ist eine manuelle Aktualisierung zeitaufwändig und mühsam. Diese Diplomarbeit untersucht verschiedene Technologien und Standards, die eine Fernkonfiguration von Sensorknoten in der Heim- und Gebäudeautomation ermöglichen. Ein weiterer Fokus liegt auf den Themen Sicherheit und Zuverlässigkeit eines Over-the-Air-Systems. Basierend auf der Forschung wird eine Reihe von Anforderungen für ein sicheres und zuverlässiges Over-the-Air-System entwickelt. Besonderes Augenmerk liegt auf der Designauswahl der Kombination aus dem Zephyr-Echtzeitbetriebssystem für die Sensorknoten und der drahtlosen Thread-Technologie für das Mesh-Netzwerk. Grund ist die fehlende Forschung zu den Over-the-Air-Fähigkeiten dieser beiden Technologien. Eine Beispielarchitektur wird implementiert und anhand der gegebenen Anforderungen evaluiert. Darüber hinaus werden Bewertungen der Aktualisierungszeiten und der Zuverlässigkeit des entwickelten Systems durchgeführt und Vorschläge für weitere Verbesserungen präsentiert.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Abstract

Sensor nodes present an integral part of Home and Building Automation (HBA). These devices are usually connected with Wireless Sensor Networks (WSN). Due to the large number of deployed sensor nodes and a wide area they have to cover, updating them manually requires time and effort. This thesis investigates different technologies and standards that allow remote configuration of sensor nodes in HBA. Additional focus is given on the topics of security and reliability of an Over-the-Air (OTA) system. Based on the research, a set of requirements for a secure and reliable OTA system is developed. Special focus lies on the design choices of the combination of Zephyr Real-time operating system (RTOS) for the sensor nodes and Thread wireless technology for the mesh network. Reason is the lacking research on the OTA capabilities of these two technologies. An example architecture is implemented and evaluated against the given requirements. Furthermore, evaluations of update times and reliability of the developed system have been carried out and suggestions for further improvements are presented.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

Introduction

1.1 Motivation

As one of the trending terms and fields, the Internet of Things (IoT) has established itself as one of the many technologies that are significantly changing and shaping our society in the present century [1]. The basic idea of the IoT concept is the pervasive presence of a variety of things or objects around us - such as sensors, actuators, wearables or mobile phones - which, through unique addressing schemes, can interact with each other and cooperate with their neighbors to reach common goals [2]. The IoT can be seen as an umbrella term that encompasses an extensive scope of fields, from consumer wearables to medical and transportation uses [2]. Closely related and often brought up in the same context is the field of Home and Building Automation (HBA). HBA covers the combination of automation systems in both private homes and commercial buildings. The initial motivation of HBA systems was improved monitoring and control of Heating, Ventilation and Air Conditioning (HVAC) systems and lighting control [3]. Nowadays, there are other additional use-cases and broadly speaking, HBA systems are concerned with improving interaction with and between devices typically found in an indoor habitat [3].

An enabling part of HBA are Wireless Sensor Networks (WSN) [4]. These networks consist of tens, hundreds or thousands of sensor nodes, which can be placed into a wide variety of environments. Other than the sensors themselves, sensor nodes are often equipped with a battery. They need to operate for a long time without being connected to the power supply and have highly constrained hardware resources, meaning small memory sizes and relatively slow processors. To make wireless communication possible, sensor nodes need to have radio capability. Nodes are connected in a network and use the sensors to perceive their environment and send the information. Figure 1.1 graphically shows some of the common use cases of HBA for a private household in this case. To achieve communication between different smart device and exchange of data between

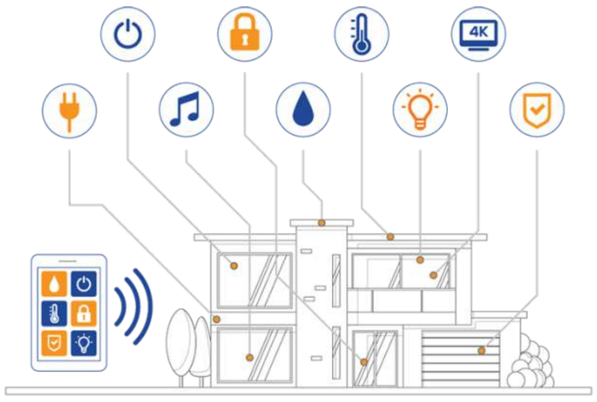


Figure 1.1: Typical HBA use cases[1]

them, these devices need be connected in a network. Common ways in which these devices can be connected in a network will be covered in Section 2.3 about topologies, but one particular topology which is suitable and popular choice for IoT and HBA is mesh. This type of topology is characterized by a rich interconnection between devices which provides more than one communication path for information and resiliency of the network [5].

One of the most challenging problems which arise with the use of WSN is its setup, maintenance and management [6]. As already mentioned, WSN can have hundreds or even thousands of nodes in the network. Setting up such a network is time-consuming and cumbersome, and it presents a task that ideally should be done only once. With the network in place, the problem of maintaining the network arises. In case of a bug in the firmware deployed on the nodes in a WSN, having an operator who has to replace the devices or manually update the firmware is time-consuming and impractical. In case of building automation, this problem is even more highlighted due to larger size of the network and with scaling of the network, the problem gets increasingly worse. Bugs in the firmware are not the only reason that would require an update; a simple software version update yields the same described issues. An interesting use case would be the addition of a new sensor on the already deployed sensor node. By enabling an automatic update of the sensor node when a new sensor is attached, a really interesting interconnection between hardware and firmware would be reached.

This problem serves as the primary motivation for investigating the ways of remotely managing and updating the sensor nodes in an already deployed WSN. The problem of Over-the-Air (OTA) updates is multilayered and requires broad knowledge to be investigated. Various design choices need to be evaluated before building a system that supports such updates. A typical OTA system contains three layers: management, network and node layer. Each layer of the system has its specific problems which require

[1]https://www.eletimes.com/knx-community-welcomes-the-future-of-home-and-building-automation-with-ets6

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

careful analysis. On the management level of the system, it is important to have a clear and reliable overview of the network itself. Furthermore, questions of how to prepare, possibly reduce the size and insert the update are all covered on the management level. On the network layer there exist different wireless technologies and protocols which have a great impact on the behavior and performance of the network. It is important to investigate and compare these technologies in terms of their applicability for WSN in HBA. Particularly interesting is the new standard, Thread, that is a result of a wide industry collaboration and has recently been called one of the best inventions of 2022 by the Time magazine[2]. Since sensor nodes in WSN are constrained devices without much computing power and radio transmissions are one of the most resourceful activities a sensor node does, choice of a wireless technology and network protocol is of great importance. Other problems that can arise on the node layer are update activation, detection of faulty updates and recovery mechanism, when such faulty updates occur. These problems are operating system (OS) dependent and choice of an OS which runs on sensor nodes influences the implementation and the approach to them. One more important issue that encompasses all of the layers and concerns the whole system is the reliability and security of an OTA system. In HBA, the collection and exchange of potentially sensitive data occurs and giving away control of devices to adversaries could cause various unwanted problems. That is why this concern has to be addressed in design and implementation of an OTA system.

This thesis investigates all of the above mentioned subproblems with the goal of understanding, implementing and evaluating possibilities for remote configuration of sensor nodes in HBA. By having a system for remote configuration of sensor nodes where the Firmware Over-the-Air (FOTA) update can be executed, issue of managing a WSN after deployment would be greatly simplified. Furthermore, this would also prolong the lifetime of WSN, since using such a system would even allow the use of the already deployed WSN for different use cases. The aim of the thesis, research questions and thesis outline are described further in this chapter.

1.2 Aim of the work

The aim of this thesis is to investigate technologies and protocols suitable for WSN in HBA, that allow reliable and secure update and maintenance of sensor nodes without the need for physical access. Additionally, the thesis identifies necessary requirements for such a network and presents and evaluates a potential architecture that fulfills those requirements.

To support the aim, the following three research questions have been formulated:

- RQ1: *What are the available technologies for the remote management of WSN in HBA and which communication protocols are supported?*

[2]https://time.com/collection/best-inventions-2022/6228191/thread/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- RQ2: *What are the minimum requirements necessary to enable secure and reliable automatic update of sensor nodes within a WSN in HBA?*
- RQ3: *How does a network that satisfies the minimum requirements perform regarding common performance measures like update times and recovery?*

Delimitations

The thesis doesn't try to measure the power dissipation of the OTA, or the memory footprint on the sensor nodes. These common measurements are important and should be a topic of further investigation, but they are out of scope of this thesis. Furthermore, there is a security and reliability analysis, which was carried out before the system design in order to set the requirements. Additionally to that, there was an evaluation of the implemented system against these requirements and a rudimentary test of the claimed security. But the thesis does not cover a thorough security analysis and testing different approaches in order to penetrate the network. Such a security analysis is of course important and useful and it should be conducted by the experts of the field.

1.3 Thesis outline

The thesis has the following outline that also represent the main tasks executed during the thesis work:

1. **Literature review:** First, an extensive review of the literature relevant to the topic has been conducted to develop and increase the knowledge of the topic. Literature provides insights into different ideas, currently used approaches and state-of-the-art solutions for the remote control of sensor nodes. This work was needed in order to identify all the different parts of the network that should be present and their respective roles in the big picture. Special focus was given to the research of reliability and security concerns of the OTA system.
2. **System planning and design:** After gathering the knowledge through literature review and learning the different approaches, a system design was made. A list of minimum requirements for an automatic update of a sensor node has been set. These requirements were gathered based on the most important aspects of an OTA system, which were identified during the literature review. The most important design decisions that greatly influence the implementation, like the use of mesh topology, which network technology to use and what OS should run on the sensor nodes, were discussed more extensively in the system design and the decisions taken have been justified.
3. **Implementation:** After the necessary know-how was gathered and with the system design as the starting point, implementation of the system had started. Due to the complexity of the task, a careful plan of smaller steps was made and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

by following those, a final solution was reached. To accomplish this goal, it was necessary to gather the hardware needed. For sensor nodes which constitute the mesh network, nrf52840 micro-development-kit boards are used. These devices have less computational power and resources since they should expend less energy. Attached to them are Bosch BME680 sensors that are used for environment sensing. As an edge/border device Raspberry Pi 4 [7] is used, which has enough resources to accomplish the task of an edge device successfully. Following the laid out plan and taking into consideration the system requirements; development and integration of all the different layers of the system had been successfully undertaken and the implementation process was documented accordingly in Chapter 7.

4. **Evaluation:** With the system in place, it was evaluated against previously set requirements. Furthermore, measurements of update times in different scenarios and testing of the system's security and reliability were carried out. This analysis gave insights into possible future improvements that could be made and to future research that can be based on this thesis. Discussion regarding the initial goals of the thesis and the results that were reached was also carried out.

1.4 Thesis structure

Chapter 2 introduces the technical background of WSN and gives a systematic overview of the current state-of-the-art work regarding remote configuration of WSN and HBA. A special Chapter 3 focuses on the security concerns of the OTA and what measures should be taken to minimize them. After that, Chapter 4 describes the scientific methodology used in this thesis. In Chapter 5, a detailed system design is presented, investigating different parts of the system and evaluating the design choices to be made. A requirements list on which the implemented system is to be evaluated against was also defined in this chapter. A detailed analysis of the open-source version of Thread that was used, OpenThread, was given in the following Chapter 6. Chapter 7 describes the implementation of the system, which is based on combining the knowledge of Chapters 3, 5 and 6 on the available hardware. In Chapter 8, the implemented system was evaluated against the minimum requirements set and performance and reliability of the system were tested in different set ups. Furthermore, discussion about the results of the thesis is also included in this chapter, together with some guidelines for future research. Lastly, the thesis is summarized in Chapter 9.

1.5 Contributions

This thesis contributes by combining the use of Zephyr and Thread and enabling OTA updates in this setting, which was until the date of writing this thesis only experimental. Furthermore, security and reliability aspects of this Thread and Zephyr based OTA system, together with measurements regarding update times have been evaluated, which is an additional contribution to the overall knowledge in the field.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Background and related work

2.1 Home and Building Automation

Over the course of several years, use of automation in residential and commercial buildings has been on a steady rise. Several reasons are driving this increase and adoption of this complex technology will most likely be broader with the upcoming years. For start we can take the definition of building automation from the Association of German Engineers (Verein Deutscher Ingenieure) as following: Building automation is the computerized measurement, control and management of building services [8].

When talking about HBA, there are a lot of overlapping use cases, but also a clear distinction should be made between the two. When we talk about home automation, we usually think of houses or apartments that are used for residential purposes by the general public. One of the most widespread examples for that is the use of sophisticated thermostats for measuring and controlling the temperature of rooms. Another comparatively simple use case is the automatic lighting control, which is easily implemented with motion detectors and optional addition of brightness sensors to make the control adaptive to the time of day. In general, as described in Merz et. al. [9], use of automation in homes focuses on:

- Cost effectiveness/energy preservation
- Comfort and convenience
- Security

When we talk about building automation, we are thinking on a much larger scale with more complicated systems. Modern buildings contain a variety of HVAC systems, as well as numerous security systems and features. Important differentiation to the home automation is the need for flexibility in building automation. Buildings like offices,

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

shopping centers, conference centers, hospitals and others often change the functionality and the layout of particular rooms. Building automation systems should greatly simplify the adaption of different components to their new functions, by simply reprogramming them instead of dealing with the complicated rewiring. That is why, Merz et. al. [9] names the following motives as primary in building automation:

- Cost effectiveness/energy preservation
- Communication via bus systems and networks
- Comfort and convenience
- Flexibility

With the increased number of connected devices in building automation systems, electrical consumption also increases. That is why energy preservation and efficient operation are essential for building automation.

For the operation to be efficient and correct, there needs to be a way of monitoring and controlling the system. To enable that, building automation systems have traditionally been divided into Management level, Automation level and Field level. The Field level is comprised of field devices: sensors and actuators. Their tasks are switching, setting, reporting, measuring and metering. The Automation level consists of various systems in the building and its task is the control and regulation of these systems. Data to the automation level is supplied from the field and the specifications are provided by the management level [10]. The Management level is responsible for monitoring and controlling the whole system, while the information exchange between individual systems (HVAC, Sanitation system, Burglary system etc.) generally happens on the automation level, which is a so-called horizontal communication [9]. There is also a vertical communication which takes places across different levels. However, today's systems do not fully follow the classical model presented on Figure 2.1.

Communication in wired systems is done over field buses and in building automation field buses that are most used are KNX, LonWorks and BACnet. All three of these standards are open standards which have gained widespread use in Europe (in case of KNX) and around the world as well (in case of BACnet and LonWorks). For a detailed explanation of these standards an interested reader can find more information in Kastner et al. [3].

2.2 Wireless Sensor Networks

WSN are rapidly becoming the main technology in HBA in recent years and therewith replacing the previous wired technology. Reasons for this are various. The most obvious advantage of wireless technology is the removal of wires, which are not just optically bad looking, but also present an additional cost during the installation. Sometimes it is also not possible to connect the devices with the cables due to the unavailability of places

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

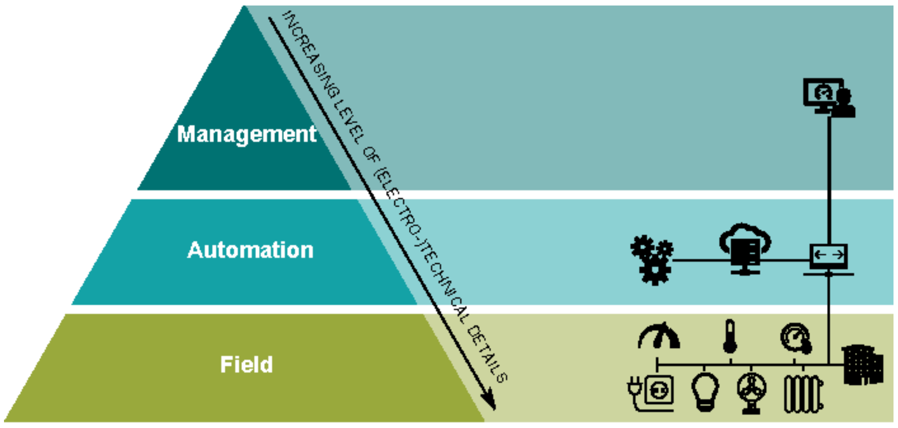


Figure 2.1: Automation hierarchy according to [3], taken from [11]

or due to safety issues. Another important advantage is the adaptability of the system. With an already existing network of devices, it is much easier to add or remove devices from a wireless network, than from a wired network.

WSN also have their disadvantages and therefore introduce some problems which we need to be aware of. When talking about wireless networks in HBA, most commonly devices we talk about do not have power wires either. They are instead powered by batteries. This makes the battery life of a device one of its most important characteristics, since there are usually numerous devices in one WSN and a frequent exchange of batteries in such a setting is not feasible [4]. Therefore, power consumption is the key characteristic that defines devices and technologies used in WSN.

With the use of an open medium, which a radio channel is, there is always a problem of interference from other resources that are also using the same medium. Two networks that use the same protocol and are located one next to another present an example for potential interference. Even bigger problem is the coexistence of different technologies and standards that use the same ISM (industrial-scientific-medical) frequency bands (e.g., 2.4 GHz frequency band) [4]. Due to this interference, WSN in general provide lower reliability in communication and higher packet loss. Furthermore, devices can fall out from a network, either by turning off due to the battery running out or by becoming unreachable due to interference.

Another important issue is the security of the data transfer, which is an additional concern that comes with the use of an open medium. There exist many devices (like SDR - Software Defined Radio) which make it possible to listen to/sniff the data being transferred with radio technology [12]. Due to the nature of data being collected in HBA, which can be extremely personal, this also presents a privacy concern. It also leaves the door open for man-in-the-middle attacks, when data is captured, modified and then passed further in system. The problem of security is highly important and it will be addressed with more care later in the thesis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

With all of the named disadvantages and potential problems that come with a switch to a wireless medium, WSN provide much more functionality and scalability. Exactly for these reasons they are the preferred technology in HBA for some time already and that will remain so for years to come. Before introducing wireless technologies and protocols that can be used, we are going to present different topologies in which a WSN can be organized.

2.3 Topologies

WSN usually consists of a large amount of sensor nodes and a base station, which acts as a bridge to the outside world (e.g., Internet). First, it is important to differentiate between two basic architecture options: single-hop and multi-hop architecture. In single-hop architecture all sensor nodes are directly connected to the Base Node, as can be seen on Figure 2.2. Because of this all data is transferred with only one single hop and there is no communication between sensor nodes. This network architecture is easier to implement, but it severely limits the scalability of the network and distance of the transmission. Because of that, a multi-hop architecture is used more often. Instead of a single link between the Base Node and sensor nodes, data can also be transmitted between sensor nodes (see Figure 2.4). This allows for much easier scaling of the network. Now specific network topologies that are used are going to be presented.

2.3.1 Star

Star topology represents the typical single-hop architecture. All the sensor nodes are directly connected to the Base Node and there is no communication links between sensor nodes. Therefore all communication has to be routed over the Base Node [13]. This gives us a single path of communication, but also a single point of failure. This is the simplest topology there is and it is really easy to set up, but as already mentioned it is not resilient and also not really scalable.

2.3.2 Tree

Networks in a tree topology have a root node at the top which acts as a central router/sink. One level below consists of sensor nodes to whom the central router acts as a parent and these nodes in turn have children nodes underneath them, thus forming a logical tree. Nodes that are at the end of the tree and have no children nodes are called leaf nodes. During data gathering, information is passed from the leaf nodes through the parent nodes to the root node. Idea behind using a logical tree is that it avoids flooding and data can be sent using unicast instead of broadcast [14]. By doing this the network consumes less power. However, a long delay for the data from leaves to root node is introduced. Furthermore, nodes that are closer to the central router are more active than the nodes further down the tree, thus they consume more power, making them more vulnerable to die. Tree topology is not resilient to node failures, when a parent node fails, the entire sub-tree is cut off from the rest of the network.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

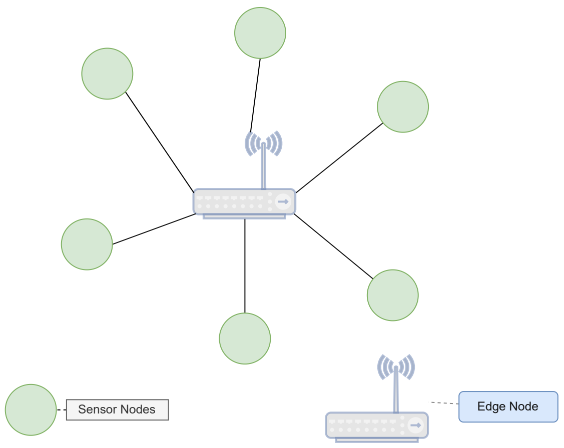


Figure 2.2: Star topology

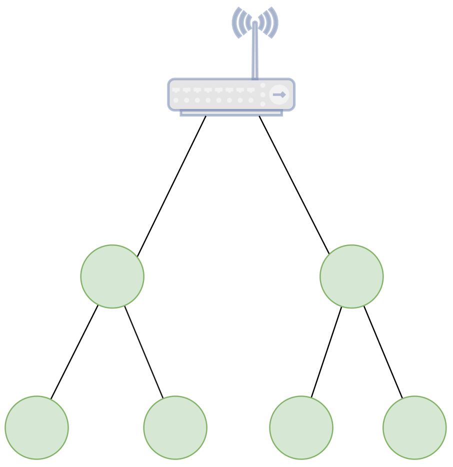


Figure 2.3: Tree topology

2.3.3 Mesh

In a mesh network there are multiple connections between sensor nodes and therefore multiple communication paths that a message can take. When compared to the previously described Star and Tree models, Mesh network does not have a central point node, which could be a single point of failure and therefore presents a security risk. Because of that, this is the most reliable and resilient network topology. Another advantage of mesh networks is that they can scale really easy. But due to multiple communication paths and redundancy it is also the most complex topology there is and consumes a lot of power [15].

A mesh network where all nodes connect to all other nodes is called a full mesh (Figure 2.5).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

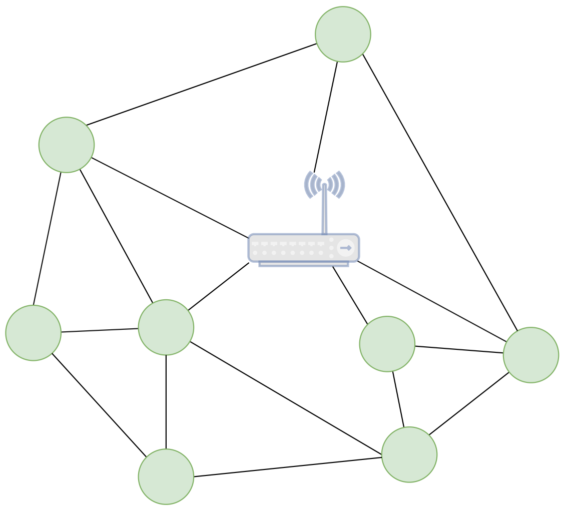


Figure 2.4: Mesh topology

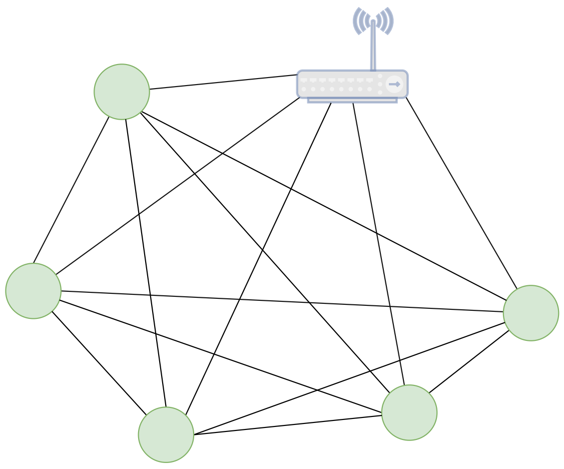


Figure 2.5: Full Mesh topology

2.4 Standards

We already mentioned some of the standards used in wired HBA systems. As with any other type of network communication, WSN also need strictly defined protocols for the network to be able to operate without problems. Some of the very popular wireless protocols are Bluetooth and WLAN (IEEE 802.11 standard). These standards are widespread in devices used by general public, but due to their complexity and high demands on energy, they are unsuitable for devices used in WSN. Because of that, IEEE 802.15.4 standard was developed for use in Wireless Personal Area Networks (WPAN).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

2.4.1 WiFi

WiFi (Wireless Fidelity) is based on the IEEE 802.11 family of standards which define the Wireless Local Area Networks (WLAN). It was developed as a wireless replacement for the popular wired IEEE 802.3 Ethernet standard and as such, it was created for Internet connectivity in mind from day one. It operates in standard 2.4 and 5 GHz bands available worldwide [16]. Standards 802.11b/g/n work on the 2.4GHz band, while standards IEEE 802.11a/n/ac use the 5GHz band. Newest 802.11ax (also known as Wi-Fi 6 and Wi-Fi 6E) version of the standard besides 2.4 and 5GHz bands also makes possible the use of a much wider 6GHz band [17]. Because it uses 14 partially overlapping 22 MHz wide bands in 2.4GHz frequency, it has massive bandwidth, and consequently, allows extremely fast data rates for a wireless medium (of 54 Mbit/s or even more) [18]. Due to its widespread use in consumer electronics and native compatibility with IP, WiFi provides a good option for reusing the already present architecture for IoT purposes. The biggest disadvantage of WiFi is its high power consumption. Devices using WiFi have to be charged (either by grid connection or solar energy), otherwise their battery will quickly discharge. This is a major problem which prevents the use of WiFi in WSN.

To tackle the problem of high power consumption, Low-power WiFi or WiFi HaLow was introduced by WiFi Alliance (as 802.11ah standard) in 2017. Its operation is in 900 MHz frequency band, offering longer range range and lower power consumption. There is also a sleep mode introduced, which additionally saves power. However, WiFi HaLow is not compatible with WiFi and it still does not provide meshing ability [18].

