Faculteit Wetenschappen

School voor Informatietechnologie

master in de informatica

Masterthesis

Sensor networks & the internet of things

Bas Van Assche
Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :
Prof. dr. Peter QUAX

2019
2020

Faculteit Wetenschappen

School voor Informatietechnologie

master in de informatica

Masterthesis

Sensor networks & the internet of things

Bas Van Assche

Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :

Prof. dr. Peter QUAX

Abstract

With the Internet of Things (IoT) becoming increasingly popular, more and more domains recognize its value and invest in new applications. In this master thesis, we perform a survey on IoT communication technologies including Sigfox, LoRa/LoRaWAN, NB-IoT, LTE-M, Zigbee, Z-Wave and Thread where we analyze and compare their inner working, security and vulnerabilities. We also perform a survey on IoT Edge technologies including EdgeX, Azure IoT Edge, Balena and Fogflow where we discuss their architecture and compare their capabilities and setup. Finally, we implement an example IoT application where we minimize the electrical bill of households by scheduling their electrical devices based on the predicted electricity price.

Acknowledgements

I would like to thank my promoter Peter Quax and my mentor Joris Herbots for their help and feedback on my thesis, without their help this would not have been possible.

Contents

Chapter 1

Nederlandse Samenvatting

Het Internet of Things (IoT) is een globaal netwerk van toestellen en machines die met elkaar kunnen communiceren. De term IoT werd voor het eerst gebruikt in 1997 en de dag van vandaag bestaat er een grote variatie aan IoT toepassingen binnen verschillende domeinen zoals gezondheid en industrie. IoT kent reeds een enorme groei met 2 miljoen toestellen in 2006 tot 200 miljoen toestellen in 2020 en de voorspelling blijven stijgen, zo voorspelt Intel dat IoT technologieën meer dan 5.47 biljoen euro waard zullen zijn in 2025.

Maar er zijn een groot aantal communicatietechnologieën die gebruikt kunnen worden, met verschillende eigenschappen en security protocollen. Het doel van deze thesis is een overzicht te geven van de meest gebruikte en meest belovende IoT communicatietechnologieën aan de hand van de volgende onderzoeksvragen: “Welke communicatietechnologieën worden er de dag van vandaag gebruikt en ”:

1. “hoe werken deze technologieën?”
2. “welke security mechanisme worden er gebruikt?”
3. “welke security vulnerabilities zijn er?”
4. “welke technologie is het meeste geschikt voor welke use case?”

Communicatie is maar een kant van het verhaal, IoT toestellen moeten ook beheerd worden en hun informatie moet verzameld worden om er de nodige waarde uit te halen. Volgens de International Data Corporation zal in 2020 meer dan 59 zettabytes aan data gemaakt, gekopieerd en gebruikt worden terwijl dit in 2010 nog maar 1 zettabyte was. Cisco voorspeld dat er in 2023 ongeveer 14.7 miljard connectie zullen bestaat waarvan 50% van IoT toestellen komen. Een probleem hierbij is dat de huidige netwerken het steeds moeilijker zullen hebben met deze hoeveelheid data te verwerken. In deze thesis kijken we daarom naar edge devices, wat toestellen zijn die taken kunnen verwerken dichterbij de IoT devices waardoor maar een klein deel van het netwerk gebruikt wordt. We bespreken een aantal van deze technologieën aan de hand van de volgende onderzoeksvragen: “Welke IoT edge technologieën zijn er de dag van vandaag beschikbaar in het IoT en ”:

1. “hoe ziet hun architectuur eruit?”
2. “hoe werken ze in de praktijk?”
3. “welke technologieën zijn het meest geschikt voor welke applicatie?”

Als laatste gebruiken we de kennis die we op hebben gedaan uit de eerste twee delen in een toepassing over energiebeheer in woningen. In de huidige elektriciteitsnetten zitten er een grote pieken in de opgewekte stroom en de verbruikte stroom die leiden tot hoge kosten voor de netwerkbeheerder en de consument. Het doel van onze applicatie is deze pieken op te vangen in de woningen door te bepalen om welke momenten welke elektrische toestellen aan staan.

1.1 Domein Studie Communicatietechnologieën

In deze studie vergelijken we de volgende IoT communicatietechnieken: Sigfox, LoRa/LoraWAN, NB-IoT, LTE-M, Zigbee, Z-Wave en Thread. Per communicatietechnologie bespreken we de architectuur, welke type toestellen we kunnen onderscheiden, hoe het netwerk eruit ziet, hoe de protocollen werken, welke packet formaten er gebruikt worden en uiteindelijk kijken we naar de beveiliging en gaten in deze beveiliging.

1.1.1 Sigfox

Sigfox is een communicatietechnologie die geschikt is voor zeer kleine data pakketten (12 Bytes) over een zeer lange afstand te verzenden. Het is vooral bedoeld voor sensoren aangezien de downlink communicatie nog meer beperkt is. Sigfox is eigendom van het bedrijf Sigfox en men kan via een abonnement gebruik maken van deze technologie om maximaal 140 keer per dag een uplink pakket te versturen. Sigfox heeft een eigen netwerk en handelt alle communicatie met de sensoren zelf af. Men kan een callback functie registeren in de Sigfox applicatieserver om de data van sensoren te ontvangen.

Elke toestel dat kan verbinden met het Sigfox netwerk heeft een licentie van Sigfox zelf. Daarbij krijgt elk toestel een eigen symmetrische key, waarmee alle communicatie tussen de sensoren en het Sigfox netwerk geëncrypteerd en gesigneerd wordt. Het encrypteren van de inhoud is daarbij niet verplicht. Men beveiligt de pakketten tegen replay attacks door deze te voorzien van een sequentienummer.

Een eerste attack op Sigfox maakt gebruikt van het beperkte aantal bits dat er gebruikt wordt voor het sequentienummer, waardoor een attacker berichten van 30 dagen geleden opnieuw kan versturen. Deze techniek kan opnieuw gebruikt worden in een Denial of Service (DoS) attack wanneer men het huidige sequentienummer ruim vergroot wordt waardoor echte berichten, met een kleiner sequentienummer, niet meer geaccepteerd worden.

1.1.2 Lora/LoraWAN

LoRa is een datalink protocol dat wordt gebruikt in LoRaWAN, beide zijn het technologieën met opnieuw een groot bereik en kunnen er maar kleine hoeveelheden aan data verzonden worden. In tegenstelling tot Sigfox is LoRaWAN een open protocol dat ook in privé netwerken gebruikt kan worden.

Elke toestel beschikt bij de installatie over een ID en aan applicatie key waarvan session keys worden afgeleid om de inhoud van berichten te encrypteren en signeren. Er wordt opnieuw een sequentienummer gebruikt tegen replay attacks.

In LoRaWAN 1.1 is het mogelijk om een DoS attack uit te voeren door twee opeenvolgende join requests te jammen en dan het eerste join request opnieuw te verzenden en het antwoord aan het IoT toestel te geven. Hierdoor zullen de sequentienummer niet meer overeenkomen en zullen foute beveiliging keys aangemaakt worden waardoor het IoT device niet meer kan communiceren met het LoRaWAN netwerk.

We voeren ook een experiment uit waarbij we een prive LoRaWAN netwerk opzetten en dit gebruiken om een tracking applicatie te maken. Daarbij gebruiken we twee Lopy4 toestellen als sensoren die beide hun locatie doorgeven aan onze applicatie gebruikt makend van LoRaWAN. Dit is eenvoudig te realiseren dankzij de ChirpStack opensource implementatie van de netwerk - en applicatieserver van LoRaWAN.

1.1.3 NB-IoT en LTE-M

NB-IoT en LTE-M zijn twee protocollen die gebaseerd zijn op Long Term Evolution (LTE) en kunnen ook in bestaande LTE netwerken met een software update gedeployed worden. Beide technologieën zijn sterk gelijkend met LTE, ze gebruiken onder andere een SIM kaart, maar een aantal features zijn weggelaten en een aantal nieuwe technieken worden gebruikt om het batterijverbruik te verkleinen. Zowel NB-IoT als LTE-M bieden een hogere bandbreedte van respectievelijk 200 Kbps en 1 Mbps en ook een hogere maximale data limiet. NB-IoT heeft een lager energieverbruik dat LTE-M en is enkel geschikt voor niet mobiele apparaten in tegenstelling tot LTE-M. De technologieën zijn afgeleid van LTE maar zullen beide onderdeel zijn van de 5G standaard.

Een symmetrische key is aanwezig in de simkaarten die nodig zijn om met NB-IoT en LTE-M netwerken te verbinden. Van deze symmetrische key worden session keys afgeleid voor de encryptie en authenticatie van de communicatie. Deze session keys worden opgeslagen op de simkaart en de simkaart is enkel toegankelijk door hun API waardoor deze afgeschermd worden van het toestel.

Het is mogelijk een DoS attack uit te voeren mits de attacker gebruikt maakt van een eigen basestation. Er zijn een aantal pakketten die verzonden kunnen worden voor de authenticatie van de gebruiker waaronder een "attach reject". Wanneer een IoT toestel een "attach reject" pakket krijgt, zal deze niet meer proberen opnieuw met dat netwerk te verbinden. De attacker zorgt er in dat geval voor dat het toestel geen netwerkverbinding kan maken.

1.1.4 Zigbee

Zigbee is een protocol voor op een kortere afstand van 10 tot 100 meter. Zigbee is een open standaard en wordt vooraf gebruikt in smart devices voor huizen en kantoren. Het protocol maakt gebruik van een mesh netwerk om gemakkelijk het hele gebouw te kunnen dekken en heeft een maximale doorvoer van 250 Kbps.

Het toestel dat het Zigbee netwerk start, de Zigbee controller, maakt een willekeurige netwerk key aan welke uitgedeeld wordt aan de toestellen die toegevoegd worden aan het netwerk. Tijdens deze uitwisseling wordt de netwerk key geëncrypteerd, gebruikt makend van een globale welbekende key of een link key gebruikt die via een "out of band" methode wordt opgesteld. De eerste methode is minder veilig dan de tweede methode aangezien de globale key ook gekend is door attackers. Wanneer de netwerk key bekend is, kunnen er nieuwe keys gegenereerd worden om het verkeer tussen twee specifieke toestellen te encrypteren. Zigbee maakt gebruik van AES encryptie, frame counters om replay attacks te voorkomen en integrity checks zodat het bericht niet kan aangepast worden.

De reeds eerder geziene attack, waarbij een attacker een pakket zendt met een sterk verhoogd sequentienummer en zo een DoS attack uitvoert op het Zigbee toestel, is hier ook van toepassing. Bij een tweede mogelijke attack wordt de netwerk key onderschept: als de globale welbekende key wordt gebruikt om de netwerk key te verzenden naar nieuwe toestellen, kan de netwerk key onderschept worden en kunnen ook volgende keys afgeleid worden.

1.1.5 Z-Wave

Z-Wave is een ander protocol dat net als Zigbee op korte afstand werkt en een mesh netwerk gebruikt. Maar in tegenstelling tot Zigbee is Z-Wave een gesloten protocol en is de pakketgrootte en doorvoer (9-100 Kbps) meer beperkt.

Wanneer een netwerk wordt aangemaakt, zal er door het eerste toestel een netwerk key worden opgesteld. Deze netwerk key wordt verzonden naar nieuwe toestellen door deze te encrypteren met een vaste key die hardcoded in de Z-Wave firmware zit. Dan wordt er gebruik gemaakt van AES voor de encryptie, een integrity code en een sequentienummer om replay attacks te vermijden.

De netwerk key kan onderschept worden aangezien er enkel een hardcoded key gebruikt wordt om deze naar nieuwe apparaten te verzenden. Omdat onderliggende lagen in het Z-Wave protocol niet worden beveiligd, is het ook mogelijk om source adressen te spoofen en zo foute routing informatie te verspreiden waardoor er een DoS attack kan uitgevoerd worden.

1.1.6 Thread/OpenThread

Thread is open standaard die ook gebruikt wordt voor IoT oplossingen in woningen op korte afstand. Net als Z-Wave en Zigbee, maakt Thread gebruik van een mesh netwerk. Openthread is een opensource implementatie van Thread. Thread werkt in tegenstelling tot Zigbee en Z-Wave op basis van IP, meerbepaald IPv6. Maar het IPv6 pakket formaat wordt aangepast (bv. kortere IPv6 adressen) zodat dit minder overhead geeft. De onderliggende lagen zijn identiek aan de onderliggende lagen van Zigbee waardoor het pakketformaat en de doorvoer gelijkaardig is aan Zigbee.

Er wordt een master key gegenereerd door het toestel dat het netwerk opzet, welke naar de nieuwe toestellen verzonden wordt over een DTLS verbinding. DTLS is hetzelfde als TLS maar maakt gebruik van UDP in plaats van TCP. Van de master key worden andere keys afgeleid voor het encrypten en authenticeren, waarbij men gebruik maakt van AES. Om replay attacks te vermijden maakt Thread gebruik van sequentienummers.

Thread wordt gezien als een relatief veilig protocol tot op de dag van vandaag. Het is wel mogelijk om een DoS attack uit te voeren door de handshakes van DTLS te laten falen omdat hier zware berekeningen moeten uitgevoerd worden op toestellen met een meestal relatief beperkte rekencapaciteit. Verder is het mogelijk pakketten te jammen en niet geautoriseerde ACKs te verzenden waardoor men opnieuw een DoS attack kan uitvoeren.

1.1.7 Vergelijking

Sigfox, LoRaWAN, NB-IoT en LTE-M zijn protocollen voor op een lange afstand, waarbij Sigfox en LoRaWAN eerder een beperkte hoeveelheid data kunnen versturen maar daardoor ook zuiniger op vlak van energieverbruik. Sigfox en LoRaWAN zijn geschikt voor toepassingen als geluidsmonitoring, luchtmonitoring, temperatuur monitoring en waterlekdetectie. NB-IoT en LTE-M zijn IP gebaseerd en hebben meer doorvoercapaciteit wat hun beter geschikt maakt voor toepassingen als smart parking, verkeersmonitoring en security systemen.

Zigbee, Z-Wave en Thread zijn technologieën voor toepassingen binnenshuis. Z-Wave heeft een gelimiteerde netwerkcapaciteit en heeft ook een aantal ernstige beveiligingsrisico's. Het grootste verschil tussen Thread en Zigbee is dat Thread gebruik maakt van de IP stack en daardoor gemakkelijker kan communiceren met het internet.

1.2 Domein Studie IoT Edge Technologieën

In deze studie beschrijven we EdgeX, Azure IoT edge, Balena en FogFlow aan de hand van hun architectuur, waar ze gebruikt kunnen worden en hun setup aan de hand van een experiment.

Voordien hebben we eerst in een eigen experiment een IoT edge technologie proberen te ontwikkelen om hiermee kennis te maken en een aantal doelen te kunnen zetten voor andere technologieën.

1.2.1 Experiment

Hier maken we een "aggregator" die data vanuit verschillende sensoren kan combineren alvorens deze naar de cloud te verzenden. In het experiment ontwikkelde we een merge operator die twee sensorgegevens binnen krijgt en deze combineert tot één resultaat.

Deze merge operators kunnen in docker containers automatisch gedeployed worden waar dit nodig is. Dus als een sensor in een bepaald regio actief worden zal er pas op dat moment een merge operator gedeployed worden naar het edge device waarmee het in verbinding staat.

Het nadeel van deze aggregator was dat er geen downlink communicatie mogelijk was. Aangezien we de use cases voor deze toepassing te beperkt vonden, hebben we verdere ontwikkelingen stopgezet. Uit dit experiment leren we dat het belangrijk is onmiddellijk vanuit een edge device feedback te kunnen geven op bepaalde events zonder daarvoor eerst naar de cloud te moeten gaan.

1.2.2 EdgeX

EdgeX is een framework dat kan worden gedeployed op één enkel edge toestel. Met het framework kan men eenvoudig IoT toestellen beheren en ook het formaat van uplink - en downlink messages bepalen. Er is ook de mogelijkheid om rules toe te voegen die getriggerd worden door een bepaald event en daarop een bepaalde actie naar een device verzenden.

Het nadeel van deze technologie is dat het framework maar één edge toestel ondersteund en dus geen managementtaken biedt over verschillende edge toestellen heen.

1.2.3 Azure IoT Edge

Azure IoT edge biedt een volledige oplossing van het managen van communicatie tot het beheren van IoT toestellen en IoT edge toestellen. Er wordt gewerkt met modules in de vorm van docker containers die gedeployed kunnen worden vanuit de cloud op een aantal edge toestellen. Deze modules kunnen subscriben op gegevens die vanuit IoT toestellen worden verzonden. De modules kunnen op basis van hun eigen logica hier actie aan koppelen. Een voorbeeld van een module is een brandalarm, waarbij de module de temperatuur van alle ruimtes in de gaten houdt en het blussysteem in gang zet mits een van de ruimte te warm wordt. In tegenstelling tot EdgeX wordt deze logic volledig in code geschreven en niet in de vorm van rules wat een zeer krachtige setup is.

Het nadeel van Azure IoT edge is dat het besturingssysteem van de edge toestellen niet beheerd wordt door Azure IoT. Hiervoor moet er dus nog een andere technologie gebruikt worden.

1.2.4 Balena

Balena is een framework dat gebruikt kan worden om een grote hoeveelheid Linux toestellen te beheren en hierop eenvoudig software te deployen. Balena voorziet een volledig besturingssysteem dat na de boot automatisch zal verbinden met de Balena cloud en vanaf dan volledig te besturen is vanuit het Balena platform. Zowel het besturingssysteem als de cloud module zijn open source.

De toestellen beheerd door Balena worden volledig up to date gehouden met de laatste versie van hun besturingssysteem en applicaties worden naar de toestellen verzonden in de vorm van een docker container. Balena voert enkel deze twee taken uit en zorgt dus niet voor IoT device beheer en communicatiebeheer.

1.2.5 FogFlow

FogFlow is een fog computing framework dat helpt om IoT toestellen en fog nodes (vergelijkbaar met edge toestellen) te beheren. Vanuit IoT devices is het mogelijk om berichten naar het fog netwerk te sturen via een software library. Op fog nodes kunnen modules gedeployed worden, welke fog functions genoemd worden, die op basis van de data van IoT toestellen bepaalde acties kunnen uitvoeren.

Het verschil met Azure IoT edge is dat fog functies niet standaard op alle edge toestellen gedeployed zijn maar dynamisch over het netwerk van fog nodes gedeployed worden waar deze het meeste nodig zijn.

1.2.6 Vergelijking

FogFlow en Azure IoT edge zijn twee frameworks die zowel management taken uitvoeren als de communicatie afhandelen. Een verschil is dat FogFlow een open source framework is en dat Azure IoT edge sterk gekoppeld is aan de Azure cloud. Een ander verschil is dat FogFlow dynamisch applicaties (of fog functions) deployed en dat bij Azure IoT vast staat. Het voordeel van Azure IoT is dat de documentatie en het support veel beter zijn. EdgeX doet minder managementtaken, en Balena doet enkel management taken. Balena kan in combinatie met de andere frameworks gebruikt worden om de besturingssystemen up te date te houden.

1.3 Gebruiksscenario

Tot slot maken we een IoT toepassing gebruik makend van de kennis opgedaan over de twee voorgaande domein studies. In de huidige elektriciteitsnetten zit er grote pieken in de opgewekte stroom en de verbruikte stroom dat leidt tot hoge kosten voor de netwerkbeheerder en de consument. Met deze IoT toepassing verlagen we de ballast op het elektriciteitsnetwerk en verlagen we de prijs voor de consument.

We gebruiken een edge toestel om in woningen de volgende IoT toestellen aan te sturen:

- Centrale verwarming
- Airco
- Ijskast
- Wasmachine, droogkast and afwasmachine
- Zonnepanelen
- Batterij

- Keukenvuur

In het edge toestel wordt er een optimale planning opgesteld voor wanneer welk toestel actief is. Die planning wordt opgesteld op basis van een voorspelling van de elektriciteitsprijs voor de komende 24 uur. Verder wordt er rekening gehouden met de volgende zaken: Het temperatuurmodel van het huis, rekening houdend met de buitentemperatuur, binnentemperatuur, zonnestralen en de verwarming. Dit model wordt gevormd op basis van historische data. Het omgekeerde temperatuurmodel wordt gebruikt voor de ijskast en airco. De voorspelde energieproductie van de zonnepanelen op basis van de weersvoorspelling. De capaciteit van de batterij.

Er wordt gebruik gemaakt van simulated annealing om de optimale indeling te maken voor elk van de toestellen. Deze planning wordt door middel van Zigbee naar de toestellen gecommuniceerd.

Chapter 2

Introduction

The Internet of Things (IoT) refers to a global network of devices and machines that can communicate with each other without the need for human-to-computer interaction. The term IoT was first used in 1997 and today lots of applications exist in health care, manufacturing, retail and households. The number of connected devices is growing fast, from 2 billion connected objects in 2006 to 200 billion in 2020. Intel predicts the global worth of IoT technologies could be as much as 5.47 trillion euro's in 2025 [1].

There are however a large variety of communication technologies out there with different abilities and security protocols. The goal of this thesis is to provide an overview of the communication technologies that can be used in the Internet of Things, discuss their inner working and analyze their security and known vulnerabilities. To achieve these goals we do a domain survey of the IoT communication technologies by formulating the following research questions: "What communication technologies are available and used today in the Internet of Things and "

1. "how do these technologies work?"
2. "what security mechanism do they use?"
3. "what are their vulnerabilities?"
4. "what technologies are best suited for what use cases?"

Communication in only one side of the IoT, the IoT devices must be managed and their information must be combined to provide useful information to the end-user. According to the International Data Corporation, more than 59 zettabytes of data will be created, captured copied and consumed in the year 2020 [2]. In 2010 the mark of 1 zettabyte was passed. The Cisco Annual Internet Report predicts there will be 14.7 billion Machine to Machine connections by 2023 and 50 % will be from IoT devices [3]. The current networks will have trouble handling these massive amounts of data, and therefore technologies that handle data closer to the IoT end-devices are needed, which are known as IoT edge technologies. The goal is to provide an overview of IoT edge technologies, describe their architecture and discuss their setup. To achieve these goals we do a domain survey of IoT edge technologies by formulating the following research questions: "What IoT edge technologies are available and used today in the Internet of Things and "

1. "what does their architecture look like?"
2. "how do they work in practice?"
3. "what technologies are best suited for what type of applications?"

Finally, we use the knowledge gathered from the communication survey and edge survey in a usage scenario about indoor power management. Modern electrical power grids have a lot of variation in production supply and customer demand, which leads to high costs for the retailer and consumer. The goal of our IoT application is to minimize the variations in production and demand to reduce the costs for the consumer and retailer.

Chapter 3

Survey IoT Communication Technologies

IoT communication technologies provide a data channel between IoT devices such as sensors, actuators, gateways and the cloud. In this chapter, we analyze and compare some of the best known and most promising IoT communication technologies. We focus only on low power IoT technologies because they provide the best battery life and are best suited for sensors and actuators. Therefore we do not analyze protocols such as Institute of Electrical and Electronics Engineers (IEEE) 802.11 and LTE. This survey includes the following protocols: Sigfox, LoRa/LoraWAN, NB-IoT, LTE-M, Zigbee, Z-Wave and Thread/OpenThread.

For each of the technologies, we start by briefly introducing the technology. Next, we discuss the architecture: what type of devices are used and what the network layout looks like. Then we explain how the different layers of the protocol work together and what packet format is used. Finally, we discuss the security features of the protocol and what vulnerabilities are known to the technology.

In Section 3.7 we compare the technologies, included in this survey, based on their features, security protocols and known vulnerabilities and provide an overview of those features.

3.1 Sigfox

Sigfox is a long-range communication protocol developed for sensor networks. The hardware used in Sigfox devices is open but users have to pay for a Sigfox subscription to use the technology. The subscription provides a global network that at the time of writing is still being rolled out. Europe is already covered [1]. All communication from and to end-devices is handled by the Sigfox core network. All the end-devices and data are managed on the Sigfox platform, the received data can be gathered through a callback [4].

The Sigfox architecture is shown in Figure 3.1. The end-devices send and receive frames wirelessly from the Sigfox radio hubs using the Sigfox radio protocol. From there on out, all communication is done over the Internet Protocol (IP). The radio hubs are part of the Sigfox core network. An application server can be connected to receive the gathered data through a callback [5].

3.1.1 Sigfox Radio [5]

The Sigfox radio protocol is named **3D-UNB** in the specifications. UNB stands for Ultra Narrow Band and 3D stands for triple diversity: diversity in time, frequency and space. 3D-UNB operates in a license-free spectrum between 868 MHz and 923 MHz. However, due to frequency band restrictions by countries, 3D-UNB uses Radio Configurations to define the local radio parameters. A 3D-UNB device must implement all configuration parameters for the countries it is intended to be used in. The Sigfox core network broadcasts the regional profile to help mobile devices discover the correct settings.

[1] `https://www.sigfox.com/en/coverage`

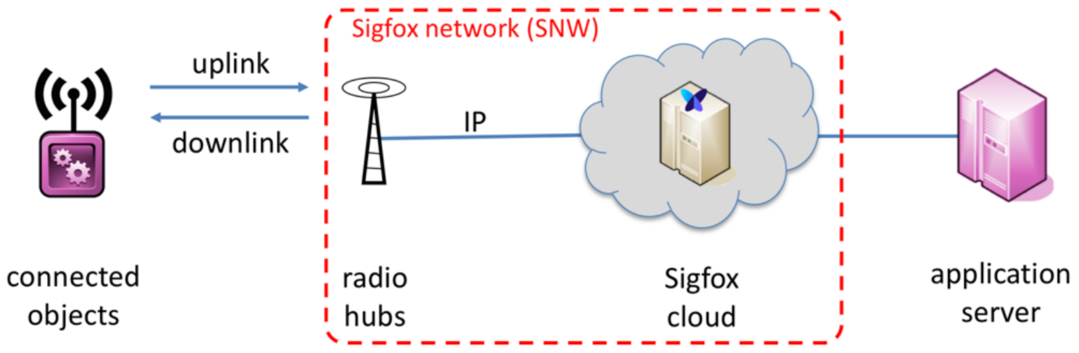


Figure 3.1: Sigfox architecture. Source: [5]

Each radio configuration contains an uplink and downlink channel of 192 kHz wide. Depending on the country configuration, an end-device gets to pick an uplink throughput of 100 bps or 600 bps. The availability depends on the country configuration. The downlink channel always uses 600 bps.

Because the Sigfox protocol is open on the device-side and hidden on the server-side we will only discuss the device side.