2.4.2 Bluetooth

Bluetooth is a short-range wireless technology, named after an ancient Scandinavian king and invented by Ericsson in 1994 as a standard for wireless communication between phones and computers [16]. Link layer was previously standardized by IEEE as 802.15.1, but the standard is no longer maintained by IEEE today. Bluetooth is now managed and controlled by Bluetooth Special Interest Group (SIG), which has more than 35000 member companies [19]. It operates in the 2.4GHz frequency band, has a short range of approximately 10 meters, and due to that also requires less power than WiFi. Maximal number of devices in one network, which has a master-slave structure, is eight. These devices form a piconet and by sharing common nodes between piconets, a scatternet can be created. A lot of consumer electronic devices already have Bluetooth transceivers and this widespread adoption, combined with low power consumption, make Bluetooth a popular choice for IoT uses in HBA [20].

Bluetooth Low Energy (BLE) (also known as Bluetooth Smart) is a recent addition to the Bluetooth specification, added with the 4.0 version. It is designed for lower data throughput and significantly reduces the power consumption of Bluetooth devices, making it possible to have much longer operation for battery powered devices. Unlike standard Bluetooth, BLE can support unlimited number of nodes and allows only point-to-point and star topologies [18].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Bluetooth 5.0 was introduced in June 2016, with the intention on increasing the functionality of Bluetooth for IoT. It has quadruple the range, double the speed and increased data broadcasting capacity [18]. It also defines a mesh connectivity protocol with flooding used for routing. Flooding means that the packets have a limited number of transfers (hops) and are repeated by each device, node by node, until they come to their destination. To be able to do that, a node must constantly be in operation and not "sleeping", which means constant and increased power consumption [12].

2.4.3 Z-Wave

Z-Wave is a proprietary wireless technology developed by the Danish company ZenSys, licensed by Sigma Designs and promoted by the Z-Wave alliance [18]. It uses a very low-power RF radio and is designed for control and monitoring applications in HBA. Characteristics of the protocol show us the difference to previous two wireless technologies and highlights the fact it was purposefully developed for use in WSN. Z-Wave operates in a sub-1 GHz band and therefore is immune to the interference of 802.11 and 802.15.1 devices. Additionally, the standard allows full mesh topology, but the size of the network is limited to 232 nodes [21]. There exist two types of devices in a Z-Wave network: controllers and slaves, where slaves execute and reply to the commands sent by the controllers. Maximum payload of the packets is 64 bytes and allowed transmission rates are 9.6 Kb/s, 40 Kb/s and 100 Kb/s. Since recently it also has the support for IPv6 and uses AES-128 encryption [18].

2.4.4 IEEE 802.15.4

IEEE 802.15.4 is a short-range wireless technology, developed to provide applications with relaxed throughput and latency requirements [21]. It can be said that unlike the previous two technologies, this one was specifically designed for the implementations of WSN. Standard is being maintained by the IEEE 802.15 Working Group (https://www.ieee802.org/15/). When looking at the OSI model of network communication (see Figure 2.6), IEEE 802.15.4 standard covers the two lowest levels of it, Physical and Data Link layer. The layers above are left unspecified.

Physical layer
IEEE 802.15.4 operates in three different unlicensed bands according to the geographical area where the system is deployed [21]. A total of 27 half-duplex channels are specified and they are divided across the three frequency bands[21]:

- 868 [MHz] band - one channel with data rate 20 kbps is available
- 915 [MHz] band - ten channels with data rate 40 kbps are available
- 2.4 [GHz] band - sixteen channels with data rate 250 kbps are available

IEEE 802.15.4 allows devices to be in a "sleep" mode, with both transceiver and receiver disabled, which presents the biggest advantage of the standard.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

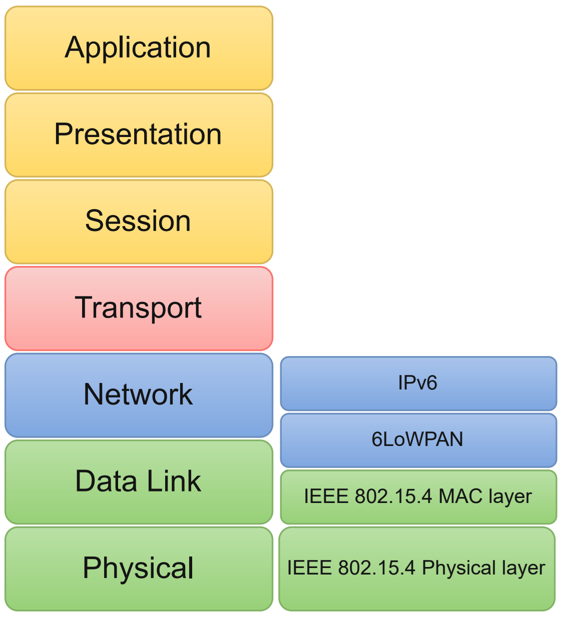


Figure 2.6: OSI network model and place of IEEE 802.15.4 and 6LoWPAN

MAC layer

The standard defines two operational modes, *beacon-enabled* and *non beacon-enabled*, which correspond to two different channel access mechanisms [21]. In a beacon-enabled mode, access to the channel is managed by a superframe, starting with a packet, called beacon, transmitted by WPAN coordinator. In a non beacon-enabled mode, an unslotted CSMA/CA (Carrier-sense multiple access with collision avoidance) protocol is used by nodes for the channel access [22].

Two types of devices in a network are defined: a full-function device (FFD) and reduced-function device (RFD). The difference between them is that FFD is capable of serving as a personal area network (PAN) coordinator, while RFD does not have this ability. Every network must include at least one FFD and every RFD only associates with only FFD at a time [22]. Both standard network topologies (star and peer-to-peer) can be used, as well as the combinations of these like cluster tree and mesh. Physical data frames have at most 127 bytes and such packets are small to minimize the probability of errors [18].

Since the standard does not define any requirements for the higher levels, this leaves a lot of room for different protocols to be built on top of it. They use the 802.15.4 standard for the lowest two layers and expand on it to define the specifications for higher levels. This is the case for ZigBee, ISA100.11a, WirelessHART, MiWi and Thread [18]. A couple of these industry standards and their characteristics are going to be described later in the chapter.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

2.4.5 6LoWPAN

In 2007 Internet Engineering Task Force (IETF) introduced IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) (RFC 4944). As already mentioned, IEEE 802.15.4 defines bottom two layers of the OSI model and the idea was to bring IP connectivity over 802.15.4. After the initial release (RFC 4944 [23]), further updates were given with RFC 6282 [24] (Header compression), RFC 6775 [25] (Neighbor discovery optimization), RFC 8931 [26] (selective fragment discovery) and some smaller changes added in RFC 8025 [27] and RFC 8066 [28]. 6LoWPAN describes an efficient adaptation layer between IEEE 802.15.4 and Internet Protocol Version 6 (IPv6), which can also be seen on the Figure 2.6. This adaptation compresses the 60 bytes long headers of IPv6 to 7 bytes and fragments the 1280 bytes long IPv6 packets to fit 127 bytes long 802.15.4 packets.

Internet Protocol Version 4 (IPv4) is not supported. Reason for choosing IPv6 over IPv4 is a much larger address space, which is essential for IoT devices and WSN, where there is a large amount of individual nodes. There is no standard organization to run certification programs for 6LoWPAN solutions [16].

By using 802.15.4 for bottom two layers, 6LoWPAN shares the technical specifications with it (large network size, 868 MHz/915MHz/2.4GHz ISM bands, mesh network topology, reliable communication and low power consumption). The biggest importance of 6LoWPAN is the IP interoperability, meaning that a 6LoWPAN device can communicate with any other IP-based server or device on the Internet. Only requirement is that a 6LoWPAN network has an Ethernet or Wi-Fi gateway to access the Internet. Since IPv4 is still much more present than IPv6 on the Internet, this 6LoWPAN gateway typically includes an IPv6-to-IPv4 conversion protocol [16].

2.4.6 ZigBee

Maintained, promoted and managed by The ZigBee Alliance, ZigBee technology is named after the Waggle Dance done by the bees to communicate the distance, direction and type of food they found. This is analogical to the way that data hops from node to node in multiple directions and paths in a mesh network [16].

ZigBee is a 802.15.4 based technology [29] and shares its technical characteristics. Mainly operating in 2.4 GHz ISM band, it also supports the 868-MHz and 915-MHz bands. It can deliver up to 250KBps of data throughput, but is usually used at lower data rates (20-40 KBps). It also has capability for very long sleep intervals and low operation duty cycles. Some of the newer ZigBee devices have energy harvesting techniques which can enable battery-less operation [16].

ZigBee is a transport-oriented protocol where logical links are established before any data transmissions occur. It supports large networks, with up to 6000 devices in one network [20]. One of the limitations is that it does not enable IP-addressing. Instead it assigns 16-bit addresses to the nodes that later must be translated into the IP-world [29]. Another major disadvantage is a lack of interoperability with other standards, meaning

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

ZigBee devices can only work with other ZigBee devices. This presents a big limitation in HBA. There is no path diversity either, meaning that if a path is broken, a new one must be set up [30].

2.4.7 Thread

In July 2014, a "Thread Group" alliance was announced by a working group with companies Nest labs (a subsidiary of Alphabet/Google), Samsung, ARM Holdings, NXP Semiconductors/Freescale, Silicon Labs and Yale as founders. This makes Thread the youngest protocol of all presented here. Since then, Thread Group has grown to more than 230 members including Apple, Amazon, Qualcomm, Siemens, Nordic Semiconductors and Silicon Labs among others [31].

Like ZigBee, Thread is also a 802.15.4 based technology and a direct competitor to ZigBee. Because of that Thread shares the technical specifications of 802.15.4 for the lower two layers. This gives us pretty similar characteristics to ZigBee, operation in 2.4 GHz frequency band, with data rate of 250 kbps. What differentiates Thread from ZigBee is the integration of 6LoWPAN.

Thread has integrated 6LoWPAN and uses its compression layer and link-layer forwarding [29]. Due to this integration, all sensor nodes in a Thread network have one or multiple IPv6 addresses. This means that devices can also directly connect to the Internet, which is a big advantage to ZigBee and other wireless protocols. Standard also uses UDP (User Datagram Protocol), which is a connectionless transport protocol. Hence it must rely on application layers like CoAP to cope with un-sequenced packets and retransmissions [29].

2.5 WSN lifecycle

Dwivedi and Vyas [32] have identified four main phases in the lifecycle of one WSN.

- Planning WSN - This phase consists of inspection of the deployment area, careful selection of the devices to be used, topology they should be arranged to, as well as operating system that is going to run on the sensor and edge nodes.
- Deploying WSN - An actual deployment of the network during which is important for sensor nodes to continually send their wireless connection quality and route (if applicable for the chosen technology) to the base.
- Monitoring WSN - Can also be thought of as the normal operation of the WSN. In this phase, the focus mainly lies on the values read and sent by the sensor nodes, as well as making sure the WSN is functioning properly.
- Controlling WSN - In this phase, network commands the devices to increase or decrease the time between messages, completely stop sending them, change the links and topology of the network and finally, update the firmware running on the sensor nodes.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Table 2.1: Comparison table, data taken from [18]

Wireless Technologies List						
Technology	Wi-Fi HaLow	Bluetooth	Bluetooth LE	Z-Wave	ZigBee	Thread
Standardization	IEEE 802.11ah	IEEE 802.15.1	IEEE 802.15.1	Proprietary	IEEE 802.15.4	IEEE 802.15.4
Frequency	900 MHz	2.4 GHz	2.4 GHz	900 MHz	868, 915 MHz, 2.4 GHz	2.4 GHz
Range (m)	<1000	1,10,100	50	30	10-100	20-30
Data rate	150-400, 650-780 Kb/s	1, 2, 3 Mb/s	1 Mb/s	9.6, 40, 100 Kb/s	20,40,250 Kb/s	250 Kb/s
Throughput	>100 Kb/s	0.7-2.1 Mb/s	305 Kb/s	-	10-115.2 Kb/s	250 Kb/s
Power consumption, mA	-	<30	<12.5	<23	<40	<40
Multiplexing	OFDM	FHSS	FHSS	FHSS	DSSS	DSSS
Modulation	BPSK, QPSK, 16-QAM, 64-QAM, 256-QAM	GFSK, $\pi/4$-DQPSK, 8DPSK	GFSK	FSK, GFSK	OQPSK, BPSK	OQPSK, BPSK
Security algorithm	WEP, WPA, WPA2	E0, E1, E21, E22, E3, 56-128 bit	AES-128	AES-128	AES-128	AES-128
Topology	star(one-hop)	p2p, scatter-net	p2p, star	star, mesh	star, tree, mesh	mesh

2.6 OTA update

First a clear distinction between software and firmware should be made. Software is usually thought of as a program, or a set of instructions used by a computer to perform a specific task. Firmware is generally a type of software used to control hardware devices. While software runs on top of the OS and has no direct interface with hardware, firmware is low-level software that stands between the hardware and the OS. Different types of software are application software, shareware, system software, etc. Examples of firmware are BIOS (Basic Input/Output system), EFI (Extensible Firmware Interface) for common motherboards used in personal computers, or TV firmware, router firmware and similar when talking about other consumer electronics [33]. Since the focus of the thesis are WSN in HBA, software on these devices usually interacts with sensors to get the data and can be considered as firmware. Since firmware is a type of software, both software

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

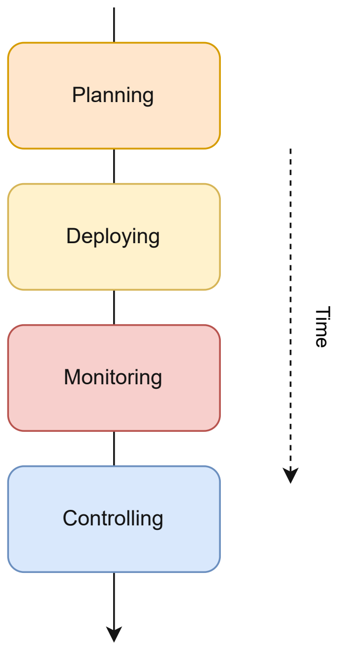


Figure 2.7: Life cycle of a WSN

OTA and firmware OTA terms are used in the thesis, but they are referring to the same process.

The problem of software update on the nodes can be divided into three separate parts as mentioned in [34]:

- Preparation of the software update
- Protocols for dissemination of the update
- Sensor node execution environment

These parts can be noticed on the software update system architecture picture (Figure 2.8).

Additional very important aspect that should be mentioned as a separate problem which can be added to the three mentioned above is the **reliability of the software update**. This aspect encompasses behavior of the nodes in a case of an unsuccessful or a faulty update. Furthermore, the **security of OTA update** and protection against the unwanted eavesdropping or other malicious attacks is of great concern. These problems are described in more depth in Chapter 3.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

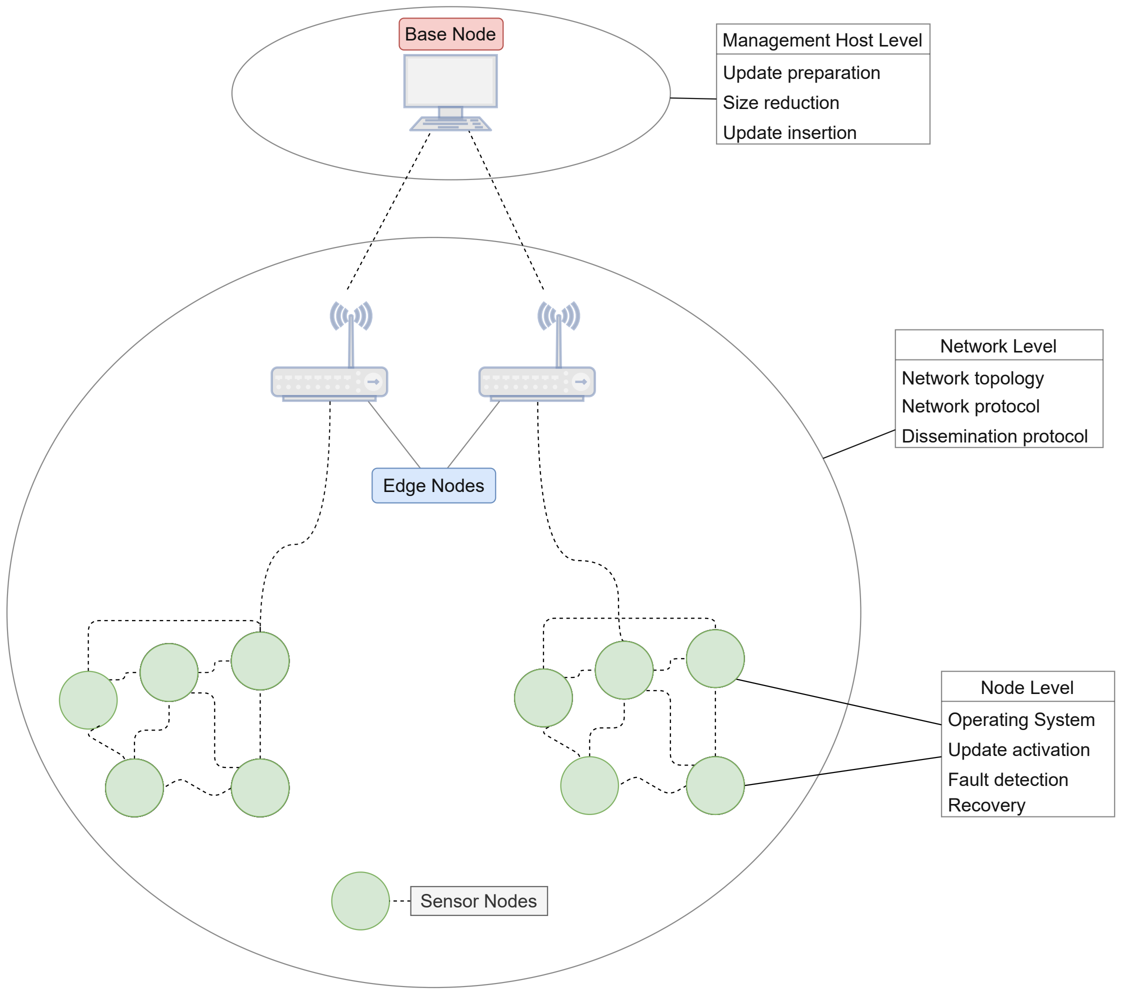


Figure 2.8: Architecture of a software update

2.6.1 Update preparation

Before inserting an update into the network, there are certain optimizations that could be done at the Base Node in order to make the whole process more efficient. Most of the energy used by a sensor node is during the time when its transceiver is operating. This means that the longer one node stays in the transmission, the more energy it consumes. Therefore, energy used is directly proportional to the size of the update that is being spread.

The least complex way to update a node is to build a full new image of the new update on the Base Node and to distribute it to every node. This is also the costliest way to do an update. It does not differentiate between having a completely new image and having just a couple of lines of code difference between the versions. It will always send the full image and in that sense be the least energy efficient way of conducting the update.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Improvements on this approach are trying to reduce the traffic caused by an update. Three main approaches according to Brown and Sreenan [35] are:

- Modularity
- Size reduction and
- Network coding

Modularity refers to the use of modular software, where only changed or new modules need to be transmitted. Size reduction is a method of comparing two versions of the software and performing a differential update, where just the changes from the previous version are transmitted and not the full image. This approach is specially efficient when there are just a few changes between versions, like is usually a case with small bug fixes, but it is not so useful with major updates where there are big differences between the versions. Finally, use of network coding maximizes the benefits of overhearing in a broadcast environment with loss, and thus reduces the traffic. Network coding is a networking technique used in multicast networks, where transmitted data is encoded at the routing nodes and then decoded at their destinations. In network coding, when a node receives two packages from two different sources, instead of simply transmitting both sources, it uses algebraic algorithms to merge those two messages and forward the result to the destination [36]. This means that fewer transmissions are required to transmit the data, but with more computational processing on the nodes. This method was initially introduced in 2000, in the seminal paper by Ahlswede, Cai, Li, and Yeung [37].

2.6.2 Update dissemination

When we look at the Figure 2.8, we can see that the network level is engaged in the task of data dissemination. Protocol for dissemination should distribute the update to all the nodes in the network. Furthermore, in the process of doing that it should use as little energy and resources as possible. A couple of well known data dissemination protocols are going to be presented and analyzed now.

Jeong and Kim [38] from the University of California developed the **XNP (Crossbow in-Network Programming)**, which is a native TinyOS OTA code distribution algorithm. It is a single-hop only mechanism, meaning that the base station transmits code to all nodes in the broadcast domain; thus, nodes outside the single hop boundary cannot be programmed. After transmitting the entire image, the base station polls each node for missing code capsules. Nodes scan the contents of EEPROM to find gaps and reply with NACKs if necessary, and then the base station unicasts missing capsules as required. Another big problem with XNP is that it sends the whole program rather than just the difference when it updates the program code with another version. This complete update approach leads to longer update times even when the difference between software versions is sparse. To improve this problem, Reijers et al. [39] proposed an energy-efficient code distribution scheme for the wireless update of the code in a sensor network. Instead

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

of distributing the whole image, only the changes to the currently running code are distributed, thus saving energy. It is possible to update all software on the nodes, including the operating system and the code distribution scheme. This scheme by Reijers et al. [39] notably reduces the amount of communication compared to the way of XNP and is an example of an improvement in the software update preparation part of the remote software update problem.

Mate [40] is a communication-centric middleware layer for WSNs, based on a virtual machine architecture, which runs on TinyOS. **Bombilla** is a virtual machine that runs under Mate and its successor. They present a completely different approach to others, since they are stack-based virtual machines. Mate includes three execution contexts, with two stacks per context, while Bombilla programs consist of capsules, with each capsule having up to 24 instructions. Nodes can forward capsules to other nodes. Every node will install a capsule that has a newer version number than one currently used [41].

As already mention, problem with the TinyOS native XNP algorithm is that it only supported single-hop OTA reprogramming. Stathopoulos et al. [41] developed a multihop network programming mechanism called **MOAP (Multihop Over-the-Air Programming)**. Resources that are focused on are energy consumption, memory usage and latency. The challenge of multihop network programming is avoiding saturation of the network while propagating the program codes over multiple sensor networks [42]. Stathopoulos et al. use a Ripple dissemination protocol to regulate the traffic. The ripple protocol disseminates the program code packets to a selective number of nodes without flooding the network [42]. This approach should lead to a 60 to 90% performance improvement in required transmissions compared to flooding. While MOAP advances the data dissemination problem, it still disregards many design decisions. It requires nodes to receive the entire code image before making advertisements, and it does not allow the use of spatial multiplexing to leverage the full capabilities of the network [42].

Trickle is an algorithm proposed by Levis [43] (who also developed Mate and Bombilla in 2002, which we previously described) and it uses a so-called "polite gossip". Nodes periodically advertise metadata info of their software to their neighbours. In a case that the node receives an advertisement of software which is at an older version than the current software on the node, it sends a code update to the "gossiper". If a node hears the metadata identical to its own, it stays silent. To achieve lower maintenance, period of metadata advertising changes. If there were no changes for some time, then the advertising period increases and if there are changes occurring, the node advertises more often. Time is broken down into intervals and at a random point in the interval, a node considers broadcasting its code metadata. If up to this point the node has already heard neighbors advertising the same metadata, it will not go further and broadcast its metadata. If a node hears that a neighbor has an older version, it broadcasts the code update to everyone nearby and brings them up to date. If a node hears that there is a newer version of the code, it then advertises its own metadata and following the first rule, other nodes will broadcast the code update accordingly. A limitation of Trickle is that it assumes that the nodes are always on. For conserving energy, long-term node

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

deployments often have very low duty cycles (1%), meaning that nodes are rarely able to receive messages [42].

Chlipala et al. [44] proposed **Deluge**, a reliable data dissemination protocol for propagating large amounts of data (more than can fit in RAM) from one or more source nodes in a multihop WSN. It builds on top of Trickle [45]. Representing the data object as a set of fixed-size pages provides a manageable unit of transfer, which supports spatial multiplexing and provisions for incremental upgrades. Deluge is similar to SPIN-RL [46] in that it makes use of a three-stage (advertisement-request-data) handshaking protocol. SPIN-RL is designed for broadcast network models and provides reliability in lossy networks by allowing nodes to make additional requests for lost data. This property is kept in Deluge, and it also allows it to be robust to widely varying connectivity scenarios. Since there is no need to maintain the state of neighbouring nodes, nodes may move, and connectivity can vary without requiring nodes to adapt to such changes. Further improvements to Deluge are Rateless Deluge and ACKless Deluge, both designed and implemented by Hagedorn et al. [47]. Both of these are rateless OAP protocols that replace the data transfer mechanism of Deluge with rateless analogs [42]. With increasing network size and density, performance of Deluge quickly drops. This drop of performance can mainly be attributed to negative acknowledgment (NACK) mechanism used to satisfy the reliability requirement. This mechanism requires every destination to notify the source about missing packets, which becomes a problem on a contended shared channel and leads to a so-called "NACK implosion problem" [47]. With use of rateless coding, receivers do not need to indicate which specific packets require retransmission. They instead just need to receive a sufficient number of independent packets, which can further be used to decode the original message, thereby contributing to communication and energy savings due to lower control messages overhead [47].

2.6.3 Sensor node execution environment

As explained in [35], execution environment on the sensor node governs how the software update may be propagated and executed. Some kind of low-level support to write to program memory always needs to be present for the software update functionality, but the rest of the functionality can be implemented in device-specific firmware, in the operating system, in a middleware layer, or as an application [35]. This is graphically shown in Figure 2.9, also taken from the same authors.

Most important software choice to be made for the sensor nodes is the operating system to be used. This determines the behavior of sensor nodes, standards and dissemination protocols that can be implemented, power consumption and thereby longevity of the network too, etc. Significant parameters that should be taken into consideration when choosing the OS for WSN will be presented now.