Modulation

3D-UNB uses **Differential Binary Phase Shift Keying (D-BPSK)** for the uplink. D-BPSK is a modulation technique where the phase of the signal changes depending on the data bits being sent. If the data bit is 0, the phase of the signal is not reversed. If the data bit is 1, the phase is reversed. An example of a D-BPSK signal can be seen in Figure 3.2 [6].

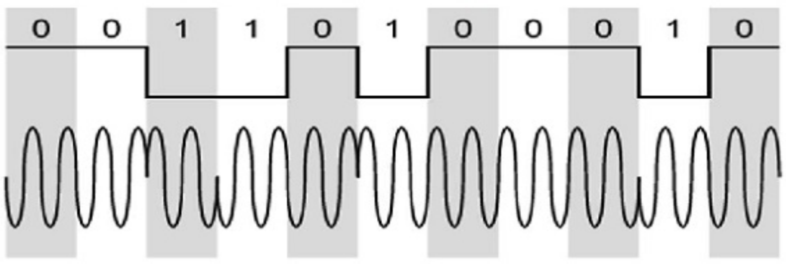


Figure 3.2: Example signal of Differential Binary Phase Shift Keying. Source: [6]

Before the transmission of the first bit, a radio may start with a ramp-up. The signal will slowly build up in strength until the fixed transmission power is reached. After the last bit, the transmission can end with a ramp-down. The ramp-up and ramp-down contain no phase shifts (no bits are sent) and each shall be less than the time to transmit two bits.

For the downlink, **Gaussian Frequency Shift Keying** is used as the modulation technique. GFSK is a special case of **Frequency Shift Keying (FSK)**. In FSK, the frequency of is signal is increased if a 1 is sent and decreased if a 0 is sent. An example of an FSK modulated signal is shown in Figure 3.3. GFSK is FSK with a Gaussian filter. This filter will smooth out the abrupt changes in frequency [7].

As in the uplink transmission, ramps are used at the start and end of the transmission. The difference is that in the downlink the maximal duration is limited to half of the time it takes to send a bit.

Uplink Frame format

An uplink message can consist of multiple frames. The format of an uplink 3D-UNB frame is shown in Figure 3.4. The message content consists of application data or control data. 3D-UNB is built

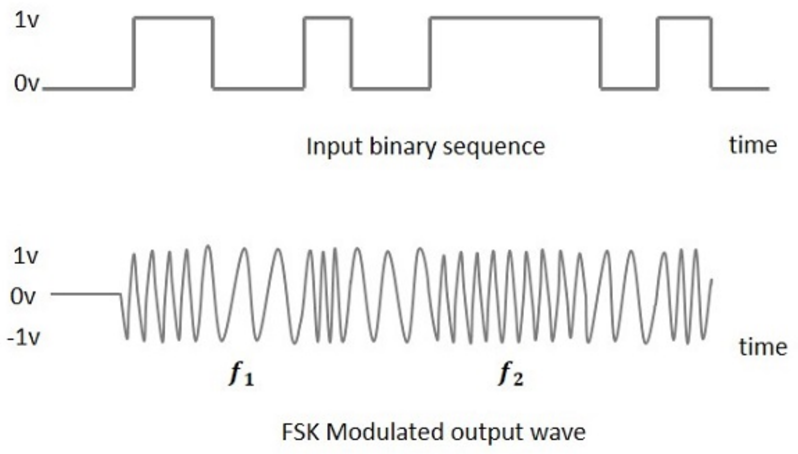


Figure 3.3: Example signal of Frequency Shift Keying. Source: [8]

in different layers. The physical layer is, among other things, responsible for error detection and correction, modulation and frequency selection. The Medium Access Control (MAC) layer is responsible for authentication, replay protection, integrity checks and uplink and downlink procedures. The last layer is called the applicative/control layer which can be seen as a mix of the network and application layer. This layer is responsible for the transportation and encryption of user -and control messages as will be seen in Section 3.1.3.

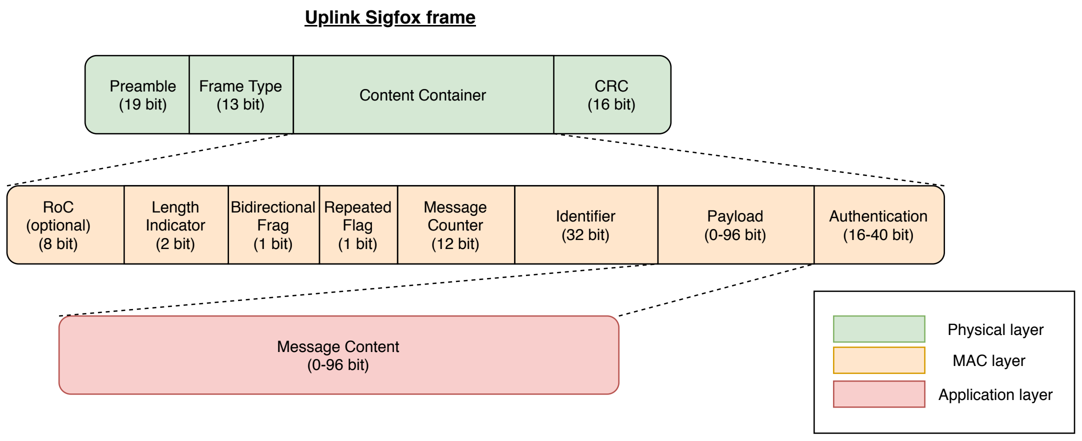


Figure 3.4: Sigfox uplink frame format.

The applicative/control layer contains the message content. The format of applicative data is decided by the user, while the format of control messages is specified by Sigfox. Sigfox defines two types of control messages: a keep-alive and a confirmation (of downlink message) message.

The MAC layer consists of 7-8 fields. The Rollover Counter (RoC) field is only added when payload encryption is active. The length indicator is used to specify the length of the authentication field and the payload. What length corresponds to what length indicator is specified in the specification. The bidirectional frame bit is set to 0 if no downlink message is expected as a response and set to 1 if the end-device expects a downlink response frame. The message counter starts at zero and is incremented with each uplink message. If a specified maximum value is reached, the counter will restart at 0. The counter is the same for all frames belonging to the same message and is used to prevent replay attacks. Two identical messages are not accepted by the receiver. The identifier contains the identifier of the end-device. The authentication field is based on all other fields of the MAC layer, which is explained in more detail in Section 3.1.3.

The physical layer consists of 4 fields. The first field is the preamble, which is a 19-bit field with a fixed value which is used for the synchronization between two devices. The Cycle Redundancy Check (CRC) field is used for the error detection of the container content. After calculating the CRC, the CRC and the container content are encoded using convolutional codes. Convolutional codes are used in a variety of wireless standards, their main advantage is the possibility to recover the most likely message for the encoded message. This method is a form of error correction [9]. The encoded result is placed over the content container field and CRC field. Finally, the frame type specifies three things: whether it is an application message or a control message, the size of the content container and the frame emission rank. The frame emission rank decides which convolutional code is used.

Uplink procedure

An uplink message is initiated by an end-point. Two different uplink procedures can be used: the single frame- and multiple frame procedure. The single-frame procedure sends a message consisting of one frame. In the multiple frame procedure, exactly three frames are sent with the same content only a different frame type and convolutional code.

For each uplink frame, a pseudo-random frequency is selected within the usable channel.

Downlink frame format

A downlink message can only be created in response to an uplink message and can only contain application data. The format of the downlink frame is shown in Figure 3.5. The format of the data is up to the user. The authentication field is based on the identifier of the end-device and the message counter of the uplink message. How this field is formed is explained in Section 3.1.3.

The CRC field is based on the container content and is used for error detection. ECC stands for Error Correction Code and is based on the container and the CRC field. As the name suggests, the ECC field is for error correction. Then the ECC, container and CRC field are passed through a whitening function which XORs the fields with a pseudo-random bitstream. The end-device identifier and message counter are used as a seed. Notice this whitening function does not provide encryption in any way. Its only purpose is to remove any correlation in the content. The frame type is fixed for downlink messages. The preamble in the downlink frame is a predefined bit stream of 91 bits which is almost 5 times as long in the uplink frame.

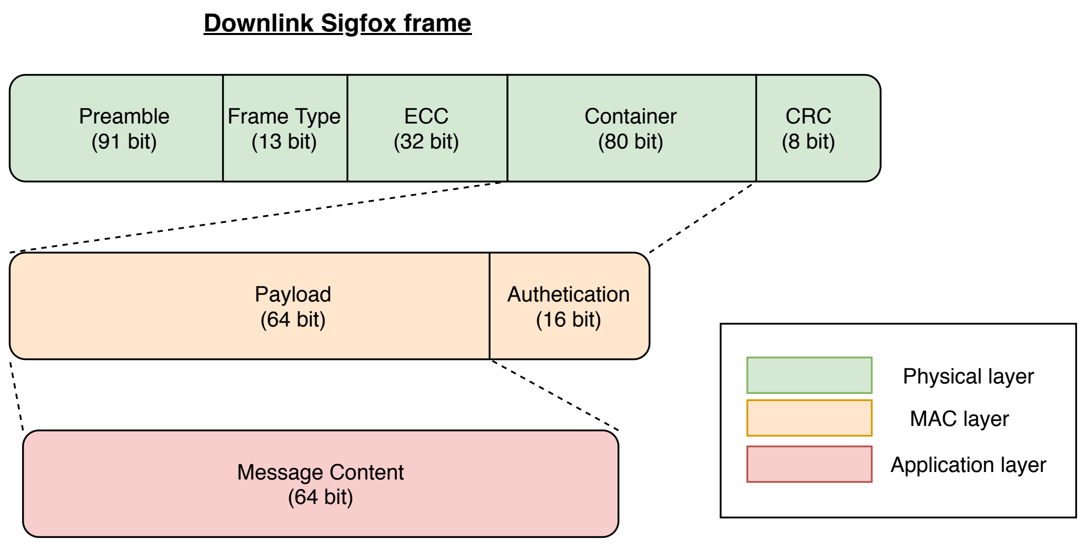


Figure 3.5: Sigfox downlink frame format.

Downlink procedure

The downlink procedure is initiated by an end-device by setting the bidirectional field to 1. The Sigfox network will send the downlink frame on a predefined frequency offset of the frequency used by the uplink frame. If in the uplink frequency a is used, in the downlink frequency $a + o$ will be used. Where *o* is the predefined offset which depends on the radio configuration being used. When the downlink message is received successfully, the end-device answers with a confirmation message.

3.1.2 Features

The throughput of the network is limited to 100 bps or 600 bps for the upload en 600 bps for the downlink. Also, there is a limit on the number of packets that can be sent/received per day. Uplink messages are limited to 12 bytes and a device cannot send more than 140 packets per day. Downlink messages can have a payload of up to 8 bytes and can be received 4 times a day. This makes the network ideal for low power sensors that don't need frequent updates [4, 10].

3.1.3 Security [5, 11, 12]

There are two important parts in the Sigfox network, see Figure 3.1. The part from the end-point to the base stations is wireless. The second part consists of the Sigfox core network and the application servers. The second part uses HTTPS and VPNs to secure all communication. For the first part, Sigfox provides the following security mechanisms: authentication, integrity, replay protection and optionally confidentiality. When a Sigfox certified device is made, it is provided with a unique symmetrical authentication key. This key is referred to as the Network Authentication Key (NAK) and never leaves the device. The NAK is used for authentication and message confidentiality.

In the uplink, frame authentication is provided by the authentication field. This is done by concatenating all other fields and encrypting the result using AES128-CBC with the NAK. Advanced Encryption Standard (AES) is a symmetric key encryption algorithm. The last two to five bytes of the encrypted result are copied into the authentication field. For the downlink message, frame authentication is similar to that of the uplink frame. The end-device identifier, the message counter and the downlink payload are concatenated in a specified order. Then AES128-CBC is used with the NAK to encrypt the concatenated blocks. The last two bytes of the encrypted result are used again for the authentication field.

Replay protection is already explained in Section 3.1.1.

Sigfox provides payload encryption as a service. Module makers can use a Sigfox provided library to encrypt the payload. The payload will be decrypted in the Sigfox core network before delivering it to the application provider. As encryption AES128-CTR is used, where the encryption key is derived from the NAK.

3.1.4 Known Vulnerabilities

The work by Coman et al. [13] shows several vulnerability and proof-of-concept attacks in LoRaWAN, Sigfox and NB-IoT. We will discuss the vulnerabilities found for Sigfox.

As already seen in Section 3.1.1, Sigfox uses a sequence number to protect frames against replay attacks. Messages arriving at the Sigfox core network with a sequence number lower than the last received message will be ignored. The problem is that this field is only 12 bits long which corresponds with 4096 possible sequence numbers. When sequence number 4095 is reached the next sequence number will be zero. Sending at a rate of 140 messages a day, the sequence number will be reset every 29 days. This allows an attacker to replay the previous 4096 packets forever [13].

The same issue can be used by attackers to perform a Denial of Service (DoS) attack on an end-device. When a frame is received by the Sigfox core network the sequence number cannot be increased more than approximately 420 (for a Platinum subscription) or otherwise the frame will be ignored. This number is defined by Sigfox. This means that by replaying 10 packets, the end-device will not be able to successfully send messages for a minimum of 30 days [13].

Another DoS attack can be performed by using the maximal sequence number gap of 420. If a minimum of 420 subsequent messages sent by a device are jammed, the device is DoS-ed for one rotation of the sequence number [13].

3.2 LoRa/LoRaWAN

Long Range (LoRa) and Long Range Wide Area Network (LoRaWAN) form another long-range low power protocol. LoRaWAN defines the data link layer and LoRa defines the physical layer. LoRa is a closed standard but LoRaWAN is an open protocol. In comparison to Sigfox, Lora allows for a little more throughput and therefore has a slightly shorter range [4].

LoRa is owned by the Semtech Corporation and LoRaWAN is a standard maintained by the LoRa Alliance. This is a non-profit organization supported by multiple organizations. Contrary to what the name LoRaWAN suggests, different data link layer protocols can be used on top of LoRa, two examples are Symphony Link and Haystack [4].

In Europe there are multiple networks, set up by KPN, Proximus, Orange . . . There is also an open network that is free to use, it is called 'The Things Network' [4].

3.2.1 LoRa

LoRa is the physical layer used in LoRaWAN. This layer is responsible for the modulation en demodulation of signals. LoRa operates in the free to use Industrial Scientific and Medical (ISM) band. This corresponds to the 868 MHz band in Europe, the 915 MHz band in the USA and the 433 MHz band in Asia [4].

Modulation

LoRa is based on **Chirp Spread Spectrum (CSS)**. CSS is known for its robustness against interference and its long range. CSS uses a windowed chirp which is a signal whose frequency changed linearly over time. The frequency can start from the lower frequency and stop at the higher frequency which is called an *up-chirp* because the frequency increases. The opposite is called a *down-chirp*, where the frequency starts at the higher frequency and ends at the lower frequency [4, 14]. An example chirp signal is shown in Figure 3.6. The first plot shows the data bits that are send, the second plot shows the effect on the frequency and the last plot shows the chirp signal itself. In this example, a 1 bit is represented by a decreasing frequency and an 0 bit is represented by an increasing frequency.

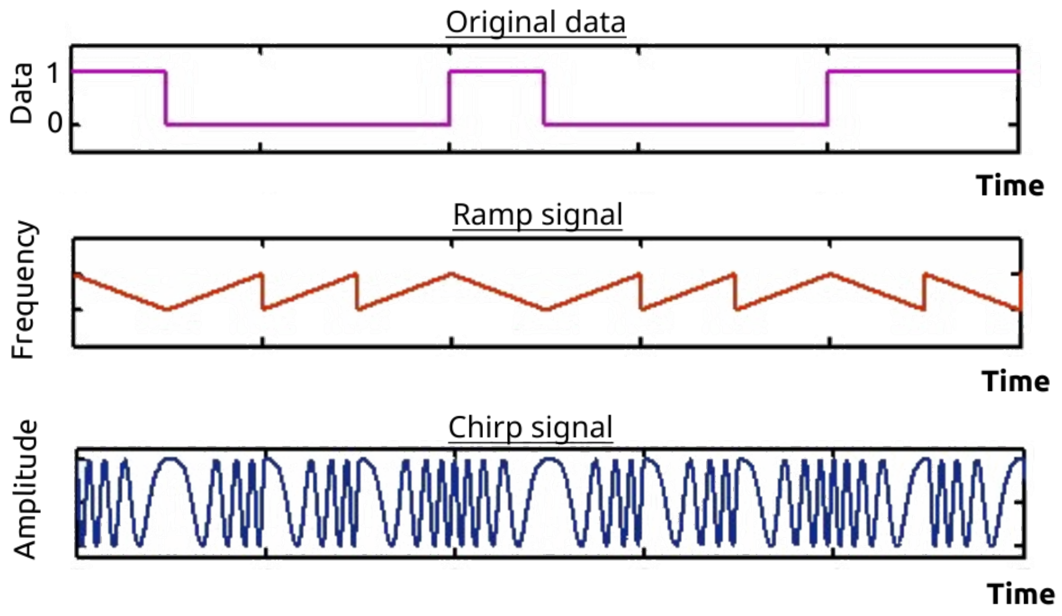


Figure 3.6: Example of a chirp signal. The uppermost plot shows the data that is being sent. The second plot shows the effect on the frequency of the chirp signal. When a 1 bit is sent, the frequency decreases and when a 0 bit is sent, the frequency increases. The last plot shows the chirp signal itself where the same effect on the frequency is visible. Source: [15]

The LoRa signal-modulation is defined by three parameters. But to define those, we need to understand following LoRa signal characteristics [16, 17]:

- *Chirp rate (R_c)*
  The chirp rate is the number of chirps transmitted per second, expressed in chirps-per-second (cps).
- *Symbol rate (R_s)*
  A symbol is formed by one or more chirps. The symbol rate is the number of symbols transmitted per second. One symbol can represent one or more bits.
- *Bit rate (R_b)*
  The number of bits transmitted per second in bits-per-second (bps).

Now we can define the three parameters of a LoRa modulated signal [16–18]:

- *Bandwidth (BW)*
  The width of the frequency band used to transmit signals, expressed in Hz. The more bandwidth is used to transmit a signal, the higher the bit rate.
- *Spreading factor (SF)*
  The spreading factor represents the number of symbols used in transmitting a piece of information. The SF ranges between 7 and 12. By increasing the spreading factor, the transmission time is increased too. The lower the spreading factor, the lower the transmission time or the higher the bit rate.
- *Coding rate*
  LoRa uses Forward Error Correction to recover bits. The Coding Rate defines the proportion of the bits that are used to carry information:

  $$\textit{portion of relevant bits} = CR = \frac{4}{4 + \textit{Coding rate}}$$

  The LoRa coding rate ranges between 1 and 4 which means CR ranges from 4/5 to 4/8. A higher coding rate means more recovery bits and less real information, in other words, a lower data rate but also a lower probability for errors.

Based on these parameters, it is possible to calculate the chirp rate, the symbol rate and the bit rate [17]:

- The chirp rate is defined equal to the bandwidth. If for example the bandwidth is 125 kHz, the chirp rate is 125 kcps.
- The symbol rate is defined as follows:

  $$R_s = \frac{BW}{2^{SF}}$$

  Thus a symbol is represented by 2^{SF} chirps.
- The bit rate is defined as follows:

  $$R_b = SF * \frac{1}{\frac{2^{SF}}{BW}}$$

  when using forward error correction we add the *portion of relevant bits*:

  $$R_b = SF * \frac{CR}{\frac{2^{SF}}{BW}}$$

Message Format

The format of a LoRa packet can be seen in Figure 3.7. The packets start with a predefined preamble to synchronize the receiver. The size can vary but the default size is 12 symbols [18].

There are two packet modes defined for LoRa: explicit header mode and implicit header mode. In **explicit header mode** the packet contains a header and a header Cycle Redundancy Check (CRC) for error correction. The header contains three chunks of information: the payload length, the coding rate and if the payload contains a CRC. In **implicit header mode** the packet does not contain a header to lower transmission times and save power. The information which is normally presented in the header should be known by the sender and receiver [18].

The payload can be up to 222 bytes. If the CRC is present, it is appended to the payload [4, 18].

The header and payload have different coding rates. When talking about the coding rate of a LoRa packet, people usually mean the payload coding rate. For the header, $CR = 4/8$ is used which means the number of information bits equals the number of error correction bits. For the payload, a different coding rate is used which is specified in the header [18].

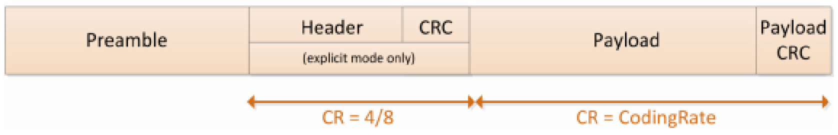


Figure 3.7: Lora packet format. Source: [18]

3.2.2 LoRaWAN

LoRaWAN is a datalink layer protocol that is built on top of LoRa. LoRaWAN is just like LoRa optimized for low power usage [4, 19].

An overview of a typical LoRaWAN network is shown in Figure 3.8. **End nodes** or **end-devices** are typically sensors with a limited energy capacity that send messages at a regular interval e.g. every hour. The end nodes form a star topology around the gateways. A LoRaWAN **gateway** is usually a device with a strong network connection like LTE or ethernet. The gateway consumes more energy than the end-devices and is usually plugged into the wall. LoRaWAN packages are used to send data from the end-devices to the gateway. The gateway forwards the message over the normal network connection to a central **network server**. It is the task of the network server to distribute the payload to the **application server** associated with the end node [19].

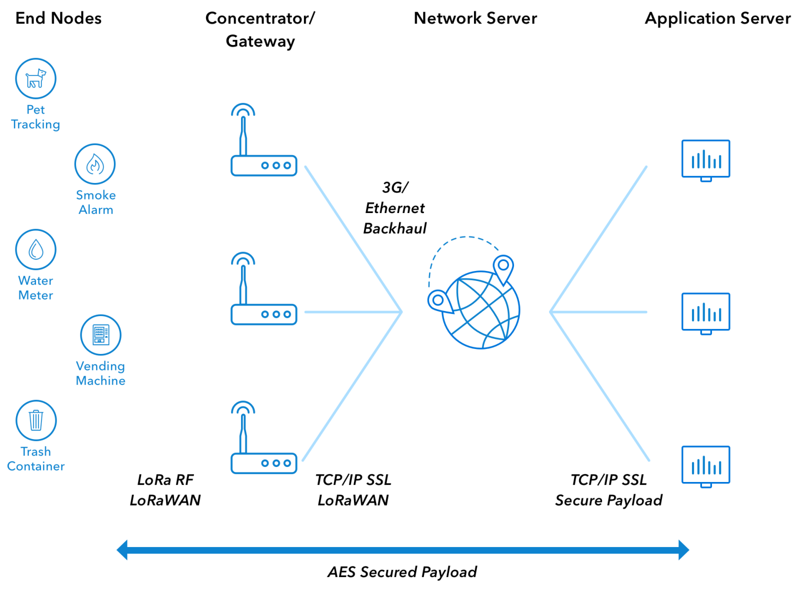


Figure 3.8: LoRaWAN architecture overview. Source: [20]

An **uplink message** is created by and end-device and is sent to the network server. The message can be received by multiple gateways, it is the task of the network server to detect duplicate messages. Downlink messages are sent from a network server to an end-device. This message passes a single gateway. The gateway that was the fastest in the uplink is typically used for the downlink message [19].

LoRaWAN Classes

LoRaWAN uses classes A, B and C to specify different end-device requirements. All LoRaWAN devices need to implement class A; class B and C are optional. Here is an overview of the LoRaWAN classes[19]:

- **Class A**
  Class A is the most power-efficient. Downlink messages can only be send after an uplink message. This allows the end-devices to enter deep sleep mode.
- **Class B**
  Class B allows end-devices to receive downlink messages at regular intervals. The end-devices open a receive window at scheduled times. To synchronize the end-device and gateway, the gateways sends beacons.
- **Class C**
  Class C devices have an always-open receive window. This allows for low latency communication but requires more power than class A or B.

For the rest of this section, we will focus on Class A devices since these techniques require the least amount of power and are therefore preferred in low power sensors.

Message Format

The format of a LoRaWAN message is shown in Figure 3.9. The entire message is placed in the LoRa payload. A LoRaWAN message is typically called the PHYPayload which is short for Physical payload. A LoRaWAN message contains a payload which is called the MAC payload [19].

There are 8 different LoRaWAN messages. The first 3 bits of the PHYPayload specify the message type. Three types are used to join, accept and rejoin a network. Four types are used for normal traffic: up or down and whether the message should be acknowledged or not. Messages that require an acknowledgment are called **confirmed data messages**. The last type specifies a proprietary message which can be used to implement nonstandard messages. The major version is used to specify the LoRaWAN standard [19]. The following bytes contain the MAC payload and the last 4 bytes contain the Message Integrity Code (MIC). This is used to check if the message is changed in any way during the transmission. The MIC is calculated on the header and payload [19].

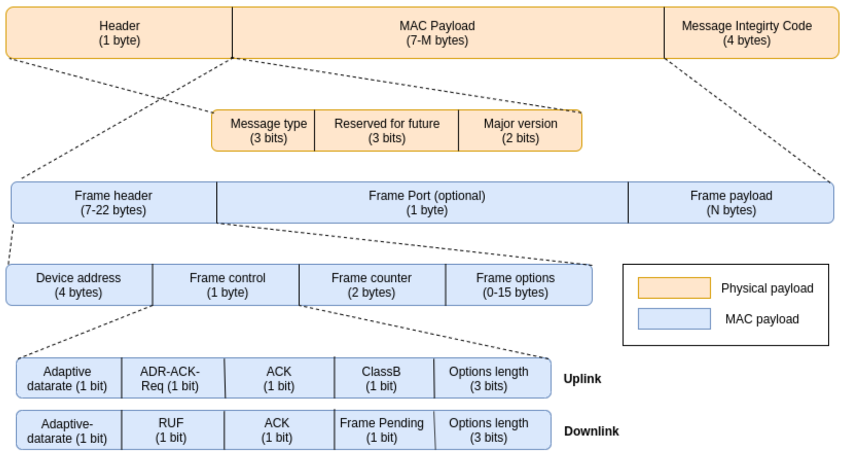


Figure 3.9: LoRaWAN message format.

The MAC payload consists of a frame header, an optional frame port and the frame payload. The frame header is 7 bytes long but can be extended by frame options to 22 bytes. The first 4 bytes specify the address of the end-device. For the next three bytes, one is used for the frame control and two are used for a frame counter [19].