Architecture

The heart of an operating system is its kernel. There exist different variations of the kernel and they impact the size of application programs, as well as the way services are

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

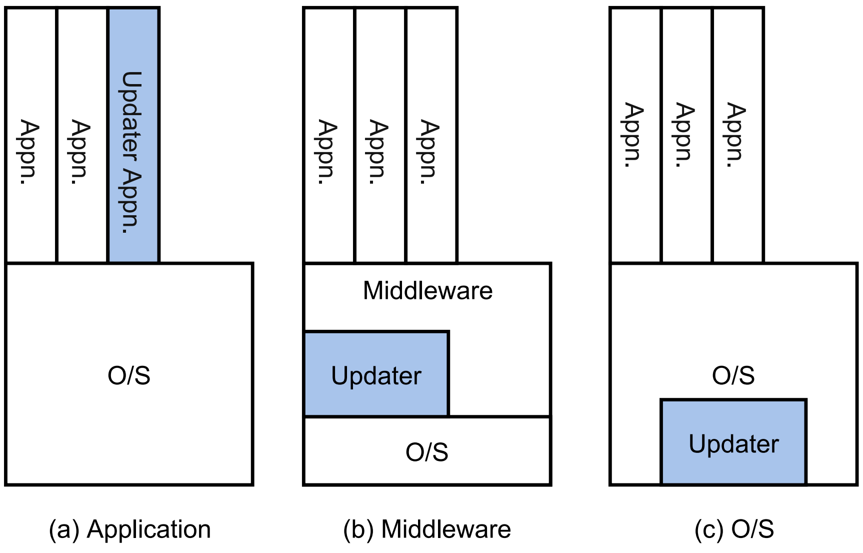


Figure 2.9: Execution environments possibilities, taken from [35]

provided. In [48] as important existing architectures are monolithic, microkernel, virtual machine and layered architecture named.

The monolithic architecture has the entire operating system working in kernel space. It is just a single process which lies in a single address space in memory. All the different services are combined in one system image. This results in a smaller memory footprint and the performance is usually higher. But the code is therefore longer and more complex, making it difficult to understand and configure. Another problem is that the failure of one component in a monolithic kernel leads to the failure of the entire system [49].

The microkernel architecture is in contrast to monolithic architecture. Here, the minimum functionality is provided in a kernel. A lot of the functionality of the OS is added through user level servers like a file server, a memory server, etc. This architecture provides higher reliability, easier customization and ease of expansion. But it does run slower than the monolithic architecture because of the frequent crosses between a user level and a kernel level [48].

In layered infrastructure services are implemented in the form of layers. The hardware is considered as the bottom layer, while the topmost layer is the user interface. Between them there are layers for scheduling, memory management, I/O buffers etc. The layers are designed in such a way that each layer uses the functions of the lower-level layers only. This architecture is manageable and easy to understand. It provides higher reliability when compared to the monolithic architecture, since a single module failure does not result in a complete system crash. Main disadvantage is that it adds restrictions to the design of the OS. Data needs to be modified and passed on each layer, which also adds overhead to the system [48].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Main idea of the virtual machine architecture is to export virtual machines to user programs. These virtual machines resemble hardware, since they have all the needed hardware features. Biggest advantage is portability, but this OS architecture typically has really poor system performance [48].

Scheduling

Scheduling refers to the algorithm used by the Central Processing Unit (CPU) that decides the order in which different processes should be executed. The latency, throughput, fairness and waiting time depend on the scheduling algorithm. IoT devices can be used in applications that have real-time requirements. If this is the case, then a real-time scheduling algorithm must be used, but since that is not always the option, there are a lot of operating systems that do not have a real-time scheduler. An operating system can have more than one scheduler and they can be selected at build time [48].

Memory management

Memory management refers to the process of allocating and de-allocating memory for different programs. The memory allocation can be static or dynamic. Static memory allocation is simpler, but it does not provide the flexibility that a dynamic allocation provides. On the other hand, dynamic memory management leads to a more complex system. Usually functions like *malloc* and *calloc* (which are used for memory allocation in C), do not work in real-time, so having a dynamic memory management conflicts with real-time requirements. To combat this, custom implementations of memory allocators like TLSF are needed. Many IoT operating systems do not have Memory Management Unit (MMU) and Floating Point Unit [48].

Networking

The networking stack is one of the most important aspects of an OS designed for constrained devices. All of the data that is being collected needs to get to the Internet so it could be recorded, analyzed and useful. Networking stack is also a way of maintaining an already deployed remote system. Conventional TCP/IP stacks are not suitable for IoT devices because they are not capable of operating with low power consumption. Mechanisms like 6LoWPAN, RPL (IPv6 Routing Protocol for Low-power and Lossy Networks) and Constrained Application Protocol (CoAP) are designed for low-power systems. Header compression and inclusion of minimal features help in keeping the protocols viable for IoT [48].

Portability

There is an extremely wide choice of microcontrollers and other hardware devices that are used in WSN and IoT. These range from 8-bit to 32-bit architectures. Due to the numerous use cases where WSN find place, there is no one-size-fits-all solution. Therefore it is good to give developers the choice to find the best device for a given purpose. That is why the operating system should be able to run on as many different architectures and microcontrollers as possible. It should also be easily portable, so making it compatible with a new architecture is possible without many problems [49].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Energy efficiency

As already said multiple times before, energy efficiency is the most important factor for devices in WSN. The devices used are usually battery-powered and for prolonged lifetime of the network it is crucial to use as little energy as possible. This also applies to the OS in use. It is difficult however to know exactly how much energy is used by a given OS on a given architecture, but due to the nature of the field where they are applied, all of the developed operating systems are optimized to be as little "resource-hungry" as possible [49].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

Security and Reliability

By the nature of their domain, HBA is a very security sensitive field. Due to widespread presence of electronic devices in one house or building and the sensitive data they share, it is of utmost importance to protect these devices from adverse attacks, even more so if they are equipped with actuators or other control mechanisms. One of the most vulnerable attacks in terms of resource depletion in WSN are Denial-of-Service (DoS) attacks, where legitimate users are prevented from accessing networks, services or other resources due to intentional unnecessary traffic [50]. There are other common attacks in WSN like Black hole attack, Wormhole attack, Selective forwarding attack and Flooding [50]. What is shared among these attacks is the fact that there needs to be an illegitimate node that somehow becomes part of the network. This means that ensuring that no unauthorized device gets control of the network resources is one of the major security tasks. When it comes to technology and software, security refers to the protection against deliberate harm, while reliability refers to the ability of the system to work properly under certain conditions [51]. Among usual security concerns which are present in WSN, having the FOTA capability highlights some of these security issues and adds certain reliability risks that need to be addressed. In this chapter, security and reliability concerns which are especially highlighted with the process of OTA in WSN have been presented, together with possible solutions to them. This serves the development of minimum requirements regarding reliability and security that need to be fulfilled by a FOTA enabled system, which are later used for the evaluation of the developed system.

3.1 Security

As already mentioned, security refers to the protection of the system against deliberate harm. In this sense, in order to regard a system as secure, following three security principles need to be satisfied: privacy, integrity and authenticity [52].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Privacy means that a piece of data cannot be read by unauthorized users or devices [52]. Ensuring privacy in HBA is of upmost importance in it of itself, even without having the FOTA capabilities. Sensor networks that are used in HBA are tools for collecting information, which can be personal itself, or can be used to extract personal information. With the introduction of FOTA ability to the network, ensuring the privacy of code being transferred to the devices is important. A skilled person that gets access to the image of the new update could maybe reverse engineer the image in order to gain knowledge about the source code. Since manufacturers usually don't want the competition to get knowledge about the code they produce, this presents a serious problem they would naturally want to avoid.

Authentication ensures that the firmware delivered to the sensor node is from a reliable and relevant source, not from anybody else [53]. Without authentication, it would be possible to distribute and install firmware developed by a third-party. This of course leads to an abundance of security problems and risks that come with having foreign firmware running on the devices in the network, since the control of the devices would be completely given up. Another side of the same issue is the use of genuine firmware on malicious hardware device. The device could be used for hacking purposes, reverse engineering and this also falls under a problem of authenticity.

Integrity means that the data sent from the origin is exactly same as the data received at the destination [53]. In an OTA system, it would mean that the firmware sent from the Base Node has not in any way been modified during the transmission and that it stays exactly the same on the node itself. An example of compromised integrity is if during the transport of the firmware, a third-party device gets access to the firmware, slightly modifies it so it still appears as genuine and then sends it further to the target device. This is a case of the famous man-in-the-middle attack, which results in maintaining control over data transferred among end users [54].

To summarize, if the three described security principles are not adhered to, following security problems can occur [52]:

- Use of firmware on an unauthorized device
- Use of untrusted third-party firmware
- Altered firmware
- Reverse-engineering of the firmware

3.2 Reliability

Reliability of the FOTA update could also be regarded as resilience and prevention of problems that can arise during the transport of the firmware to the nodes. It is crucial to ensure that sensor nodes have a working firmware on them at all times, since otherwise they cannot function properly or they can even be completely bricked, leaving the device useless. Difficulties of updating the nodes with OTA and possibility of bricking the sensor nodes were recognized by Strübe et al. [55] as well. Following three types of problems can arise during the transmission [52]:

- Transmission Error: During the transfer of the update, a part of the update has been damaged. This could be only a couple of bits that have been flipped, or if one or a couple of bits were not received by the device. This leaves the firmware corrupted.
- Transmission Failure: If during the transmission, a sudden power outage happens or the connection is lost due to any other reason, a device would end up with a truncated firmware. This means that a part of the update has not been successfully transferred. As a result, an application area would be corrupted and unusable.
- Information Loss: Parts of the update were not received during the transmission and the firmware on the device ends up incomplete. Everything that comes after the missing part is then corrupt.

Another important reliability aspect to consider is a behavior of the node in a case one of the above problems arises. In such cases, there should be a way of rolling back to the previous working firmware in order to avoid bricking of the device.

Possible solutions to security and reliability problems will be presented next. It is important to mention that implementing these solutions usually comes at a cost of size and speed of the system. So the most secure and reliable solution would also be the biggest and slowest one. That is why a careful analysis is needed in order to determine the right requirements regarding security and reliability.

3.3 Security solutions

As already introduced, in order to provide security, i.e., protect the device against deliberate attacks, three security principles that need to be satisfied are privacy, integrity and authenticity. If these principles are not adhered to, following dangers occur: use of firmware on an unauthorized device, use of untrusted third-party firmware, use of altered firmware and reverse-engineering of the firmware. Techniques that can be used in order to fulfill each of the three security principles will be presented now.

3.3.1 Integrity

Integrity can be violated either by purposeful modification of the firmware, or due to accidental modification. Latter one is an issue of reliability and is typically solved by using error detection codes (see Section 3.4.1) [52]. For checking if the integrity of firmware was purposefully violated, there exist several different techniques which are going to be presented now.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Hash Function

Idea of a hash function is to provide a digital "fingerprint" for every piece of data [52]. In order to verify the integrity of a firmware, its digital fingerprint is calculated and attached to the file. The bootloader receives both the firmware and the fingerprint, recomputes the fingerprint itself and compares the two. If they are different, that means that the firmware has been altered and that integrity has not been preserved.

For a string of any length on input, a hash function gives a fixed-size output called a message digest. It has the "good diffusion" property, which means that for slightly different input strings, digests that are generated vastly differ, a term first proposed by Shannon [56] in 1949. Since the output is fixed in size and the input string can be any length, one output has to be same for more different strings. This does not present too big of a problem however, since the hash functions ensure that it is almost impossible to find two different messages which will have the same digest. It is a requirement for a hash function that it should be computationally infeasible to find two distinct messages that hash to same value. That is why the problem of getting the same digest for different firmware is bound to the hashing algorithm used and with the use of newer and safer algorithms (SHA-2 and SHA-3 algorithms), the threat of getting the same digest for different string inputs is practically eliminated.

Problem with hash functions is that anyone can calculate them [52]. With a man-in-the-middle attack, an adversary could modify the firmware, calculate the new digest and send it further. A sensor node would not be able to tell that a modification was made. Nonetheless, hash functions can still be used at runtime, in order to avoid running a damaged application.

Digital Signature

Because hash can be computed by anyone, it is necessary to encrypt it. That is exactly what the digital signature does, digest of a firmware is computed using the hash function and then it is encrypted using public-key cryptography. Public-key (also called asymmetric) encryption uses two keys. The manufacturer uses his private key to encrypt the signature, while a device uses the corresponding public key do decrypt it.

The private key is known only to the manufacturer, meaning nobody except the manufacturer is able to produce the signature. In this way, the problem described above that can happen during a man-in-the-middle attack is avoided. Since the attacker does not have the private key, it cannot encrypt the digest and fake the signature. Still, anybody with the public key of the manufacturer can verify the signature.

Message Authentication Codes

Message Authentication Codes (MACs) have the same functionality and ideas as digital signatures, but they use private-key cryptography instead. Unlike public-key encryption which uses two different keys, private-key cryptography uses just one secret key, which is

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

shared both by manufacturer and the nodes. MAC will be computed and verified faster than digital signature due to private-key cryptography being much faster than public-key cryptography [52].

Having just one private key, MACs have two important differences to digital signatures. Firstly, anyone who can verify a MAC can also produce it. Secondly, if the private key inside the device gets exposed, then the security of the whole system is compromised. This means that if the adversary is able to retrieve the private key from the bootloader (which is supposed to be locked using security bits), then he will be able to modify the firmware and encrypt it as well. This would be then accepted at the sensor node, since it can not notice the difference to the case where the manufacturer encrypts the firmware with the same key.

This threat is also present with the digital signatures. If the private key of the manufacturer is exposed, the same problems arise as with the MACs. Adversary can modify the firmware, encrypt it with the private key and send it further. Only difference is that the private key is present in all devices of the network when using MACs, where private key is only present at the manufacturer when using digital signatures.

3.3.2 Authentication

Authentication is the process of verifying the identity of the sender and the receiver of a message. When applied to FOTA in HBA, it means verifying that the firmware has been issued by the Base Node and that the target node is a genuine one. It solves the security issue of use of an untrusted third-party firmware on a genuine device and the issue of using the authorized firmware on an unauthorized device [52].

Digital signatures and MACs, which are introduced as methods for ensuring integrity, are also methods for ensuring the authentication. The way how they do that is going to be presented in this section.

Digital Signature

Working principle of digital signatures has already been introduced. Since the Base Node is the only one that has the private key, a simple decryption using the accompanying public key is the proof that the firmware is coming from the Base Node and nobody else. An issue of course arises if the private key of the Base Node is exposed and known by an adversary, then the same adversary could encrypt any firmware with this key.

Message Authentication Codes

Unlike digital signatures, MACs theoretically cannot be used to authenticate that the sender is the one who created the message [52]. Since in MACs both a sender and a receiver have the same private key, it is indeed possible that either the sender or the receiver have generated any message. But practically, in a non-compromised system, use of MACs would guarantee the same authentication that is provided by the digital

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

signatures. This is because in such a FOTA system, the Base Node is the one and only part of the system that is building new firmware images and distributing them, a sensor node would never do that by itself.

The property of MACs that is different from digital signatures is the authentication of a target device. Since only valid nodes and the Base Node have the private key, there is no publicly known key, an unauthorized device is not able to decrypt the data. Again, if an adversary gets a hold of the private key, the whole authentication is compromised.

This shows that the biggest problem in both approaches, digital signatures and MACs, is making sure that the private key used for encryption never gets exposed.

3.3.3 Privacy

Data privacy is ensured by using encryption. By using an encryption key and a cryptographic algorithm, a cipher text is generated from the original one. Reading this cipher text without decryption is not possible, since the node just sees nonsense and is not able to extract any information from it. By enforcing privacy of the firmware, possibility of reverse engineering the source code is taken care of.

To ensure privacy, private-key encryption system has to be used. Use of public-key system does not in any sense ensure the privacy, since the public key is known by everyone and it can thus be deciphered by anyone [52]. Thus the private key is identical and only known by the Base Node and the sensor node.

3.4 Reliability solutions

In order to prevent reliability-related issues the following techniques can be used. As already mentioned, reliability problems mostly present the errors and losses which can happen during the transmission of the update to the sensor node.

3.4.1 Communication Protocol Stack

One of the systematical approaches to dealing with transmission reliability is the use of protocol stacks [52]. The standard OSI model with its seven layers represents the protocol stack. Each layer is responsible for providing certain functionality. This was explained in more detail in Section 2.4. Transmission reliability is usually implemented at the transport layer, which lies above the network layer protocols that were described.

Error Detection/Correction

In order to notice if the transmitted packet is corrupted or not, use of error detection and error correction codes is possible. Detection codes compute a mathematical value for the data packet to be sent (for example CRC (Cyclic Redundancy Check)). This value is transmitted together with the original packet itself. When the receiver gets the data, it calculates the same value using the same mathematical functions. If the calculated

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

values are equal, the transfer was successful. Otherwise, the packet is corrupted and the initial sender is asked to retransmit the packet. Correction codes in addition to the detection also have the ability to correct some of the errors. By doing so, request for the retransmission is not needed anymore, saving up time and energy. In the case of FOTA, the firmware is split up into parts and they are sent as packets. Error detection can be carried out on every packet, but error correction is not really necessary. That is more suitable for the cases where a lot of the data should be retransmitted, which is not the case with FOTA [52]. Furthermore, error correction codes introduces more overhead than error detection codes and shouldn't be implemented if not rally necessary [52]. Use of error detection/correction prevents the transmission errors to arise. There are still limitations to these codes, since they have a maximum number of errors that can be detected.

Block Numbering

As the name suggests, block numbering simply adds a sequence number to every transmitted packet [52]. The purpose of it is to notice the loss of a block of data or notice if two blocks arrive in the wrong order. Since the blocks are ordered, if the receiver gets a packet of a higher order than the one it was expecting, it can communicate to the sender that a packet is missing. This is crucial in a file-oriented transfer like firmware downloading is, since packets that would go missing would make the received code unusable. Use of block numbering solves the Information Loss problem.

Packet Acknowledgment

Another way of making sure that no packets get lost is the packet acknowledgment [52]. For every packet that the sender sends to the receiver, it would wait for the acknowledgment signal (ACK) from the receiver. If there is no acknowledgment after a certain period of time (timeout), then the sender assumes that the package has been lost and retransmits it. No further packets should be sent until the ACK signal arrives, to prevent receiving the packets in a wrong order. Just like block number, packet acknowledgment solves the Information Loss problem.

3.4.2 Memory Partitioning

The main idea of memory partitioning is to always have a copy of working firmware somewhere in the memory [52]. By doing this, there is always a possibility of rolling back to the working version of the firmware, if there process of update is not safely finished.

Memory and Memory Banking

Memory banking refers to the separation of the memory into one or several banks (or planes), which is common with microcontroller memory [52]. Usually, one or two banks are used, which are called single bank and dual bank respectively.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Single Banked Partitioning

When only one bank is used, it is partitioned in three different regions at all times [52]:

- Bootloader region, which is at the beginning of the memory and it is never rewritten or updated
- Application code region, where the running firmware is stored
- Buffer zone for downloading the new firmware, which has the same size as the application code region. If the firmware passes the verification tests, it can be copied into the application code region.

Dual Banked Partitioning

In dual banked partitioning, memory is divided into two banks of the same size and each one of these banks has the same copy of the bootloader at the beginning, followed with the firmware area [52]. When booting from one bank, firmware which is stored in this bank represents the current version, while the other bank is used as the upgrade buffer. After downloading the update into the other bank, new firmware version is verified and the boot banks are switched. Reboot follows, booting from the bank that has the updated firmware and the other bank can now be used as the upgrade buffer. In this approach, there can be two working firmware version in memory at the same time. There is a decrease in update time in comparison with single banked partitioning, since there is no need for any copying between banks or regions. However, since there has to be a bootloader in every bank, application memory is a bit smaller than in single banked solution [52].

3.5 Summary

To sum everything up, digital signatures and MACs present a pretty solid way to ensure all three security principles. Any FOTA enabled HBA system should use these in order to guarantee the security of the system. Nevertheless, information that was given in this chapter represents the base for developing the security requirements for the system, which will be laid out in Chapter 5. One important thing to keep in mind is that there exists no system that is completely secure. Outside of the software solutions that are going to be presented, there are also many hardware-based attacks like micro-probing, power analysis and timing analysis. These enable a hacker to break through software protections. In order to protect against these attacks, it is best to use a dedicated secure chip [52].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 4

Scientific Methodology

The research method applied in this thesis follows a design and creation research strategy and builds upon academic literature, Internet research and relevant specifications as well [57]. In information and computing research, the result of a design and creation strategy is a new IT product called artefact [58]. The artefact could be a construct, model, method and instantiation [59], although in creation of new knowledge a combination of several artefacts is usually used. Since artefacts are often computer-based products, a clear distinction from product development is required. That is why design and creation research strategy emphasizes on the analysis, explanation, argumentation, justification and critical evaluation of the results.

Design and creation research, focuses either on the artefact itself, (e.g., the IT application incorporates a new theory), the artefact as a vehicle to create new knowledge (e.g., the IT application in use) or on the process to create an artefact to create knowledge [60]. Within this thesis, the focus lies on gaining knowledge in the process of connecting two artefacts, addition of new functionality to the newly created combined artefact and the later evaluation. Specifically, addition of remote management to a combination of a new meshed network protocol with a new OS for embedded sensor nodes, creates new knowledge.

Another essential part of design and creation research is its problem-solving nature which utilizes the principles of system development [61]. The process involves typically five steps—awareness, suggestion, development, evaluation and conclusion. Whereby the steps are not rigid in order instead form an iterative cycle. As a significant effort in computer-based research often concentrates in the development step, it is necessary to apply a specific system development method (analysis, design, implementation and testing). Before system development, requirement specification was developed based on the information analysis. This research project uses partly a waterfall and prototyping methodology. Within both methods, analysis, design, implementation and evaluation is applied in an iterative way. Section 1.3 describes in more detail the simple steps.

Figure 4.1 visualizes the research method and the underlying applied strategies and data sources applied in this research project.

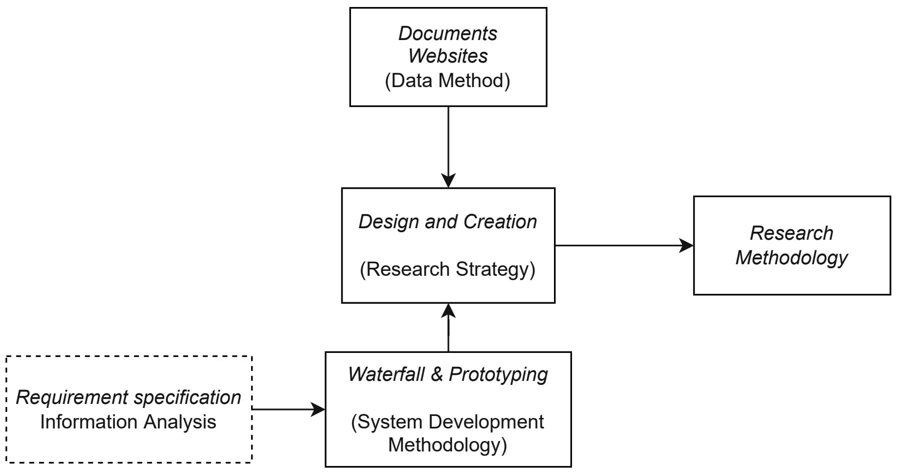


Figure 4.1: Research method overview and elements based on [57]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

System Design

Chapter 2 presented the technologies used in the field of HBA and showed how hard and diverse the problem of FOTA really is. Furthermore, Chapter 3 put special focus on the security and reliability issues of implementing such systems in a WSN. By following the aim of the thesis, this chapter uses knowledge from these previous chapters in order to set the minimum requirements list for the development of a secure and reliable automatic update of sensor nodes in WSN in HBA. Some of the design choices are also contained in the requirements and the chapter explains the reasoning behind narrowing down from the overall picture to these design choices.

5.1 System requirements

After detailed research of all the technologies that make up the secure FOTA system and with keeping the already made design choices in mind, a list of system requirements is now going to be presented. It would be preferable for the FOTA system to fulfill all of the following requirements:

- **Sensor nodes can be successfully updated**: Base requirement for the system is the ability to update the sensor nodes OTA. Without fulfillment of this requirement, none of the other listed ones cannot be reached and the goal of the thesis would not be reached.
- **Mesh topology**: Since the particular focus of this thesis is the use of WSN in HBA, mesh topology presents the best possible topology for the setting. Detailed explanation of different topologies in Section 2.3 serves as base for this requirement, but further explanation for the topology of choice is given in Section 5.2.
- **Operating system suitable for HBA**: Sensor nodes should run an OS suitable for their characteristics. This means low energy use and low memory footprint.

Furthermore, the selected OS should have good portability and IPv6 support, which is important for HBA. Further discussion of the OS design choice follows in Section 5.3.

- **Wireless technology suitable for HBA**: Building on the knowledge presented in Section 2.4, system should be developed using the appropriate wireless technology that supports the use of IPv6. This design choice is discussed further in the Section 5.5.

- **Possibility of selective updating**: A particularly practical feature that would be useful for real-world scenarios is the possibility of selective update. By having this option it would be possible to have various firmware on the nodes in the network, while maintaining a centralized management system. This is useful for both home and building automation systems, since it would allow presence of nodes particularly tailored for the specific room/area of the building. Hence, the usability of the network would be increased.

- **Reliability requirement 1: Resilience against transmission errors**: The system developed should be resilient against transmission errors. If errors arise during transmission, firmware on the device is corrupted and the normal operation is not possible.

- **Reliability requirement 2: Failed update recovery:** Due to operation in wireless medium, it is to be expected that there will be connection losses from time to time. This can happen during the update process. It is therefore a requirement to set up a reliable system that can adequately recognize when these events occur and that always leaves the node in a working condition.

- **Security 1: Protection against use of firmware on an unauthorized device**: It should be prohibited to get the firmware distributed in the network onto an unauthorized device.

- **Security 2: Protection against use of untrusted third-party firmware:** In no case should it be possible to run any other firmware besides the one authorized by the Base Node. If this requirement is not met, security of the whole network is compromised.

- **Security 3: Protection against alteration of the firmware**: Similar to the previous requirement, it is important to preserve and protect the firmware image from being altered at any point.

- **Automatic probing for the update**: The biggest benefit of FOTA is the ability to update the sensor node without having physical access to it. This implicitly means that a node automatically asks the Base Node if an update is available, without any interference from anyone else.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

- **Triggered probing of the update after attaching the new sensor**: A nice feature to have would be a triggered update after attachment of a new sensor. A sensor node runs a certain version of the firmware and when a new sensor is attached, it immediately asks for a newer version of the firmware which makes use of this new sensor.
- **Sending of an incremented update instead of a full image:** By having the ability of sending the difference between two firmware images, performance improvements can be achieved. Due to the lower image size, time needed for an update would also be shorter.

5.2 Topology

Why was a mesh topology chosen?

We have to think of the characteristics that are most important for HBA. Since the networks in use have to cover a large surface area, choice between a single-hop (star) and multi-hop architecture is relatively easy. The star architecture is limited by the maximal possible distance between the Base Node and the sensor node. There is no way of extending the reach of the network, which makes it a bad choice for our needs. Additionally, as already said in Section 2.3, this type of topology has a single point of failure, which presents a significant reliability concern. Because of that we have to chose among multi-hop architectures.

Both tree and mesh topologies allow the expansion of the network reach. Tree networks have advantage of consuming less power in comparison to the mesh networks, but like star networks they suffer from the single point of failure problem. When a parent node in tree dies, the entire branch is cut off, which is unacceptable in HBA. Mesh is more complex and uses more energy, but it provides multiple paths for data transmission and is much more resilient to node failures. These characteristics are extremely important and that makes this topology best suitable for our application, despite higher energy consumption, which is undesirable for WSN. Due to this reasoning, mesh networks are already popular and a first choice for HBA and IoT applications.

5.3 Operating system

Why was Zephyr OS chosen?

When taking into account characteristics presented in Section 2.6.3, it is evident that the right choice of operating system depends of sensor nodes is not a clear-cut decision. Some of the famous operating systems used in WSN are Contiki, TinyOS, RIOT, mbedOS, or Zephyr. All of them satisfy the base requirement of lower power usage and low memory footprint, but a lower subset of them satisfies the requirement for the network of choice. Table 5.1 shows the comparison of the operating systems, based on the work of Jaskani et al. [49].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Table 5.1: Comparison of different IoT OS as referred in on [49]

OS	Min RAM	Min ROM	Real-time	Architecture	Scheduler
Contiki	<2kB	<4kB	No	Monolithic	Cooperative, Preemptive
Tiny OS	<1kB	<30kB	No	Monolithic	Cooperative
RIOT	~1.5kB	~5kB	Yes	Microkernel	Preemptive, Priority based
Zephyr	~2kB to ~8 kB	~50kB	Yes	Microkernel	Preemptive, Priority based
Mbed	~5kB	~15kB	Yes	Monolithic	Preemptive

In the end, Zephyr turned out to be the OS of choice. It is younger than other comparatively famous operating systems like Contiki, TinyOS and RIOT. In recent years, it has been getting more and more attention from the academical circles. In addition to that it has a large industrial backing with companies like Intel or Google being involved and supporting the project. It satisfies the energy efficiency and networking requirements, with an impressive portability, since it can run on almost anything[1]. Another bonus is that it is a Real-time operating system (RTOS). When thinking about dissemination protocols from Section 2.6.2, it is noticeable that majority of described algorithms were initially developed for TinyOS, the most popular choice some years ago. There is also little to none research on the OTA possibilities for the Zephyr OS. All of this makes it a really good choice for the thesis.

5.4 Failed update recovery

What should happen in a case of a failed update?

In the process of updating the firmware of the device, due to some expected or unexpected reasons, it is possible for the device to not finish the update successfully. Particularly in a wireless environment, there is a high likelihood of a loss of connection happening during the lifetime of one network. Various problems can also arise in the process of update itself, not only during the dissemination of the update. In these cases, we are talking about a failed update. It is very important to think about is going to happen in this case.

If a system is not designed to be resilient to these circumstances, then it can easily happen that a device loses connection to the rest of the network, or in a worst case is completely bricked. The safest action for the device is to rollback to the previous version,

[1]https://docs.zephyrproject.org/3.1.0/boards/index.html

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

that was already running on the device. Some further actions a device can take is to retry the update, notify others of a failed update and ask for a new version. In any case, a device has to be left operational and in a safe state.

5.5 Protocol

Why was the Thread protocol chosen?

The choice of protocol to be used is crucial for the whole system. WiFi is the most widespread technology of all. It is to be found in almost every household and it is so wide-known that among the general population it became synonymous with the word "wireless". But with its immense power consumption it is not at all suitable for our purpose. WiFi HaLow has a much lower power consumption, but since it is not compatible with WiFi, it doesn't benefit from the already deployed WiFi infrastructure. Besides IP integration, it does not bring a lot of benefits in comparison to other technologies and it also lacks meshing ability, which is our preferred topology. Bluetooth is also extremely widespread and used by many people in different IoT gadgets like headphones, smartwatches, other wearables etc. It is definitely a better choice than WiFi for constrained devices if we take into consideration BLE version of Bluetooth. The meshing ability is also added in Bluetooth 5.0 specification. But when it comes to WSN, it still does not match the performance of other described protocols and is therefore not the popular choice for WSN and also not our choice.

The other three protocols that we introduced in Section 2.4 are Z-Wave, ZigBee and Thread. All these protocols are purposefully developed for WSN and present a better choice than the previous two "traditional" wireless protocols. All three of them have the meshing ability, which satisfies our set requirement. Z-Wave is an efficient and established protocol, which from recently also has the support for IPv6 and AES-128 encryption as already stated. A major throwback for our purpose and for the research community is the fact that Z-Wave is a proprietary technology. This makes it undesirable, since it would be very much preferred to work with an open source protocol.

Both ZigBee and Thread can fulfill this criterion. These protocols are very much alike, since they are both built on top of the IEEE 802.15.4 standard. This means that they share a lot of the technical characteristics between them. ZigBee is older and until now more used and accepted protocol. Thread, although younger and still not so widely accepted, is being backed by a Thread Group, which consists of an impressive list of members (already listed before). An additional distinction between two protocols is the integration of IP, which is present in Thread, due to the standard making use of 6LoWPAN, and lacking in ZigBee. These two facts are a driving factor for the increasingly fast adoption of Thread in popular products in the market. There is also an option to incorporate ZigBee into the Thread network at the application, making them interoperable and a complementary product in the future. Hence, Thread is the protocol of choice for our use, too. A much more detailed explanation of the Thread protocol, the way it works and its characteristics are described in the following Chapter 6.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

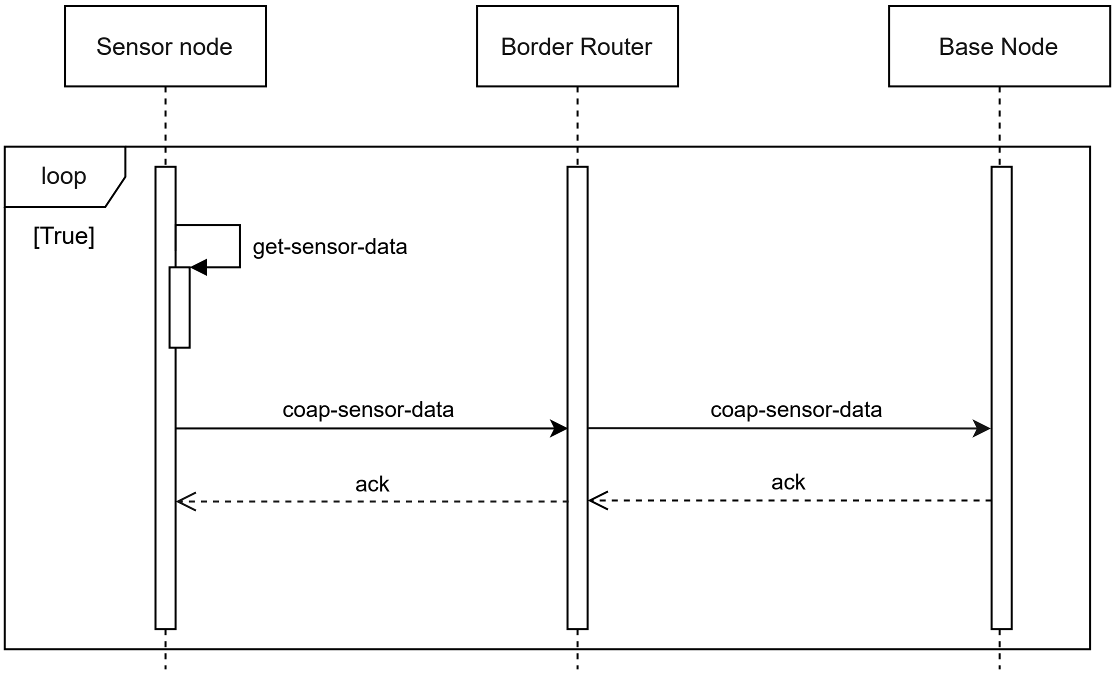


Figure 5.1: Sequence diagram for normal operation

5.6 System overview

With all the requirements for the system set, what is still left to explain is the overall view of the system and the interactions between the elements. During the normal operation of the network, sensor nodes are collecting information about the environment and sending this data over the Border Router to the Base Node. The UML sequence diagram describing the normal operation is shown in Figure 5.1.

In the case a new sensor is attached to a sensor node, a bit more complicated sequence of action is carried out. After the sensor is detected by the sensor node, it checks for the update and triggers the update in case there is one available. In a case there is no available update, sensor node continues the normal operation, but periodically probes the Base Node for updates. When there is an update available, the download of it starts, which is a scenario shown in Figure 5.2. Depending on if the update was successful and valid, the sensor node will continue the normal operation either with the old, or with the new firmware image.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

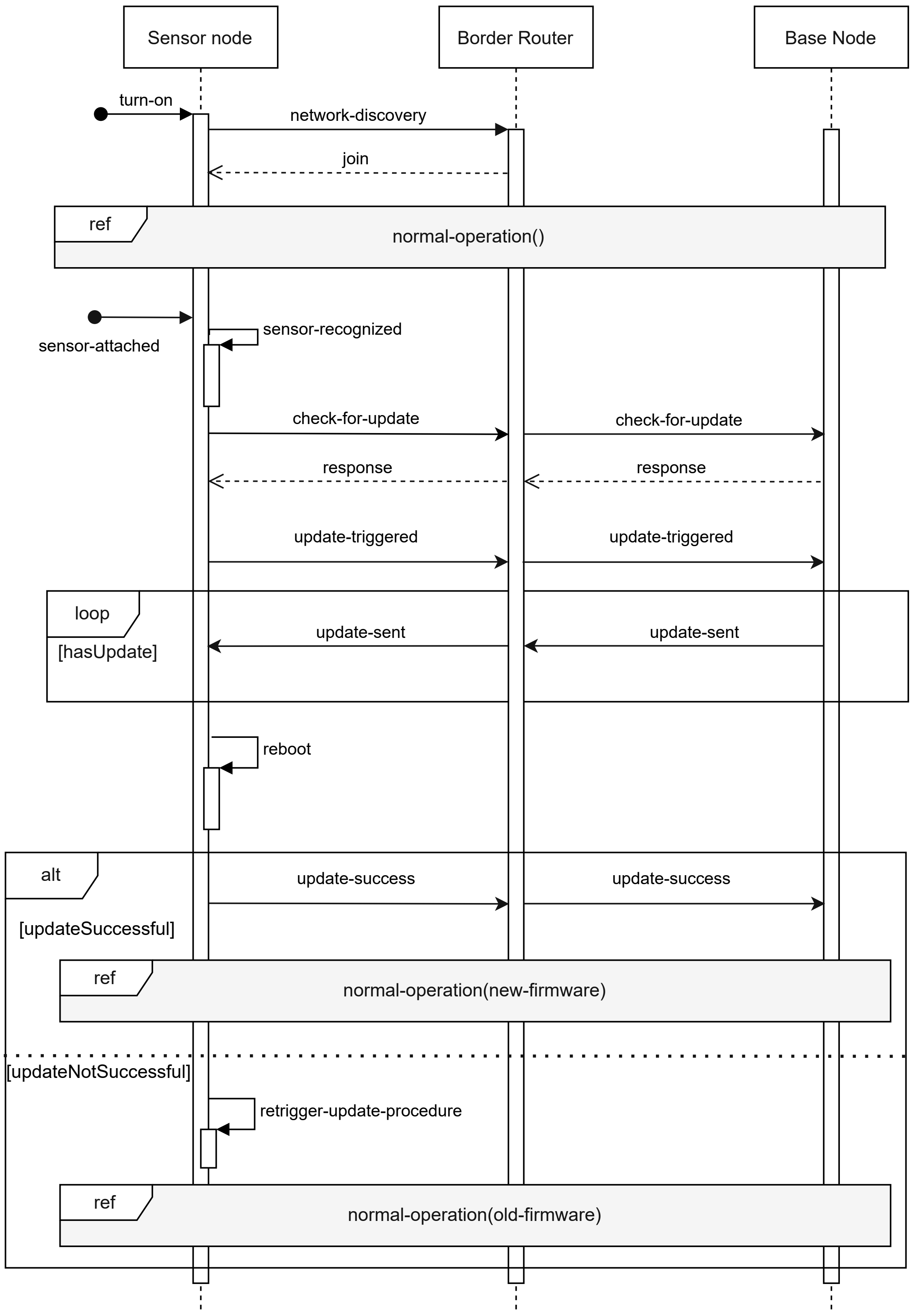


Figure 5.2: Sequence diagram for automatic update

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 6 Thread

Thread is an IPv6 networking protocol targeted for use in low-power, embedded consumer and commercial IoT devices. The Thread Group alliance, formed in July 2014, develops, maintains and drives the adoption of Thread to an industry network standard for IoT applications. It also provides an official certification process, after which a formal certificate (see Figure 6.1) is issued for the product.

BUILT ON THREAD
THREAD CERTIFIED COMPONENT
BUILT ON THREAD BORDER ROUTER
BUILT ON THREAD REQUIRES BORDER ROUTER

Figure 6.1: Thread certifications[1]

6.1 Thread layers

Thread networking stack will be presented now, starting from the lowest layers and moving upwards. Figure 6.2 presents the overall networking stack.

6.1.1 Physical layer and Data link layer

As already mentioned, the Thread specification uses the IEEE 802.15.4-2006 specification for the lowest two layers. This technology was briefly described in Section 2.4.4 and here only some further relevant details will be presented.

[1]https://www.threadgroup.org/What-is-Thread/Certification

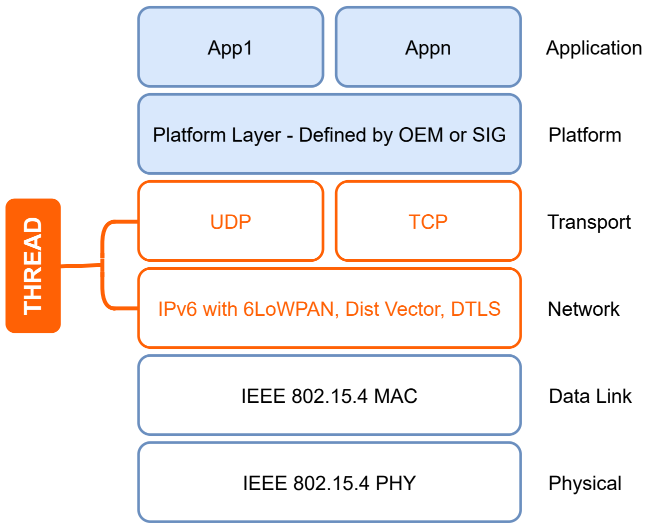


Figure 6.2: Thread Stack Layer Levels, based on [65]

IEEE 802.15.4 PHY provides the interface between the physical radio channel and the MAC. PHY interfaces provide access to transceiver hardware and its firmware. PHY Management Entity (PLME) provides management functions for the layer, including important parameters, objects and other manageable information of the PHY. This information is stored in a PAN Information Base (PIB). Important PIB attributes for the frequency band used by Thread (2400-2483.5 MHz) are [22]:

- Chip Rate 1600 kchip/s
- Modulation O-QPSK. DSSS (Direct Sequence Spread Spectrum)
- Bit Rate 250 kbit/s
- Symbol Rate 62.5 ksymbol/s
- 16 channels

Other important functions supported by PHY are Energy Detection, Link Quality Indicator and Clear Channel Assesment.

Every packet, also called PHY protocol data unit (PPDU), contains a preamble, a start of packet delimiter, a packet length and a payload field. The payload length of a PHY packet can vary from 2 to 127 byes. PHY header is composed by 4 Bytes Preamble + 1 byte Start Packet Delimiter + 1 byte for Length field. On Figure 6.3, a 802.15.4 packet structure can be seen [22].

Enforcing smaller sized packets is a characteristic that is beneficial for lowering the power used for transmission and to limit the BER (Bit Error Rate). MAC layer payload can

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek, Your knowledge hub
WIEN

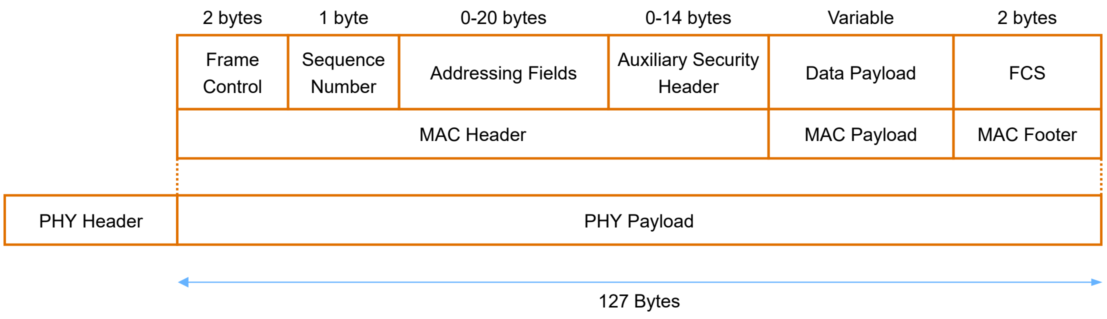


Figure 6.3: 802.15.4 PHY and MAC payloads [62]

vary depending on the security options and addressing type as illustrated in Figure 6.3. As already mentioned in section 2.4.4, IEEE 802.15.4 MAC layer supports two modes: **Non Beacon Mode** and **Beacon Mode**.

6.1.2 Network layer

IPv6

Internet Protocol version 6 (IPv6) makes up the Network layer of Thread and is precisely the reason what makes Thread an attractive choice for HBA. The Internet Engineering Task Force (IETF) developed IPv6 in order to combat the IPv4 address exhaustion problem. Due to IPv4 being made of 32 bits, it provides an addressing capability of 2^{32} or around 4.3 billion addresses [63]. With the rapid growth of Internet, it became quickly evident that this amount of addresses is not enough. Especially with IoT, number of connected devices is rising on scale larger than ever before. IPv6 addresses consist of 128 bits, which theoretically allows 2^{128}, or approximately 3.4×10^{38} addresses and solves the address shortage problem. In addition to a larger address space, IPv6 has other technical improvements on IPv4 as better security due to native integration of IPsec, easier configuration of networks due to Stateless address autoconfiguration (SLAAC) and expanded and simplified use of multicast addressing among others [63].

An IPv6 packet consists of two parts, a header and a payload. The header itself consists of a fixed part required by every IPv6 packet and optional extensions for special features. Fixed part of the header occupies the first 40 bytes of the packet and consists of IP version number, traffic class, flow label, payload length field, Next Header field for optional extensions, hop count, source address and destination address respectively. Extension headers carry additional information important for a packet like routing, fragmentation or security. Payload itself must be less than 64 kB in a normal case [63]. When using Jumbo Payload extension option, the payload must be less than 4 GB [64].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

802.15.4 PHY/MAC hdr	Mesh hdr	Fragmentation hdr	Compression hdr	6LoWPAN Payload

Figure 6.4: 6LoWPAN Packet containing a Mesh Header, a Fragmenation Header and a Compression Header, based on [62]

6LoWPAN

This technology was already introduced in Section 2.4.5. As a reminder, main goal of 6LoWPAN is to act as an adaptation layer between IPv6 network layer and 802.15.4 link layer. It solves the issue of transmitting an IPv6 Maximum Transmission Unit (MTU) (1280 bytes) by fragmenting the IPv6 packet at the sender (into 127 bytes long packets) and reassembling it at the receiver.

Here a more detailed explanation of all the most important functionalities provided by the 6LoWPAN adaptation layer will be presented [62]:

- **IPv6 packet encapsulation:** In order to accomplish other functionalities described below, 6LoWPAN encapsulates IPv6 packets using encapsulation headers. There are three types of 6LoWPAN headers: Mesh Header, Fragmentation Header and Header Compression Header. Each of these headers is used for its accompanied functionality. When more than one header is used, they must appear in order they were listed above and like shown in Figure 6.4. Before all of the 6LoWPAN headers comes the 802.15.4 PHY/MAC header used by 802.15.4 layers below.

- **IPv6 header compression:** This mechanism reduces the IPv6 header size sent over-the-air and consequently reduces transmission overhead. By reducing the transmission, less energy is consumed by devices. An IPv6 header is 40 bytes long, together with a User Datagram Protocol (UDP) header of 8 bytes, it gives the length of 48 bytes. In the best case compression, combined length of headers can be reduced from 48 bytes to just 6 bytes.

  Thread uses two types of header compression: IPHC (Improved Header Compression) and NHC (Next Header Compression). IPHC is an improvement on the HC1 header compression and that is where it gets its name from. It provides a compression technique for global IPv6 addresses. NHC is used for UDP header compression and it comes after the IPHC header. Number of compressed bytes varies and is dependent on following factors: what IPv6 addresses are used, type of 802.15.4 addressing modes and whether or not network contexts are available.

  In order to use these Thread-specific network contexts, CID (Context Identifier Extension) flag of IPHC has to be set to one. In a Thread network, 6LoWPAN layer reduces the size of IPv6 headers by making use of network wide known information, such as the mesh-local and global prefixes. These prefixes are known and managed by the Network Leader and each prefix has its corresponding network context. (e.g., mesh-local prefix -> Context Id 0, Global prefix 2001::/64 -> Context Id 1, Global

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

prefix 2003::/64 -> Context Id 2, etc.) By replacing source and destination prefixes with context Ids, significant header reductions are achieved.

- **IPv6 packet fragmentation and reassembly:** In a wort-case, IEEE 802.15.4 MAC layer provides a payload of length of 88 bytes. This payload size additionally shrinks when IPv6 header (40 bytes) is calculated in. Furthermore if for example UDP is used, that takes additional 8 bytes for the UDP header. In this case that leaves just 40 bytes for the payload. When header compression is used and best case scenario is reached, it still gives only 82 bytes for payload and that means that fragmentation of IPv6 is a necessity.

  There exist two types of fragments in use by the 6LoWPAN abstraction layer: FRAG1 and FRAGN. FRAG1 is the initial fragment. It contains the IPv6 compressed header and can also contain a part of the payload. Every other fragment is of FRAGN type. They contain rest of the payload.

  In order to reassemble the initial IPv6 message, all fragments have to reach the destination. But it does not matter in which order they arrive at the destination, as long as all of the fragments are there, original message can be reassembled. There is a risk of running out of memory for operation on a device due to fragment loss. To avoid this risk, device starts a timer when the first 6LoWPAN packet arrives. If the timer expires, memory used for already received fragments is freed.

- **Link layer packet forwarding:** Last important feature of 6LoWPAN is the link layer packet forwarding ability. Thread uses IP layer routing and link layer packet forwarding of 6LoWPAN. By doing so it avoids sending the packet up to the network layer in order to forward it. More precisely, Thread uses the mesh header mechanism defined in 6LoWPAN specification to forward packets between devices. Mesh header information is first filled with the originator 2 byte address and final destination 2 byte address. Then the next hop 2 byte address is looked up in the Routing Table, packet is sent to this next hop address and the Hops Left count in 6LoWPAN Mesh header is decremented. Process is continued until the final destination is reached or until the hop count reaches 0.

6.1.3 Transport layer

For messaging between devices during mesh establishment and maintenance Thread devices use UDP as defined in [RFC 768] [65]. Transmission Control Protocol (TCP) is also supported, as well as any other IPv6-based transport protocol. UDP is faster, simpler and more efficient in comparison with TCP, but due to lack of acknowledgments, retransmission it is much less reliable. This is done so by design. Since TCP is a connection-oriented protocol and UDP not, UDP has no overhead for establishing, maintaining and terminating a connection, making it faster and more fitting to mesh networking used in Thread.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Implementation of TCP is therefore optional for a Thread device. However, a Thread Certified Component must implement the TCP initiator and listener roles and must implement a well-defined API for the use of these roles [66].

6.1.4 Application layer

Thread as a standard does not define the application layer (see Figure 6.2 again). It instead focuses on the network management side, leaving other technologies to provide the application level functionality. This allows customers to build different application layer functions to fit their own requirements, but base them on the Thread network specifications. Because of that different apps can concurrently run on a single Thread network. For example, application layer for ZigBee devices can be used on top of a Thread network, allowing for interoperability with other ZigBee devices.

Thread still provides some basic application services like multicast messaging, which refers to sending the same message to multiple nodes on a Thread network, be it neighbor nodes, routers or an entire network. Application layers using IP services like CoAP and UDP and other similar IP servers are also supported by Thread [67].