The format of the frame control bits is different for an uplink and downlink message. The first bit in both messages is used to specify whether the sender can use an adaptive data rate. As seen in Section 3.2.2, an adaptive data rate is a mechanism to control the data rate and transmission power.

This is different for each individual end-device and is usually controlled by the network server using ADR command messages. If the ADR bit is set in a downlink message, the network server can send ADR commands. If the ADR bit is set in an uplink message, the end-device informs the network server whether it should receive ADR messages. The next bit in the uplink frame control is used to request a response for the network server, this way the end-device knows if the gateway is still able to receive uplink messages. The next bit in the frame control is used for acknowledgments. Conformed data messages require an acknowledgment. When a conformed message is received, the network server or end-device has to set the ACK bit in the next message. Acknowledgments correspond to the last message. The fourth bit in a downlink message is the frame pending bit, it is used to inform end-devices there is more data pending. By setting the bit, the network server asks the end-device to open a new receive window as soon as possible. The fourth bit in an uplink message informs the network server that the end-device is switching to class B [19].

The frame port is one byte long and is used to specify the payload type. If a message contains a payload, the frame port must be set. When the frame port is set to zero the payload contains a MAC message. MAC messages are used to send ADR messages, change the channel, request a status, rejoin, etc. MAC messages are not passed on to the application layer. The 223 other frame port values are application-specific and are passed to the application layer. The remaining frame port values are kept for tests and future standards [19].

Adaptive Data Rate

By varying the bandwidth and spreading factor, LoRa bit rates can range from 0.3 kbps to 50 kbps. To optimize the network capacity, the bit rate can be set for each end-device individually. This is called the **adaptive data rate**. For a short distance, a small spreading factor can be used because the signal strength is better. This means a higher data rate for short-distance transmissions. For transmissions over a longer distance, a larger spreading factor is used to enhance signal quality [4, 19].

3.2.3 Security

End-device IDs and Keys

Each end-device requires a set of keys and IDs to connect to a network. In most LoRaWAN devices the keys and IDs will be preinstalled on the device. The following keys and ID are required [19]:

- **Join EUI**
  A 64 bit unique global application identifier. In previous version of LoRaWAN this is called the Application EUI.
- **Device EUI**
  A 64 bit unique global device identifier that is hardcoded into the LoRa chip.
- **Network Key and Application Key**
  Are AES-128 bit keys that are used to derive session keys. Both keys are kept securely on the device.

All keys are needed to perform an Over-The-Air Activation (OTAA). After activation more information is stored on the end-devices [19]:

- **End-device address**
  This is a 32-bit address that identifies the end-device within the network and is allocated by the network server.
- **Forwarding network session integrity key**
  This is used to calculate the MIC for uplink messages. This key is derived from the network key after the join procedure.
- **Servicing network session integrity key**
  This is used to verify the MIC of downlink messages. This key is derived from the network key after the join procedure.
- **Network session encryption key**
  This key is used to encrypt and decrypt MAC commands and frame options. This key is derived from the network key after the join procedure.

- **Application session key**
  This key is used to encrypt and decrypt the frame payload of application messages. This key is derived from the application key after the join procedure.

Join Procedure

LoRaWAN defines two types of join procedures: Over-The-Air Activation (OTAA) and Activation by Personalization. Activation by Personalization directly ties an end-device to a network. The device address and session keys are stored on the device. If the device address and session keys are not stored on the device, an OTAA procedure is necessary to join the network [19].

The OTAA procedure proves that both the end-device and the LoRaWAN network have the same Application Key. OTAA is initiated by the end-device by sending a join or rejoin request. The request contains the Join EUI, the device EUI and a **Number - used Once (NONCE)** which is incremented with every request. The nonce in this case has the same function as a sequence number: prevent replay attacks. The network server will drop messages where the nonce is not incremented. The join request is not encrypted but is authenticated [19].

The network server will respond with a join- or rejoin-accept message if the end-device is permitted to join. If the end-device is not permitted to join, the network server does not respond. The accept message contains a server nonce, a network identifier, the end-device address and other specific parameters. Notice the server nonce is different from the nonce sent by the device. The server nonce is a device-specific counter value that is used to derive the session keys. This server nonce is incremented with every join-accept message [19].

Encryption

In a LoRaWAN message, the frame options and frame payload are encrypted using the Advanced Encryption Standard (AES) with a key length of 128 bits. The frame options are encrypted using the Network session encryption key. The frame payload is encrypted using the Application session key if the payload is application-specific. The frame payload is encrypted using the Network session key if the payload is a MAC command [19].

First, the options and payload part of the message is padded with zeros at the end to create a multiple of 16 bytes. Then the encryption algorithm defines some blocks A_i with $i \in [0, k]$ and $k = len(options + payload)/16$. In other words, k corresponds to the number of 16-byte blocks in the message. The blocks are 16 bytes long and have the following format [19]:

Size(bytes)	1	4	1	4	4	1	1
Content	1	0	Direction field (O or 1)	Device address	Frame counter	0	i

Block A_0 is used for the option field, if present and block $A_{0/1..k}$ is used for the payload. Every block A_i is encrypted using the corresponding key K [19]:

$$S_i = aes\_128(K, A_i) for i = 0..k$$

All blocks S_i are concatenated to S [19]. S has the same length as the frame options (if present) concatenated with the frame payload. Finally S is XORed with the options field (if present) and the payload. For decryption, S is created in exactly the same way but S is XORed with the encrypted option field and payload [19].

Because all the blocks A_i contain the device address, the message is signed. The frame counter prevents attackers from using a repay attack.

Message Integrity

The MIC is 4 bytes long and is calculated over the header and the MAC payload of the MAC message. First, a block of 16 bytes is created containing the device address, the frame counter, the length and some specific information which depends on whether it is an uplink or downlink frame [19].

The entire message is XORed with the block. The MIC is calculated using the AES-128-CMAC. For uplink messages, the Servicing network session integrity key is used. For downlink messages, a combination of the servicing and forwarding network session integrity key is used [19].

3.2.4 Known Vulnerabilities

Coman et al. [13] show several vulnerabilities for LoRaWAN. According to their work, a large number of vulnerabilities were fixed in version 1.1 including a vulnerability where the network server was able to read the messages and a vulnerability where a replay attack was possible for join requests. However, some vulnerabilities still apply in version 1.1. The first issue is that LoRaWAN considers the security protocols as being ideal, this is however not always the case. For example, LoraWAN 1.1 uses AES in ECB mode for its join and accept messages, however, ECB mode has a flaw where two identical messages produce the same encrypted payload.

A second possible vulnerability is that LoRaWAN assumes a trusted infrastructure. The network server for example can alter an encrypted message. A message is integrity-protected between the end-device and the network server and between the network server and the application server. The network server itself can however alter the message. Notice the message is still encrypted using the application key hence the content is not readable [13].

In the work of Butun et al. [21] another security analysis was performed on LoRaWAN version 1.1. They came up with several issues.

Using low-cost hardware, it is possible to jam a gateway or node which can be exploited in a DoS attack. When the attacks use a device that jams a wide frequency band, this attack is easy to detect. If the attacker however uses a small selected frequency, this attack is hard to detect [21].

Butun et al. also demonstrated another DoS attack by capturing a join request from a device whilst jamming it so it is not received by any gateway. When the join request is not answered, the end-device will increase the device nonce (see Section 3.2.3) and send a new join request. This second attempt is also jammed by the attacker. Next, the original join request is resent by the attacker and the join accept will be received by the end-device. The end-device supposes the join accept is based on the second join request and therefore uses the second device nonce to create the session keys (see Section 3.2.3). The result is that the sessions key generated by the end-device and network do not match which results in a DoS attack[21].

3.2.5 Experiment

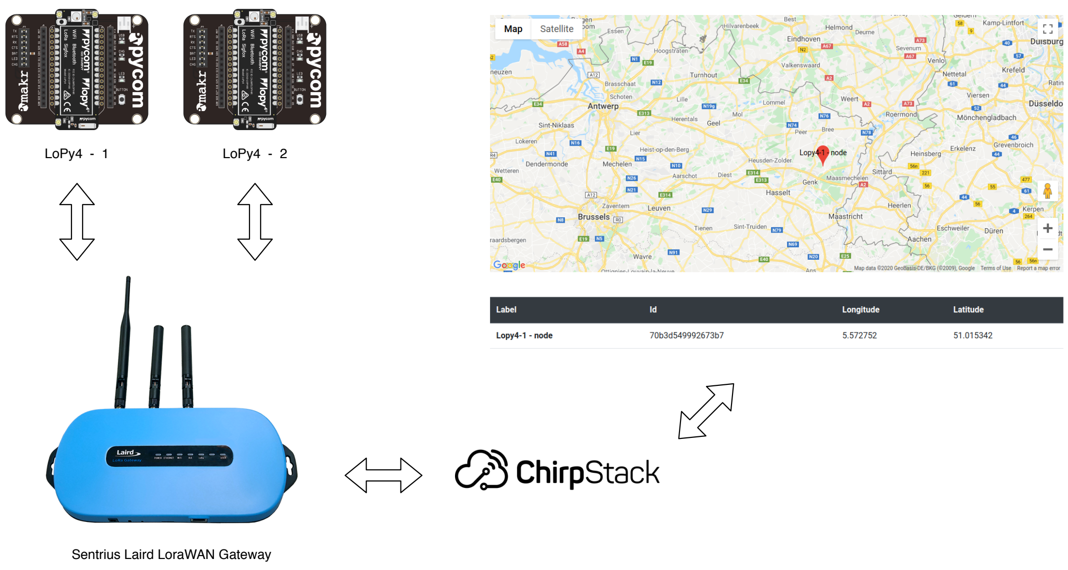
In this section, we set up a full stack LoraWAN solution. We create an application for tracking assets: sensors report their location using LoraWAN to the application server and the location of all assets is shown in a dashboard. For this experiment we use the following hardware:

- Laptop
  Runs the network server, the application server and the dashboard.
- 2x LoPy4 with pycom development board
  These devices send out a simulated location.
- Sentrius Laird LoraWAN Gateway

In this setup, we use the open-source LoraWAN network server and application server from ChirpStack [2] to manage gateways and devices. ChirpStack can be deployed on a large variety of devices and platforms but the easiest setup is through docker and docker-compose. A git repository is cloned which contains the configuration files and docker-compose file which can be used to start the full solution with a single docker-compose command. The next step it to add the gateway to the ChirpStack network server. Therefore the Laird gateway is configured to forward its network traffic to the ChirpStack network server and in the ChirpStack management panel the gateway is added. We then add a device profile in which the LoraWAN class is specified together with the LoraWAN version. Finally, we add an application in which end-devices can be added by entering the Device EUI. ChirpStack than generates the Application Key and the Application EUI which must be placed into the python code on the LoPy4 devices.

On the two LoPy4 devices we deployed a python script that connects to the network server using the provides key and publishes its location at a regular interval. Then we made a dashboard in which the live locations are visible. Finally, a callback to the dashboard server is added in the ChirpStack server to receive updates when new messages arrive. An overview of the setup is shown in Figure 3.10.

[2] `chirpstack.io`



Label	Id	Longitude	Latitude
Lopy4-1 - node	70b3d549992673b7	5.572752	51.015342

Figure 3.10: Architecture of the LoraWAN tracking experiment. LoPy4 devices provide location updated trough gateway, network server to the application server and dashboard.

The setup of this private LoRaWAN solution is relatively easy. The pycom LoPy4 devices have detailed documentation and tutorials explaining how to set up the hardware, connect to the hardware and connect to a LoRaWAN network. The same goes for the ChirpStack server, the documentation describes in great detail how to set up and deploy the solution. The most difficult part is to set up the gateway, application and devices in the ChirpStack server which is also explained in the ChirpStack documentation.

3.3 NB-IoT and LTE-M

NB-IoT and LTE-M are two other long-range protocols. They use the Global System for Mobile Communication (GSM) or Long Term Evolution (LTE) band in the licensed spectrum. They provide more throughput compared to LoRa and Sigfox, NB-IoT messages can have a payload of up to 1.6 kB with a throughput of 200 Kbps and their range, with 1 to 10 km, is slightly shorter than those of LoRa and Sigfox. LTE-M provides an even larger peak data rate of up to 1 Mbps while reducing the coverage. LTE-M is also known as eMTC (Enhanced Machine Type Communication), LTE CAT-M1 or LTE CAT-M2, the last two are specific versions [22]. In comparison to other LTE solutions, both NB-IoT and LTE-M focus on a massive amount of devices, a low device cost and a low power usage [4].

Both solutions can be implemented on an existing LTE network with a software update. The protocols are based on LTE with some features removed to lower energy usage. NB-IoT and LTE-M, in comparison to Sigfox and Lora, can be used in applications that require lower latency [10]. Several providers support NB-IoT and LTE-M in Europe including but not limited to Proximus, Orange and Telenet [23–25].

3.3.1 5G

5G or the fifth-generation mobile network is a communication standard that is in the final phase of completion at the time of writing and is designed to replace 4G-LTE. 5G consists of a set of different cellular services that focus on different use cases such as IoT, mobile and vehicular applications. The following requirements are set out for 5G [4]:

- 1 to 10 Gbps connections to UE.
- 100% coverage across the globe.
- 10 to 100x the number of devices supported in 4G.

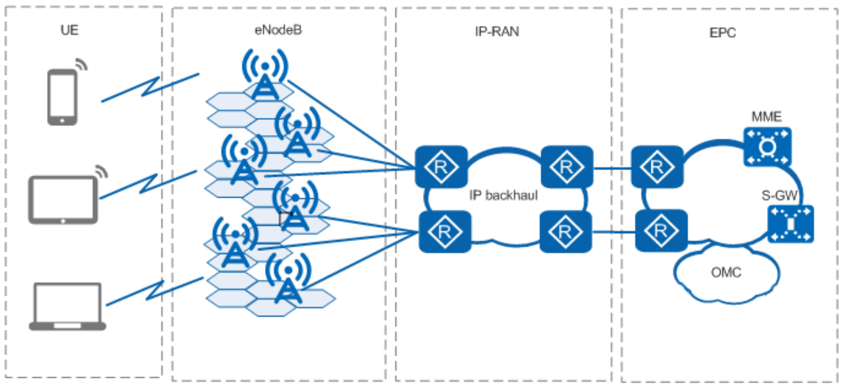


Figure 3.11: LTE network architecture. Source: [28]

- 99.999% availability.
- Less than 1 ms end-to-end round trip time.
- 90 % reduction in network energy usage in comparison to 4G.
- 10 years of battery life to IoT devices.

To provide the very high bandwidth, millimeter waves are used which are waves with a frequency of 24 to 100 GHz. However, with such high frequencies, the signal loses strength very quickly. Therefore antennas must be placed on various locations to provide a line of sight to the end-device. To provide support for more devices, Massive Multiple Input Multiple Output (MIMO) macrocells are used which provide cell towers with 256 to 1024 antennas. The best antenna is selected for communicating with a specific device, which is called beamforming [4].

While 5G aims for a very high bandwidth and an ultra-low latency it also has components for a massive IoT deployment and low power devices. The 3the Generation Partnership Program (3GPP) has indicated that both NB-IoT and LTE-M will be a part of the 5G standard and evolve with the specifications. Hence all NB-IoT and LTE-M networks deployed today are part of 5G [26].

3.3.2 Architecture

NB-IoT and LTE-M use the same infrastructure as existing LTE networks, the architecture is shown in Figure 3.11. The first column consists of User Equipment (UE), these are devices such as phones, laptops and IoT devices. The second column consists of the eNodeBs, which are the wireless base stations that handle the communication between the UE and the rest of the LTE network. Each eNodeB controls one cellular zone (or cell), in Figure 3.11 this zone is represented by a hexagon. The eNodeB is responsible for transmissions and congestion control. The next column consists of the IP-Radio Access Network (RAN) which is a full IP based network that connects the eNodeBs to the rest of the network. The last column consists of the Evolved Packet Core (EPC) which is the core of the LTE network. The EPC is built in two layers: the **user plane** passes user data and the **control plane** manages the network. The Mobility Managment Equipment (MME) is responsible for control plane traffic, it handles authentication, location and tracking. The Serving Gateway (S-GW) handles user data in the user plane, it can forward traffic back to the same network or to the internet. Traffic to the internet is passed to the Public Data Network Gateway (PGW). The Operation and Maintenance Center (OMC) is an administrator component that can manage the network in a centralized manner [4, 27].

To connect to an NB-IoT or LTE-M network, a **Subscriber Identity Module (SIM)** card must be used which runs a Java application that provides an interface to the network. Security-related information such as cryptographic keys are stored on the SIM card. It also stores the two identifiers: the Internation Mobile Equipment Identifier (IMEI) which identifies the device to the cellular network and the International Mobile subscriber Identifier (IMSI) which identifies the subscriber to the cellular network [29].

3.3.3 Layers

At the network and transport layer LTE uses the TCP/IP stack, however other layers are needed to transport the IP packets through the LTE network and control the UE. The physical layer is responsible for power management, modulation, searching cell signals, synchronization and cell measurement. The MAC layer multiplexes various packets in the same blocks, as we shall see in the next section. This channel is also responsible for channel scheduling and Quality of Service. The next layer is called the Radio Link Control (RLC) layer and is responsible for error correction, segmentation, duplicate detection and retransmission. The Packet Data Convergence Control Protocol (PDCP) is built on top of the RLC layer and performs header compression, packet reordering and packet routing through the LTE network. The security-related operations such as integrity and confidentiality protection are also performed in the PDCP layer [4, 29].

In case of a user plane packet, the IP protocol is fulfilled on top of PDCP. Control plane messages are transported using the Radio Resource Control (RRC) layer which is used to transport system information, set up connections, perform authentication, etc [29].

3.3.4 Physical and MAC layer

Just like the architecture, the physical layer of NB-IoT and LTE-M have a lot of similarities with LTE. NB-IoT requires however 180 kHz of bandwidth for both the uplink and downlink which is much smaller than LTE. This narrow band enables three possible deployment options [30]:

- **Standalone**
  A GSM channel can be replaced by an NB-IoT channel.
- **Guard band**
  NB-IoT can be deployed in the guard band. The guard band exists between two LTE channels.
- **In band**
  NB-IoT can be deployed inside a lager LTE channel. It has been proven that this will not compromise the LTE performance.

An overview of the options is shown in Figure 3.12.

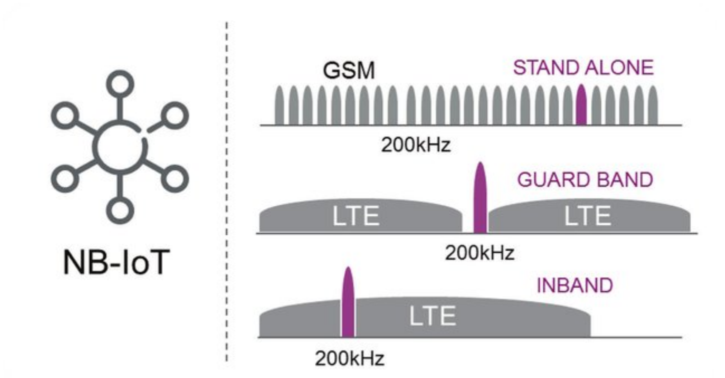


Figure 3.12: NB-IoT deployment options. Source: [31]

NB-IoT uses BPSK and QPSK as the modulation technique. BPSK is already explained in the Sigfox Section 3.1.1. In BPSK, two different phases are used to encode bits. In QPSK, 4 phases are used to encode two bits. Using QPSK more bits can be transferred in the same amount of time [32].

As NB-IoT, LTE-M also has a lot of similarities with LTE. The channel bandwidth is however reduced to 1.4 MHz which is wider compared to the 180 kHz of NB-IoT. Therefore LTE-M cannot be deployed in the guard band or in an LTE channel, instead, a single LTE channel is replayed with six LTE-M bands [4, 22].

In an LTE network, the frequency and time are divided into slots. One unit of transmission in LTE is called a Resource block which is shown in figure 3.13. The same structure is used in NB-IoT and LTE-M. A resource block consists of 84 resource elements (RE). An RE corresponds to a data symbol.

The number of bits in a data symbol depends on the modulation technique being used. In QPSK, 2 bits are encoded in an RE, in BPSK 1 bit is encoded in an RE. A resource block is divided into 12 subcarriers which are 15 kHz in width and have no spacing between them. In one time slot, 7 symbols are sent in one subcarrier during a period of 0.5 ms. In LTE, the whole resource block is allocated to a single UE but in NB-IoT a single RE can be allocated to support more devices [32].

In NB-IoT, the decision can be made to reduce the subcarrier width of the uplink connection to 3.75 kHz which corresponds to 48 subcarriers. The slot duration is in this case increased to 2 ms [32].

The coverage of NB-IoT has increased 20 bB over LTE networks which allows for better coverage in basements, tunnels and rural areas. LTE-M however does not have this extended range [4].

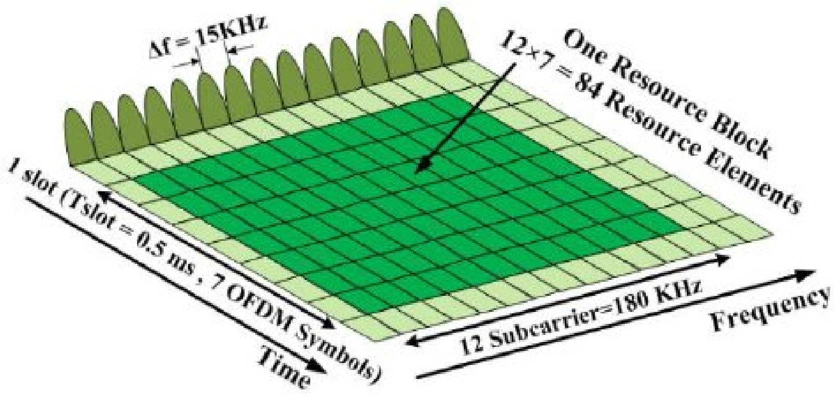

Δf = 15KHz
12×7 = 84 Resource Elements
One Resource Block
1 slot (Tslot = 0.5 ms , 7 OFDM Symbols)
Time
12 Subcarrier=180 KHz
Frequency

Figure 3.13: Long Term Evolution (LTE) resource block with 84 resource elements. Source: [33]

Another difference between NB-IoT, LTE-M and LTE is the ability to disconnect. A normal LTE device is not able to disconnect even when inactive or idle. A possibility would be to disconnect and reconnect but this would take about 15 to 30 seconds and consumes a lot of power. Power Saving Mode (PSM), which is used in NB-IoT and LTE-M, allows the device to enter deep sleep. At an interval specified in the connection setup, the device will wake up and listen for incoming messages which is called paging. The maximum time allowed in deep sleep is 12.1 days, but this varies depending on the network [4].

Due to the simplicity of NB-IoT, there is no mobility support. An end-device must remain associated with a single eNodeB, if the end-device moves to another cell, it has to reassociate which consumes a lot of energy. NB-IoT is therefore built for a more stationary deployment scenario. However, LTE-M does have mobility support and therefore can be used in nonstationary deployments [4].

3.3.5 Network and Transport Layer

In the user plane, the IP protocol is used to route the internet when the packet leaves the NB-IoT or LTE-M network through the PGW. This is a big difference between NB-IoT or LTE-M and protocols such as Sigfox and LoRaWAN where the end-device cannot use the IP/TCP stack to send and receive data. The PGW will provide an IP address to the UE devices which in most cases will be a private IP address. The PGW uses Network Address Translation (NAT) to convert the local IP address to a public IP address to route packages on the internet [4].

In the control plane, another stack is used to manage the NB-IoT and LTE-M networks and route user data packets. The Radio Resource Control (RRC) is located in the network layer and manages security keys and configurations. The Non-Access Stratum (NAS) is located in the transport layer and manages sessions between UE and MME. In LTE this layer also manages mobility between eNodeBs [4].

3.3.6 Security

Since LTE-M and NB-IoT are based on LTE, both share the same security features as LTE. As the network uses a dedicated spectrum band, interference from other radio technologies is kept to a minimum which benefits the security of NB-IoT and LTE-M [34]. The most important security mechanism is the SIM card which contains a pre-shared master key. The **master key** is stored in the memory of the

SIM card and within the core network and it never leaves those locations. All other keys are derived from the master key. Security-related operations and subscriber authentication are performed by the SIM card itself. The SIM card can only be accessed through a simple API which limits the control of an attacker. To prevent the SIM card from being inserted into another device, a PIN code is used to unlock the card after the boot [29].

In the primary authentication mechanism, both the SIM card and the core network prove that they both know the master key. The UE starts by sending its identifier to the MME which will respond with an authentication vector and Nonce. The UE calculates a result based on the authentication vector, Nonce and sequence number and sends that back to the MME. The MME calculates the same result and compares the two to validate if the UE has the correct key [29].

After the authentication mechanism is finished successfully, other cryptographic keys are generated. From the master key a Cipher key and an Integrity key is generated which is used to generate confidentiality and integrity keys for user plane - and control plane messages. Among other security mechanisms, AES in CTR mode can be used to provide confidentiality at the PDCP layer and AES-CMAC can be used for integrity protection. What security mechanisms are used, is decided by the network operator [29].

3.3.7 Known Vulnerabilities

Because NB-IoT and LTE-M use the IP to send uplink data and receive downlink data, classic attacks over IP are possible such as port scanning, IP spoofing and DNS spoofing. This in contrast to Sigfox and LoRaWAN who have their network and packet formats. However, NB-IoT and LTE-M are based on LTE networks which are already used for a long time on very large scales, hence the infrastructure has become more robust and secure throughout the years [13].

Coman et al. [13] propose several attacks. The first attack is called battery exhaustion where an attacker performs a DoS attack on an end-device. An attacker could force an end-device to send data and hence prevent the device from entering deep sleep. The simplest method is sending ping requests. However, this attack is questionable since NAT is used in the LTE core.

A second attack proposed by Coman et al. [13] is to use an end-device to enter the private network of the network provider. The first step is to have an NB-IoT end-device which can be done by simply buying a device, by stealing one or by gaining remote access. From there on, the private network is accessible through the end-device. A ping scan was performed by Coman et al. and two other NB-IoT devices where identified. From there on an attacker could scan ports on a device and find vulnerable services. However, the results of this attack are questionable since only two other devices where found. We conducted the same experiment in an LTE network and no other device in the 10.X.X.0/24 network were found with a ping scan.