6.2 Thread network architecture

In a Thread network, devices are of a certain type and they can fulfill different roles based on their type. In Figure 6.5, a typical Thread topology is presented, as well as different devices that are part of one Thread network. End users communicate with the network using their smartphone or some type of a cloud solution. This communication can be done over WiFi as seen on the Figure, or over Ethernet as well. This is because Border Router, a device in a network responsible for bridging the Thread network to Internet, can be connected to the rest of the Internet over either of the two technologies. All the different device types and roles are going to be presented now and their tasks described.

6.2.1 Node Roles and Types

Forwarding Roles

With regard to forwarding there are two types of devices (nodes) in a Thread network, Router (Parent) and End Device (Child). The task of a **Router** is to forward packets for other network devices and provide secure commissioning for devices that want to join the network. An **End Device** on the other side is connected to a single Router, does not forward packages and can additionally turn off its transceiver to reduce power and hence extend the battery life. A Router does not have this ability to turn off the transceiver, since otherwise they wouldn't be able to properly route at all times.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

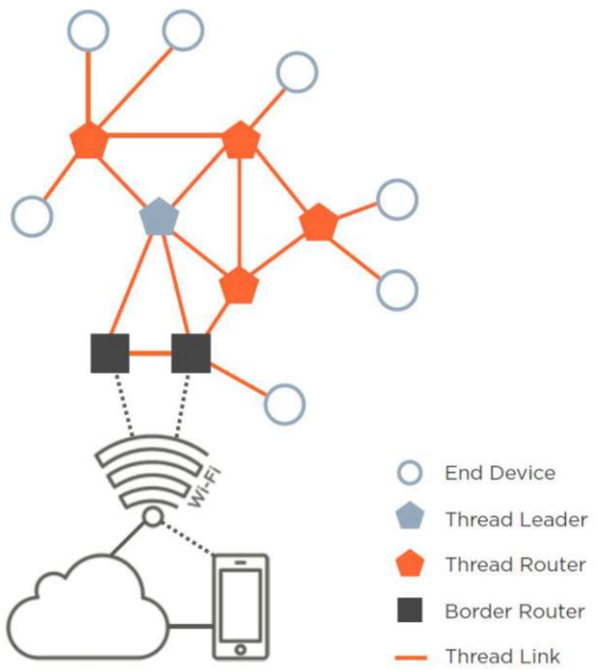


Figure 6.5: Thread network [68]

Device types

Devices inside the Thread network belong to one of the two main types: Full Thread Devices (FTD) and Minimal Thread Devices (MTD).

Full Thread Device (FTD) always has its radio on, subscribes to all-routers multicast address and maintains IPv6 address mappings. Into FTDs belong following node types: Router, Router Eligible End Device (REED) and Full End Device (FED).

Minimal Thread Devices (MTD) on the other hand do not subscribe to all-routers multicast address and forwards all messages to its Parent. That is why an MTD can only be an End Device (Child). There are two types of MTDs: Minimal End Device (MED) and Sleepy End Device (SED).

Apart from the 5 mentioned types of common devices there are two more special node types existing in every Thread network, Border Router and Leader. Let's explain what the role of every type is [65]:

- **Sleepy End Device (SED):** Sleepy End Devices can only communicate through their parent Router and cannot forward messages to other devices. SED keeps its radio turned off during idle periods and wakes periodically to communicate with its parent.

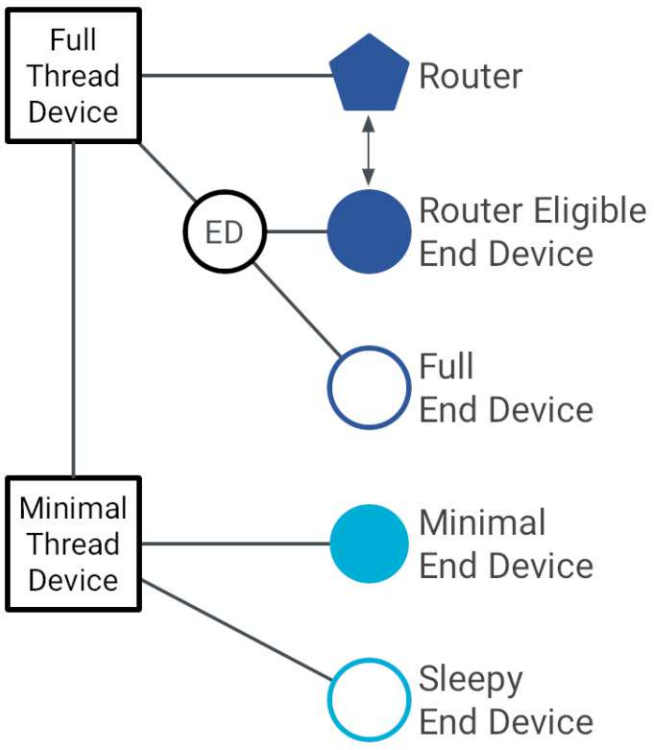


Figure 6.6: Types of Thread devices [68]

- **Minimal End Device (MED):** Minimal End Devices do not have route the packets either and they also communicate only with their parent. They have their radio turned on at all times, even when idle.

- **Full End Device (FED):** FEDs belong to Full Thread Devices, meaning that they have their transceiver on at all times. They are similar to REEDs, but they do not have the ability to act as a Router.

- **Router Eligible End Device (REED):** REEDs have the capability of becoming Routers, but are acting as End Devices and not forwarding the messages. When a REED is the only node in reach of a new End Device that wants to join the network, it can upgrade itself and operate as a Router. In the same vein if a Router has no child devices, it can downgrade itself to an End Device, becoming a REED.

- **Router:** Routers provide routing services to Thread devices in the network. Because of that they need to keep their transceivers enabled at all times. They are also responsible for commissioning devices into the network and providing security. Every Router maintains neighbor, child and routing tables, which are used for correct forwarding of the packets. In a case of Router failure, other Routers update their routing tables so messages can still be transmitted using existing nodes. This

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

is due to resiliency of mesh topology and multiple paths one message can take. Routers can downgrade their functionality and become REEDs, or take on a role of a Leader.

- **Leader:** A Thread Leader manages a registry of assigned Router IDs and processes requests from REEDs to become Routers. Every Thread network has to have a Leader and there can be only one Leader in the network. A Leader decides which device can be a Router, assigns and manages Router addresses using CoAP and forwards packets like a Router itself. Information contained on the Leader is present in other Routers as well, so in the case of Leader failure or loss of connectivity, another Router can take over the Leader role without user intervention.

- **Border Router:** Border Routers provide connectivity between a Thread network (based on IEEE 802.15.4) and other IP-bearing interfaces, like Ethernet (IEEE 802.3), Wi-Fi (IEEE 802.11) and cellular. It is a requirement to have a Border Router if there is a need to connect a Thread network to other networks. There can however be more than one Border Router in one network, unlike with Thread Leader. Border Router minimally supports the following functions [68]:

  – Bidirectional IPv6 connectivity between Thread and Wi-Fi/Ethernet networks
  – Service discovery - allowing services on the Thread network to be discovered by devices that are not on the Thread network, via mDNS Technology
  – Thread-over-infrastructure - merging of Thread partitions over IP-based links
  – External Thread Commissioning - a device outside of the Thread network (e.g., a mobile phone) can commission new device onto the network

- **Bluetooth End Device (BED)** BEDs communicate only through their parent Router, like other End Devices. Peculiarity for these devices is that their parent Router is a BLE Bridge Router and unlike other devices, these communicate over BLE and not IEEE 802.15.4.

6.2.2 IPv6 Addressing

Now that all the different node types and roles are known, the way how they actually address each other for the communication will be explained.

All Thread devices use a number of different IPv6 unicast addresses for communication. Besides link-local addresses, which are reachable with a single radio transmission, Thread defines three different types of unicast addresses:

- Routing Locator (RLOC)
- Anycast Locator (ALOC)
- Endpoint Identifier (EID)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

In a Thread network, there are also three scopes for unicast addressing [68]:

- Link-Local - all devices reachable by a single radio transmission (neighbors)
- Mesh-Local - all devices reachable within the same Thread network
- Global - all interfaces reachable from outside a Thread network

RLOC is an important unicast address that identifies a Thread interface based on its location in the network. It is generated when a device is attached to a network and since it is dependent on the location in the network, it also changes as the topology of the network changes. Its scope is Mesh-Local and it is seldom used by applications, but is rather used for delivering IPv6 datagrams by a Thread network.

ALOC is used when the specific RLOC of a destination is not known at the originator and it identifies the location of one or more Thread devices within a Thread network.

Last important type of unicast address are Endpoint Identifiers (EIDs), which unlike RLOCs identify a unique Thread interface independent of the network topology. There are many different unicast types that are EIDs, which just means they are not based on the location in the network topology.

Other common unicast address types are the following [68]:

- Link-Local Address (LLA) - an EID that identifies a Thread device reachable by a single radio transmission. This addres is not routable, it is used to discover neighbors, configure links and exchange routing information. As the name itself suggests, scope is link-local.
- Mesh-Local EID (ML-EID) - an EID used to reach a Thread device in the same Thread network/partition. This is the address which should be used by applications.
- Anycast Locator (ALOC) - identifies a Thread device via RLOC lookup, when RLOC of the destination is unknown. Scope of ALOC is also mesh-local and this address is generally not used by applications.
- Global Unicast Address (GUA) - an EID that identifies a Thread device on a global scope. It is the public IPv6 of a device.

Multicast addressing is used to communicate information to multiple devices at once. When it comes to multicasting, there are four different specific addresses reserved for multicast use. Multicast messages can be either link-local or mesh-local [68]. Furthermore they can be delivered to all FTDs and MEDs or only to all FTDs and exactly these two differences combined give the four reserved addresses, like it is shown in Table 6.1.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Table 6.1: Multicast address types, based on [68]

IPv6 Address	Scope	Devices targeted
ff02::1	Link-Local	All FTDs and MEDs
ff02::2	Link-Local	All FTDs
ff02::3	Mesh-Local	All FTDs and MEDs
ff02::4	Mesh-Local	All FTDs

6.2.3 Network Discovery and Formation

There are three unique identifiers of a Thread network. Those are [68]:

- 2-byte Personal Area Network ID (PAN ID) - example: 0xBEEF
- 8-byte Extended Personal Area Network ID (XPAN ID) - example: 0xBEEF1111CAFE2222
- a Network Name - example: yourThreadNetwork

Network Discovery

The process of network discovery is pretty simple.

1. A Thread device broadcasts an 802.15.4 Beacon Request on a specific channel.
2. If there are any Routers or REEDs in range on this specific channel, they broadcast a Beacon that contains all three above mentioned unique identifiers.
3. A Thread device further continues the same process for other channels.
4. In the end, a device can connect to one of the discovered networks or create a new Thread network if no available networks are found.

Mesh Link Establishment

Regardless of the device's decision to create a new Thread network or to join an existing one, Mesh Link Establishment (MLE) protocol will be used [68]. This protocol is used to for establishing and configuring secure radio links in IEEE 802.15.4 radio mesh networks [69].

In link configuration MLE has following uses:

- Discover links to neighboring devices
- Determine the quality of links
- Establish links to neighbors

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- Negotiate link parameters

When wanting to establish links, following information is disseminated by MLE:

- Leader data (Leader RLOC, Partition ID, Partition weight)
- Network data (on-mesh prefixes, address autoconfiguration)
- Route propagation (based on the Routing Information Protocol (RIP), a distance-vector routing protocol)

MLE is not used for the dissemination of network credentials and identifiers. This is done by Thread Commissioning first and then MLE is used to establish and configure the links [68].

It is also important to mention that Thread uses Trickle to set the advertisement message interval [70]. Trickle is familiar to the reader as one of the state-of-the-art dissemination protocols described in Section 2.6.2.

Creating a new Thread network

When a device decides to create a new Thread network, it follows the following steps:

1. It first selects the least busy Channel and a PAN ID not used by other networks.
2. Then it takes a role of a Router and elects itself as the Leader.
3. Afterwards, it sends MLE Advertisement messages to other devices to inform them about its link state and responds to Beacon Requests from other Thread devices.

Joining an existing Thread network

If a device has discovered a passing network and wants to join it, it first configures the channel and network identifiers (PAN ID, XPAN ID and network name) to match the target network via Thread Commissioning. After that it follows the MLE attach process, which consists of the followings steps:

1. Parent Request - multicast request from the joining device to discover Routers and REEDs in range
2. Parent Response - unicast response from the Router or REED with information about it
3. Child ID Request - unicast request from the joining device to the Router or REED to establish a Parent-Child relationship

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

4. Child ID Response - unicast respond from the Parent to the Child as confirmation that a Parent-Child relationship is established

Important to notice is that every new device that joins the network, starts of as an End Device at the beginning, regardless of its ability to act as a Router.

6.3 Matter

A recent development in the world of HBA is Matter[2]. Started in December 2019 as Project CHIP (Connected Home over IP) by companies like Amazon, Apple, Google, Comcast, ZigBee Alliance, Nordic Semiconductors and other, Matter is an application layer royalty-free standard for connected things at home. The goal is to create a unified application layer standard which makes it easy for manufacturers to create secure and reliable solutions that are interoperable with prevalent smart home ecosystems like Apple's Siri, Google's Assistant and Amazon's Alexa.

It can use Thread, Wi-Fi and Ethernet for transport and BLE for commissioning. Since from three protocols named for transport Thread is by far the best for HBA, it comes as no surprise that a combination of Thread+Matter arises often in articles and industry journals recently. The synergy is made possible due to the fact that Thread leaves the application standard as undefined. In addition to that companies that develop Thread are also the companies that are developing Matter as well, which gives a lot of evidence in which direction these companies see the branch developing. All Matter devices based on Thread still require BLE connectivity as well in order to be able to add new devices to a network. As can be seen be seen on Figure 6.7, there is a plan of adding further communication protocols to the list of supported ones, like cellular for example.

Security and privacy have special focus in Matter from the early development. There are five security principles on which the protocol is built and based upon [71]:

- **Comprehensive** - to provide comprehensive security means to implement it in a layered approach with authentication and attestation for commissioning, protecting every message and securing OTA updates. Functional security of Matter is self-contained and independent of the security of the communication technology on which Matter is running, be it Wi-Fi, Ethernet or Thread. There is also no need to add functional security features on top of the ones that are provided by Matter itself.

- **Strong** - a variety of state-of-the-art security algorithms are used to ensure the best possible security. AES in CCM mode is used for confidentiality and integrity with 128 bit keys. AES in CTR mode is used for protecting identifiers to preserve privacy. SHA-256 is used for integrity and ECC with the "secp256r1" curve for

[2]https://github.com/project-chip/connectedhomeip#readme

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

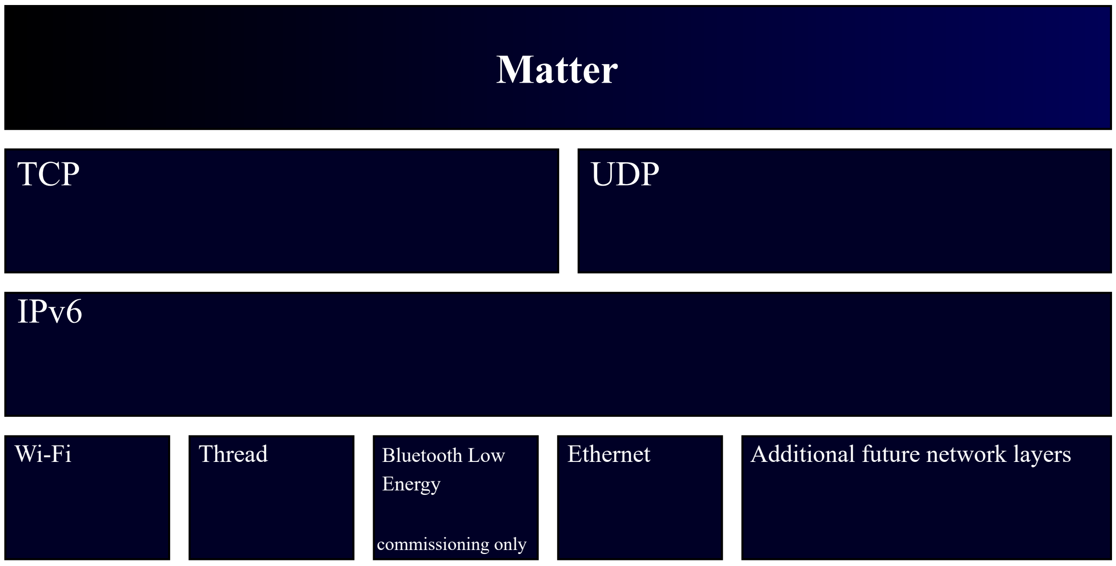


Figure 6.7: Matter's place in the OSI communication model, based on [71]

digital signatures and key exchanges, standard key derivation schemes and truly random number generators.

- **Easy** - the idea is to make smart and secure devices easier to make by manufacturers and to use by consumers. To achieve this, Matter core specification comes with test vectors and examples for each functional security aspect. There are reference implementations which offer examples of alternative integration of hardware security modules (HSM). These things help the developers to develop secure Matter products and leave the customer without worries when purchasing a product.

- **Resilient** - by providing more than one way of performing certain operations, Matter aims to have resilient security. An example is secure session establishment, where Matter attempts a shorter secure resumption protocol due to efficiency and in case it cannot be performed, or fails, full protocol is attempted. By having a self-sufficient security stack, it does not depend on the underlying security of the communications medium, increasing its independence and resiliency. One more example are several mechanisms for prevention of most common DoS attacks that have been built into Matter.

- **Agile** - having crypto-flexibility in mind, Matter can adapt and address new threats and developments that arise. Its modular design, where all cryptographic primitives are abstracted, allows a rapid adoption of new cryptographic primitives instead of changing the whole specification.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

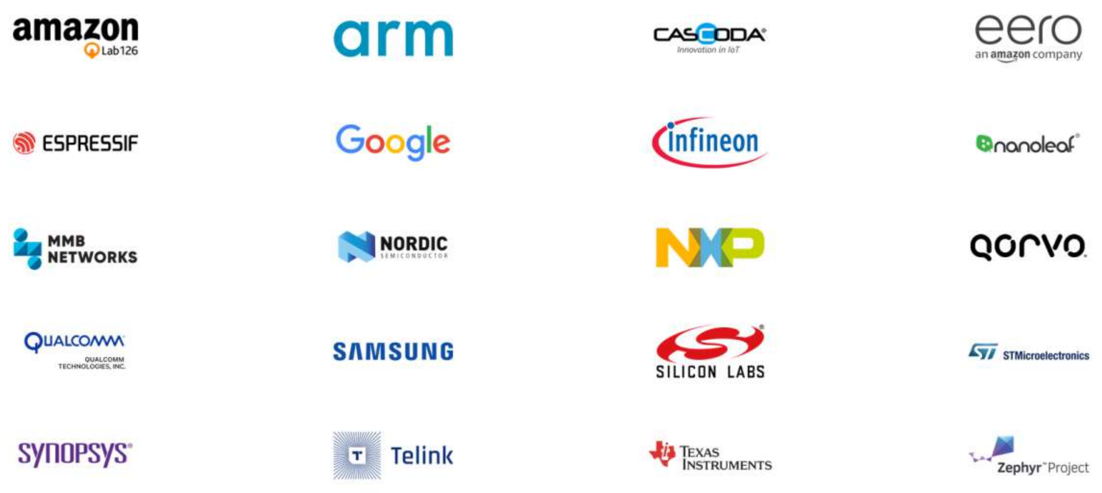


Figure 6.8: OpenThread supporters[3]

6.4 OpenThread

OpenThread is an open-source implementation of the Thread protocol developed by Google Nest. With OpenThread included in its product palette, Nest wants to speed up the adaptation of Thread network. Outside of Nest there is a large and still growing group of supporters of OpenThread that includes established silicon providers like arm, Infineon, NXP, Samsung, Texas Instruments, Qualcomm and others. Full list of supporters is presented in Figure 6.8.

Being open-source, supported by such a large group of established industry leaders and rising in popularity and use, OpenThread was chosen for the implementation in this thesis as well.

6.4.1 OpenThread CLI

OpenThread CLI (Command Line Interface) enables configuration and management APIs through command line interface. It is present on the Border Router and can be accessed easily by executing ot-ctl as a superuser. On the Github repository[4] there is a reference guide for all commands. More information about OpenThread will be given in Section 7.2, where the Border Router implementation in OpenThread is described.

[3]https://openthread.io/
[4]https://github.com/openthread/openthread/tree/2bce9557e62c451307d1a4d944c3eaf69f71573c/src/cli

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 7

Implementation

This chapter describes the implemented system, presents individual parts of it, their characteristics and connection to other parts. Figure 7.1 presents the system from the hardware point of view and shows the connections between different layers. There are a lot of technologies that are combined in the final implementation: Thread, Zephyr, mcuboot, IPv6, CoAP and UpdateHub. The chapter explains how these technologies are used at different layers of the network and their roles in the overall picture.

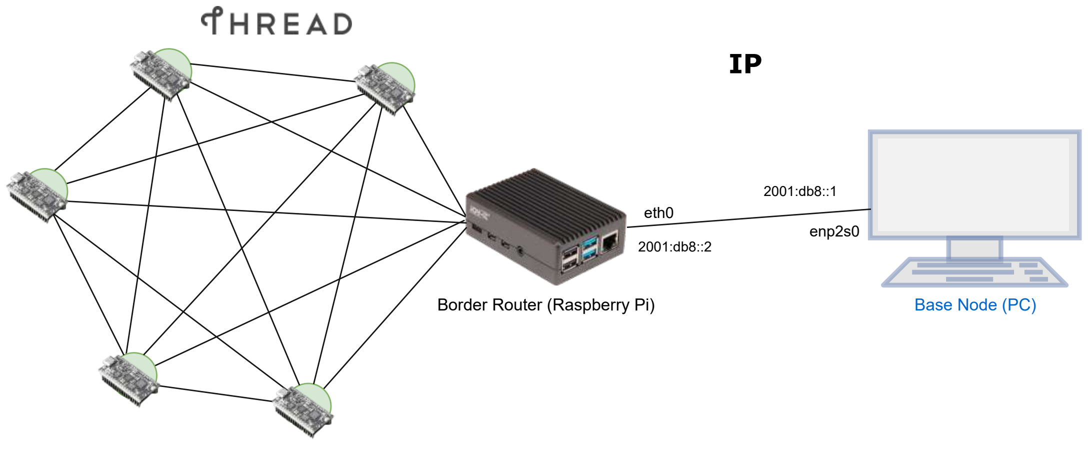


Figure 7.1: Network set up

7.1 Sensor nodes

Starting from the lowest level, sensor nodes are the devices that are responsible for the collection of data. They are the devices that should be updated OTA and present an

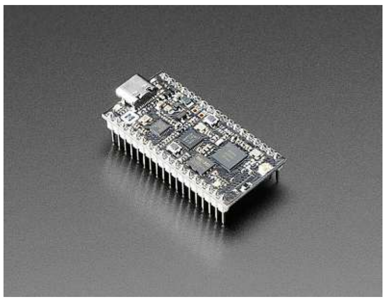


Figure 7.2: nrf52840 mdk

important part of the implementation. This section presents the hardware and software of sensor nodes used in this thesis.

7.1.1 nrf-52840 mdk

As a sensor node, the nrf-52840 mdk (mini-development kits) boards were used, which use the nrf52840 System on a chip (SoC) themselves. This is the most advanced member of the nRF series built around the 32-bit ARM Cortex-M4 CPU, with floating point unit, running at 64 MHz. There is an additional ARM Cryptocell 310 cryptographic accelerator on the chip, which provides additional security for the CPU. nrf52840 comes with 1MB Flash and 256kB of RAM memory. Its main intention is the use in IoT systems like HBA, industrial mesh networks and smart city infrastructure [72]. It enables the protocol concurrency of all the most important protocols in IoT today like Bluetooth 5, Bluetooth Mesh, Thread, IEEE 802.15.4, ANT and 2.4GHz proprietary stacks.

Additional hardware features that the development kit adds to the nrf52840 SoC are[73]:

- Use of DAPLink[1] for easy programming and debugging
- Microchip 2-Port USB 2.0 Hi-Speed Hub Controller
- 24 GPIO pins
- IF Boot/Reset button
- User programmable button and RGB LED
- On-board 2.4G chip antenna
- U.FL connector selectable for external antenna
- Reversible USB 3.1 Type-C Connector
- Dimensions: 50mm x 23mm x 13mm, with headers soldered in

[1]https://daplink.io/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

7.1.2 Zephyr

Zephyr is an open-source real-time operating system supported by companies like Google, Intel, Facebook and NXP among others. It is licensed using the Apache 2.0 license[2].

Based on a small-footprint kernel, it is designed for use on resource-constrained and embedded devices. Following architectures are supported by the Zephyr kernel:

- ARCv2 (EM and HS) and ARCv3 (HS6X)
- ARMv6-M, ARMv7-M, and ARMv8-M (Cortex-M)
- ARMv7-A and ARMv8-A (Cortex-A, 32- and 64-bit)
- ARMv7-R, ARMv8-R (Cortex-R, 32- and 64-bit)
- Intel x86 (32- and 64-bit)
- MIPS (MIPS32 Release 1 specification)
- NIOS II Gen 2
- RISC-V (32- and 64-bit)
- SPARC V8
- Tensilica Xtensa

A full list of supported boards is listed on the Zephyr website[3] and there exists a board porting guide for adding support of a new board[4].

For enabling such a wide compatibility, Zephyr C code uses a devicetree for hardware configuration. More precisely, it is used for describing available hardware on a certain target board, together with its initial configuration. A devicetree is a hierarchical data structure. There are two types of devicetree input files: devicetree sources and devicetree bindings. Sources contain the devicetree and bindings describe its contents, including data types. Out of sources and bindings, a generated C header is created during the build system. The generated C header is then abstracted by the *devicetree.h* API, which can be included and used by all Zephyr and application source code files. The interested reader can find further information about the structure and syntax of devicetree on the following link[5].