In Cichonski et al. [29] the LTE security is analyzed and some weaknesses are explained with there possible mitigation. Some messages are allowed to be sent without security protection: identity request, authentication request, authentication reject, attach reject, detach accept, tracking area update reject and service reject which are trusted by the UE. When a rogue base station sends an attach reject to a UE, it is trusted by the UE and the UE will no longer attempt to attach to the network. This causes a DoS until the UE reboots. The same messages can be used to track UE and perform a downgrading attack to a less secure GSM connection [29].

3.4 Zigbee

Zigbee is a low power, short-range IoT protocol. Zigbee is an open standard and operates in the unlicensed spectrum (868 MHz and 2.4 GHz) [4, 35]. But to ensure interoperability, brands can become a member of The Zigbee Alliance and certify their products [36].

Today, Zigbee is used in many smart devices. These include but are not limited to light control, temperature control and door locks [37].

3.4.1 IEEE 802.15.4 [4, 38]

Zigbee uses the IEEE 802.15.4 protocol as its physical layer and MAC layer. Zigbee provides the network and application layer. 802.15.4 is a standard for wireless personal area networks. It is a low cost and low power protocol.

Modulation

Depending on the selected frequency, Zigbee utilizes one or 16 channels for the 868.3 MHz band and 2.4 GHz band respectively. In the 868.3 MHz band, Zigbee uses one channel and in the 2.4 GHz band, Zigbee uses 16 channels. For modulation, Offset Quadrature Phase-shift keying (O-QPSK) is mostly used. To share the frequency with other technologies, 802.15.4 uses **Carrier Sense Multiple Access Collision Avoidance (CSMA/CA)** where a sender will listen to the channel before transmitting and if the channel is busy, the sender will backup of a random period of time.

Device Types

802.15.4 used two types of devices:

- **Fully functional device (FFD)**
  This device is always on and communicates with multiple devices. Since this device is always on, it can route messages. If for example device A wants to send a message to C but both can only reach device B, device B can forward the message from A to B.
- **Reduced function device (RFD)**
  These devices can sleep and can only talk to a parent. Due to the limited functionality, the devices can be battery powered.

The task of the **Personal Area Network (PAN) coordinator** is to manage the PAN and store node information. The PAN coordinator is constantly sending and receiving, therefore it usually has a dedicated power line. The PAN coordinator is always an FFD. IEEE 802.15.4 supports multiple network topologies: star network, tree network and mesh network. In a star network, all devices are connected to the PAN coordinator. This network setup has a low complexity but has a limited range. In a tree network, all devices have a single parent device and the final parent device is the PAN coordinator. This setup has a wider range but all traffic still needs to pass the PAN coordinator which creates a bottleneck. Typically Zigbee uses the mesh network topology. In a mesh network, devices are connected to all reachable devices. A mesh network has the advantage it can reach further and has multiple paths to the destination.

Packet Format

The packet format of 802.15.4 is shown in Figure 3.14. The frame control field is used to specify the frame type, enable security, specify if the sender has more data, whether acknowledgments are required and what type is used for the addresses. The sequence number specifies the sequence identifier for the frame. The address info can contain the source and destination address and the PAN ID. The bits after the payload are used for a frame check sequence. This field is calculated over the MAC header and payload [39].

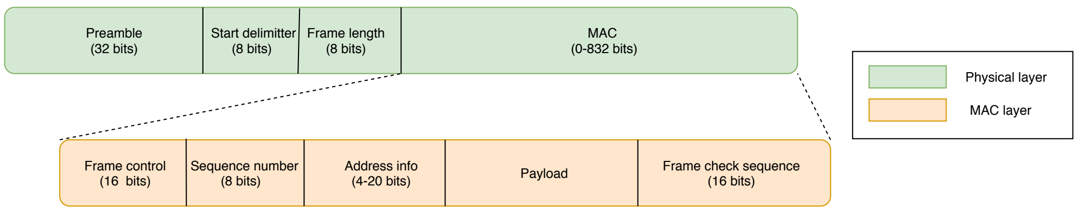


Figure 3.14: IEEE 802.15.4 packet format of PHY and MAC layer.

3.4.2 Zigbee

Zigbee is a standard located in the transport and network layer. Zigbee is used to manage devices, network structure and security. A Zigbee network consists of three components [4, 38]:

- **Zigbee Controller (ZC)**
  This Zigbee controller fulfills the role of the PAN coordinator. This device is used to set up the network. After the network is created this device is used as a ZR.
- **Zigbee Router (ZR)**
  The Zigbee router is an FFD that does the routing of messages. The Zigbee router can also assign addresses and add or remove devices from the network. The device can be integrated with the Zigbee controller, but it is also possible to use dedicated Zigbee routers to offload the Zigbee controller. It is not the case that Zigbee uses dedicated routers, a Zigbee router can be a sensor or actuator.
- **Zigbee End Device (ZED)**
  An end-device has no routing logic, it will relay any message that is not destined for that device. An end-device can only communicate with the coordinator. Both an FFD or an RFD can be used as an end-device.

An example of a Zigbee mesh network is shown in Figure 3.15.

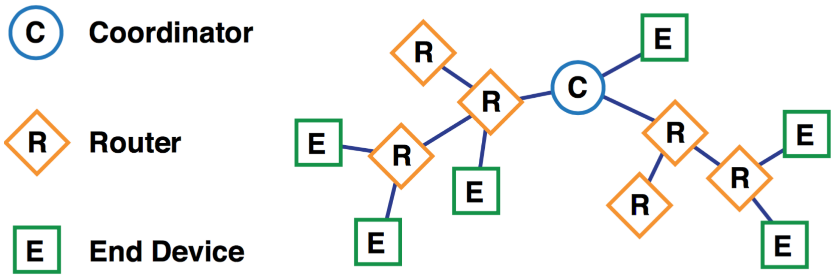


Figure 3.15: An example Zigbee mesh network. Source: [40]

Addressing

Every IEEE 802.15.4 device has a globally unique 64-bit address. The address is assigned by the manufacture and cannot be changed. When a device joins a Zigbee network, it receives a unique 16-bit local address which is called a **network address**. It is used to route packages within a network. The shorter address is used to save energy. If a device leaves the network, the network address can be reassigned. The coordinator in the Zigbee network has device address 0 [4, 41].

Network Layer

The first layer on top of 802.15.4 is the network layer which performs device management and route discovery. An overview of the Zigbee network layer packet format is shown in Figure 3.16. The first 16 bits of the network layer form the Frame control field. This field contains information such as the protocol version, whether security is used or not, whether it is a multicast message or not and whether the IEEE 802.15.4 addresses are used in the header. The destination address field contains the 16-bit network address of the destination. In case of a multicast message, a group ID is used to specify the destination multicast group. The source address contains the 16-bit network address of the sender. The radius can be compared to a time to live value, it is a 8-bit value that is decreased by each passed device. When the value becomes zero the packet is dropped. The sequence number is a value which is increased with every transmitted frame. After the radius the full destination and source IEEE 802.15.4 address is optionally specified [41].

The network layer is responsible for routing messages, therefore it manages two tables: a neighbor table and a routing table. The neighbor table consists of the directly reachable nodes. The routing table has entries for other nodes in the network. A routing entry consists of a destination node together with the next node on the shortest path to that destination. If a node receives a message that is not destined for itself, it will look up the entry of the destination in the routing table and forward the message to the next hop. If no entry exists for the destination, a route will be created by a process called **route discovery**. First, the node (the source) will broadcast a route discovery to all nodes in the network. Only the destination node will answer the route discovery with a reply, which is sent back to the source node. All nodes along the path of the reply will add a routing entry for the destination node. When

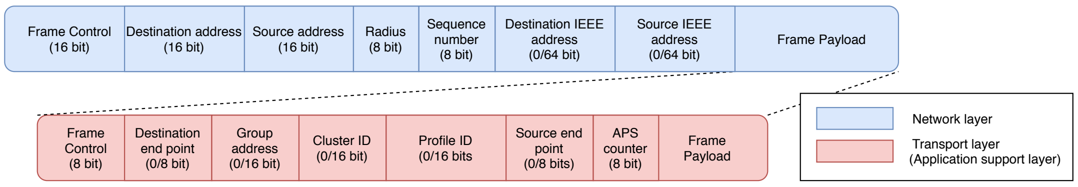


Figure 3.16: Zigbee frame format for network and transport layer. The Zigbee transport layer is also called the Application support layer.

the source node receives the reply it also adds the entry in its routing table and sends the message to the next hop. The nodes along the path to the destination will already have the routing entry in their table [42].

Transport Layer

The transport layer in Zigbee is called the application support layer. Applications in Zigbee are modeled by an application object, such as a light switch or an LED light. Each application object is identified by an endpoint address which is an addressable component in a device. A light will typically contain a single application object but a controller may contain an application object to control the lights and another to control the heating system. In Zigbee, a maximum of 254 different application objects can be registered. The first endpoint is reserved for the Zigbee Device Object (ZOD) which is used for device management including key management and device roles. The last endpoint is used for broadcasting data to all application objects and 241-254 are reserved by Zigbee for Zigbee specific functions and future use [41].

Communication between endpoints use data formats which are called clusters. Clusters are identified by a cluster-ID. Each message sent between devices is defined in form of a cluster. A cluster consists of a set mandatory and optional attributes, a command and a description. As an example, we take the Temperature Measurement Sensor cluster which is a cluster defined by Zigbee itself. This cluster supports the following attributes: measured value, minimum, maximum, tolerance and the cluster revision. It specifies that the cluster server should be a temperature sensor and the cluster client should be a device that wants temperature data. Also, information about the format of the data is included [41, 43].

Application profiles are used to describe the messages and actions between devices and are run by the application object. These profiles are necessary to create interoperable applications on separate devices. Application profiles use clusters to describe message formats. An application profile consists of the following specifics: a set of devices with a functional description for each device, a set of clusters and to which device they belong [41].

The general packet format of the transport layer is shown in Figure 3.16. The frame control field contains information about the message type, whether the package is unicasted or broadcasted, whether an ACK is expected and whether security is used. The destination endpoint field specifies the endpoint of the final receiver. If the message is sent to a group, the group ID field is used instead of the destination endpoint field. The next field is the cluster identifier which is used for the interpretation of messages at the receiver. The profile identifier is used to filter messages at the receiver side. The source endpoint field specifies the endpoint that sent the message on the sender device. The APS counter is a value incremented for every new transmission and is used to detect duplicate packages [41].

There are several possible transport layer message that can be sent including read attribute request and response, write attribute request and response, messages to configure being notified when a value reaches a specified interval and discover request- and response messages [43].

3.4.3 Security [4, 44]

Zigbee is a protocol that is designed with security in mind. However, some security protocols are poorly implemented to keep a low energy usage and keep devices highly compatible. Zigbee uses three security mechanisms:

- 128 bit AES encryption.

- Frame counters to prevent replay attacks.
- Frequency switching to prevent jamming attacks.

In a Zigbee network, the **Trust Center (TC)** is responsible for the distribution of keys that enable end-to-end security between all devices. The TC is usually the Zigbee controller.

Security Models

Zigbee supports two security models. In a **centralized security network** there is a single trust center (usually the controller). In a **distributed security network** there is no central node or trust center. In this model, a single network key is used to encrypt all messages. When a new device joins the network, the network key is sent to the new device. All routers and end-devices must have a link key to encrypt the network key. In the centralized security model, an out-of-band method can be used to initialize the link key. In the distributed setup the key is known to all devices. The centralized security model is therefore more secure than the distributed security model.

We will describe the centralized model in more detail in the following sections.

Keys

Zigbee uses security at two different layers. For the network layer a 128-bit **network key** key is used to encrypt communication between all nodes. This key is randomly generated by the trust center before any node joins the network and is shared by all devices in the network.

For the transport layer (application support), multiple link keys are used. A **link key** is a 128 bit key shared by two devices. The first two link keys are used to set up network-level encryption:

- **Global link key**
  This key is pre-configured on all devices. This key is the same for all nodes in the network. It can be the Zigbee-defined key or a key used only by a specific manufacturer.
- **Unique link key**
  This key is also pre-configured or pre-programmed on a device but is different for each device. In Zigbee 3.0, this comes in the form of an install code. Older Zigbee devices usually only have the global link key.

Once the network key is exchanged, a **trust center link key** is created. This key is used the encrypt traffic between the trust center and a unique node. The trust center link key is created in the trust center and sent to the new node. The key is therefore encrypted using the network key and, if available, the unique link key.

To secure communication between two nodes in the network a **link key** can be used. Four possible keys exist: the global or unique link key, a trust center link key or an application link key. We already discussed the first three keys. The application link key is generated by the trust center and sent to the two nodes individually. This exchange is encrypted using the network key and the trust center link key.

The network key is updated periodically and send to the nodes encrypted with the trust center link key.

Encryption [45]

Zigbee uses 128-bit AES for encryption and CBC-MAC for authentication. This combination is called **CCM***. CBC-MAC produces a tag that is appended to the message. It is comparable to a hash of the messages with a key. Zigbee assumes an **open trust** model. Which means that different layers in the stack trust each other. There is only cryptographic protection between devices and not between layers. Different layers can use the same key. The MAC -, network - and application support layer encrypt their payload using the keys provided by the upper layers.

The network layer gives the network key to the MAC layer to encrypt the MAC payload. MAC layer encryption is shown in Figure 3.17. In the network layer, this key is also set in the upper layers. When available, a link key will be used to encrypt the network layer payload. Network layer encryption is shown in Figure 3.18. The application support sublayer is responsible for the encryption of the application layer. The network key or link key is used for encryption. The application support layer encryption is shown in Figure 3.19.

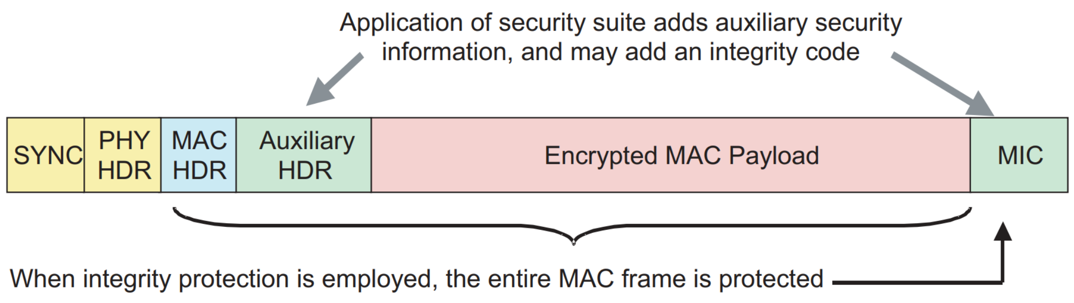

Application of security suite adds auxiliary security information, and may add an integrity code
SYNC
PHY HDR
MAC HDR
Auxiliary HDR
Encrypted MAC Payload
MIC
When integrity protection is employed, the entire MAC frame is protected

Figure 3.17: MAC layer encryption of the MAC payload. The network key is used which is passed down from the network layer. Source: [45]

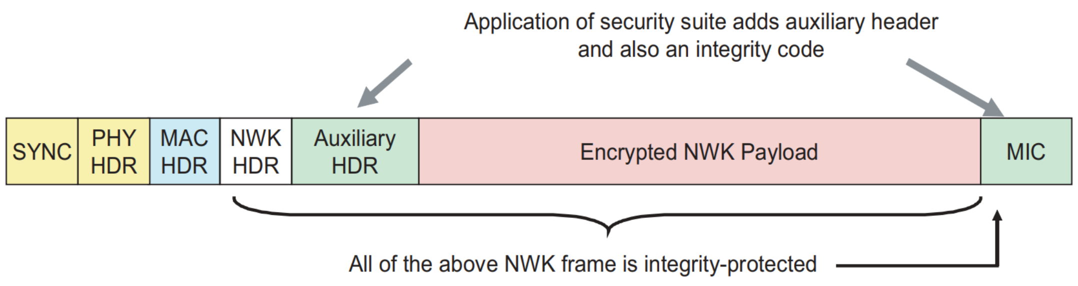

Application of security suite adds auxiliary header and also an integrity code
SYNC
PHY HDR
MAC HDR
NWK HDR
Auxiliary HDR
Encrypted NWK Payload
MIC
All of the above NWK frame is integrity-protected

Figure 3.18: Network layer encryption of the network payload. If available an appropriate link key is used. Source: [45]

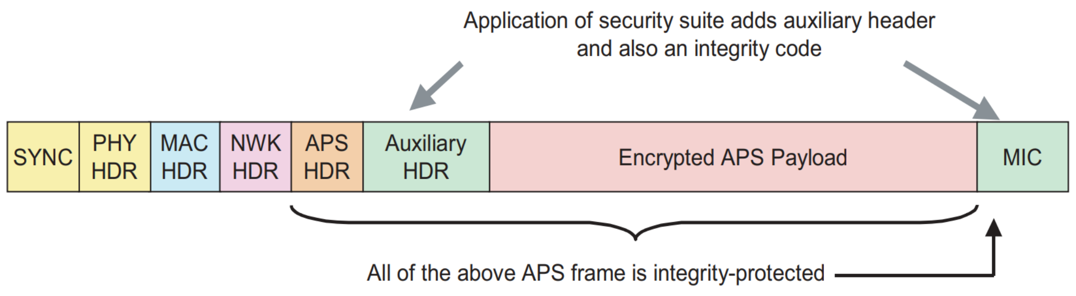

Application of security suite adds auxiliary header and also an integrity code
SYNC
PHY HDR
MAC HDR
NWK HDR
APS HDR
Auxiliary HDR
Encrypted APS Payload
MIC
All of the above APS frame is integrity-protected

Figure 3.19: Application support layer encryption of the payload. The network of link key is used. Source: [45]

3.4.4 Known Vulnerabilities

In the work of Vidgren et al. [46] an overview of existing attacks is given including two novel attacks against the Zigbee security: the Zigbee end-device sabotage attack and the Zigbee network key sniffing attack.

The end-device sabotage attack is a DoS attack where end-devices consume more power than needed until they run out of battery power. The attack impersonates a Zigbee router or Zigbee controller in the network. A ZED will poll there parent at regular intervals to check if data is received. The attacker in this case sends replies to all polls containing random data. Because there is data the device will stay powered on and send a new poll request every 100 ms. Since the device is powered on at all times it consumes a lot of power and this will eventually cause power failures [46].

The network key sniffing attack described in Vidgren et al. [46] does not apply anymore to the latest Zigbee 3.0 standard. In the attack, the network key is intercepted when it is sent unencrypted from the trust center to the joining device. As seen in Section 3.4.3 the initial network key exchange is now encrypted using a link key. If however the global link key is used, the attack still applies because the global link key is public. Ones the network key is known by the attacker, the application keys can also be intercepted.

The vulnerability where a DoS attack on an end-device is performed by sending fake messages with a well-increased sequence number, as already seen in Section 3.1.4 and Section 3.2.4, also applies to Zigbee [46].

KillerBee is a python software suite that uses the AVR RZ Raven USB Stick[3] with custom killerbee firmware to sniff and inject packages in a Zigbee network. This is an open-source library and is often used to sniff and perform attacks on the Zigbee network [46].

3.5 Z-Wave

Z-Wave is another short-range protocol that is used primarily for home automation. The Z-Wave Alliance is responsible for the standard. They claim there are over 3000 interoperable Z-Wave products and over 94 million are sold since 2001. Like Zigbee, Z-Wave uses a mesh network to control all devices. In Europe, Z-Wave operates in the unlicensed spectrum at 868.4 and 869.85 MHz [4, 47].

The physical - and MAC layer of Z-Wave are based on ITU G.9959. The network - and application layer are proprietary. In the work of Badenhop et al. [48] a portion of the routing protocol is reverse engineered and a part of the application layer is revealed in OpenZwave, which implements a Z-Wave controller in software. We use this work to explain the inner workings of the protocol [48].

3.5.1 Device Types

In the Z-Wave protocol, two main devices can be distinguished: controller devices and slave nodes. A controller device performs routing and can send commands to other nodes. A slave device executes commands from controller devices and is not able to communicate with other slaves unless otherwise instructed by a controller [4].

The controller that starts the network becomes the **primary controller**. It controls the network topology and the inclusion or exclusion of nodes. Other nodes are called **secondary controllers**. Both type of controllers have a full routing table of the network. Controllers can also be classified according to movement. A static controller's location must not be changed and is always on. An example of a static controller is a device plugged into the wall. A portable controller is not fixed in location and its location has to be estimated to calculate the best route through the network. An example of a portable controller could be a remote control device. The primary controller can also assign specific tasks to other controllers. A bridge controller for example is a static controller that can act as a gateway to other networks and an installer controller is a portable controller that helps in network quality testing and advanced network management [4].

Slave devices can be upgraded to a routing salve or an enhanced slave. A **routing slave** stores some routing information and is therefore able to send messages to other nodes. An **enhanced slave** also stores some routing information, has a clock and has memory to store application data [4].

3.5.2 Addressing

Z-Wave uses a 32-bit Home ID to distinguish networks from each other. A home ID is pre-programmed in all controller devices. The initial controller uses its ID as the home ID. When a new slave or controller device joins the network, the primary controller provides the new device with the home ID [4, 49].

An 8-bit node ID is used to address unique nodes within the network. In Z-Wave the number of nodes is limited to 232 due to some reserved addresses. Nodes receive the node ID from the primary controller when they join the network [4, 49].

3.5.3 Protocol Stack

Z-Wave is built in five layers. The lowest layer is the physical layer which manages signal modulation, channel assignments and preamble detection. The MAC layer is built on top of the physical layer and manages the Home ID and Node ID. The MAC layer is also responsible for collision avoidance and

[3]`https://www.element14.com/community/docs/DOC-67532/l/avr-rz-usb-stick-module`

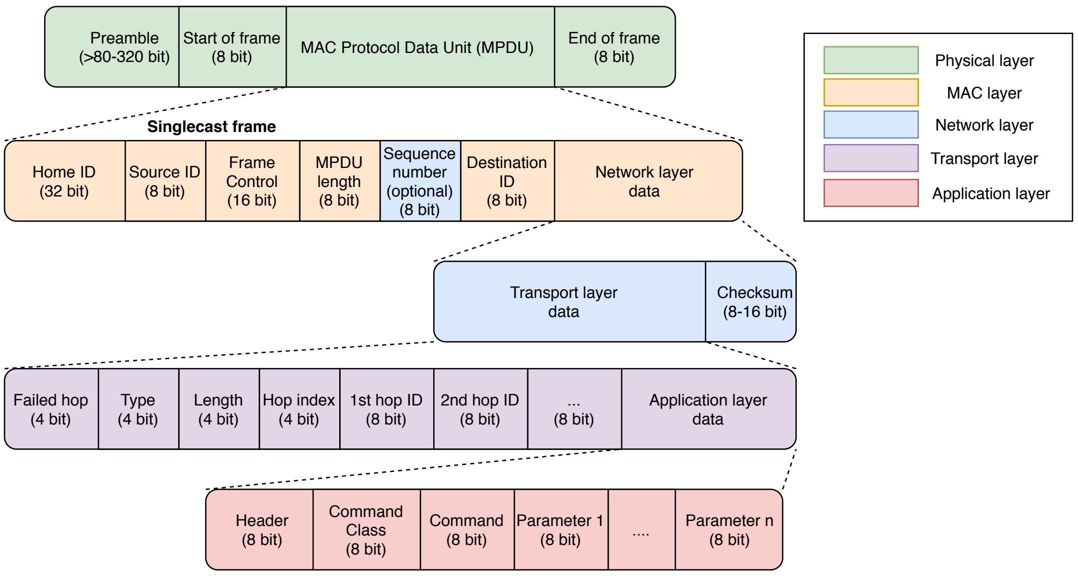


Figure 3.20: Z-Wave packet format. The physical and MAC layer are defined by the ITU G.9959 protocol [49]. Layers up from the network layer are defined by Z-Wave. The network layer and transport layer are respectively called the transfer layer and routing layer in Z-Wave [4, 48].

congestion control on the channel. As said in the introduction, the physical and MAC layer are defined in ITU G.9959. The network layer in Z-Wave is called the transfer layer and manages retransmissions, acknowledgments and checksum bindings. The transport layer in Z-Wave is called the routing layer and performs routing and topology scans. The last layer is the application layer which provides an interface to applications and data [4].

3.5.4 Physical Layer

Z-Wave specifies three channel-configurations. These correspond to the following data rates: 9.6 Kbs, 40 Kbps and 100 Kbps. Channel configuration one and two use Frequency Shift Keying (FSK) as the modulation technique. This technique is already explained in more detail in Section 3.1.1. When channel configuration three is selected, Gaussian Frequency Shift Keying (GFSK) is used. As already seen in Section 3.1.1 about the Sigfox modulation, the Gaussian filter smooths the frequency shifts [4].

To share the frequency band with other technologies, Carrier Sense Multiple Access Collision Avoidance (CSMA/CA) is used. This means, as already seen in Section 3.4.1, the sender will first listen to the channel before transmitting. If the channel is busy, the sender will back off and try again after a random period of time [4].

The physical layer is responsible for modulation, channel usage and synchronization between the sender and receiver. The Z-Wave packet format of a singlecast frame is shown in Figure 3.20. The preamble is used for symbol synchronization at the receiver side. The bits alternate between 0 and 1. Depending on the frame type, the preamble's minimum size is between 10 and 40 bytes. The start of the frame is a fixed number of bits as is the end-of-frame field. A physical frame is called a Physical Protocol Data Unit [49].

3.5.5 Physical and MAC Layer

The MAC layer defines MAC Protocol Data Units (MPDU) which can be seen in Figure 3.20. The header of the MPDU is shared with higher layers. These fields are carried transparently by the MAC layer. The home ID specifies the Z-Wave network that is used. The source ID corresponds to the node ID of the sender. The home ID and node ID specify a single device. The frame control field defines the frame type and other control flags. There are four types of MPDUs: a singlecast frame, a multicast frame, an acknowledgment frame and a routed frame. A **Singlecast frame** is sent to a specific node, a **multicast frame** is sent to multiple nodes, an **acknowledgment frame** is sent to conform a successful reception of a singlecast frame and a **routed frame** is frame with a specific route. An acknowledgment can be requested by setting a bit in the frame control field. The length field specifies the length of the entire MPDU in bytes. The sequence number is set by the upper layers. The field distinguishes different frames. Only in channel configuration three is this 8-bit field used. In channel configuration one and two, a 4-bit sequence number is set in the frame control field. The destination ID specifies the node to which the frame is sent. Therefore the node ID is used, which specifies a single device in combination with the home ID. Byte 0xFF can be used to broadcast the frame to all nodes in the Z-Wave network. In the multicast frame, the multicast control field indicates the length and an address offset. Together with the following fields, a list of node IDs can be formed to which the message can be sent. The data payload is defined by the upper layers. The checksum field validates the entire MPDU. Channel configuration one and two use an 8-bit XOR checksum. Channel configuration three uses a cycle redundancy code of 16 bit which provides more certainty that the MPDU is not altered [49].

When a packet arrives at the destination and an acknowledgment is requested and ACK the send back to the sender. When the sender does not receive an ACK, it will retransmit the packet up to three times before giving up [48].

3.5.6 Network and Routing Layer

The network layer in Z-Wave is called the transfer layer. This layer is responsible for the transfer of data between two nodes, retransmission, acknowledgments and the checksum. The routing layer controls the routing between nodes. This layer also maintains a routing table and network topology. The packet format of both layers is shown in Figure 3.20 [48].

Z-Wave uses source routing, where the source or sender specifies the path the message has to travel. Therefore Z-Wave puts the IDs of the hops, along the predefined path, in the transport layer of the message. We now discuss the fields of the transport layer in more detail, an overview of the transport layer is shown in figure 3.20. The first four bits represent the failed-hop field which is used to specify the hop where an error occurred, otherwise it is zero. The type field specifies whether it is an application packet, command packet, route ACK or route NACK. The route ACK is sent by the destination node when an application packet is received, the route NACK is sent by an intermediate node when a forwarding error occurred. The length field specifies the number of hops and the hop index specifies the current node in the list of nodes that follows this field. The list of nodes consists of 8-bit node IDs [48].

When a node receives a packet, the node determines if it is responsible for forwarding the packet. The hop index provides the ID of the current hop. The node will increase the hop index, recalculate the frame checksum and transmit the packet to the next hop. For an ACK or NACK packet reverse routing is used which means the hop index is decreased on every hop [48].

An ACK packet if formed by copying the MAC layer, network layer and transport layer into a new frame. Then the source and destination field of the MAC layer are swapped and the type field of the network layer is set to the ACK type. If the sender does not receive an ACK packet, it will retransmit the frame up to three times. As explained in Section 3.5.5, the MAC layer uses acknowledgments to confirm the arrival of a packet, which can be used between two hops on the path. If a node on the path failed to reach the next hop on the path, a NACK is sent back to the sender. The hop index of the node that failed to respond is copied into the failed hop field. By using this NACK packet, the sender is informed that the packet failed to reach the destination because of an unreachable hop in the path [48].

Every node (controller or slave) has a list of all the devices it can reach directly, which are called the neighbors. The controller devices can request this information from all devices and build an adjacency table of the full network. Based on the adjacency table a routing table can be built, therefore a controller builds a route to all devices in the network. Routing slaves or enhanced slaves do not have a full routing table. If a routing slave or enhanced slave needs to send a message, it requests a route to that destination from a controller device. Because the controller device has the full adjacency table it can create a route from the slave to the destination and inform the slave about this route. The routes are cached, therefore the slave device does not need to repeat this process with every message [48].

3.5.7 Application Layer

The application layer can transport application-specific data or node information for the assignment of home and node IDs. The application layer uses command classes to specify the data format and available commands. A command class is identified with a unique ID, managed by Z-Wave. An example is the Binary Switch command class, which provides a get, set and report command. This class could be used by a light switch for example. Other examples of command classes are light dimming, thermostat, etc. Each device belongs to one or more command classes and announces them during the paring operation [50].

The application layer format is shown in Figure 3.20. The first 8 bits belong to the header. The second field is the command class which is explained above. The command field specifies which command, defined in the class, should be executed. The command field is followed by its parameters.

3.5.8 Security [51]

Z-Wave uses the following security mechanisms:

- 128-bit AES encryption to hide application data.
- Message Authentication Code (MAC) to protect integrity.
- 8-byte nonce to protect against replay attacks.

All messages used to exchange security-related information are defined in the Security command class. In the beginning, the primary controller will generate a random **network key**. When a device joins the network, the network key is sent to the joining device. The exchange is encrypted using a predefined key which is hardcoded in the firmware. This only provides a limit encrypting, but notice this exchange only takes place during the joining procedure which limits the time an attacker has to intercept the key. Another security benefit is that the joining procedure can only be initiated with a user intervention [48].

From the network key, two 128-bit keys are derived: the **frame encryption key** and the **data origin authentication key**. As the name suggests, the first key is used to encrypt the payload and the second key is used to authenticate the message. Both keys are identical for all the nodes in the Z-Wave network.

When a secure message must be transmitted from a node to a destination node, first an 8-byte nonce is requested from the destination. This nonce is used to prevent replay attacks and we will discuss how it is used shortly.

The payload of a message is encrypted using AES in Output Feedback (OFB) mode with the frame encryption key. AES uses an initialization vector to ensure the payload is always encrypted in a unique way. This ensures that two identical payloads result in two different encrypted payloads. The 16 bytes initialization vector is created using 8 randomly generated bytes and the nonce received from the destination. Since the destination always provides the sender with a different nonce, the destination can discard messages with an identical nonce which prevents replay attacks. The destination node can use the same key and the initialization vector to decrypt the payload.

In secure-transmission mode, an 8-byte authentication header is added to the end of the packet just in front of the checksum. Part of the header is the MAC. This MAC is calculated by concatenating the security header, the source node ID, the destination node ID, the length of the encrypted payload and the encrypted payload itself. The result is encrypted using AES in Cycle Block Chaining (CBC) - MAC mode with the data origin authentication key. At the receiver side, the MAC is calculated ones more and is compared to the MAC in the packet which ensures the content of the packet remains unchanged and the message originates from the sender.

3.5.9 Known Vulnerabilities

While Z-Wave provides the necessary security mechanisms it is not mandatory, therefore it depends on the functionality of the device whether security is used. Devices such as lights, switches and thermostats usually have no security mechanisms enabled. The Z-Wave security class is also rather limited, firstly only the application layer is encrypted and only a part of the transport layer fields are integrity protected.

This means that the remaining layers are visible to an attacker and some of the fields can even be altered [48].

In Badenhop et al. [48] several against the Z-Wave security are listed. Because security is not required, it is possible to spoof the source address of packages. This allows an attacker to alter the neighbor list of a node in the Z-Wave network which can cause routing failures. It also allows the attacker to gather the full network layout, or even deliver false routing information to nodes on purpose. By altering the routing information in a node, an attacker can force all traffic to pass a controlled node which can be used to inspect network traffic or alter messages. In the black hole attack, this method is exploited to drop packages between the source and destination which causes a DoS [48].

3.6 Thread/OpenThread

Thread is an Internet Protocol version 6 (IPv6) based low power networking protocol. This open standard is designed for connected home applications. Thread was launched in 2014 by the Tread Group Alliance. This group is backed by companies such as Google, Samsung and Qualcomm. OpenThread is an open-source implementation of Thread created by Google. Like Zigbee, Thread uses the IEEE 802.15.4 standard on the physical and MAC layer and is also mesh-based. The main difference between Thread and Zigbee is that Thread is IP addressable. The overall architecture of Thread is based on 6LoWPAN, which is an acronym for IPv6 over Low power Wireless Area Networks. 6LoWPAN was formed in 2005 but the working group closed, the standard however is made public [4]. Thread is used in several Nest products like thermostats, doorbells and smoke detectors [52].

3.6.1 IEEE 802.15.4

Thread uses the IEEE 802.15.4 standard on the physical and MAC layer. Thread operates in the 2.4 GHz band at 250 Kbps [4]. The IEEE 802.15.4 standard is already explained in Section 3.4.1. In Thread the 802.15.4 MAC layer provides Carrier Sense Multiple Access and retries between neighbor devices. The Thread layers provide the keys for MAC layer encryption and message integrity [53].

3.6.2 Network Structure

An example Thread network with all the device types is shown in Figure 3.21. Thread devices can be categorized based on forwarding. A **router** forwards packets and is always on. An **end-device** is a node that does not forward traffic to other nodes. It can be powered off to save power. The routers are represented by a rhombus, the end-devices by a circle in Figure 3.21. A **Thread Leader** is a special Thread router. The Thread Leader is responsible for managing the routers and distributing the network configuration. This device is chosen through an election process. When the Tread Leader goes offline, a new device will be elected. The **Border Router** can forward messages between Thread networks and non-Thread networks using WiFi or Ethernet for example [54].

Thread nodes can also be divided into categories based on whether they listen to the all-routers multicast address. An **Full Thread device (FTD)** always has its radio on and subscribes to the all-router multicast address. A **Minimal Thread device** does not listen to the all-route multicast address, it only communicates with its parent router. When the parent of a node is not available anymore, the end-device can select another parent [53, 54].

There are three types of full Thread devices. The first type is a router, it forwards traffic. The second type is a **Route Eligible End Device (REED)** which is an end-device but can be promoted to a router. The last type is a **Full End Device (FED)** which cannot be promoted to a router [54].

The minimal Thread devices can be subdivided into two categories. The first type is a **Minimal End-Device (MED)** which always has its receiver on and therefore can received all messages instantaneously through its parent router. A **Sleepy End-Device (SED)** in contrast, can be turned off to save energy. When the device wakes up, it polls for new messages at its parent router [54].

3.6.3 Addressing

As said before, Thread uses IPv6 addressing. This makes it easier to communicate with end-devices from outside the network. By using IPv6, the addressability is more than large enough to cover the

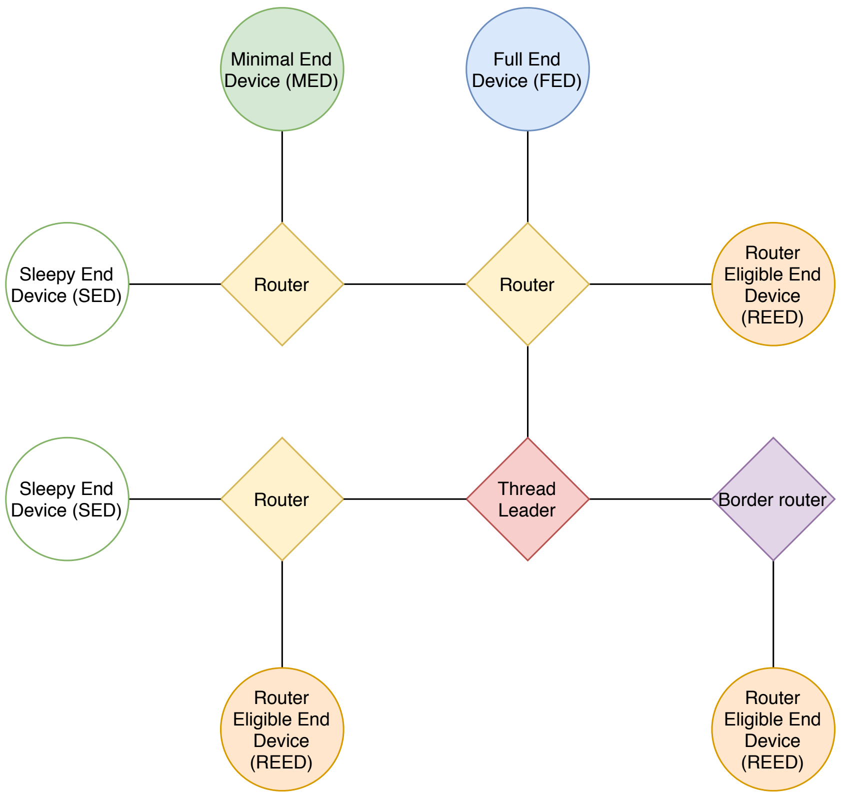


Figure 3.21: An example Thread network with all types of devices.

next generation IoT devices. While communication and addressability are big advantages, using 128-bit IPv6 addresses in low power technologies is not ideal. A normal IPv6 header is 40 bytes long and a 802.15.4 packages can only contain 127 bytes. Therefore header compression is needed [4].

First, let us take a look at the different types of addresses. A **Link-Local** address starts with *fe80::/16* and addresses the devices that are reachable with a direct radio transmission. The **Mesh-Local** address defines the devices that are reachable within the same Thread network. Mesh-Local addresses start with *fd00::/8*. And finally, the **Global** addresses define de devices reachable from the internet [54].

An IPv6 address is 128 bits long. The first 64 bits are called the prefix. This prefix is a Locally assigned Global ID and is chosen by the Thread Leader when it creates the network and is used throughout the Thread network. Border routers may have other prefixes that can be used to create Global addresses. The last 64 bits are assigned locally in the network [53].

As specified in the IEEE 802.15.4 standard (see Section 3.4.1), each device joining the Thread network is assigned a 16-bit short address. This address is used to route messages through the Thread mesh. The short address consists of two parts: a router ID and a Child ID which can be seen in Figure 3.22. The Router ID identifies the router to which the device belongs and the child Id identifies a single device along all devices connected to that router. The Child ID for a router is always 0. This short address is part of the **Routing Locator (Routing Locator) address**. The RLOC is an IPv6 address that identifies a single Thread device and is based on the location of the Thread device in the network. If for example a device switches between two parent-routers, the short address changes, and hence the RLOC address changes. The last 16 bits of the RLOC address are formed by the short address. Because the short address is part of the RLOC, the short address consisting of the router ID and child ID is also called the **RLOC16**. To save data in the Tread messages only the RLOC16 (16 bits) is included in a Thread packet [54–56].

The mesh-local **Endpoint Identifier (EID)** identifies a device independent of the network topology. This address must be used in applications when addressing a single device. The first 64 bits are formed by the mesh prefix and the last 64 bits of the address are chosen randomly after the device joins the network. Because the address cannot be used for routing, a full Thread Device must translate it to an RLOC address. A similar address can be assigned on a global scale, this is called the **Global**

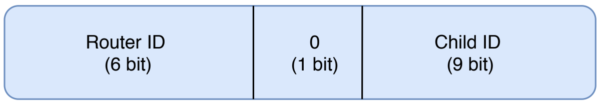


Figure 3.22: The last 16 bit of the RLOC address.

Unicast Address (GUA), but this is optional since not all devices must be reachable from the internet [54].

Several multicast addresses are defined to send the same message to multiple devices in the Thread network. For example *ff03::1* sends a message to all FTDs and MEDs and *ff02::2* sends a message to all FTDs [54].

3.6.4 6LoWPAN

There are some issue when sending IPv6 packets over the IEEE 802.15.4 protocol. The IPv6 protocol requires that lower layers support a minimum payload size of 1280 bytes while that maximum physical packet size of 802.15.4 is 127 bytes. The other problem is already explained in the previous section, the IPv6 header with its 40 bytes in size provides a significant overhead. These issues are solved using the 6LoWPAN adaptation layer [56].

6LoWPAN provides header compression techniques, as already seen in Section 3.6.3 where these IPv6 addresses are replaced by shorter 16-bit addresses. 6LoWPAN also provides a fragmentation layer, which can fragment packages at the source if they are too big to fit in an IEEE 802.15.4 packet and reassemble them at the destination. Another feature of 6LoWPAN is the ability to forward packages based on IEEE 802.15.4 header information. As already seen in Section 3.4.1, IEEE 802.15.4 contains the short source and destination address which are uses by 6LoWPAN to forward frames to the next router. Hence 6LoWPAN is capable of routing without the need to fully reconstruct the original IPv6 frame which is more efficient and can save energy [56].

Thread support the Internet Control Message Protocol version 6 (ICMPv6) and User Datagram Protocol (UDP) through the use of 6LoWPAN [56].

3.6.5 Packet Format

As said in previous sections, the physical and MAC layer are already explained in Section 3.4.1. The network layer of a Thread packet is shown in Figure 3.23 [56].

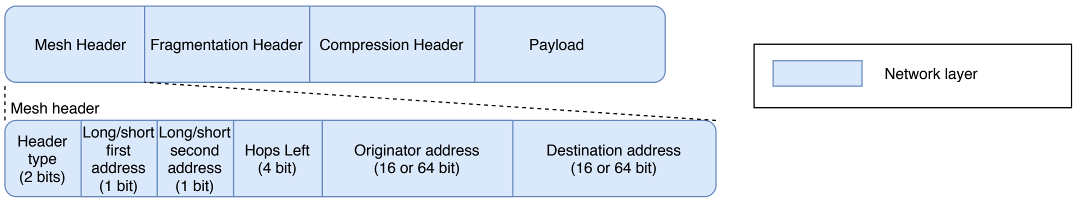

Mesh Header
Fragmentation Header
Compression Header
Payload
Network layer
Mesh header
Header type (2 bits)
Long/short first address (1 bit)
Long/short second address (1 bit)
Hops Left (4 bit)
Originator address (16 or 64 bit)
Destination address (16 or 64 bit)

Figure 3.23: Network layer of Thread packet [56].

6LoWPAN uses stacked headers to compress the headers from other protocols. A stacked header always starts with a header type followed by the header itself. Thread uses the mesh header, the fragmentation header and the compression header.

The first two bits specifies the header type, which is 10 in case of the mesh header. The first two bits of the mesh header specify whether the short or long address is used for the source/destination address. The hops left field specifies the time to live, which is decremented when the packet is given from one hop to the next. When the hops left field reaches zero, the packet will be dropped. The Originator Address field specifies the source address, and the next field contains the Destination Address. Whether the short 16 bit or longer 64-bit address is used, depends on the first two bits of the mesh header [56].

The fragmentation header is used when the IPv6 packet is too big to fit into an IEEE 802.15.4 packet. The fragmentation header consists of a tag, a size and an offset. The tag is identical for all fragments belonging to the same IPv6 packet, the size specifies the size of the original IPv6 packet and the offset is used to specify the order in which the fragments should be reconstructed [56].

The compression header contains the compressed IPv6 header, the exact format would go beyond the scope of this thesis. Also, the UDP header can be compressed [56].

3.6.6 Routing

In a Thread network, next-hop routing is used for sending packets from device A to device B. This means all routers maintain a routing table of the Thread network. Each row of the routing table consists of a destination router, the cost to that destination and the next hop on the shortest path to that destination. The cost is the sum of the link costs to the destination. When a package arrives at a router, the router will look at the router ID in the 16-bit destination short address and the package will be forwarded to the next-hop router. If the router is the destination router, the message will be forwarded to the corresponding child [53].

The routing table is periodically exchanged with neighbor routers through Mesh Link Establishment (MLE) messages. When a router receives the routing table of a neighbor device it will update its routing table based on the new information. MLE messages are also used to exchanged other configurations for example during the joining procedure [53].

Between two devices, MAC level retries are used to deliver messages reliably. Because Thread uses UDP as a transport protocol, it is up to the application to create a reliable message delivery system [53].

3.6.7 Join Procedure

Joining a Thread network requires a device to perform three steps. Firstly the device must discover the Thread network. The next step is commissioning where the device proves it is allowed to join the network. And the last step is called attaching where the device will contact the nearest router and exchange network information [53].

During the discovery phase, a device will send a beacon request on each channel. All devices that are in range will answer with a beacon containing their PAN ID, XPAN (extended) ID and the network name. A neighboring router is selected to perform commissioning [53].

During the commission phase an out of band method or a commissioning application on a smartphone/tablet is used to authenticate the device [53].

In the last phase, MLE messages are used to determine neighbors, negotiate parameters and create links. To establish a link between parent and child the following messages are sent [54]:

1. The child sends a multicast Parent Request.
2. Neighboring Routers and REEDS send a parent response. This message contains among other fields the RLOC16 address of the router, the leader and signal quality.
3. The child chooses the best parent device and sends a Child Request ID.
4. The parent answers with a Child Id Response.

3.6.8 Security

Thread uses a **network key** that is shared by all devices in a Thread network and is used for MAC layer encryption and authentication using the IEEE 802.15.4 protocol which is already explained in Section 3.4.3. This network key is updated regularly and passed to all the Thread devices using MAC security. On top of MAC layer security, Thread uses individual frame counters for all its neighbors to protect against replay attacks. Thread also allows applications to use any secure internet protocol for end-to-end encryption [57].

All the managements tasks, such as adding or removing a device, are performed by a management application which needs a human administrator. This application can run on any internet-connected device and connects to the border router using Datagram Transport Layer Security (DTLS). DTLS is similar to TLS, but packages are sent over UDP instead of TCP. If the management device is however part of the Thread network, this connection is not needed. A Thread network has a management password that allows the management application to perform its tasks [57, 58].

The **commissioner device** stores the network key and is responsible for delivering the key securely to joining devices. This can even be a device outside the Thread network, in which case the device must connect to the border router by a DTLS connection. A joining device must be authenticated, therefore a DTLS connection is set up between the joining device and the commissioner, then the administrator can validate that the correct device tries to join the network and then the network key is passed securely to the new device [58].

As we have seen in Section 3.4.4, some attacks against the IEEE 802.15.4 protocol are known, they are however not applicable to Thread. Replay attacks are not applicable since Thread uses a 4 bytes frame counter which takes 49.7 days before it has to reset at one packet per millisecond. However, the keys are reset before the 49.7 days are over. The DoS attack, where the frame counter is well increased by an attacker so that legitimate messages with a smaller frame counter are ignored, is not applicable in Thread since the frame counter is taken into the integrity protection calculation. This attack would therefore result in an authentication failure [59].

3.6.9 Known Vulnerabilities

In Liu et al. [59] a taxonomy for security assessments is proposed and applied to the Thread protocol. The conclusion is that Thread is secure to most attacks against data and network functions except for a few DoS attacks that can affect network performance. Due to the time complexity of the security handshakes in DTLS and the limited processing power of the devices, the joining time increases in orders of magnitudes with handshake errors happening. Thread is vulnerable for radio jamming, which can lead to a DoS attack. Similar attack can be performed where the number of packets is sent to a particular node to drain the battery or an attack where the CSMA/CA algorithm is skipped which will greatly affect the network performance. Also, an ACK frame can be forged to hide the dropped messages [59].

3.7 Comparison

In this section, we compare the IoT communication protocols discussed in the previous sections. An overview of the features is shown in table 3.1.

The communication technologies can be divided into two categories: the Wide Area Network (WAN) and the Personal Area Network (PAN). The WANs are for outdoor usage or long-range applications, they typically have a low data rate since the signal must be able to travel further. Sigfox, LoraWAN, NB-IoT and LTE-M are protocols that can be used in WANs. The PANs are for a shorter range or indoor applications because their range is limited, they have however a higher data rate available. Zigbee, Z-Wave or OpenThread are examples of PAN protocols.

If the application requires a long-range technology, Sigfox, LoRaWAN, NB-IoT and LTE-M are three options that can be used. The biggest difference between these technologies is their data rate and data limit, NB-IoT and LTE-M can transmit more data at a higher data rate while LoRaWAN and Sigfox are strongly limited to the number of messages per day. Sigfox however is also limited in packet size, only 12 Bytes are allowed per uplink message. Sigfox is therefore well-suited for monitoring applications, some examples are sound monitoring, air quality monitoring, snow level monitoring, temperature monitoring, tank level monitoring and asset tracking. LoRaWAN allows more data per uplink message, has a lower latency and allows for more downlink packages. This makes it well suitable for more urgent applications or applications where more downlink capabilities are required, some example applications are fire detection, water leakage detection, fall detection and selective irrigation. Some of the LoRaWAN capabilities highly depend on the provider, The Things Network for example has a fair access policy where the rules can be compared to the Sigfox network [60]. However, applications described for Sigfox can also be implemented by LoRaWAN. LoRaWAN also has the capability to set up your network where there is no fair access policy which makes it suitable for application including but not limited to perimeter access control, microclimate control in greenhouses and monitoring on the company premises. NB-IoT allows for even more data at a higher data rate. Because it uses the IP protocol, it can be easily integrated into existing web applications which eases the development process. However, NB-IoT does consume more energy in comparison to Sigfox and LoRaWAN. NB-IoT is well suited for applications where more data is required such as smart parking and traffic monitoring. LTE-M provides an even higher data rate, a lower latency and has mobility support in comparison to NB-IoT. LTE-M has however a lower battery live and the range is limited in basements and tunnels. LTE-M with the higher data rate and lower latency is capable of supporting Voice Over LTE (VoLTE) applications which makes LTE-M suitable for: smartwatches, security systems and Smart Grids.

Zigbee, Z-Wave and Thread are mostly used for indoor applications. The biggest differences are data rate, network size and security. Z-Wave is has a lower data rate and a limited network size in comparison to Zigbee and Thread. Based on the security analysis and security vulnerabilities in Section 3.4.3, 3.4.4, 3.5.8, 3.5.9 and 3.6.8 we can conclude that Tread and Zigbee are more secure than Z-Wave. The biggest difference between Thread and Zigbee is the stack, Thread is IP based and can therefore easily communicated with the internet. Zigbee on the other hand defines its own network stack including application layer which makes it more difficult to communicate with the internet. Zigbee provides however a uniform interface to data and hence gives more structure to the data and messages.