The major goal of devicetree use in Zephyr is to have a single source for all hardware information and to provide compatibility with other operating systems[6]. By obtaining all

[2]https://github.com/zephyrproject-rtos/zephyr/blob/main/LICENSE
[3]https://docs.zephyrproject.org/latest/boards/index.html#boards
[4]https://docs.zephyrproject.org/latest/hardware/porting/board_porting.html
[5]https://docs.zephyrproject.org/latest/build/dts/intro.html
[6]https://docs.zephyrproject.org/latest/build/dts/design.html

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

the hardware information exclusively from devicetree, a lot of the confusion is avoided, especially in the case of Zephyr, which should work on so many different platforms. Linux, among others, also uses devicetree and the tooling used by Zephyr for devicetree is interoperable by the tooling used by Linux.

Apart from devicetree, Zephyr also uses Kconfig configuration files to configure the source code of the application. The whole build process of Zephyr is based on CMake[7]. During the build process, both the application and Zephyr itself are built and compiled into a single binary image. Zephyr's own source code, kernel configuration options and build definitions are to be found in the Zephyr's base directory and are used to build the kernel source tree. Files in the application directory are used for the source code, configuration of the application itself and linking of the application to Zephyr.

The directory of a simplest application has the following content[8]:

```
 <home>/app
- CMakeLists.txt
- prj.conf
   /src
    - main.c
```

- **CMakeLists.txt**: This file is used for finding other application files and linking of the application directory with the Zephyr's CMake build system.
- **prj.conf**: This is an example of a **Kernel configuration file**, mentioned earlier as Kconfig. It doesn't have to be named prj.conf. Kconfig configuration files specify application specific values of kernel configuration options and together with board-specific settings (overlay files) produce a kernel configuration.
- **main.c**: In subfolder *src* all the source code files are to be found. These can be written either in C, or in assembly language.

After all of the files have been defined, application build artifacts that are created after the build process are saved in a **build directory**. The default build tool in Zephyr is *west*[9]. Another build tool that can be used is *ninja*, or *cmake* can be run directly if preferred as well. After a successful build, the application can be flashed on the target device using *west* or *ninja*, or it can be run on the emulator.

OTA possibilities in Zephyr

The task of an OTA update on Zephyr can be split into two parts, namely the download part and the installation part. The download part is responsible for getting the software

[7]https://cmake.org/
[8]https://docs.zephyrproject.org/latest/develop/application/index.html
[9]https://docs.zephyrproject.org/latest/develop/west/index.html#west

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

update to the sensor node, and the installation part is responsible for enabling the software update.

For the installation part, it is necessary to use the **MCUBoot** bootloader [74]. This bootloader splits the memory into parts, making space for two different software images to be present in the memory. A more detailed explanation of how this bootloader works will be presented in the following subsection.

The job of the download part of Zephyr OTA is to get the software image into the second slot of MCUBoot. For this, there already exist some approaches like **mcumgr** [75], **UpdateHub** [76] and **Hawkbit** [77]. These are applications which run on top of Zephyr.

mcumgr allows remote management of nodes running Zephyr over the following transports:

- BLE (Bluetooth Low Energy)
- Serial (UART)
- UDP over IP

Hawkbit can be used with following technologies:

- BLE (Bluetooth Low Energy)
- 802.15.4
- OpenThread

And finally **UpdateHub** should work with following technologies:

- WiFi
- modem
- BLE IPSOP
- 802.15.4
- OpenThread (experimental)

As already said, for any of the above mentioned approaches for getting the software image to work, MCUboot needs to be present. Due to the choice to use Thread as the communication protocol, mcumgr is not a passing solution for this thesis. UpdateHub and Hawkbit are pretty similar in the way they work and the options they give to users. They both offer Docker solutions, which provide a web interface that allows the user to upload packages, plan roll outs and update the devices. None of them work with OpenThread "out of the box", but are in experimental phase. A fork of UpdateHub was made and after a lot of modifications, it could be used as a part of the implementation.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

7.1.3 MCUboot

MCUboot is a bootloader for 32-bit microcontrollers developed by mcu-tools[10]. It is comprised out of two packages: the bootutil library (boot/bootutil) and the boot application. Most of the functionality of the bootloader is performed by the bootutil library. The only missing piece is the final step of jumping to the main image. This is implemented by the boot application. The reason for it is because a library can be unit tested and an application can't, so everything is delegated to the library whenever possible [78].

MCUboot uses flash partitions for its workings. The flash memory is partitioned into multiple slots, as can be seen in Figure 7.3. Start of the flash memory is reserved for the bootloader itself. There are two slots reserved for the firmware images, a primary and a secondary slot. Usually, the bootloader simply runs the image in the primary slot (unless direct-xip or ram-load upgrade modes are selected). In the case there is an image in the secondary slot that needs to be executed, it needs to be copied into the primary slot either by swapping the images or by overwriting the content of the primary slot. This is configured at build time.

There are two possible algorithms for swapping of an image: swap using scratch and swap without scratch [78]. When using swap-using-scratch, there is an additional scratch area in flash memory, just like how it is presented in Figure 7.3. The size of this scratch area needs to be large enough to store the largest sector that is going to be swapped, since the swapping of images is being done sector by sector.

Other algorithm does not use scratch, but it has an additional sector in the primary slot to make the swap possible. By following the algorithm described in Listing 7.1, images get completely swapped.

Listing 7.1: Swapping without scratch in mcumgr

```
IN primary slot MOVE all sectors to position + 1
N = 0
DO
        FROM secondary slot COPY sector N TO sector N in primary slot
        FROM primary slot COPY sector N+1 TO sector N in secondary slot
        INCREMENT N
UNTIL images completely swapped
```

In both algorithms, after the correct image is in the primary slot, it then gets executed.

Instead of swapping images, there is also option of using direct-xip mode, where images can be run from both primary and secondary slot. Decision which image should be run is made by using an active image flag, which is set active at one of the two slots. By using this method, there is no need for image swapping. In this case however, an image update client needs to know which slot has the active image and which slot can be used for the

[10] https://github.com/mcu-tools

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

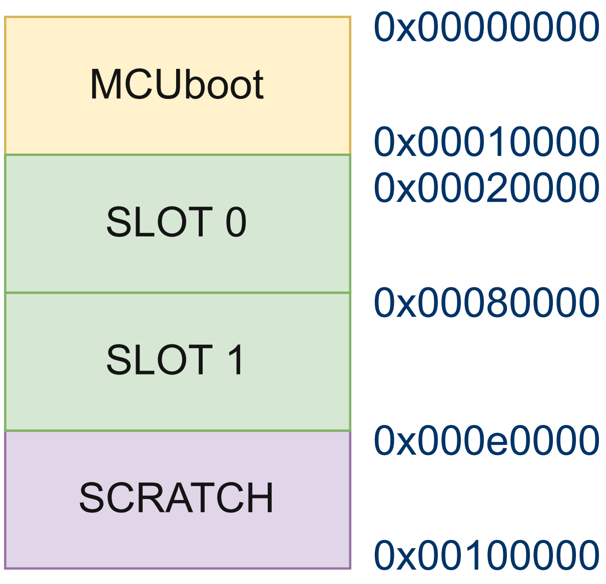


Figure 7.3: Example of flash memory layout with MCUboot, based on [74]

upgrade image and it is also responsible for loading the proper image into the proper slot. This also means that images need to be built to be executed from a certain memory address. Both direct-xip mode and overwriting mode are much simpler to implement than the swapping algorithms.

Before flashing the image, they need to be signed in order to work with MCUboot. Signing the image adds a header and a trailer to the binary or Intel hex format. With MCUboot comes a development key which can be used for testing. Since this key is widely used and known, it should never be used for production or any other setting outside of initial testing. A new production key should be generated by using the Python program imgtool.py that comes with the bootloader. This tool currently supports rsa-2048, rsa-3072, ecdsa-p256 and ed25519 keys.

In order to use MCUboot with the application used in Zephyr, next steps need to be followed[11]:

- Flash partitions which are required by MCUboot need to be defined
- Flash partition needs to be specified as the chosen code partition, like it is shown in Listing 7.2
- Kconfig option CONFIG_BOOTLOADER_MCUBOOT needs to be enabled in the .conf file of the application

[11]https://docs.zephyrproject.org/3.2.0/services/device_mgmt/dfu.html

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- MCUboot itself needs to be built and flashed
- Application needs to be built, signed and flashed at the correct offset, which is right after the bootloader

Listing 7.2: Specifying the *slot0_partition* as the chosen code partition

```
chosen {
        zephyr,code-partition = &slot0_partition;
};
```

7.1.4 Environment sensing application

Main task of HBA is usually concerned with measuring and controlling various environmental values. For the thesis, an environment sensing application is implemented by using the BME680 sensor from Bosch, which measures temperature, humidity, pressure and gas.

To use and enable the BME680 sensor, information about it needs to be added into the overlay file of the board. The information that has to be added can be seen in Listing 7.3.

Listing 7.3: BME680 description in the overlay file of nordic nrf82540 mdk

```
&i2c0 {
        bme680@76 {
                label = "bme680";
                compatible = "bosch, bme680";
                reg = <0x76>;
                status = "disabled";
        };
};
```

Sensing information about the environment is sent to the CoAP server that is on the Base Node. Time period of sampling and sending is 3 seconds, but this can be configured. The idea is to implement the ability, that automatically triggers an update from UpdateHub when a new sensor is attached onto the sensor node.

7.2 Border Router

A Border Router (or an Edge Router) is a Router that sits at the edge of the network and routes between two different kinds of networks. In the network used for this thesis, an OpenThread implementation of the Border Router is used, with Thread network on one side and a connection to Ethernet on the other side.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Figure 7.4: RPi 4 cased

7.2.1 Hardware

Raspberry Pi

A device used as the Border Router is Raspberry Pi 4 Model B. This is a much more powerful device than the devices used for sensor nodes. It comes with a quad-core Cortex-A72 (ARM v8) microprocessor and with either 1, 2 or 4 GB of RAM memory. It runs the Raspberry Pi OS, which is a Debian-based OS.

The most important technical features of the device are:

- Quad core 64-bit ARM-Cortex A72 running at 1.5GHz
- 1, 2, 4 and 8 Gigabyte LPDDR4 RAM options
- dual micro-HDMI display output up to 4Kp60 resolution
- 802.11 b/g/n/ac Wireless LAN
- Bluetooth 5.0 with BLE
- 1x Gigabit Ethernet port
- 28x user GPIO supporting various interface options (UART, I2C, SPI, SDIO, DPI, PCM)
- ARMv8 Instruction Set

There are more features besides the listed specifications[12]. It is a pretty powerful device when compared to other embedded devices. Due to the always-on requirement it needs to fulfill in order for the Thread network to function properly and high computing power required for running, Raspberry cannot run solely on battery. It requires a constant, external power supply like a power outlet.

[12]https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

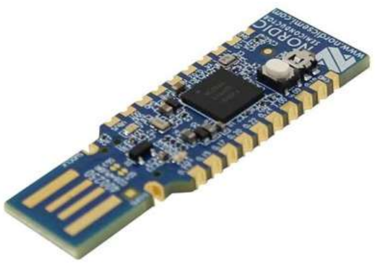


Figure 7.5: nrf52840 Dongle

nRF52840 Dongle

We are using the nRF52840 Dongle, that has the same SoC as the sensor nodes, as a Radio Co-Processor (RCP).

7.2.2 OpenThread Border Router

Running on the device is an OpenThread implementation of Thread Border Router (OTBR). In Section 6.2.1, minimal functions that need to be supported by the Border Router in a Thread network were listed. All these requirements are fulfilled by OTBR and it provides a number of other additional features [79]:

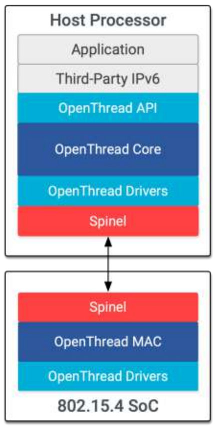
- Web GUI for configuration and management
- Thread Border Agent that supports external commissioning
- DHCPv6 Prefix Delegation to obtain IPv6 prefixes for a Thread network
- NAT64 for connecting to IPv4 networks
- DNS64 to allow Thread devices to initiate communications by name to an IPv4-only server

Two different co-processor designs are supported by OpenThread: Radio Co-Processor (RCP) and Network Co-Processor (NCP).

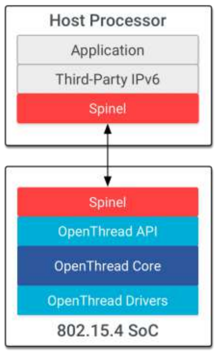
In the standard NCP design, 802.15.4 SoC has all the Thread features and only runs the application layer on a host processor. Communication between the NCP and the host processor is done by wpantund through a serial interface using the Spinel protocol. For more information about Spinel, see the Internet Draft for Spinel Host-Controller protocol [80]. By separating the functionality in this way, a high-power host can sleep while the low-power OpenThread device remains active and keeps its place in the Thread network. This design is useful for gateways and devices that have processing demands like IP cameras and speakers.

In the RCP design, 802.15.4 SoC only has a minimal MAC layer with the Thread radio. The core of OpenThread is on the host processor. Communication between the RCP and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



(a) RCP architecture



(b) NCP architecture

Figure 7.6: Two design options for OTBR [79]

the host processor is in this case done by OpenThread Daemon over an SPI interace, but using the same Spinel protocol. With this design, the host processor doesn't sleep and is more suitable to devices less sensitive to power constraints. That is also the reason why this design is used for the implementation of the Border Router for the thesis.

OTBR provides support for BeagleBone Black from Texas Instruments and for Raspberry Pi 3B or newer versions. It is important to highlight that OTBR on Raspberry Pi 3B with a nordic nrf52840 NCP is a Thread Certified Component [79].

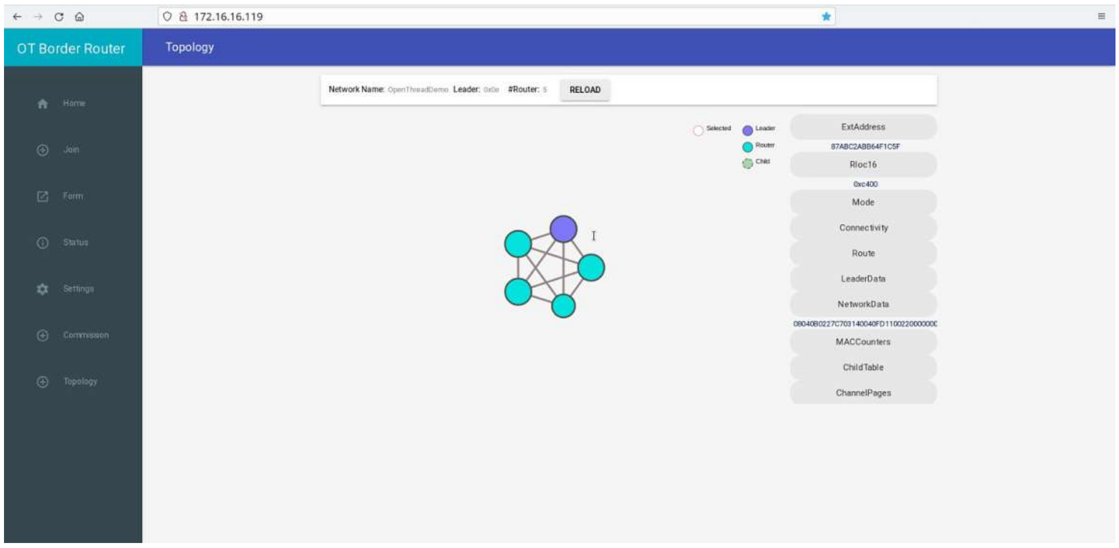
One of the most helpful features that comes with OTBR is definitely the Web GUI. It can be used to configure, form, join and check the status of a whole Thread network. Additionally, it features a graphical representation of the whole network like the one presented on Figure 7.7.

7.2.3 IPv6

As it was already mentioned, Thread is an IPv6 based networking protocol. This means that all members of the network use various IPv6 addresses. For the setup used that enables the OTA to work, the initial problem is to make sure that different parts of the network can communicate with each other.

Just running the UpdateHub server and an application with a correct IPv6 address of the UpdateHub server is not successful without appropriate configurations that need to be made. The Border Router is the central part of the whole system, since it is the one that is connected to an OpenThread network on one interface and to the Base Node (PC) by Ethernet over the other interface. Before having a working connection between a sensor node and the Base Node, there need to be working connections between both sensor node and Border Router, as well as between Base Node and Border Router. A tool that is regularly used in computer networking to check if there is a connection between different

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



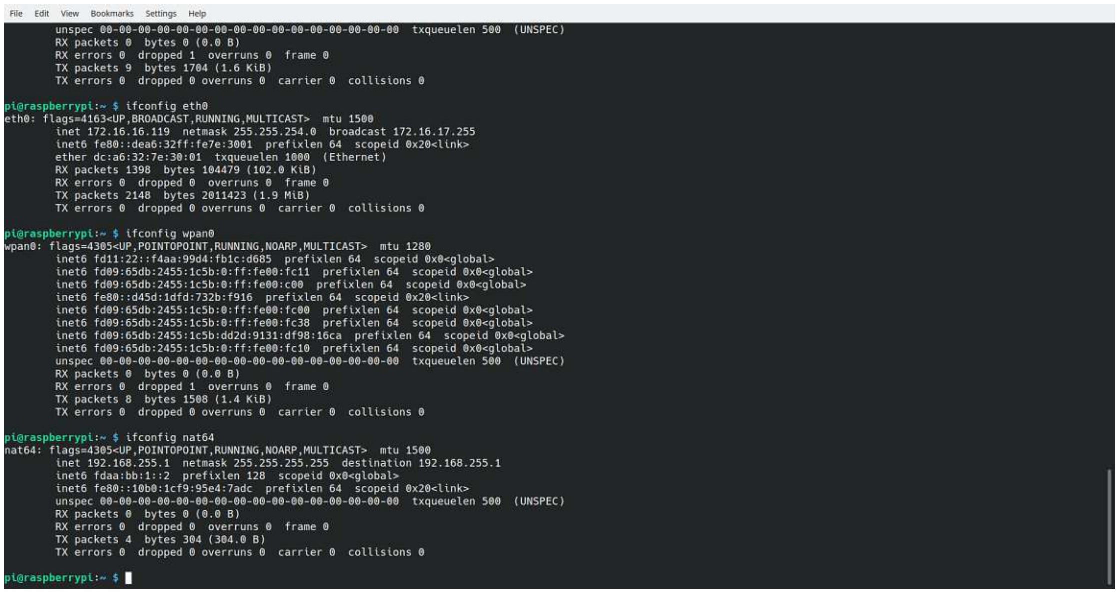
Network Name: OpenThreadDemo Leader: 0x0e #Router: 5

ExtAddress: 87ABC2ABB64F1C5F

Rloc16: 0xc400

NetworkData: 08040B0227C703140040FD11002200000000

Figure 7.7: OTBR Web GUI with network topology



```
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 500  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 1  overruns 0  frame 0
        TX packets 9  bytes 1704 (1.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

pi@raspberrypi:~ $ ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.16.119  netmask 255.255.254.0  broadcast 172.16.17.255
        inet6 fe80::dea6:32ff:fe7e:3001  prefixlen 64  scopeid 0x20<link>
        ether dc:a6:32:7e:30:01  txqueuelen 1000  (Ethernet)
        RX packets 1398  bytes 104479 (102.0 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2148  bytes 2011423 (1.9 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

pi@raspberrypi:~ $ ifconfig wpan0
wpan0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST>  mtu 1280
        inet6 fd11:22::f4aa:99d4:fb1c:d685  prefixlen 64  scopeid 0x0<global>
        inet6 fd09:65db:2455:1c5b:0:ff:fe00:fc11  prefixlen 64  scopeid 0x0<global>
        inet6 fd09:65db:2455:1c5b:0:ff:fe00:c00  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::d45d:1dfd:732b:f916  prefixlen 64  scopeid 0x20<link>
        inet6 fd09:65db:2455:1c5b:0:ff:fe00:fc00  prefixlen 64  scopeid 0x0<global>
        inet6 fd09:65db:2455:1c5b:0:ff:fe00:fc38  prefixlen 64  scopeid 0x0<global>
        inet6 fd09:65db:2455:1c5b:dd2d:9131:df98:16ca  prefixlen 64  scopeid 0x0<global>
        inet6 fd09:65db:2455:1c5b:0:ff:fe00:fc10  prefixlen 64  scopeid 0x0<global>
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 500  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 1  overruns 0  frame 0
        TX packets 8  bytes 1508 (1.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

pi@raspberrypi:~ $ ifconfig nat64
nat64: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST>  mtu 1500
        inet 192.168.255.1  netmask 255.255.255.255  destination 192.168.255.1
        inet6 fdaa:bb:1::2  prefixlen 128  scopeid 0x0<global>
        inet6 fe80::10b0:1cf9:95e4:7adc  prefixlen 64  scopeid 0x20<link>
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 500  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4  bytes 304 (304.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

pi@raspberrypi:~ $ █
```

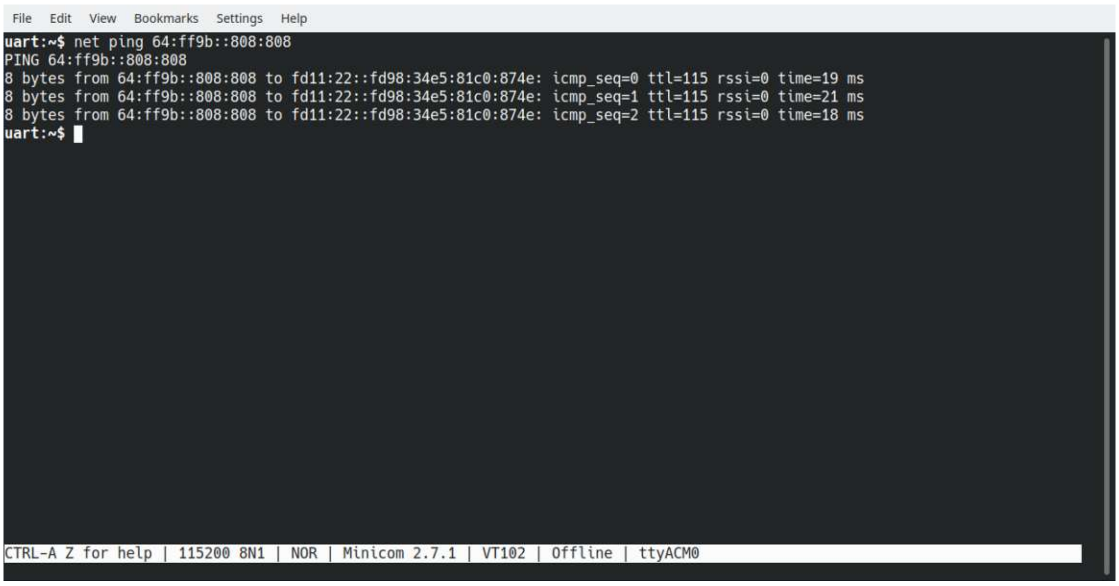
Figure 7.8: Interesting network interfaces on the Border Router

hosts is pinging. By using ping, the status of a connection between different parts of the system can be tested.

First, important network interfaces on the Border Router need to be examined. A list of important interfaces for this thesis is shown on Figure 7.8.

Interface *eth0* is the Ethernet connection from Raspberry Pi to the network where the Base Node (PC) is also connected. *wpan0* is the Thread network interface and *nat64* is

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



```
uart:~$ net ping 64:ff9b::808:808
PING 64:ff9b::808:808
8 bytes from 64:ff9b::808:808 to fd11:22::fd98:34e5:81c0:874e: icmp_seq=0 ttl=115 rssi=0 time=19 ms
8 bytes from 64:ff9b::808:808 to fd11:22::fd98:34e5:81c0:874e: icmp_seq=1 ttl=115 rssi=0 time=21 ms
8 bytes from 64:ff9b::808:808 to fd11:22::fd98:34e5:81c0:874e: icmp_seq=2 ttl=115 rssi=0 time=18 ms
uart:~$

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyACM0
```

Figure 7.9: Pinging the Google Public DNS server

a NAT64 gateway, which is a translator between IPv4 and IPv6 protocols.

7.2.4 NAT 64

NAT64 is an IPv6 transition mechanism that facilitates communication between IPv6 and IPv4 hosts by using a form of network address translation (NAT) [81].

As a test to see if a device inside of an OpenThread network can reach another address on the Internet, a ping to the Google Publich DNS can be done. Google Public DNS is a service from Google introduced on December 9, 2009, which provides recursive DNS servers to outside world. It has the 8.8.8.8 IPv4 address.

To do so, NAT64 as a gateway for translation between IPv4 and IPv6 addresses needs to be used. The well-known prefix reserved for the NAT64 service is 64:ff9b::/96. To ping an IPv4 address from a sensor node, it just needs to be converted it to its hexadecimal representation and add it as a sufix to the well-known prefix. 8.8.8.8 in hexadecimal representation is 808:808, so the address we are trying to ping is 64:ff9b::808:808.

For this ping to work, the Border Router must have its IPv6:LocalAddress set. If this address is not set after a reboot or due to any other reason, a new Thread network should be created with this address set. As can be seen on Figure 7.9, **ping is successful**.