	Sigfox	**LoRa / LoRaWAN**	**NB-IoT**	**LTE CAT-M1**	**Zigbee**	**Z-Wave**	**Thread / OpenThread**
Spectrum band	ISM	ISM	Licensed LTE	Licensed LTE	ISM	sub-1 GHz	ISM
Maximum payload	12 B (up), 8 B (down)	243 B	1280 B [61]	1280 B [61]	127 B	64 B	127 B
Data limit	140 messages/day (up), 4 messages/day (down)	depends on provider	depends on provider, 1-30 MB per month (orange[4])	depends on provider, 1-30 MB per month (orange[5])	no limit	no limit	no limit
Peak data rate	100 bps (down), 600 bps (up)	0.3 - 5 Kbps	200 Kbps	1 Mbps	250 Kbps	9.6/40/100 Kbps	250 Kbps
Range	3-10 km (urban), 30-50 km (rural)	5 km (urban), 15 km (rural)	1 km (urban), 10 km (rural)	1 km (urban), 10 km (rural)	10-100 m	30 m (indoor), 100 m (outdoor)	10-100 m
Theoretical network size (nodes)	n/a	n/a	n/a	n/a	65,536 [4]	232 [49]	65,536 [4]
Sleep current	$1.5\mu A$ [4]	$1.5\mu A$ [4]	$3\mu A$ [62]	$8\mu A$ [62]	$2.16\mu A$ [63]	-	$2.16\mu A$ [63]
Idle current	$0.5mA$ [64]	$1.4mA$ [62]	$6mA$ [62]	$9mAA$ [62]	$3.11mA$ [63]	-	$3.11mA$ [63]
Receive current	$18mA$ [64]	$12mAmA$ [62]	$46mA$ [62]	-	$12.84mA$ [63]	-	$12.84mA$ [63]
Transmit current	$19 - 49mA$ [64]	$24 - 44mA$ [62]	$74 - 220mA$ [62]	$100 - 490mA$ [62]	$16.07mA$ [63]	-	$16.07mA$ [63]
Mobility	mobile	mobile	limited to single base station	mobile	limited local network	limited to local network	limited to local network
Latency	$<$ 60 s	0.5-2s	1.6-10s	0.010 - 0.015 s	$<$ 0.005 s	$<$ 0.036 s (at 40 Kbps)	0.005 s
Private network available	No	Yes	No	No	Yes	Yes	Yes
Example hardware	ATA8520E [65]	Semtech SX1276 [66]	Quectel BC95 [67]	SARA-R4 series [68]	AT86RF231 [69]	-	nRF52840 [70]
Example dev-kit	ATA8520-EK3-F	Dragino LoRa/GPS HAT [71]	FiPy [72]	FiPy [72]	Rasp module [73]	USB dongle [74]	nRF52840 DK [75]

Table 3.1: Overview IoT communication protocols: Sigfox, LoRa, NB-IoT [4, 10, 76, 77]
, ZigBee, Z-Wave [4, 78–80], Thread [4, 80]

Chapter 4

Survey IoT Edge Technologies

According to the International Data Corporation, more than 59 zettabytes of data will be created, captured copied and consumed in the year 2020 [2]. In 2010 the mark of 1 zettabyte was passed. The Cisco Annual Internet Report predicts there will be 14.7 billion M2M connections by 2023 and 50 % will be from IoT devices [3]. The current mobile networks will have trouble handling these massive amounts of data. In fully cloud-based applications, all data is sent to the cloud and the decision making is also done in the cloud. For some applications, this will not even be possible because of the high bandwidth and low latency requirements. Another concern is privacy, it may not be legal to send data to the cloud e.g. in case of a video surveillance system at home. This is where **edge computing** comes in handy. In edge computing, a device is introduced close to the end-devices which may be as close as one hop away from the IoT devices, such as a gateway or WiFi access point. This edge device can store, manage and process the data generated by the end-devices. The most interesting case is where the edge device can react to certain events and send out a command to other local end-devices. By using edge devices a part of the work, otherwise send to the cloud, can be performed locally. This gives lower latency, provides more bandwidth, reduced the cost of communicating with the cloud and keeps privacy-sensitive data local [81].

There exist similar concepts to edge computing. **Fog computing** refers to computing, storage, networking and management devices located anywhere from in the cloud to on the end-devices themselves. Fog computing is hence a more general term than edge computing. The computing paradigm where the processing of data is done on the device itself is called **Mist computing** [81].

4.1 Applications

In Fog Computing: Platform and Applications [82] and A Survey of Fog Computing: Concepts, Applications and Issues [83] a list of possible edge applications is published. We describe some of the application scenarios.

Cameras are commonly deployed in cities. The camera images are analyzed using object recognition and tracking to provide security and traffic management for example. By using edge computing images can be analyzed locally so there is no need to send images to the cloud. If there is a store for example with some cameras pointing to different regions of the room, all camera streams could be analyses on a single edge device for the entire store. The edge device could detect incidents and alert emergency services. The biggest advantages of this approach are network cost reductions, real-time analytics and that privacy-sensitive data can be kept locally.

Edge computing can help in handling big data analytics. Data acquisition, aggregation and preprocessing can be performed at the edge. For example in a large scale monitoring system. The edge nodes can analyse the data and provide a quick response. This is beneficial in case of emergencies. For more advance and more computational insensitive tasks, the important data can still be forwarded to the cloud. The main advantages here are quick feedback and offload work, initially done in the cloud, to edge devices.

I smart homes a lot of different devices are used for a large number of applications. Because different vendors and communication technologies are used it is hard to interconnect devices. An edge device can be installed to solve this compatibility problem and let the IoT devices work together. An edge device could detect for example through a video stream that someone knocked over a glass of milk and command the cleaning robot to clean it up. This application does not need to communicate with the cloud. An other example that does need the cloud is for example a company that places insulation and wants feedback on their products. The edge device could calculate the insulation index based on inside and outside temperature and the state of the central heater. This insulation index can be sent anonymously to the cloud. The biggest advantage here is privacy, the data is analyzed locally and actions are performed locally.

4.2 Edge Technologies

Edge technologies can help in making applications for the edge. Usually, the architecture of an edge application will look like Figure 4.1. The bottom of the architecture is referred to as the **south**. This is where the IoT end-devices are. The top of the architecture is referred to as the **north**. This usually is the cloud. Messages going to the north are called **northbound traffic** and messages going to the south are called **southbound traffic** [84, 85].

Edge devices interact with the end-devices. End-devices can be sensors that report data and actuators that perform some action. The edge devices can perform simple actions like if the temperature is to high turn on the cooling device or more advanced actions like detect if a person falls and notify the emergency services. Edge devices can be connected to the cloud but this is not mandatory and some edge technologies will even deal with an unreliable uplink connection.

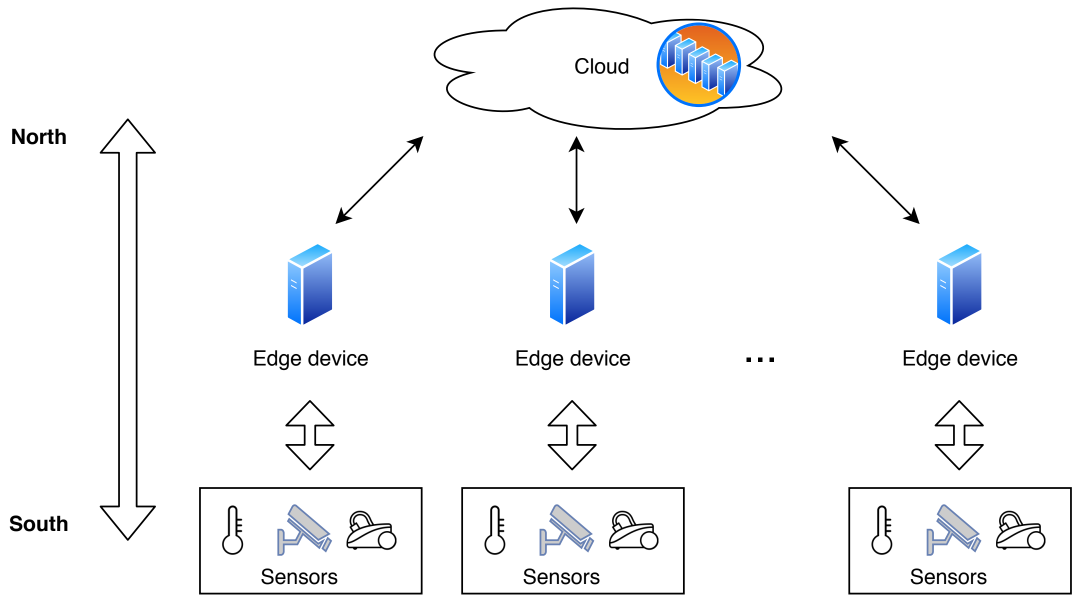


Figure 4.1: General edge computing archtecture.

In every edge application there must be dealt with several problems:

- Management of edge devices.
- Management of IoT end-devices.
- Communication between end-devices and edge devices.
- Communication between edge and cloud.
- Provide a setup in which the application logic can be developed.
- Provide security throughout all communication.

Edge technologies help in performing these common tasks. In the following sections, we will compare several IoT edge technologies. Every technology will be briefly introduced firstly. Then we will discuss the architecture and where the technology can be used. Third, we discuss a more practical side of how applications can be set up and developed by performing an experiment. And finally, we discuss how well the problems defined above are solved.

4.3 Preliminary Experiment

In this section, we experiment by creating an edge solution on our own. We do this to familiarize ourselves with the topic and to set some goals to other edge solutions. In our solution, we limit ourselves to northbound (uplink) traffic. Because this solution does not support downlink traffic we call this an aggregator instead of a full edge solution. Due to a limited number of use cases we prematurely stopped the development of this solution.

We define the following requirements for an aggregator:

- An aggregation operator is fully customizable
- Easy to add/replace sensors
- Sensors can deliver messages with different priorities
- Support for multiple IoT technologies
- Easy to deploy aggregator
- Support for a large number of devices

We propose an architecture with merge operators that run on independent devices such as edge devices. A merge operator gets multiple messages as input and delivers a single or a limited number of output messages. These input messages come from the south and the output messages go to the north. A merge operator can be placed at multiple levels in the network architecture. An aggregator close to an IoT gateway can process the raw data from sensors. An aggregator placed close the network provider can combine messages from different lower-level aggregators. Here messages from aggregators closer to the sensors can be merged. This reduces traffic to the final destination and distributes computing power. An overview is shown in Figure 4.2.

This solution is useful in applications where a large amount of sensor data is gathered but only a part of the data is useful to the application. As an example, we discuss a fictive example: street lighting. To maintain the lights a company installs a monitor in every light that reports the status of the light every minute. If the status report is not received from the light, it is probably broken and an electrician is sent out to fix that specific light. For passing the messages they use a wireless technology because a wired solution would be too expensive. If every light would send out its status to the central server, this would require a lot of data. In our solution, an aggregator could be installed on every street. This aggregator could use a wireless technology to aggregate the data from all the lights. A free of charge technology can be used to connect the lights to the aggregator. The aggregator can then check the status of the light and report only the lights that are not sending a status update. For the uplink connection of the aggregator, a more capable paid connection can be used such as a wired connection, an NB-IoT connection or even an LTE connection. Due to the limited number of uplink messages, the uplink connection does not require a lot of bandwidth. Also, the number of messages the cloud server of the company must handle is reduced.

4.3.1 Architecture

The architecture consists of different components. The **main server** controls all applications. An application consists of a merge operator and the list of sensors that can use this merge operator. Data flows from the IoT sensors through the aggregators until it is finally received by the main server. The task of the **aggregator** is to combine messages and reduce the amount of data that must be used. An aggregator device consists of a handler and a set of applications. The **handler** maintains the applications. All messages arriving at an aggregator are processed by the handler and forwarded to the corresponding application.

An application is built in a docker container and this container is registered on the main server. The use of containers eases the deployment and provides the necessary security for the aggregator device. When a sensor message arrives at the aggregator that does not belong to an application known by

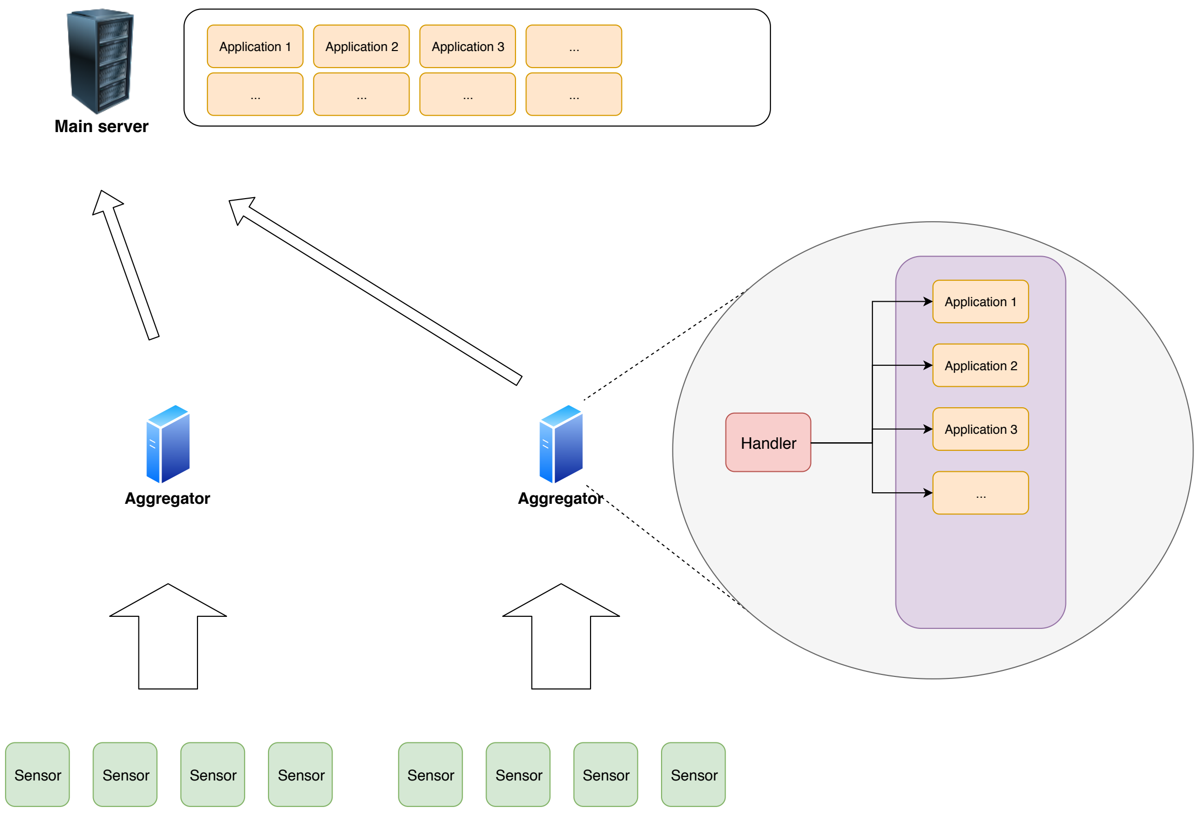


Figure 4.2: Aggregator architecture.

the aggregator, the aggregator will request the application from the main server. This handler will download the docker container from the main server and start the container locally on the aggregator. This ensures an application is only deployed on aggregators that use the application.

The merge operator is provided as a python library. It has a simple API to add new messages, poll messages, merge messages and convert messages to the uniform format.

To provide a flexible architecture, uniform messages are used between aggregators and between an aggregator and the main server. This allows for hierarchically placed aggregators.

4.3.2 Adapter Architecture

To support multiple technologies as communication between the sensors and aggregator we introduced an adapter layer. The inner working of an adapter is shown in Figure 4.3. An aggregator uses the adapter to convert messages to a uniform format. Typically the initial message from the sensor to the first aggregator differs from the other messages send between aggregators. Therefore, the application must provide a conversion routine to convert a sensor message to the format used by the aggregators.

The task of the adapter is to communicate with sensors and extract the relevant data from senor messages while keeping the configuration to a minimum and keeping the data hidden from attackers.

4.3.3 Conclusion

The overall architecture was developed first. Until the point where it was possible to pull containerized application on demand and merge data from virtual sensors until it was received at the main server. We started developing the LoraWAN adapter for which we used the chirpstack[1] application server.

During the development, we realized there were a limited number of use cases for this aggregator solution. The main reason being that there was no support for southbound communication, no commands could be sent from the cloud back to the end-devices. For this reason, we stopped further development

[1] `chirpstack.io`

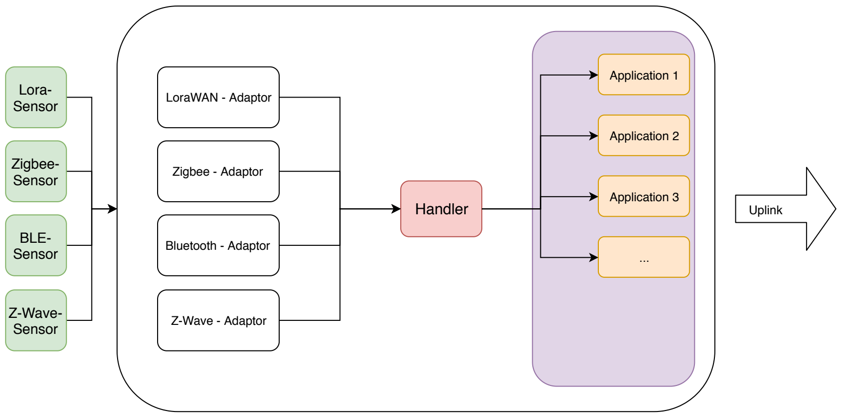


Figure 4.3: Inner working of aggregator. To support multiple IoT technologies, adapters are used to convert messages to a uniform format.

of this experiment. But we can draw some important conclusions. For an edge device to be valuable it must be able to respond to events and send commands back to end-devices. Also, we can see some features developed in this experiment come back in some existing edge technologies which means they are valuable to IoT edge development. On-demand deployment of applications on edge devices is also used in FogFlow and contained environments are used in Azure, Balena and FogFlow.

4.4 EdgeX

EdgeX is an IoT framework that allows bidirectional data- and command flow between end-devices and the cloud. EdgeX is also able to work offline if the network connection drops. The project is open source and hosted by the Linux foundation. The goal of EdgeX is to facilitate the hardware interoperability in the IoT ecosystem [84].

4.4.1 Architecture

EdgeX provides a module that can be deployed on a single edge device (e.g. a gateway). EdgeX consists of multiple layers. An overview of the EdgeX architecture is shown in Figure 4.4. The bottom of the architecture is referred to as the south side. The sensors and actuators (devices that can perform an action) belong in this area. The top of the architecture is referred to as the north side which is usually the cloud. Communication can flow in two directions. Northbound traffic is usually sensor data and southbound traffic are commands to actuators. For the communication between different layers REST APIs are used, usually in JSON format [84].

The bottom layer consists of **device services**. This layer communicates with the sensors and actuators. The protocol used for communication is device-specific. These services are responsible for the translation between the device data format and the EdgeX data format. There are several device services already implemented by EdgeX. Examples are Representational State Transfer (REST), Message Queuing Telemetry Transport (MQTT), Low Energy Bluetooth and Zigbee. It is also possible to create specific device services [86].

The second layer is called **core services**. This layer consist of multiple components. The core data component ensures persistent storage for sensor data and management data. The command component receives commands from the north side and passes these to the south side. The metadata component manages information about devices and sensors. It knows what type of data can be sent to and received from devices. It knows for example a temperature sensor will send a JSON with a “temperature” attribute and a float value. This is after it has been converted by the device services (first layer). The metadata component also manages what commands are available on each device. The last component is called registry and configuration. This component knows all active services in the setup and their configuration. The registry is used when services need to find each other [86].

The third layer is named **supporting services**. This layer provides edge analytics and intelligence. Other tasks such as logging and data cleanup are also performed in this layer. The two most important

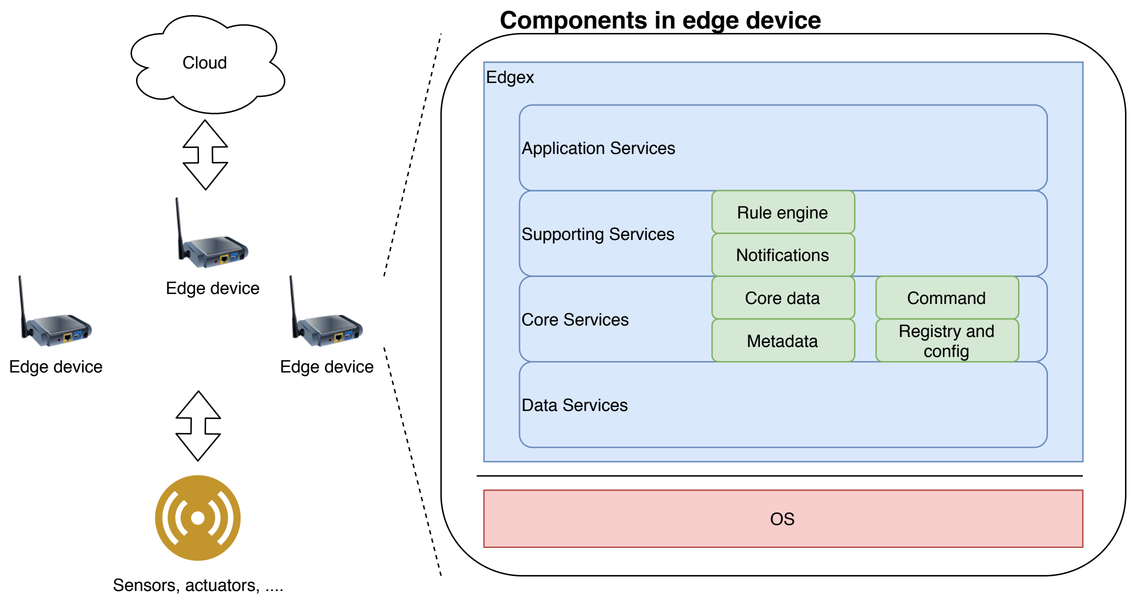


Figure 4.4: EdgeX architecture

components are the rule engine and the notification service. The rule engine can inspect sensor data and issue a command back to a device if necessary. This allows for a quick response at the edge. The notification component can generate a critical alert or informative notifications and send those to a subscriber. EdgeX currently supports mail and HTTP callback notifications [86].

The **application services** layer is the last layer and is responsible for transforming EdgeX data and sending it to an endpoint of choice. This layer is relatively new and replaces the older export layer. The EdgeX data is transformed using a pipeline of functions before sending it to the endpoint, which is usually the cloud. Different protocols can be used such as REST or MQTT [86].

Two other type of services are **security services** and **management services**. The security services protect the data and commands managed by EdgeX. The system management services provide an external interface for managing en monitoring the services [86].

4.4.2 Experiment

All services described in the previous section have their docker container. All containers including their setup are provided in a docker-compose file which can be used to start EdgeX with docker-compose [2]. A web GUI is also provided which can be easy to get started.

The next step is to configure the format of all the data received from and send it to an IoT device. For example, the description of the outdoor temperature would say it is a float, it can range from -60 to 80 degrees, is expressed in degrees Celsius and defines the outdoor temperature. By defining what the data looks like EdgeX can parse the data. And EdgeX can pass this information to other services so they know the format and the meaning of the data.

Now we need to select the right device services to be able to communicate with the sensors and actuators. A custom device service can be built in C or Go using a provided setup or it can be built from scratch. Some device services already exist MQTT and REST. If MQTT is used, the data must be published on a specified topic by your sensor. Commands will be sent on another topic. The REST device service currently only supports sensor data and cannot send commands to actuators [87]. A well-implemented device service can autodetect new IoT devices. If not, a list of devices can be provided in a configuration file loaded by the device services. Take for example a device service that communicates with a Zigbee thermometer. The device service will parse the Zigbee data and convert it to the data format that has been agreed with EdgeX.

The next step is to configure the northbound communication. This will usually be a device in the cloud

[2]`https://docs.docker.com/compose/`

to store sensor data. Multiple cloud setups are provided such as Azure or Google. But also a custom setup is possible. A protocol (REST or MQTT), IP address, port and login must be provided.

Communication from the sensor to the cloud and vice versa is now possible. Sensors can pass data to EdgeX, who passes it on to the cloud. And a command can be initiated from the cloud, send to an EdgeX edge device who passes it to an actuator.

4.4.3 Edge Capabilities

At this moment EdgeX does little to no management of the edge devices themselves. This is a big disadvantage in the setup. In future releases, they are planning on bringing more management features to the setup [86].

EdgeX defines communication protocols that can be used to connect to the cloud or end-devices. However, the end-device component or cloud component is not provided by EdgeX and should be implemented by the developer.

By default, EdgeX comes with a basic rules engine. Like all other EdgeX components the rule engine can be accessed through a REST API. The rule engine uses Drools [3] as its core [86]. Rules are created by defining the corresponding device, the property on which the rule applies, a check and a corresponding action. The check consists of two operands and an operator. In the action, a device and a command are specified which is executed if the check is value. An example is: IF the temperature of device *X* is GREATER THAN *25*, THEN send command *setAirco('on')* to device *Y*.

This rule engine is basic but is enough to perform simple actions. There are also situations where more advanced rules are required. A more advanced action could be: if 20% of the crops have not enough water, turn on the irrigation system and give a variable amount of water to each section of the field depending on their needs. Since EdgeX is an open-source system all components can be replaced, this is also the case for the rules engine. As said in Section 4.4.1, all components communicate through REST APIs, this makes it easier for novice programmers to create the necessary components. All APIs can be found in the online documentation of EdgeX.

4.5 Azure IoT Edge

Azure IoT Edge is a service from Microsoft that brings cloud workloads to the edge. Azure IoT Edge provides store and forward communication to the cloud and back. In comparison to EdgeX the communication flow is more programmable by default. With the Azure IoT Hub the management of the IoT Edge devices is done in the cloud. When the IoT Edge is installed and connected to an Azure IoT Hub no direct interaction with the IoT Edge device is needed anymore. All IoT devices such as sensors and actuators are managed from the cloud in the IoT Hub. To speed up the development of IoT solutions Azure provides templates of common IoT solutions such as tracking and inventory. Docker containers are used install the IoT logic on the IoT Edge devices [88].

4.5.1 Architecture

The architecture of an Azure IoT Edge setup is shown in Figure 4.5. The IoT Edge runtime is installed on an edge device. An edge device can be very small, such as a Raspberry Pi, or can be a big industrial server depending on the needs. In the cloud, the Azure IoT Hub manages all the IoT devices and the Azure IoT Edge devices [85].

An edge device runs on an operating system of choice. The **IoT Edge runtime** component makes it an Azure IoT Edge device. The runtime consists of multiple modules, which are called runtime modules. Two modules belong to the system and the rest of the modules are application-specific. The first preinstalled module is the **IoT Edge Agent**. This module is responsible for the deployment and monitoring of application-specific modules. These modules are listed in a deployment manifest as a docker image. The agent will download the image, start the image and report the status back to the IoT Hub [85].

The second preinstalled module is the **IoT Edge Hub**. This module is responsible for all communication. It must not be mistaken with the IoT Hub which runs in the cloud. Both the IoT Edge Hub and

[3] `www.drools.org`

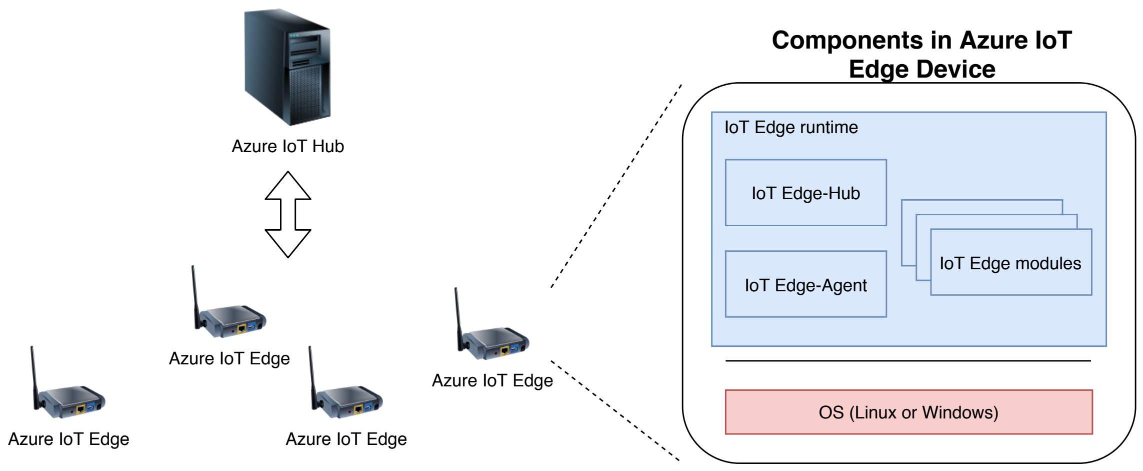


Figure 4.5: The architecture of an Azure IoT Edge setup.