7.2.5 Routing

Figure 7.10 presents the simple architecture used for testing and establishing of the system. IP addresses used in the following subsections are shown, next to the actors they belong to.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

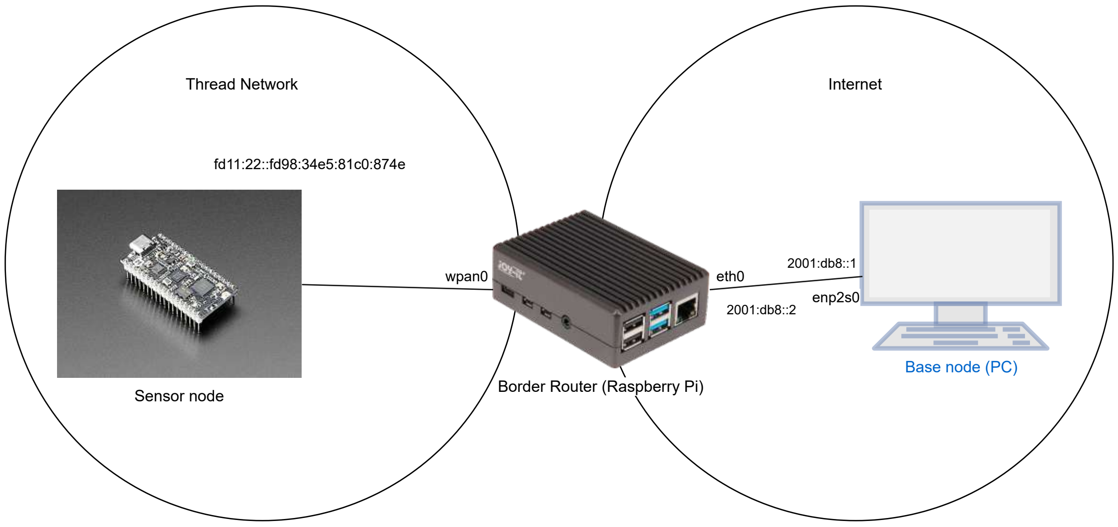


Figure 7.10: Network set up for testing

Connection between Border Router and Base Node

If we further examine the *eth0* interface of Border Router presented on Figure 7.8, the Raspberry has an IPv4 address 172.16.16.119 and a link-local IPv6 address fe80::dea6:32ff:fe7e:3001. This link-local address is, as the name suggests, only accessible on the link itself, meaning that pinging it from the Base Node will work, but a ping outside of this interface cannot work.

On Figure 7.11 the *enp2s0* interface of the Base Node is shown, which is the Ethernet interface. The Name of the interface is slightly weird, because one would expect it to be *eth0*, but that is just how Linux assigns interface names. It also has an IPv4 address 172.16.16.35 and a link-local IPv6 address fe80::e265:56c3:4de:5f51.

IPv4 ping between the Base Node and Border Router works, but this technology is not too interesting to the implementation. In order to get a successful IPv6 ping to the Border Router from a sensor node in Thread or from somewhere outside of the *eth0* link, there needs to be a global IPv6 address assigned.

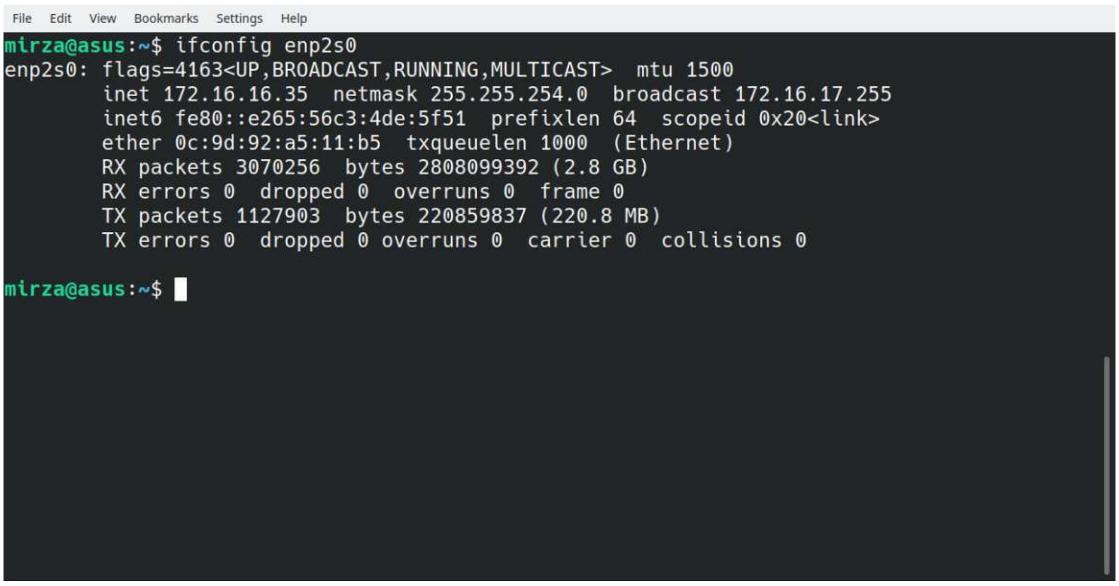
By running the following command, a global IPv6 address 2001:0db8::2/64 will be assigned to the *eth0* interface of the Border Router (see Listing 7.4).

Listing 7.4: Adding a global IPv6 address to the ethernet interface of the RPi

```
sudo ip -6 addr add 2001:0db8::2/64 dev eth0
```

There also needs to be a global IPv6 address on the Base Node itself. A similar command should be run to assign a global IPv6 address to the *enp2s0* interface as well (see Listing 7.5).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



```
mirza@asus:~$ ifconfig enp2s0
enp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.16.35  netmask 255.255.254.0  broadcast 172.16.17.255
        inet6 fe80::e265:56c3:4de:5f51  prefixlen 64  scopeid 0x20<link>
        ether 0c:9d:92:a5:11:b5  txqueuelen 1000  (Ethernet)
        RX packets 3070256  bytes 2808099392 (2.8 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1127903  bytes 220859837 (220.8 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mirza@asus:~$
```

Figure 7.11: enp2s0 interface of the PC (Ethernet)

Now that both Ethernet interfaces have global IPv6 addresses, an IPv6 ping between them is successful.

Listing 7.5: Adding a global IPv6 address to the ethernet interface of the PC

```
sudo ip -6 addr add 2001:0db8::1/64 dev enp2s0
```

Connection between Thread network and Border Router

On a sensor node, ping is a command under the net program. There are no further configurations that need to be done in order to ping the *eth0* interface of the Border Router. Simply pinging the global IPv6 address of the Border Router, like it is shown in Listing 7.6 is enough.

Listing 7.6: Pinging of the Border Router from a sensor node

```
net ping 2001:db8::2
```

Connection between Thread network and Base Node

Testing the connection between the Thread network and the Base Node is done in the same way as it was done for Thread and Border Router. By simply pinging the IPv6 address like it is done in Listing 7.7.

Listing 7.7: Pinging of the Base Node from a sensor node

```
net ping 2001:db8::1
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

But just running this ping does not give a successful result. After further analysis, it was evident that the ping from a sensor node reaches the Base Node, but the response is not propagated. The sensor node has a global IPv6 address of fd11:22::fd98:34e5:81c0:874e. All of the devices in the network have the global addresses of the prefix fd11:22::/64. The problem is, that the Base Node does not know to which interface it should send the packets, which reach the addresses with this prefix. To know this, a prefix fd11:22::/64 needs to be added as a route to the *enp2s0* interface.

Listing 7.8: Enabling routing of messages through enp2s0 interface

```
sudo ip -6 route add fd11:22::/64 via 2001:db8::2 dev enp2s0
```

By executing the code in Listing 7.8, a route for all the messages sent to devices with the prefix fd11:22::/64 will be made. Without this route, pings sent from a sensor node to Base Node will get a network unreachable error. In the command, the IPv6 address of the Border Router is specified as the next hop. If this was to be omitted from the command, the network unreachable error would disappear as well, but there would be a ping timeout instead. After adding the route above to the *enp2s0* interface of the Base Node, there is finally a working IPv6 connection between the Thread network on one side of the Border Router and the Ethernet network on the other side of the Border Router. This makes network communication fully possible and also allows the connection between sensor nodes and the UpdateHub server which is now going to be described.

7.3 UpdateHub

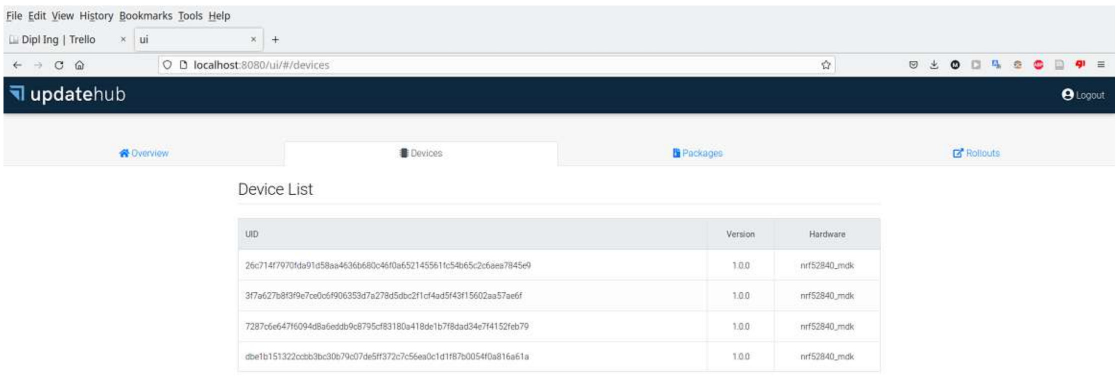
UpdateHub is the chosen API for providing the update to the devices. It comes in two editions, community and enterprise. Community edition is free to use without any royalties and is the choice for this thesis.

As already said UpdateHub works with following protocols: WiFi, modem, BLE IPSOP, 802.15.4 and OpenThread. It is noted on the website that functionality with OpenThread is experimental and there is no support and guidance that makes it work. Probably because of that UpdateHub is not working with Zephyr directly out of the box. A json parsing error is preventing the update to be carried out. There is a known bug fix that is available on GitHub[13]. Due to the nature of Thread, the presence of a Border Router is a requirement to mediate between an IP network and an OpenThread network.

The used UpdateHub server is provided within a docker image that runs on the Base Node. It features a visual GUI that enables the user to see the devices that have the connection, upload packages and create rollouts. Figure 7.12 shows how the server looks like. It allows a selection of individual nodes to be updated or an update of all of the devices. Packages that are uploaded to the UpdateHub server need to comply to a certain format. These packages need to be created using UpdateHub Utilities (uhu)[14].

[13]https://github.com/zephyrproject-rtos/zephyr/issues/41933
[14]https://github.com/UpdateHub/uhu

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.


File Edit View History Bookmarks Tools Help
Dipl Ing | Trello
ui
localhost:8080/ui/#/devices
updatehub
Logout
Overview
Devices
Packages
Rollouts

Device List

UID	Version	Hardware
26c714f7970fda91d58aa4636b680c46f0a652145561fc54b65c2c6aea7845e9	1.0.0	nrf52840_mdk
3f7a627b8f3f9e7ce0c6f906353d7a278d5dbc2f1cf4ad5f43f15602aa57ae6f	1.0.0	nrf52840_mdk
7287c6e647f6094d8a6eddb9c8795cf83180a418de1b7f8dad34e7f4152feb79	1.0.0	nrf52840_mdk
dbe1b151322ccbb3bc30b79c07de5ff372c7c56ea0c1d1f87b0054f0a816a61a	1.0.0	nrf52840_mdk

Figure 7.12: UpdateHub server

There exist two modes in UpdateHub, polling and manual mode. In polling mode, a sensor node periodically asks the UpdateHub server if there is an update available. In manual mode, there is no fixed time period when the node checks the server, the check is rather triggered on an event. This is usually done by executing *updatehub run* command in UpdateHub CLI. To do this, one needs to have access to the sensor node.

7.4 CoAP

CoAP (short for Constrained Application Protocol) is a specialized protocol for use with constrained nodes and constrained networks, like the one we encounter in IoT and WSN. Specified in Request for Comments (RFC) 7252 [82], which was published in June 2014, CoAP is a RESTful protocol similar to Hypertext Transfer Protocol (HTTP). It has the client/server architecture and is designed to easily translate to HTTP for simple integration with the web. Communication in CoAP is done with Requests and Responses. Requests can be either Confirmable or Non-Confirmable. Responses can either be Acknowledgement or Reset, which gives four message types in total. Methods used in the protocol are: GET, POST, PUT and DELETE, which shows the similarity to HTTP.

UDP, which is native to Thread, is a transport protocol and it is not meant for programming at the application layer. Furthermore, it is unreliable by design and there are no confirmations of received messages. TCP is reliable, but it is not native to Thread. It is possible to have TCP over Thread, but it requires complex adjustments. That is why MQTT, another machine-to-machine (M2M) protocol which usually requires TCP (or

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek
WIEN Your knowledge hub

a similar ordered, lossless, bi-directional transport protocol), is a bad choice. HTTP is well-known, but it is also TCP based and thus unsuitable for WSN.

Due to all reasons mentioned above, CoAP becomes the natural choice for the data exchange in the network. It is integrated with OpenThread and there is an API which can be used by importing the CoAP library into the main source file. Sensor nodes are the CoAP clients, while the server can be either the Border Router or the Base Node. For the implementation of the server, the aiocoap[15] Python library was used. As was the case with UpdateHub (which also uses CoAP msessages), the most important thing is to correctly set up the Border Router and routing tables on the Raspberry and on the PC. In the final implementation, the sensor nodes (CoAP clients) send the environment sensing data to the Base Node, which is a CoAP server.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

[15]https://github.com/chrysn/aiocoap

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 8

Evaluation and Discussion

In this chapter, first the evaluation of the implemented system, which is described in Chapter 7, against the set minimum requirements is done. Then a couple of evaluations testing the systems security and measuring the performance have been carried out, described and presented together with the results. Lastly, discussion of the investigated topics, developed system and evaluation are presented, together with the discussion of possible future improvements and research based on the results of this thesis.

8.1 Evaluation against system requirements

In this section, the system is evaluated against the system requirements that were set before the start of the implementation.

8.1.1 Sensor nodes can be successfully updated

A base requirement after all other requirements follow has been successfully met.

8.1.2 Wireless technology suitable for HBA - Thread protocol

Due to the superiority of the Thread protocol for HBA in comparison with other protocols that were investigated, the first requirement was to establish a Thread network. To achieve this, investigation of the protocol itself, reading of the available specification and experimenting was required in order to get the necessary knowledge. The Chapter 6 was dedicated to the introduction of the Thread standard. In the end, a working Thread network together with a Thread Border Router was deployed. That is why this requirement can be marked as met.

8.1.3 Operating system suitable for HBA - Zephyr RTOS

For the OS running on the sensor nodes, one requirement was to run the Zephyr RTOS and the justification for this was given in the system design. The main reason for this requirement was the lack of research about FOTA with Zephyr OS. This requirement was met and this thesis can serve as a starting point for the future research about FOTA possibilities in Zephyr.

8.1.4 Mesh topology

By programming sensor nodes that are used in the Thread protocol upgrade to the Router roles, it was possible to establish and show the meshing ability of Thread. In Chapter 7 Figure 7.7 shows the established mesh network on the Border Router Web GUI.

8.1.5 Possibility of selective updating

This requirement is fully met. On the UpdateHub server it is possible to see all the nodes in the network together with the firmware versions currently running on them. It is possible to select any node freely and update the firmware that runs on it.

8.1.6 Reliability requirement 1: Resilience against transmission errors

For every packet sent during the software update, a certain hash is calculated. This hash is also recalculated at the sensor node when received. By doing so, this reliability requirement of resilience against transmission errors is met.

8.1.7 Reliability requirement 2: Failed update recovery

As the bootloader on the sensor nodes, mcuboot was used. Workings of this bootloader have been explained in Section 7.1.3. The memory is split into slots and this exactly shows the practical application of memory partitioning, which was introduced as a solution to this reliability problem. In this way, it is guaranteed that there is always at least one functional firmware version on the node. If the update is for any reason unsuccessful, the functional firmware is going to be run. This is the rollback mechanism, which provides recovery from an unsuccessful update and makes it possible to try the update process again.

8.1.8 Security 1: Protection against use of firmware on an unauthorized device

By use of public-key cryptography this issue has not been solved, since every device can use the public key to decrypt the firmware. With use of private-key cryptography this requirement would be met. This is the first requirement which was not met.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

8.1.9 Security 2: Protection against use of untrusted third-party firmware

With the use of public-key cryptography this requirement has been met. Only the Base Node has the private key and therefore it is the only member of the network that can correctly encrypt the data, effectively preventing the use of untrusted third-party firmware. The use of private-key cryptography does not theoretically prove that the Base Node was the creator of the firmware, since it can be anyone that has the private key. Practically, the private key is only held at the sensor nodes and at the Base Node. Since nodes are not building and distributing new firmware, private-key cryptography would also meet the requirement.

8.1.10 Security 3: Protection against alteration of the firmware

In order to protect the firmware from alteration, a principle of integrity needs to be followed. It checks for both purposeful and accidental modification of the firmware. Use of digital signatures is a good way of ensuring integrity and that is why this requirement has also been met.

8.1.11 Automatic probing for the update

It is possible to program the nodes in such a way as to probe the UpdateHub server for updates after a certain time period. If there is an active rollout, the process of update will be automatically started. So the requirement is fully met.

8.1.12 Triggered probing of the update after attaching the new sensor

It is possible to manually probe the UpdateHub server for an update. By doing this, an update can be triggered after a certain condition is met, for example attachment of a new sensor. After attachment of a new sensor, the sensor node asks the server for the update, which is then downloaded if the server has made it available. This makes this requirement fulfilled.

8.1.13 Sending of an incremented update instead of a full image:

The implemented system always uses the full image during an update. Hence, this requirement has not been met. Reason for this is the use of mcuboot bootloader, which requires two slots for two valid and working images. Performance of the system is lower because of that, but on the other hand the reliability of the system is ensured.

8.2 Evaluation of the system

In order to measure the performance of the implemented network, a couple of evaluations were carried out. The same hardware presented in Chapter 7 was used in evaluation set ups. For sensor nodes, nordic nrf82540-mdk devices were used, while Raspberry Pi 4B

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

served as a Border Router. The most interesting metric to measure is the time it takes to successfully complete one update. That is why two evaluations are concerning the update times and one additional evaluation is regarding security.

8.2.1 Evaluation setup 1

The purpose of the first setup is to test the routing capabilities of sensor nodes and to measure the performance drop with an increase of hops between the Base Node and the sensor node being updated. When the sensor node being updated is a direct neighbor of the Border Router, that is regarded as an update over a single hop. Since Thread is a multi-hop network, it is possible for a sensor node to be updated even when it is not directly connected to the Border Router. In this set up, update times for one, two, three and four hops were measured. Sensor nodes in this case take a role of a Router in the Thread network. The distance from the Border Router to the first node, which is its neighbor, was 5 meters. Furthermore, the same distance was kept between all other nodes, meaning that with every hop, distance from the Border Router increased for 5 meters.

The sensor nodes were running a base version of the firmware, without any sensor attached to them. Then, a BME680 sensor was attached to the sensor nodes. This triggered the polling of the server for new firmware which contains necessary code for these sensors. The new firmware has already been made available by the UpdateHub server, thus starting the automatic update of the sensor node.

CoAP payload times

Since the time for an update is increasing with every hop, it starts to become increasingly difficult to carry out repeated measurements for more than two hops. The update is distributed in CoAP packet payloads of length 64 bytes. The time it takes for one such packet to be downloaded from the server is measured and an average has been calculated. In order to measure time taken for every packet, a function called *k_uptime_get()* at the start of the packet download and also when the packet was successfully downloaded. This function receives a timestamp which represents the time that passed (in ms) since the boot up. By simply subtracting the first timestamp from the second timestamp it is possible to get the time between those two function calls. Average payload times for one 64 byte CoAP packet are following:

- 1 hop - 45 ms
- 2 hops - 104 ms
- 3 hops - 171 ms
- 4 hops - 233 ms

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

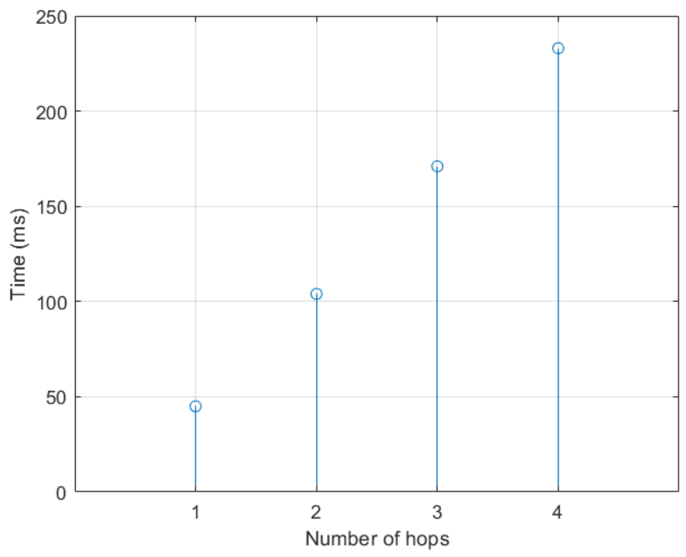


Figure 8.1: Time needed for the download of one 64 bytes packet

When looking at the plot of these values on Figure 8.1, a clear linear progression can be noticed.

Since 1kB has 1024 bytes, dividing 1024 with 64 gives a total of 16 payload blocks per 1kB. By simple multiplication it is possible to get the time needed for 1kB of data with the increase of hops.

- 1 hop: 45 ms * 16 = 720 ms = 0.72s
- 2 hops: 104 ms * 16 = 1664 ms = 1.664s
- 3 hops: 171 ms * 16 = 2736 ms = 2.736s
- 4 hops: 233 ms * 16 = 3728 ms = 3.728s

Expected transmission times

From measured average times needed for one payload, it can easily be calculated how long would the download of an image of particular size take. The total size of the image that was downloaded to the sensor nodes was 374560 bytes. Since it is downloaded in blocks of 64 bytes, it means that there are $\frac{374560}{64} = 5852.5$ blocks needed to download the whole image. Since the time needed for every 64 block is approximately 45 ms, it gives a total of 5852.5×45 ms $= 264362.5$ ms, which is $\frac{264362.5}{1000}$ ms $= 263.36$s. That means that the expected download of the whole image for one hop is $\frac{263.36}{60}s = 4.38$ min or 4 minutes and 23 seconds.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Table 8.1: Measured times of updates

	Measurement	**Time(min:s)**	**Total time(min:s)**
1 hop	Download time	4:36	4:36
	Reboot time	0:38	5:04
	Confirmation	1:00	6:14
2 hops	Download time	11:07	11:07
	Reboot time	0:38	11:45
	Confirmation	1:00	12:45

Total measured times

After downloading the image, a sensor node needs to boot the correct image for the download to be successful. Furthermore, there is a delay between the rebooting and confirmation on the UpdateHub server. On Table 8.1, firmware download, reboot and confirmation times are presented.

Total measured download time is 4 minutes and 36 seconds, which represents an error of approximately 4.95%, which shows that a pretty good estimate was made. Thereby, the value 45 ms could be used for calculating the download time of images of any size. On the same Table 8.1, measured times for a network of 2 hops have been presented.

8.2.2 Evaluation setup 2

The second setup is pretty similar to the first one. Difference this time is that there is always only one node that is being updated and which is directly connected to the Border Router. What is changing is the distance between the Border Router and the sensor node. The goal was to measure how much does the upload time depend on the sole distance from one node to the other. The nearest distance tested is 10 cm, which is almost directly next to the Border Router. As farthest distance a location between the devices of 500cm has been assumed.

As can be seen on Figure 8.2, times needed for the download of one blocks do not vary greatly. They are mostly in range from 40 to 45 ms. This means that a lot of the performance drop encountered in the previous set up comes from routing the message over Routers. What was observed when the sensor node was located 5 meters or even more from the Border Router are occasional loss of connections. However, when the connection was back, time for one payload did not differ too much from time at lower distances.

8.2.3 Evaluation setup 3

Third evaluation served to test of how well the system fulfills the reliability and security requirements that were already discussed. Basically, by signing the firmware image using asymmetric cryptography, which is later checked by the mcuboot bootloader, the system

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

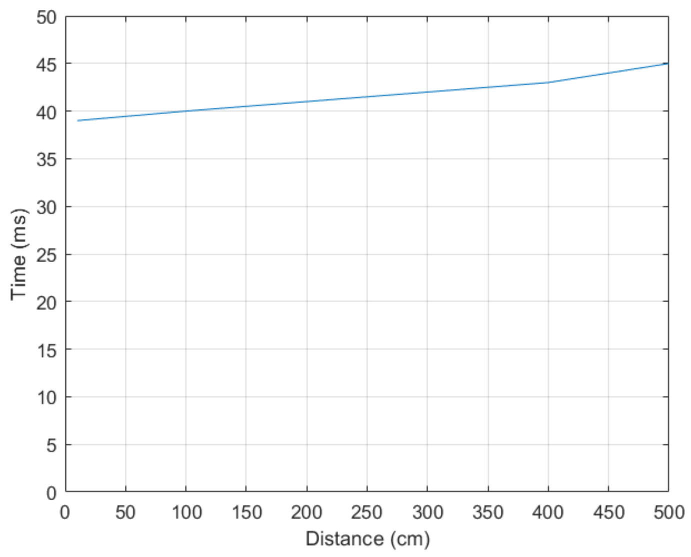


Figure 8.2: One node measurements

is secured against alteration of the authorized firmware and against use of untrusted third-party firmware. This is in case the Base Node and sensor nodes are the only ones that can sign the image, which is why it is important to sign the image with something else than the default key that comes with mcumgr. To test if that is really the case, in third setup an unsigned image was uploaded to UpdateHub and rolled out to one sensor node.

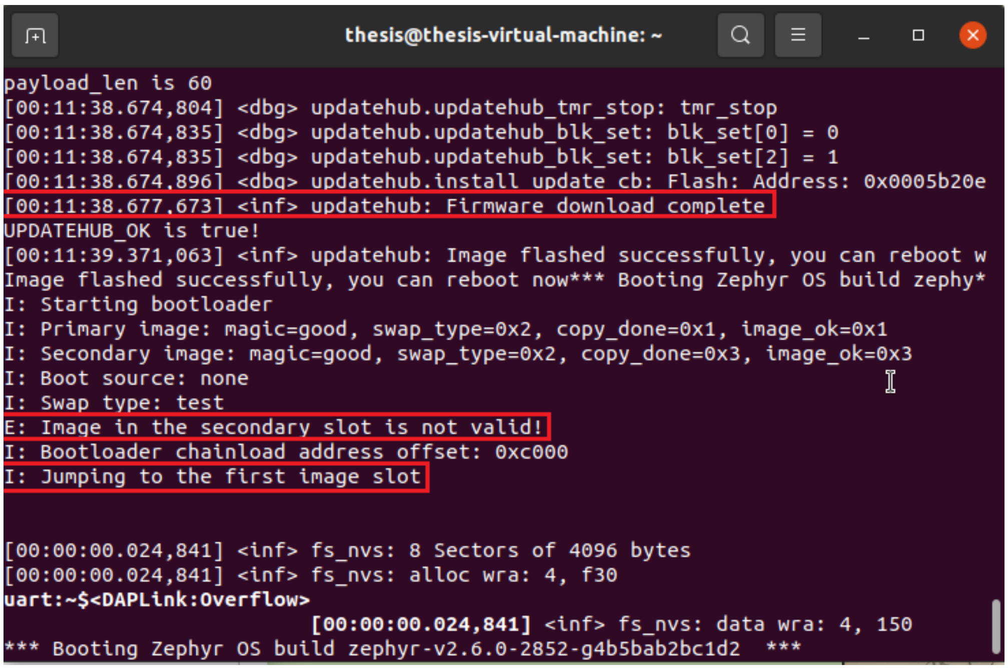
The server allows the upload of an unsigned image. Triggering of the update and the download part go in the same way as was the case with a signed image. After the whole image was downloaded, reboot of the sensor node was initiated. Then the mcumgr bootloader checked the newly downloaded firmware image and after checking the image in the secondary slot it announced that the image is not valid, which can be seen on Figure 8.3. Furthermore, booting of the previous valid image, which was contained in the first slot, was done. This is exactly the behavior which was desired.

8.3 Discussion

This section presents a discussion on the topics covered by this theses, provides a reflection on the aim and the research questions and talks about limitations and further work.

Guided by the first research question: *RQ1: What are the available technologies for the remote management of WSN in HBA and which communication protocols are supported?*, conducted literature research showed that remote management of WSN in HBA is a multilayered problem, with many different aspects that need to be taken into consideration. As the field of HBA is becoming more relevant every day, there exist various technologies

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.


thesis@thesis-virtual-machine: ~