IoT Hub have similar functionality, in fact, the IoT Edge Hub has the same interface as the IoT Hub in the cloud. The IoT Edge Hub decides what messages are passed along to the cloud and what message can be handled on the edge device itself. To enable offline capabilities, the IoT Edge Hub stores the messages until a connection has been reestablished. The IoT Edge Hub module is also responsible for module-to-module communication. Application-specific modules specify a URI on which they accept messages and on which they output messages. These input and output channels can be linked with each other or passed to the cloud. These links are called routing rules [85].

4.5.2 Experiment

The first step is to create an IoT Hub and register an edge device in the Azure cloud. The registration of the edge device will give a connection string, which is used to connect the IoT Edge device to the IoT Hub. On the edge device, IoT Edge can be installed and configured with the connection string. From this moment the device will be connected to the cloud and no further configuration on the device is necessary. Monitoring and the deployment of modules can be initiated from the IoT Hub in the cloud.

The next step is to pick or develop modules to deploy to the edge devices. There is a large set of modules already available such as a rule engine for filtering messages, Stream analytic engines, vision AI modules and Anomaly detection modules [89]. Custom modules can be build in C, C#, Java, Node.js and Python [85]. The Azure sdk can be used to subscribe to messages from sensors, other modules, and messages from the cloud. It is also possible to publish messages. Message streams are identified by a URI. The flow of messages is defined by routing rules. A routing rule links two message flows together. To push modules to IoT Edge devices, which are docker containers, they must be uploaded to a docker registry online.

4.5.3 Edge Capabilities

Azure IoT Edge provides a full-stack solution for edge applications. The monitoring of end-devices and edges devices and the management of jobs can be done from the Azure dashboard. A drawback is that the software stack offered for the edge devices does not include an operating system. As explained in the previous sections a docker container must be deployed on the edge devices. This can be an advantage if the device also runs other applications but it is a drawback if it is a standalone device. The OS must be kept up to date and monitored and this task is not performed by Azure.

Communication between end-devices and edge and further to the cloud is handled securely by Azure. Azure also provides offline support in case the internet connection is not reliable between the edge and the cloud [85].

The application logic is written in docker modules which can be deployed on the edge devices. Since all the application logic is defined in code a wide variety of applications can be developed. This is a very powerful setup in the case complex logic is needed. But it takes more time to develop simpler logic such as if-this-then-that logic.

4.6 Balena

Balena provides a set of tools to manage, develop on and deploy on a large fleet of Linux devices. This makes it very useful in IoT edge development. Balena provides an OS which can be installed on the edge devices, it is called balenaOS. BalenaCloud, as the name suggests, runs in the cloud and is the central component on which the edge devices are managed and software is deployed. BalendaCloud is a paid service, but it is also available as an opensource tool: OpenBalena. Docker containers are used to deploy IoT applications on the edge devices. [90]

4.6.1 Architecture

An overview of the devices and component in a Balena setup is shown in Figure 4.6.

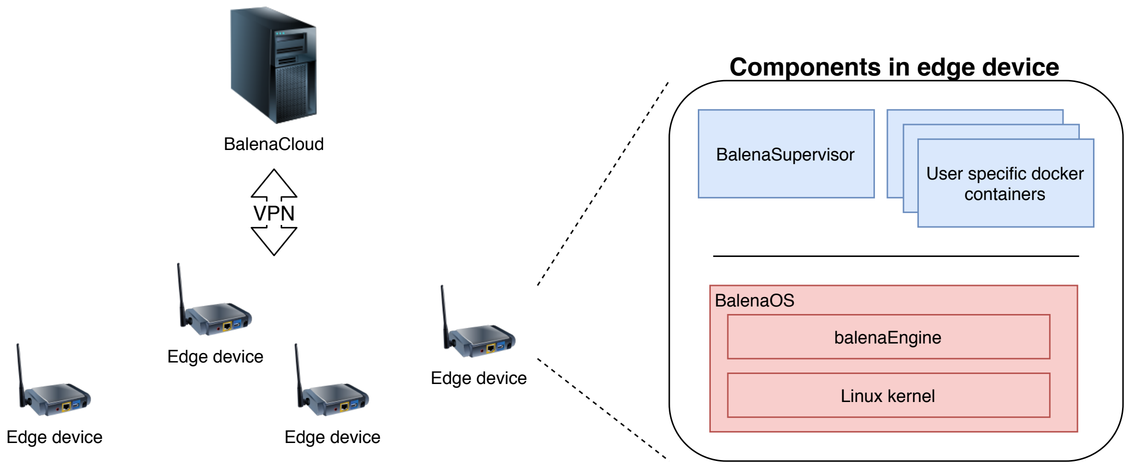


Figure 4.6: Overview of the Balena architecture using the balenaCloud and balenaOS device [91].

BalenaOS is Linux based OS that must be installed on the edge devices. All the management tasks are performed in the **balendaCloud**. An application can be created in the balenaCloud. Devices can be linked to exactly one application [91].

An edge device consists of multiple components. The components are shown on the right side of Figure 4.6. The **balenaEngine** is an open-source docker-engine developed by Balena. A docker-engine provides a mechanism to run docker containers. The balenaEngine promises to be more resource-friendly than the default docker setup [90]. The rest of the components are added as docker containers that run on top of the balenaEngine. The **supervisor** stays in contact with the balenaCloud. OpenVPN is used to encrypt all outgoing network traffic. The supervisor is responsible for monitoring and making changes to the edge device. If for example, a new version of the OS is available, it will download and install the update. The last component consists of docker containers that are deployed on the device by the user. These containers are application-specific. It is the task of the supervisor to download the docker container, stop/stop it and report the status back to the user [91].

4.6.2 Experiment

The first step is to create an application on BalenaCloud or OpenBalena. From there on out, new devices can be added. Balena provides a fully setup image for a large set of devices. This image can be installed on the edge devices for example a Raspberry Pi or an Intel NUC. When the devices are powered on and provided with an internet connection, they will automatically connect to the balenaCloud. The devices are now in full control of the balenaCloud and no interaction is needed anymore with the device. Their state, IP, software versions, services, device type, logs, etc. are provided in the dashboard. An ssh session to an edge device can be established to solve a device-specific issue.

The next step is to develop an application in a docker or docker-compose environment. Public ports in the docker container are reachable from the outside the device. An edge device can be switched in local mode to enable local development. By pushing to a specified git branch the containers are pushed onto all the devices in production mode.

4.6.3 Edge Capabilities

Balena is built for device management and deploying software on those devices. This makes Balena ideal for the management of edge devices. Balena supports a large number of devices and the devices are kept fully up to date. This ensures that no human intervention is needed at the device itself. But Balena on its own is not enough, in an edge device, more functionality is needed. Think about data going from sensors to the cloud and commands from the cloud to IoT actuators. Think about sensor management and the different communication methods they use. Therefore this technology should only be used in combination with other technologies that provide those missing requirements.

4.7 FogFlow

FogFlow is an IoT fog computing framework that helps to manage and devise tasks over a large set of FogFlow nodes. The framework provides edge software and cloud software for a full-stack edge solution [92].

4.7.1 Architecture

The architecture of FogFlow is shown in Figure 4.7. The main component runs in the cloud. The fog nodes can run on edge devices. The fog nodes can communicate with the IoT end-devices. The application logic is contained in fog functions. A **fog function** is a docker container with some logic. The input and output data of a fog function have a well-defined format (entity). The fog functions are deployed on the edge devices where the data is present. For example, if a fog function takes temperature data as input, the function will only be deployed when a sensor reports the temperature to the system. And the function will only be deployed on the edge device that is the closest to the temperature sensor. This is in contrast to other edge technologies such as EdgeX and Azure IoT Edge where a fixed set of functions is deployed [92].

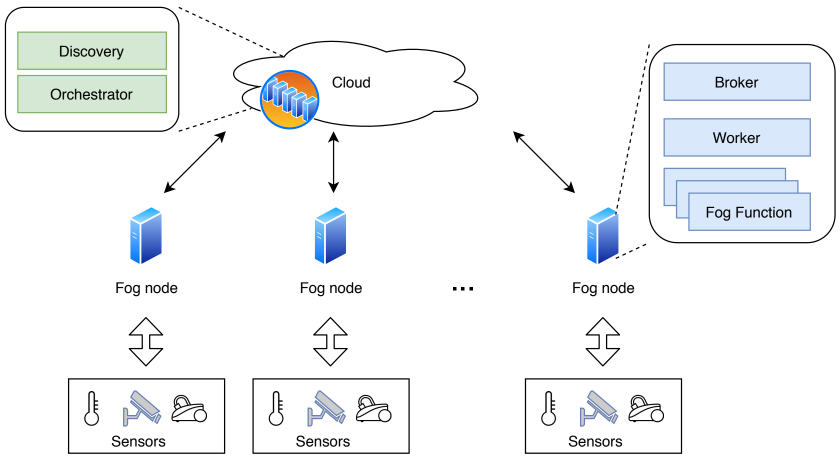


Figure 4.7: FogFlow architecture. FogFlow nodes can be edge devices. Those communicate with other edge devices and the cloud. The main component runs in the cloud and consists of a discovery component and an orchestrator. Fog function run the application logic which can be deployed on fog nodes [93].

Each node consists of a broker and a worker. The **worker** starts and stops fog functions on the fog node itself and it manages the data flow between fog functions. The **broker** manages all data formats or entities. The main component is usually placed in the cloud. It consists of a discovery component and an orchestrator. The **orchestrator** manages all the fog functions. It keeps a list of the functions that are available together with their input and output. The orchestrator also decides which fog function is deployed on which fog node. A fog function will be deployed to the fog node that is the closest to the end-device. The **discovery component** can be used to discover fog nodes, data and available resources at fog nodes. The brokers work together with the discover-component to provide de necessary information [92, 93].

The components work together in the following way [93]:

1. A developer registers a fog function, with a well-defined input - and output format, at the orchestrator.
2. The orchestrator subscribes for the input format of the fog function at the discovery component.
3. A broker on a fog device detects new data. *The new data can be generated by a sensor or can be the output data of another fog function.*
4. The broker does not have a corresponding fog function. Hence it reports the data format to the discovery component in the cloud.
5. Since the orchestrator is subscribed to this input data format, it is notified.
6. The orchestrator will deploy the fog function on the fog node that is the closest to the data source.
7. The worker of the selected fog node will download and start the container of the fog function.
8. For the generated output data the process is repeated.

4.7.2 Setup and Workflow [92]

The components described in the previous section are all containerized in Docker environments. To make development easier, a designer GUI component is added. From here the full architecture is shown including end-devices, fog node, brokers, worker, etc. Fog functions can be added from the dashboard and also their status can be shown.

The typical steps would be to deploy the containers on the corresponding devices. The broker and worker are usually deployed on an edge device, the discovery and orchestrator (master) are usually deployed in the cloud. The GUI container can also be deployed in the cloud for development purposes. Sensor devices can be added by sending an HTTP request to the broker.

The next step is to develop the application logic. This is done by creating a fog function. Therefore a docker environment must be created which uses the FogFlow SDK to process the input data and report the output data. To be able to use the fog function, the container should be published on DockerHub. Next, the fog function is registered at the FogFlow orchestrator together with its input data format. The data format is for example Temperature data. When this data format is detected the fog function will be deployed.

FogFlow also provides a monitoring setup. This setup uses Metricbeat, Elasticsearch and Grafana to extract CPU, network, ram, disk usage and other statistics from the fog nodes and show them in a dashboard.

All the containers are also available in a docker-compose file for a single-device setup.

4.7.3 Edge Capabilities

FogFlow is a full-stack fog computing framework, it provides a management tool for fog node and end-devices and there is also a monitoring setup as described in the previous section. Also, the communication between end-devices, fog nodes and the cloud is completely handled by FogFlow. Just like Azure IoT Edge, FogFlow does not provide management for the OS of the fog nodes [92].

The application logic is defined in code which uses the FogFlow SDK to communicate with the FogFlow network as described in the previous section. This is just like in Azure IoT a very powerful setup where more difficult rules can be implemented.

4.8 Discussion

FogFlow and Azure IoT Edge are two full-stack IoT edge/fog technologies. Both provide management tools and handle the communication between end-devices, edge and cloud. However, both tools do not provide management capabilities for the operating system. The main difference between Azure IoT Edge and FogFlow is in the way applications are pushed to the edge or fog nodes. In Azure IoT Edge, every application is deployed to all the edge devices in that solution while in FogFlow, Fog functions are only pulled to the fog node that needs the function. FogFlow is in this case more flexible and resource-friendly. Another difference is that Azure IoT Edge is only partially open source and heavily depends on the Azure Cloud while FogFlow is completely open-source and can be deployed on a system of choice. Both solutions have their pro and cons, Azure IoT Edge has better support, better documentation and their management platform is more user friendly while FogFlow is free of charge, is managed by the open-source community and does not have a vendor lock-in.

EdgeX only provides a component for a single edge node, it does not perform the management tasks Azure IoT Edge and FogFlow do. The main difference between on one hand EdgeX and the other hand Azure IoT Edge and FogFlow, is the way messages are defined and application logic is implemented. In EdgeX, application logic is implemented by rules while in Azure IoT Edge and FogFlow application logic is defined in code. The first makes the development process easier, the second makes application logic more powerful. The format of messages in EdgeX is predefined in contrast to Azure IoT Edge and FogFlow, which makes communication and data more structured.

Balena only performs management and deployment of applications to devices. This is however performed on OS level, which means the full device can be managed remotely. This makes Balena ideal to combine with EdgeX, Azure IoT Edge or FogFlow. The combined solution has the capabilities to manage the edge/fog devices from OS to the application level and have the benefits of communication tools, application logic and end-device management.

Chapter 5

Usage Scenario

In Chapter 3 we have surveyed IoT communication technologies and in Chapter 4 we have surveyed edge technologies. We now implement a use case using the lessons we learned in both survey studies.

Modern electrical power grids have a lot of variation in production supply and customer demand. Typical customer demand is higher during the day than the night with a peak in demand during the evening. Next to consumption peaks, production peaks can also happen, especially during a windy or sunny day due to the presence of solar panels and wind turbines. This leads to a high cost for the retailer because the grid has to be capable of those high variations. Low production in the case of no wind or solar also has a negative impact on the reliability of the power grid. But currently, consumers are still charged a constant price [94]. Looking to the future this problem will only amplify due to an increase of electrical cars, electric heat pumps, fields of solar panels, bigger wind turbines, etc. In an ideal scenario, the electricity production and consumption should be equal at all times. This enhances the efficiency of the power grid which also reduces the total cost of electrical energy.

We focus on the client-side of the problem. The objective is to minimize the electrical bill in a household. We created a predictive control device for households that minimizes their electricity costs. It is based on real-time price prediction and the weather forecast. We determine the best schedule for each electrical device controlled by a central unit using Zigbee. An overview of the setup is shown in Figure 5.1.

Since July 2019, the Belgian energy retailer “Fluvius” started rolling out digital energy consumption measurement devices such as the T211-D. This device is capable of measuring power consumption in real-time [95, 96]. Due to real-time measurement, energy retailers are capable of charging households with a real-time electricity price. It is the task of the retailer to implement this.

5.1 Related Work

A cloud-enabled thermostat was created by Perez et al. [97] to control the electrical home heating system. The indoor temperature, outdoor temperature, solar radiation and the heater state are used to predict the indoor temperature for the next 24 hours. The temperature is allowed to vary in a range of 2 °C around a fixed point. They ran an experiment using an actual house for 3 days in a row and managed to reduce the heating cost by 31%.

Biegel et al. [98] used an aggregator to control a portfolio of domestic electrical heat pumps. The indoor temperature, outdoor temperature, the input power of the heat pump, the daily load profile and some disturbance are used to predict the indoor temperature. The temperature is also allowed to vary 2 °C around a fixed point. A simulation is carried out based on real heat pump data, historical electrical prices and predictions. They showed savings of 18 % and 20% if they strategically bid in on the electricity market.

In Qian et al. [94] a real-time pricing scheme is proposed to reduce the peak load in smart grids. The scheme consists of a retailer and a house. The price is decided after a limited number of messages between the retailer and houses. The retailer will optimize its profits using a simulated annealing-base price control algorithm. The retainer will announce a price proposal to the house that returns the predicted energy usage. Based on this usage the retailer continues to make a new price proposal until

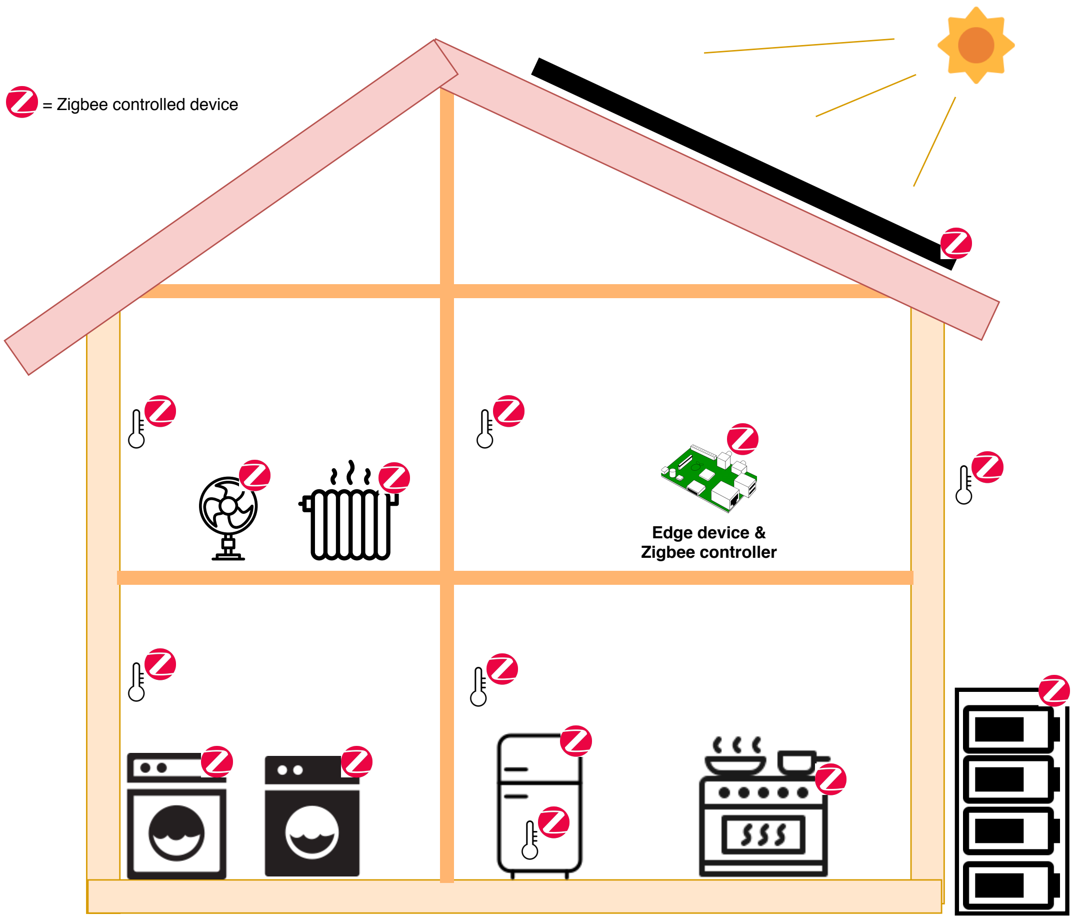


Figure 5.1: The architecture of the electrical device management usage case. The central component in this case is a raspberry pi which also forms the Zigbee controller. The other devices such as the solar panels, battery, fridge, washer, dryer, heater and cooler are controlled by the central component to optimize the electrical bill.

its profit is maximized. In the house, the predicted energy usage is formed by creating an optimal schedule for each device based on the price announced by the retailer. The optimal schedule is a trade-off between comfort and the price to pay.

5.2 Architecture

To make a predictive control device that minimizes the electricity costs, we used the following architecture. The implementation consists of three main components. First, there are the devices that need to be controlled. Then, there is a set of sensors that monitor the house to make predictions such as temperature sensors and power consumption sensors. The last component is the main controller which is the brain of the operation. The controller will use the sensor data and the wishes of the user to create a device schedule that minimizes the electrical bill. An overview of the devices is shown in Figure 5.1.

The main "brain" controller will reassess the situation every 15 minutes. This means every 15 minutes a new prediction is made and the controller will send a new schedule to each device. A device schedule is a vector with for every state a boolean whether this device is turned on or off. A prediction is made for the next 24 hours. This gives us a reasonable perspective of which devices can be scheduled while the number of calculations remains limited. As we will see in Section 5.7 a typical day will consist of a low price at night and a high price during the day. Hence 24 hours allows the main controller to schedule devices during low prices at night.

5.3 Communication Protocol and Edge Technology

The application is mostly indoor and hence an indoor communication technology should be used. In Section 3.7 we concluded that Zigbee is well suited for indoor IoT applications. We use Zigbee in this setup because the network can support a large number of devices, its communication is relatively secure and there is a large amount of Zigbee compatible devices on the market.

In this application, the central component is located at the edge, in the house of the user and no application logic must be performed in the cloud. It is however essential to be able to update application logic on the edge device and update the device itself. As discussed in Section 4.8 this task is best performed by Balena.

5.4 Sensors and actuators

The following devices are included in the design of the main controller:

- Central heater
- Central cooler
- Refrigerator
- Laundry and dish washer
- Solar panels
- Battery
- Kitchen stove

These devices are chosen because they are an important share in the electrical bill of a typical household [99], see Figure 5.2. The first five devices are also controllable in a reasonable period. It does not matter for example if the dishwasher runs immediately or during the night. Devices such as the heater and cooler can also operate whenever it suits them as long as the temperature stays in a certain range. The solar panels are taken into account because they can produce a large amount of energy which is dangerous for overloading the electrical grid. By including them in the design, the main controller can use the energy indoor and hence the current should not flow back to the grid. The battery device will become an important factor in energy management because it can spread out the energy produced by the solar panels during the days to consumers in the evening and night. Therefore the battery can lower the load on the electrical grid and hence this device is also taken into account. Devices such as a TV and a PC are not included in the design because when their usage depends on the user.

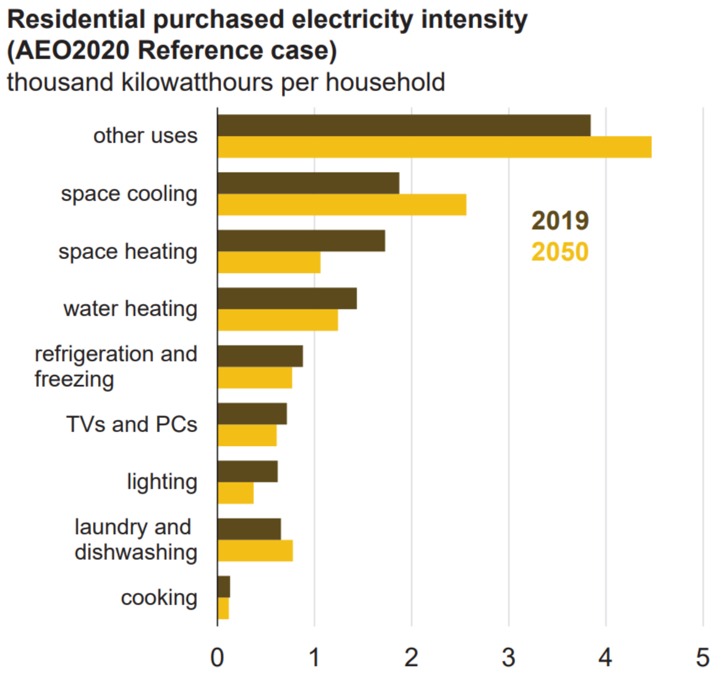


Figure 5.2: Residential electricity purposes. Source: [99]

Sensors are needed to optimally control these devices. For the central heater and cooler, the temperature in the house must be kept within a certain range defined by an upper and lower threshold. In Perez et al.[97] and Biegel et al.[98] a range of 2 °C around a center point is used. The indoor temperature is influenced by several factors. In Perez et al.[97] the following factors are used to predict the indoor temperature:

- Current indoor temperature
- Current outdoor temperature
- An indication of whether the heater is turned on
- Solar radiation

The indoor and outdoor temperatures are measured by two Aquara Xiaomi Zigbee temperature sensors. The solar radiation is determined using a weather API. By controlling when the heater and cooler are used, the price can be minimized.

The same temperature model can be used to optimize the electrical bill of the fridge. Notice solar radiation will probably have a minimal impact on the temperature inside the fridge. A third Aquara Xiaomi temperature sensor is placed inside the fridge. To control the fridge a 2300 W INNR Zigbee wallplug is used. A big advantage of this controller is it reports live voltage, ampere and wattage output. This allows real-time monitoring of the connected device.

The washer, dryer and dishwasher are referred to as semi-inelastic appliances by Qian et al. [94]. This category consumes a fixed amount of energy but the period they operate can be chosen freely within a reasonable time. In this implementation, the device will be scheduled in the next 24 hours from that moment. A device must not be interrupted during its program since this can harm the device or its content. The safest way to control these devices is through their API. Some of the more recent washers have their own API [100, 101]. We simulate these APIs with an OSRAM Smart + Zigbee wallplug.

As the main controller, a raspberry pi 3B+ is used with the Phoscon Conbee 2 Zigbee USB dongle. This forms the controller in the Zigbee network. The Conbee 2 is compatible with devices of multiple brands: Philips Hue, IKEA TRÅDFRI, Xiaomi Aqara, OSRAM SMART+, Busch-Jaeger, GIRA, JUNG, Paulmann and Paul Neuhaus. Communication with Zigbee sensors and controllers is enabled through a REST API. The WebSocket API can be used to receive real-time updates from the devices [102] .