```
payload_len is 60
[00:11:38.674,804] <dbg> updatehub.updatehub_tmr_stop: tmr_stop
[00:11:38.674,835] <dbg> updatehub.updatehub_blk_set: blk_set[0] = 0
[00:11:38.674,835] <dbg> updatehub.updatehub_blk_set: blk_set[2] = 1
[00:11:38.674,896] <dbg> updatehub.install_update_cb: Flash: Address: 0x0005b20e
[00:11:38.677,673] <inf> updatehub: Firmware download complete
UPDATEHUB_OK is true!
[00:11:39.371,063] <inf> updatehub: Image flashed successfully, you can reboot w
Image flashed successfully, you can reboot now*** Booting Zephyr OS build zephy*
I: Starting bootloader
I: Primary image: magic=good, swap_type=0x2, copy_done=0x1, image_ok=0x1
I: Secondary image: magic=good, swap_type=0x2, copy_done=0x3, image_ok=0x3
I: Boot source: none
I: Swap type: test
E: Image in the secondary slot is not valid!
I: Bootloader chainload address offset: 0xc000
I: Jumping to the first image slot


[00:00:00.024,841] <inf> fs_nvs: 8 Sectors of 4096 bytes
[00:00:00.024,841] <inf> fs_nvs: alloc wra: 4, f30
uart:~$<DAPLink:Overflow>
                          [00:00:00.024,841] <inf> fs_nvs: data wra: 4, 150
*** Booting Zephyr OS build zephyr-v2.6.0-2852-g4b5bab2bc1d2  ***
```

Figure 8.3: Unsigned image update attempt

and standards that can be found in practice. This thesis introduced and compared the most popular wireless technologies which enable the use of WSN in HBA. What quickly became evident is that some of the technologies which are extremely widespread and used in almost every modern household, like Wi-Fi, present a particularly bad choice for constraint devices used in WSN. Other network technologies that were specifically designed for WSN, like Z-Wave, ZigBee and Thread, provide much better network characteristics for constrained devices. With the trend of making all the more common household devices "smart", importance of supporting the IP stack is rising. That is why networks that support 6LoWPAN have an advantage over the others.

Since WSN exist for some time already and are not a recent phenomenon, there already exists research of various parameters. OTA process has been researched and improved upon on years, with different dissemination standards improving on one another. But what is lacking is the research of this field when taking into consideration the design choices that were made. The thesis examined and showed that having an OTA system with these choices is possible. It was therefore clear that major design choices of having a meshed Thread WSN with sensor nodes running Zephyr were a right choice.

Regardless of the technology used, if the OTA capability is to be implemented into a real world system, it has to be secure and reliable. To ensure a secure and reliable automatic update of sensor nodes, it is important to be fully aware of what security and reliability is in the first place. Questions like serve to answer the second research question, which asks: *RQ2: What are the minimum requirements necessary to enable secure and reliable*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

automatic update of sensor nodes within a WSN in HBA? In Chapter 3, important principles were described, common problems that happen when these principles where not uphold were presented and from these observations, minimum requirements that should be met in order to ensure secure and reliable updates of sensor nodes were set. Besides design and security requirements, additional requirements which should enable automatic updates of nodes with the attachment of a new sensor were set. With all these requirements in place, it was clear how a developed network should look like. The thesis itself did not cover the rigorous testing of the security and reliability chapters. Most of the security solutions have been solved by design, by actually using digital signatures. Nevertheless, a thorough analysis of the network could be a topic of further research.

As it was already discussed previously in this chapter, the implemented system meets most of the requirements set in Chapter 5. Such a network could be implemented in a real world setting and that is why the reliability and update times of such a network are of particular interest, which is a topic covered by the last research question: *RQ3: How does a network that satisfies the minimum requirements perform regarding common performance measures like update times and recover?* Since Thread is a self-healing and self-organizing network, the implemented system is pretty reliable and does not suffer from problems due to connection losses. Sensor nodes continue their normal operation during the download of new firmware image. If the connection has been lost during download, the device simply continues its normal operation and resumes the download after connection to the server has been reestablished. In case newly downloaded firmware does not pass the checks done by the mcuboot bootloader, a previous version of firmware will be run, making the device still operational. This proves that the reliability of the system is solid. The second important characteristic of the network is the time needed for executing the OTA update. Update times that were observed during evaluations for one-hop and multi-hop network set ups have been presented previously in this chapter. These results show that these update times are longer than one could expect. The biggest bottleneck is a small payload of CoAP blocks used for transmitting the firmware from the Base Node to the sensor nodes. It should be mentioned that enabling the nodes to form mesh connections will probably increase the performance of the network, since there would be multiple paths which could be used for data packets.

8.4 Future improvements

As is the case with every system, there exist various ways in which a system designed and implemented in this thesis could be improved. Design and security of the overall system are satisfactory, so much so that such a system could be implemented in a real world setting without too many adjustments.

A fundamental and valuable improvement would be replacing the UpdateHub with a software solution developed from scratch. UpdateHub provides good and reliable performance, particularly important is the possibility to use it over Thread and for the selective updating of nodes. One problem that was noticed after thorough testing was

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

that once a node was discovered in UpdateHub, it would stay permanently listed on the list of devices on the UpdateHub web GUI. If a node is shut down, or there is a loss of connection due to any reason, it does not disappear from the list of active nodes on the UpdateHub GUI. This gives the system administrator information inconsistent with the actual state in the network.

There is Hawkbit, which might behave better in this sense, but it does not solve the dependency on an already existing solution. Furthermore, there are currently two separate software instances running on the Base Node, the UpdateHub server and the CoAP server, which serves as the main application, collecting the sensor data. UpdateHub server is using CoAP as well, meaning that there are two parallel CoAP servers running on the Base Node. An ideal solution would only have one main application running, which combines both the data collecting functionality of the current CoAP server and the software update functionality of UpdateHub. This would simplify the solution and provide better control of the whole system.

One thing that could always be better is performance. As it was seen in the metrics presented previously in this chapter, the overall time of update is not particularly fast. It is indeed not expected or referable to update the WSN too often. Meaning that slight improvement of update time should not noticeably affect the productivity of the network. But in some networks with very high number of nodes, it might present a problem.

One possibility to improve the update time would be to send the difference between the images, instead of sending the whole image. This approach is not possible with use of mcumgr bootloader, since it requires the separation of the flash for more images. This separation of flash is good, since it allows for always having one working firmware on the node. If there were no flash partitions and only the incremental update of the code would be carried out, this would sacrifice the stability of the system for performance. If there is a way where the rollback mechanism can work and delta updates are possible, that would be present a major improvement of the system.

Finally, the system in itself serves as a base for more complicated and creative applications that can be built on top of it. Matter, an application protocol that runs on top of Thread, was already described in this thesis. In recent months it is starting to get increasingly more popular. This trend is looking to continue in the upcoming months and years, so it would be a good test to use a system developed here to test one or more different Matter applications.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 9

Conclusion

This thesis investigated different technologies, standards and design decisions required for remote update and management of sensor nodes in WSN in HBA. Special focus has been given on important steps that ensure reliability and security of such a system have and key requirements that enable secure automatic update of sensor nodes have been identified. To keep the thesis up to date with the industry trends, for the development of a system it was decided to focus on using the Thread protocol, which is backed by major industry companies and enjoys an increasing support and adoption. In the end, reliability and performance of the implemented system were tested, which gave insights in how a system that fulfills the needed requirements behaves in practice. Reliability of the system proved itself to be pretty good, leaving the nodes operating during connection losses and running an older firmware version in case new one has been corrupted. Thread's self-healing and self-organizing capabilities also contributed to the reliability of the system, since recommisioning of the nodes is possible without the need for human intervention. Performance of the developed FOTA system in terms of update times has not been on par with the reliability of the system. Since downloading of 1 kB of data increases approximately by 1 second for every 5 meter hop, time needed for uploading nodes which are farther away from the Border Router quickly rises. Main reasons for that is the small payload of the packets that distribute the update and the fact that the whole image is downloaded every time for an update. This is a result of the way how the bootloader that enables OTA updates in used Zephyr OS behaves. An important performance improvement would come with use of incremental updates, instead of transferring the whole image. Future research could be done on this problem, investigating ways that enable incremental updates in Zephyr. Another interesting topic for further research would be thorough security and reliability testing of such a system. This would cover this important issue, especially in the field of HBA, in much more depth.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Bibliography

[1] Aaron Hurst. How IoT is driving industrial innovation. https://www.information-age.com/how-iot-driving-industrial-innovation-123496990/. Accessed on 05.09.2021. 1

[2] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, October 2010. 1

[3] Wolfgang Kastner, Georg Neugschwandtner, Stefan Soucek, and H. Michael Newman. Communication systems for building automation and control. *Proceedings of the IEEE*, 93(6):1178–1203, June 2005. Conference Name: Proceedings of the IEEE. 1, 8, 9, 91

[4] Christian Reinisch, Wolfgang Kastner, Georg Neugschwandtner, and Wolfgang Granzer. Wireless Technologies in Home and Building Automation. In *2007 5th IEEE International Conference on Industrial Informatics*, volume 1, pages 93–98, June 2007. ISSN: 2378-363X. 1, 9

[5] Antonio Cilfone, Luca Davoli, Laura Belli, and Gianluigi Ferrari. Wireless Mesh Networking: An IoT-Oriented Perspective Survey on Relevant Technologies. *Future Internet*, 11(4):99, April 2019. Number: 4 Publisher: Multidisciplinary Digital Publishing Institute. 2

[6] Sunita Dixit Indu. Wireless sensor networks: Issues & challenges. *International Journal of Computer Science and Mobile Computing (IJCSMC)*, 3:681–85, 2014. 2

[7] Raspberry Pi 4 Model B. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. Accessed on: 29.09.2021. 5

[8] Verein Deutscher Ingenieure. Gebäudeautomation: Kommentar zu VOB/C: ATV DIN 18299, ATV DIN 18386 (Beuth Kommentar), May 2017. 7

[9] Hermann Merz, Thomas Hansemann, and Christof Hübner. *Building Automation*. Signals and Communication Technology. Springer International Publishing, Cham, 2018. 7, 8

[10] TÜV Süd. Building automation 1: For sustainable and profitable buildings. `https://www.tuvsud.com/en/industries/real-estate/buildings/building-automation`. Accessed on: 04.07.2022. 8

[11] Malte Burkert, Heiko Krumm, and Christoph Fiehe. Technical management system for dependable Building Automation Systems. In *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–8, September 2015. ISSN: 1946-0759. 9, 91

[12] Wojciech Rzepecki and Piotr Ryba. IoTSP: Thread Mesh vs Other Widely used Wireless Protocols – Comparison and use Cases Study. In *2019 7th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 291–295, August 2019. 9, 14

[13] Divya Sharma, Sandeep Verma, and Kanika Sharma. Network Topologies in Wireless Sensor Networks: A Review 1. 10

[14] Quazi Mamun. A Qualitative Comparison of Different Logical Topologies for Wireless Sensor Networks. *Sensors*, 12(11):14887–14913, November 2012. Number: 11 Publisher: Multidisciplinary Digital Publishing Institute. 10

[15] Manish Kumar Singh, Syed Intekhab Amin, Syed Akhtar Imam, Vibhav Kumar Sachan, and Amit Choudhary. A Survey of Wireless Sensor Network and its types. In *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, pages 326–330, October 2018. 11

[16] Gil Reiter. Wireless connectivity for the Internet of Things. *Europe*, 433(868):466, 2014. 13, 16

[17] Wi-Fi Alliance. Wi-Fi 6E expands Wi-Fi® into 6 GHz. Technical report. 13

[18] Oleh Horyachyy. *Comparison of Wireless Communication Technologies used in a Smart Home : Analysis of wireless sensor node based on Arduino in home automation scenario*. 2017. 13, 14, 15, 18, 93

[19] Bluetooth. About Us. `https://www.bluetooth.com/about-us/`. Accessed on: 05.06.2022. 13

[20] Salim Jibrin Danbatta and Asaf Varol. Comparison of Zigbee, Z-Wave, Wi-Fi, and Bluetooth Wireless Technologies Used in Home Automation. In *2019 7th International Symposium on Digital Forensics and Security (ISDFS)*, pages 1–5, June 2019. 13, 16

[21] Chiara Buratti, Andrea Conti, Davide Dardari, and Roberto Verdone. An Overview on Wireless Sensor Networks Technology and Evolution. *Sensors*, 9(9):6869–6896, September 2009. Number: 9 Publisher: Molecular Diversity Preservation International. 14, 15

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[22] IEEE Standard for Low-Rate Wireless Networks. *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)*, pages 1–800, July 2020. Conference Name: IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015). 15, 46

[23] Gabriel Montenegro, Jonathan Hui, David Culler, and Nandakishore Kushalnagar. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. Request for Comments RFC 4944, Internet Engineering Task Force, September 2007. Num Pages: 30. 16

[24] Pascal Thubert and Jonathan Hui. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. Request for Comments RFC 6282, Internet Engineering Task Force, September 2011. Num Pages: 24. 16

[25] Carsten Bormann, Zach Shelby, Samita Chakrabarti, and Erik Nordmark. Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). Request for Comments RFC 6775, Internet Engineering Task Force, November 2012. Num Pages: 55. 16

[26] Pascal Thubert. IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Selective Fragment Recovery. Request for Comments RFC 8931, Internet Engineering Task Force, November 2020. Num Pages: 28. 16

[27] Pascal Thubert and Robert Cragie. IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Paging Dispatch. Request for Comments RFC 8025, Internet Engineering Task Force, November 2016. Num Pages: 8. 16

[28] Samita Chakrabarti, Gabriel Montenegro, Ralph Droms, and james woodyatt. IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) ESC Dispatch Code Points and Guidelines. Request for Comments RFC 8066, Internet Engineering Task Force, February 2017. Num Pages: 9. 16

[29] Roberto Sandre. Thread and ZigBee for home and building automation. page 12, 2018. 16, 17

[30] Seema Kharb and Anita Singhrova. Review of Industrial Standards for Wireless Sensor Networks. In Daya K. Lobiyal, Vibhakar Mansotra, and Umang Singh, editors, *Next-Generation Networks*, pages 77–87, Singapore, 2018. Springer. 17

[31] Thread Group. Thread Group. `https://www.threadgroup.org/thread-group`. Accessed on: 30.10.2022. 17

[32] Anuj Kumar Dwivedi. Wireless Sensor Network: At a Glance. August 2011. 17

[33] Tim Fisher. What Is Firmware? `https://www.lifewire.com/what-is-firmware-2625881`. Accessed on: 30.10.2022. 18

[34] S Brown and C J Sreenan. Updating Software in Wireless Sensor Networks: A Survey. page 14. 19

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[35] Stephen Brown and Cormac J. Sreenan. Software Updating in Wireless Sensor Networks: A Survey and Lacunae. *Journal of Sensor and Actuator Networks*, 2(4):717–760, December 2013. Number: 4 Publisher: Multidisciplinary Digital Publishing Institute. 21, 23, 24, 91

[36] Tracey Ho and Desmond Lun. *Network Coding: An Introduction*. Cambridge University Press, April 2008. Google-Books-ID: rKr9B3vn3a4C. 21

[37] R. Ahlswede, Ning Cai, S.-Y.R. Li, and R.W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, July 2000. Conference Name: IEEE Transactions on Information Theory. 21

[38] Jaein Jeong, Sukun Kim, and Alan Broad. Network reprogramming. *University of California at Berkeley, Berkeley, CA, USA*, 8, 2003. 21

[39] Niels Reijers and Koen Langendoen. Efficient code distribution in wireless sensor networks. In *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, WSNA '03, pages 60–67, New York, NY, USA, September 2003. Association for Computing Machinery. 21, 22

[40] Philip Levis and David Culler. Maté: A tiny virtual machine for sensor networks. *ACM Sigplan Notices*, 37(10):85–95, 2002. Publisher: ACM New York, NY, USA. 22

[41] Thanos Stathopoulos, John Heidemann, and Deborah Estrin. A Remote Code Update Mechanism for Wireless Sensor Networks. Technical report, California University Los Angeles, Center for Embedded Network Sensing, November 2003. 22

[42] S A Quadri and Othman Sidek. Software Maintenance of Deployed Wireless Sensor Nodes for Structural Health Monitoring Systems. 3(2):29, 2013. 22, 23

[43] Philip Levis, Neil Patel, Scott Shenker, and David Culler. Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. page 15. 22

[44] Adam Chlipala, Jonathan Hui, and Gilman Tolle. Deluge: data dissemination for network reprogramming at scale. *University of California, Berkeley, Tech. Rep*, 2004. Publisher: Citeseer. 23

[45] Jonathan W. Hui and David Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 81–94, New York, NY, USA, November 2004. Association for Computing Machinery. 23

[46] Joanna Kulik, Wendi Heinzelman, and Hari Balakrishnan. Negotiation-Based Protocols for Disseminating Information in Wireless Sensor Networks. *Wireless Networks*, 8(2):169–185, March 2002. 23

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[47] Andrew Hagedorn, David Starobinski, and Ari Trachtenberg. Rateless Deluge: Over-the-Air Programming of Wireless Sensor Networks Using Random Linear Codes. In *2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)*, pages 457–466, April 2008. 23

[48] Muhammad Omer Farooq and Thomas Kunz. Operating Systems for Wireless Sensor Networks: A Survey. *Sensors*, 11(6):5900–5930, June 2011. Number: 6 Publisher: Molecular Diversity Preservation International. 24, 25

[49] Fawwad Hassan Jaskani, Saba Manzoor, Muhammad Talha Amin, Muhammad Asif, and Muntaha Irfan. An Investigation on Several Operating Systems for Internet of Things. *EAI Endorsed Transactions on Creative Technologies*, 6(18), January 2019. Number: 18 Publisher: European Alliance for Innovation. 24, 25, 26, 39, 40, 93

[50] Mahesh Pawar and Jitendra Agarwal. A literature survey on security issues of WSN and different types of attacks in network. *Indian J. Comput. Sci. Eng*, 8(2):80–83, 2017. 27

[51] Stuart Foster. Software Safety vs. Security: What's the Difference? `https://www.perforce.com/blog/kw/software-safety-vs-security-whats-different`. Accessed on: 11.11.2022. 27

[52] Atmel AT02333: Safe and Secure Bootloader Implementation for SAM3/4. 27, 28, 29, 30, 31, 32, 33, 34

[53] Pyrgelis Apostolos. Cryptography and security in wireless sensor networks. *FRONTS 2nd Winterschool Braunschweig, Germany*, 2009. 28

[54] Mini Sharma, Aditya Tandon, Subhashini Narayan, and Bharat Bhushan. Classification and analysis of security attacks in WSNs and IEEE 802.15.4 standards : A survey. In *2017 3rd International Conference on Advances in Computing,Communication & Automation (ICACCA) (Fall)*, pages 1–5, September 2017. 28

[55] Moritz Strübe, Florian Lukas, Bijun Li, and Rüdiger Kapitza. DrySim: simulation-aided deployment-specific tailoring of mote-class WSN software. In *Proceedings of the 17th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems*, MSWiM '14, pages 3–11, New York, NY, USA, September 2014. Association for Computing Machinery. 29

[56] C. E. Shannon. Communication theory of secrecy systems. *The Bell System Technical Journal*, 28(4):656–715, October 1949. 30

[57] Briony J Oates. *Researching Information Systems and Computing*. SAGE Publications, Ltd., 2012 edition, 2005. 35, 36, 91

[58] Salvatore T March and Gerald F Smith. Design and natural science research on information technology. *Decision Support Systems*, 15(4):251–266, 1995. 35

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[59] Peter Checkland. Soft Systems Methodology: A Thirty Year Retrospective. *Systems Research and Behavioral Science*, 17:S11–S58, 2000. 35

[60] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 3 2004. 35

[61] Jim Hughes and Trevor Wood-Harper. Systems development as a research act. *Journal of Information Technology*, 14(1):83–94, 1999. 35

[62] Thread Usage of 6LoWPAN. Technical report, Thread Group, July 2015. 47, 48, 91

[63] Steve E. Deering and Bob Hinden. Internet Protocol, Version 6 (IPv6) Specification. Request for Comments RFC 8200, Internet Engineering Task Force, July 2017. Num Pages: 42. 47

[64] David A. Borman, Steve E. Deering, and Bob Hinden. IPv6 Jumbograms. Request for Comments RFC 2675, Internet Engineering Task Force, August 1999. Num Pages: 9. 47

[65] Thread Network Fundamentals v3. Technical report, Thread Group, May 2020. 46, 49, 51, 91

[66] Thread 1.3.0 Features White Paper. Technical report, Thread Group, July 2022. 50

[67] UG103.11: Thread Fundamentals. Technical report, Silicon Laboratories, 2022. 50

[68] OpenThread. Thread Primer. `https://openthread.io/guides/thread-primer`. Accessed on 27.11.2022. 51, 52, 53, 54, 55, 56, 91, 93

[69] Richard Kelsey. Mesh Link Establishment. Internet Draft draft-ietf-6lo-mesh-link-establishment-00, Internet Engineering Task Force, December 2015. Num Pages: 19. 55

[70] Hyung-Sin Kim, Sam Kumar, and David E. Culler. Thread/OpenThread: A Compromise in Low-Power Wireless Multihop Network Architecture for the Internet of Things. *IEEE Communications Magazine*, 57(7):55–61, July 2019. Conference Name: IEEE Communications Magazine. 56

[71] Matter Security and Privacy White Paper. Technical report, Connectivity Standards Alliance, March 2022. 57, 58, 91

[72] Nordic Semiconductor. nRF52840 Documentation. `https://nsscprodmedia.blob.core.windows.net/prod/software-and-other-downloads/product-briefs/old-versions/nrf52840pbv20.pdf`. Accessed on: 02.09.2022. 62

[73] Makerdiary. nRF52840-MDK Documentation. `https://wiki.makerdiary.com/nrf52840-mdk/`. Accessed on 02.09.2022. 62

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[74] mcuboot. Building and using mcuboot with zephyr. https://www.mcuboot.com/documentation/readme-zephyr/. Accessed on: 31.10.2021. 65, 67, 91

[75] Zephyr. MCUmgr — Zephyr Project Documentation. https://docs.zephyrproject.org/3.1.0/services/device_mgmt/mcumgr.html. Accessed on: 31.10.2021. 65

[76] Zephyr. Updatehub embedded Firmware Over-The-Air (FOTA) sample. https://docs.zephyrproject.org/2.6.0/samples/subsys/mgmt/updatehub/README.html. Accessed on: 31.10.2021. 65

[77] Zephyr. Hawkbit Direct Device Integration API sample. https://docs.zephyrproject.org/2.6.0/samples/subsys/mgmt/hawkbit/README.html. Accessed on: 31.10.2021. 65

[78] Nordic Semiconductor. Bootloader — MCUboot 1.9.99 documentation. https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/mcuboot/design.html. Accessed on: 31.08.2022. 66

[79] OpenThread. OpenThread Border Router. https://openthread.io/guides/border-router. Accessed on: 07.09.2022. 70, 71, 91

[80] Robert S. Quattlebaum and james woodyatt. Spinel Host-Controller Protocol. Internet Draft draft-rquattle-spinel-unified-00, Internet Engineering Task Force, May 2017. Num Pages: 78. 70

[81] Vikalp Singh. Understanding and Configuring NAT64. https://www.cisco.com/c/en/us/support/docs/ip/network-address-translation-nat/217208-understanding-nat64-and-its-configuratio.html. Accessed on: 11.11.2022. 73

[82] Zach Shelby, Klaus Hartke, and Carsten Bormann. The Constrained Application Protocol (CoAP). Request for Comments RFC 7252, Internet Engineering Task Force, June 2014. Num Pages: 112. 77

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.