5.5 Temperature Prediction

In the previous section, we defined the factors that are used to predict the temperature inside the house and fridge. In Perez et al. [97] the following equation is used to combine the parameters and predict the temperature at time $k+1$ based on the measurements at time k:

$$T_i(k+1) = a * T_i(k) + b * T_o(k) + c * H(k) + d * R(k)$$

wit T_i the indoor temperature, T_o the outdoor temperature, H the heater state en R the radiation of the sun. a, b, c en d are scaling values for the parameters. Because the equation is linear, a linear optimizer will suffice for the temperature prediction.

Scaling factors a, b, c en d depend on thermal resistance, thermal capacity, heating power, transmittance factor of the glass window, window area, and maximum solar radiation [97]. In Perez et al. [97] they are partially estimated and partially optimized based on measurements. In this implementation, we use a Support Vector Regression Machine with a linear kernel to estimate a, b, c en d and a bias e is added. The equation used in this model is:

$$T_i(k+1) = e + a * T_i(k) + b * T_o(k) + c * H(k) + d * R(k)$$

With k we step through time using the fixed step size of 15 minutes. The machine learning model is trained on recorded sensor data of the home it is deployed in. One training sample looks like this:

$$X = [indoorTemp, outdoorTemp, heaterState, solarRadiation], y = nextIndoorTemp$$

In the case of the fridge, the indoor and outdoor temperature refers respectively to the temperature inside and outside the fridge.

5.6 Weather Prediction and Solar Prediction

A weather API is used to get the forecast for the next 24 hours. Because in many free weather-APIs the solar radiation is missing we use cloud cover as an alternative for solar radiation. Some accuracy will be lost but generally, these values depend on each other. If there are clouds, the solar radiation will be low and if there are no clouds, the solar radiation will be high.

To take full advantage of the solar panels an accurate prediction of the energy production is necessary. The amount of power produced by solar panels is related to the amount of solar irradiation on the surface. Hence the angle with the sun is an important factor, but also the temperature and cloud cover [103].

5.7 Electricity Price Prediction

An hourly price forecast is currently not available for customers in Belgium. But every day a day-ahead price is published on the entsoe platform[1] as a result of bidding procedure. The price is also available through an API[2]. In this implementation a fixed model is used for the price prediction. The model is based on a 3 year history of day-ahead prices. The history is also available on the entsoe platform. This 3 year daily average is show in Figure 5.3.

The day-ahead price is only a small part of the Belgian electricity bill for households. Other parts are distribution and taxes. Only 9,03% of the energy bill is the actual energy price given by the electricity provider [104]. The electricity price for an average household is 0.2837 euro per kWh taken over the last

[1] `transparency.entsoe.eu/`

[2] `transparency.entsoe.eu/content/static_content/Static%20content/web%20api/Guide.html`

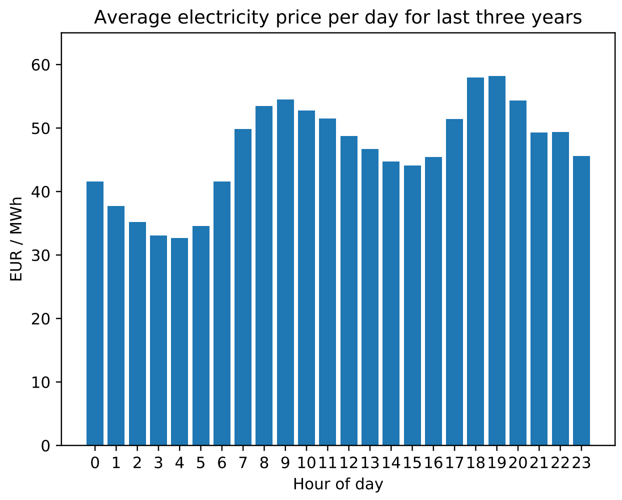


Figure 5.3: The average day-ahead electricity price based on the data published by entsoe for 2017, 2018 and 2019.

3 years [105]. The average day-ahead price over the last three years is 0.0464 euros per kWh. Assuming the remaining parts stayed constant over the years, we can form an average daily model:

$$price = 0.9097 * 0.2837\ EUR/kWh + 0.5449 * dayahead\ EUR/kWh$$

Where 0.5449 is the profit and expense margin of the electricity provider.

5.8 Price Optimization

As in Perez et al. [97], Model Predictive Control (MPC) is used to create the device's schedules. This algorithm works as follows:

1. Record all current sensor values.
2. Find the optimal schedule for each device.
3. Execute the first step in the schedule.
4. Repeat the algorithm at the next time step.

The optimal schedule is formed by making predictions for the next 24 hours in steps of 15 minutes. The result consists of a state for every device at all the time steps for the next 24 hours. It would not be possible to go through all possible schedules since there are too many possible combinations. For the heater alone there are 2^{24*4} possible combinations (on or off at every time interval of 15 minutes). Therefore the search is limited using simulated annealing.

As error function, the total cost of the 24-hour schedule is used. In every step k the power consumption for each device is summed and this total power consumption is multiplied with the predicted price for that step k. Because our model supports solar panels the total power consumption can be negative at a given moment. This can be the case when all electrical devices are turned off and the battery is full on a sunny day. In that case, the price will be negative but we reduce the absolute value with an arbitrary factor of 0.8. This is because delivering power to the grid is not free. This will push the optimizer to use the energy inhouse before putting it on the grid.

In the beginning, a random but valid schedule is created for each device. The heater and cooler will for example be turned on at random moments but keeping the temperature in the given range, the dishwasher, cloth washer and dryer will start at random moments. There are three exceptions: the solar panel will always be turned on, initially the battery is in charging mode and the kitchen stove is always on for 30 minutes in the evening.

Next, the initial state is optimized in several iterations. In one iteration all device schedules are optimized one by one. For each device schedule, a set of possible next schedules is generated while taking the annealing factor into account. The annealing factor describes the amount of randomness used to create the next possible schedules, which decreases with each iteration. We will discuss the meaning of “next possible schedules” by an example of the dishwasher. Say the current schedule turns on the dishwasher at 18:00. In the next iteration, the possible schedules turn the dishwasher on at 17:00, 17:15, ..., 18:45, 19:00. Hence, the possible schedules start the dishwasher ranging from one hour before to one hour after the previous schedule. The width of this window is chosen by the annealing factor. Then for each possible schedule, the total cost is calculated, hence including the cost of all other device schedules, then the schedule with the smallest total cost is selected as the next schedule for the dishwasher. In this case, it is probably the schedule that starts the dishwasher at 19:00 since the electrical price usually drops after 19:00. It is important to note that all possible schedules are valid for that device. The schedule of the heater for example will keep the indoor temperature between a minimal en maximum value and the schedule for the washer will consist of an integral washer program in the next 24 hours.

Notice the solar panels and kitchen stove only have one possible device schedule and hence have no next possible schedules. The battery is handled a little differently from the rest of the devices since it has the capability of storing energy. The battery schedule is created before calculating the total cost of all device schedules. The battery schedule used in this application is the following: it receives all the power from the solar panels that is not used by other devices and delivers the needed power to the first device that would otherwise need power from the grid.

In each iteration, all device schedules become better and the total cost strictly decreases. The iterations stop when the total cost does not decrease more than a certain threshold or the maximum number of iterations is reached. To minimize the possibility of landing on a local cost minimum, simulated annealing is executed multiple times with different initial schedules. The set of device schedules with the minimal total cost is chosen for execution.

5.9 Technologies

The implementation is built in a docker-compose environment. It consists of the following components:

- **Backend**
  The backend is a REST API build in Kotlin using ktor[3]. This component is responsible for creating device schedules and executing them. The temperature models from the house and the fridge are built here using history samples from the database. For training, the linear model and making predictions Smile [4] is used which is a Java library for machine learning. It supports multiple algorithms for classification, regression, feature selection, clustering, ... They promise to deliver state of the art performance. To get the weather forecast of the next 24 hours the darksky weather API is used [5]. Their API is free to use for a limited number of samples and cloud cover is included in the forecast which is important for the temperature model of the house and the energy prediction of the solar panels. To calculate the position of the sun, the commons-suncalc[6] java library is used.

- **Frontend**
  The frontend reports the device schedules to the user. The user can also see the current state of the devices and sensors plus their history. Chartjs [7] is used to show the device schedule and sensor history. Vuejs [8] is used as javascript framework.

- **Database**
  All sensor history and controller history is saved in a PostgreSQL database. This data is used to create models for the heating and cooling devices.

[3]`ktor.io/`
[4]`github.com/haifengl/smile`
[5]`darksky.net/`
[6]`github.com/shred/commons-suncalc`
[7]`www.chartjs.org`
[8]`vuejs.org`

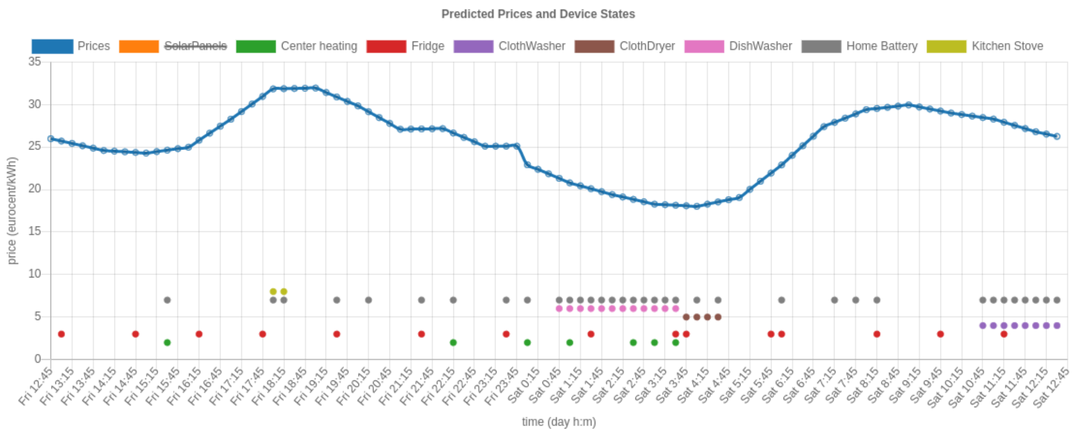


Figure 5.4: Device schedule and the predicted price for the next 24 hours. The blue line represents the predicted electricity price and the dots represent the devices that are schedules at that moment. This screenshot is taken at the same moment as the screenshot in Figure 5.5, 5.6 and 5.7.

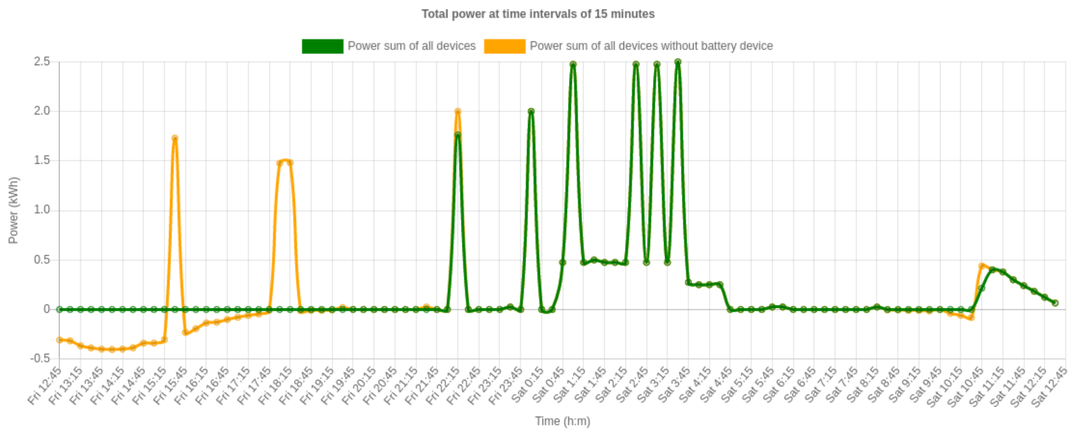


Figure 5.5: The predicted power consumption for the next 24 hours. The orange line shows the total power at each time interval of 15 minutes without the battery. The green line shows same data but with the battery.

- **Logger**
  This component implements the Conbee 2 WebSocket api[9] and saves all samples to the database.
- **ZigbeeComponent**
  This docker environment is delivered with the Conbee 2 Zigbee USB. This component provides bidirectional communication with a Zigbee device through a REST API.

5.10 Discussion

In Figure 5.4 a screenshot is shown from the home page of the application where the device schedule is shown together with the electricity price. The screenshot is taken on a cloudy day and as one can see the dishwasher and cloth dryer are scheduled at night when the electricity price is lower. The cloth washer is scheduled after the morning electricity peak because the solar panels provide enough power at that moment. The fridge is cooled regularly to keep the temperature in the given range. At the same moment, the screenshot in Figure 5.5 is taken where the total power is shown at each time interval of 15 minutes. The orange line excludes the battery power and the green line includes the battery. The battery capacity prediction at that moment is shown in Figure 5.6. As can be seen in Figure 5.5 and 5.6, the battery power is used for the center heater which is scheduled at 15:30, the kitchen stove and the fridge. Then a small amount of power is left for the central heater at 22:00. The predicted

[9]`https://dresden-elektronik.github.io/deconz-rest-doc/`

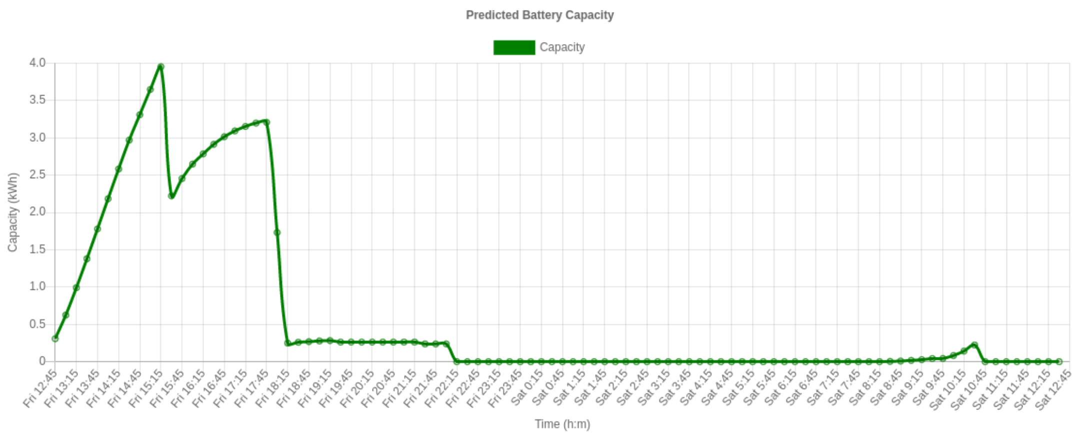


Figure 5.6: The predicted battery capacity.

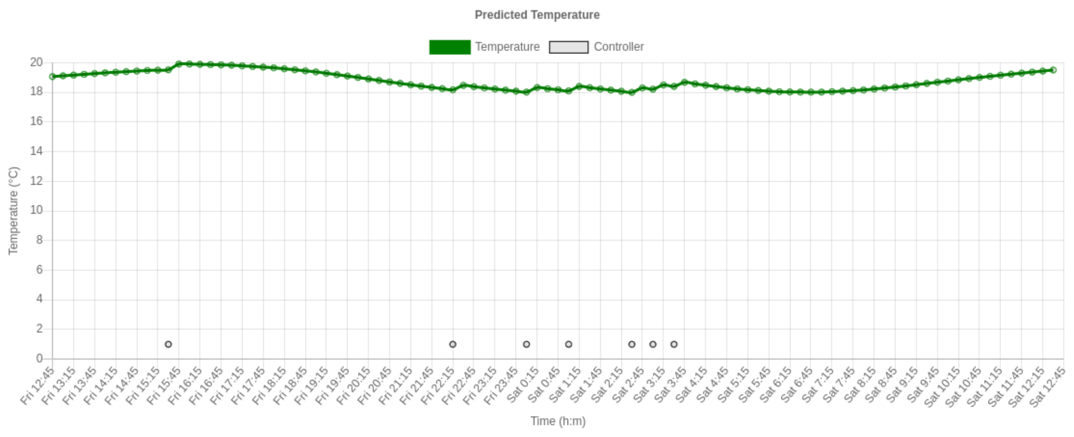
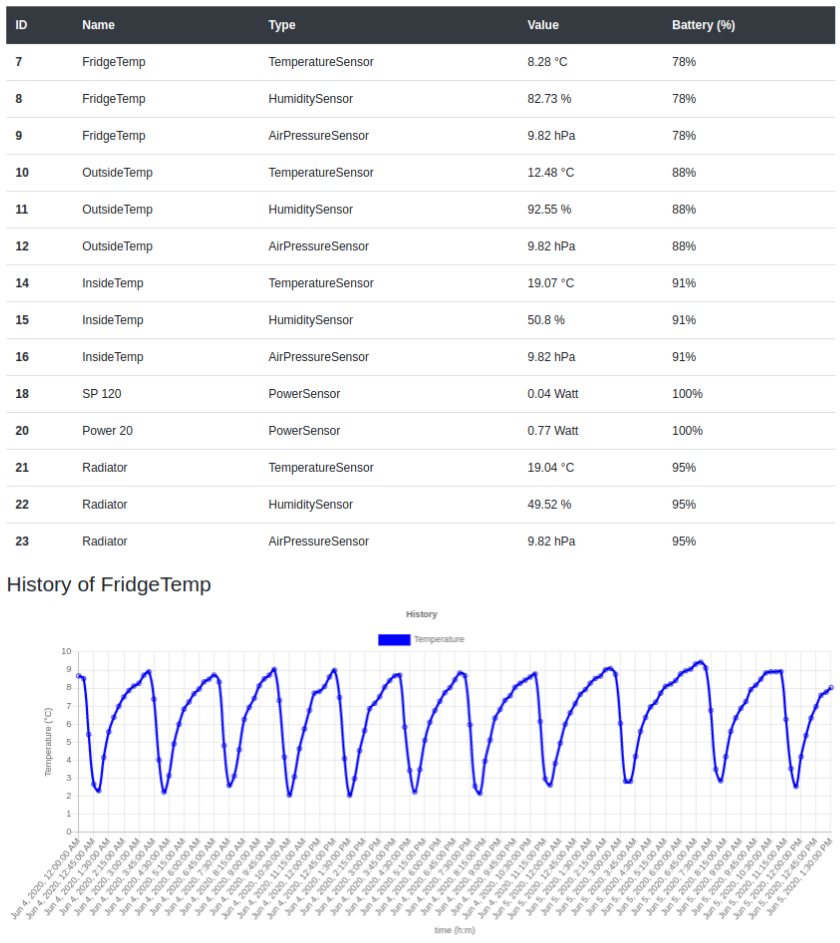


Figure 5.7: The predicted indoor temperature. The green line shows the predicted indoor temperature, the dots indicate when the central heater is active.



ID	Name	Type	Value	Battery (%)
7	FridgeTemp	TemperatureSensor	8.28 °C	78%
8	FridgeTemp	HumiditySensor	82.73 %	78%
9	FridgeTemp	AirPressureSensor	9.82 hPa	78%
10	OutsideTemp	TemperatureSensor	12.48 °C	88%
11	OutsideTemp	HumiditySensor	92.55 %	88%
12	OutsideTemp	AirPressureSensor	9.82 hPa	88%
14	InsideTemp	TemperatureSensor	19.07 °C	91%
15	InsideTemp	HumiditySensor	50.8 %	91%
16	InsideTemp	AirPressureSensor	9.82 hPa	91%
18	SP 120	PowerSensor	0.04 Watt	100%
20	Power 20	PowerSensor	0.77 Watt	100%
21	Radiator	TemperatureSensor	19.04 °C	95%
22	Radiator	HumiditySensor	49.52 %	95%
23	Radiator	AirPressureSensor	9.82 hPa	95%

History of FridgeTemp

Figure 5.8: Overview of sensors with their data history.

indoor temperature is shown in Figure 5.7. As can be seen, the central heater is schedule first at 15:30 where it is allowed to use solar energy. Then the central heater is scheduled again a few times in the evening and night where it is needed to prevent the temperature from going below 18 degrees. The next day, the model predicts that the house will heat enough thanks to the outdoor temperature and solar radiation.

The application also allows you to see the sensors and controllers which are used together with their data history. An overview of the sensors is shown in figure 5.8.

The works of Perez et al., Biegel et al. and Qian et al. [94, 97, 98] show it is possible to reduce the electrical bill of households and reduce the supply and peak demand. Our work validates this and also shows that the current IoT technologies are ready for this application. Technologies such as Zigbee allows the sensors to provide the necessary information and send commands to the actuators to turn on or of devices. Some household devices even have built-in support for remote control. Technologies such as Balena allow the deployment of IoT applications on large scale and keep the applications and devices up to date. There is however at the time of writing no variable price contract offered by retailers, which is a must for this application to work. We believe however it is in the best interest of the retailers to offer these contracts in the future to reduce the ballast on their electrical power grids.

Chapter 6

Conclusion

We started this thesis with a Survey on IoT communication technologies where we explained their inner working, security and vulnerabilities. Sigfox, LoRaWAN, NB-IoT should be used for outdoor applications. We think Sigfox is useful for monitoring applications where the state of the environment can be reported in a few very small messages to the cloud. LoRaWAN can send more and bigger messages and can therefore be used in applications where communication in both directions is required. Due to the lower latency, LoRaWAN can also be used in more urgent applications. Based on the vulnerabilities, we conclude the LoRaWAN security can be trusted more than the Sigfox security. NB-IoT and LTE-M are two protocols for applications where more bandwidth and data is required. Since these protocols require more energy, we think they should not be used in simple monitoring applications. Zigbee, Z-Wave and Thread should only be used for indoor applications or outdoor applications with a limited range. We think Zigbee and Thread are more future proof since they support more devices per network, Z-Waves limit of 232 devices seems small as the number of indoor IoT applications will only increase. Since Thread has the least amount of vulnerabilities and its network data is authenticated and encrypted by default, we conclude Thread is the most secure protocol of the three. In our opinion, Z-Wave has some serious security vulnerabilities and should therefore not be used in applications that include sensitive data or perform security-related tasks.

In the second part of this thesis, we conducted a Survey on IoT Edge technologies where we explained for each technology their architecture and their working in practice. We compared EdgeX, Azure IoT Edge, Fogflow and Balena. Balena and Azure IoT are the technologies that are the easiest to set up. They have good documentation and a limited number of components which makes them clear and easy to use. When looking to their capabilities we see that Azure IoT and FogFlow provide a full-stack solution, meaning the frameworks have components in de edge and cloud and can therefore manage IoT devices and their communication. EdgeX does not have that overall management component but this can be created easily. Balena should not be used on its own if communication to the cloud is necessary. It is however very valuable if Balena is used in combination with the other technologies to manage the OS of the edge device.

Finally, we created a price optimization application for indoor electrical devices as a usage scenario. This application shows it is possible with current IoT devices to create an application that controls indoor electrical devices and uses the predicted electricity price to reduce the electricity bill for consumers. We believe this type of application can reduce the peak ballast on the electrical grid and therefore make it less expensive and more future proof.

6.1 Future work

While we tried to include the most popular and most promising technologies in our survey, there are other technologies such as Bluetooth Low Energy, RFID, NFC and ANT(+) that were not included. It can be of interest to take other protocols into account and compare them with the current listing. The same is true for the IoT edge survey, where technologies such as Amazon Greengrass and Google Cloud IoT were not included.

Acronyms

3GPP 3the Generation Partnership Program. 28

AES Advanced Encryption Standard. 19, 25

CRC Cycle Redundancy Check. 18, 21

CSMA/CA Carrier Sense Multiple Access Collision Avoidance. 32, 38

CSS Chirp Spread Spectrum. 20

D-BPSK Differential Binary Phase Shift Keying. 16

DoS Denial of Service. 19

DTLS Datagram Transport Layer Security. 44

FED Full End Device. 41

FFD Fully functional device. 32

FSK Frequency Shift Keying. 16, 38

FTD Full Thread device. 41

GFSK Gaussian Frequency Shift Keying. 16, 38

GSM Global System for Mobile Communication. 27

IEEE Institute of Electrical and Electronics Engineers. 15

IoT Internet of Things. 13

IP Internet Protocol. 15

IPv6 Internet Protocol version 6. 41

ISM Industrial Scientific and Medical. 20

LoRa Long Range. 20

LoRaWAN Long Range Wide Area Network. 20

LTE Long Term Evolution. 27

MAC Message Authentication Code. 40

MAC Medium Access Control. 17

MIC Message Integrity Code. 23

MLE Mesh Link Establishment. 44

MME Mobility Managment Equipment. 28

MPDU MAC Protocol Data Units. 38

MQTT Message Queuing Telemetry Transport. 53

NAK Network Authentication Key. 19

NONCE Number - used Once. 25

O-QPSK Offset Quadrature Phase-shift keying. 32

OTAA Over-The-Air Activation. 24, 25

PAN Personal Area Network. 32, 45

PDCP Packet Data Convergence Control Protocol. 29

REED Route Eligible End Device. 41

REST Representational State Transfer. 53

RFD Reduced function device. 32

Routing Locator Routing Locator. 42

SIM Subscriber Identity Module. 28

TC Trust Center. 35

UDP User Datagram Protocol. 43

UE User Equipment. 28

WAN Wide Area Network. 45

ZC Zigbee Controller. 33

ZED Zigbee End Device. 33

ZR Zigbee Router. 33

Bibliography

[1] Intel. *Guide to the internet of Things.* https://www.intel.com/content/dam/www/public/us/en/images/iot/guide-to-iot-infographic.png. (Accessed on 06/09/2020).

[2] International Data Corporation. *IDC's Global DataSphere Forecast Shows Continued Steady Growth in the Creation and Consumption of Data.* https://www.idc.com/getdoc.jsp?containerId=prUS46286020. (Accessed on 05/09/2020). 2020.

[3] Cisco. *Cisco Annual Internet Report (2018–2023) White Paper.* https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf. (Accessed on 05/09/2020). 2020.

[4] Perry Lea. *Internet of Things for Architects.* Packt Publishing, 2018.

[5] Sigfox. *Sigfox Device Radio Specifications.* https://build.sigfox.com/sigfox-device-radio-specifications. (Accessed on 03/24/2020). Feb. 2020.

[6] Tutorialpoint. *Differential Phase Shift Keying.* https://www.tutorialspoint.com/digital_communication/digital_communication_differential_phase_shift_keying.htm. (Accessed on 03/23/2020).

[7] Sabih H Gerez. "Implementation of digital signal processing: Some background on GFSK modulation". In: *University of Twente, Department of Electrical Engineering* (2013).

[8] Tutorialpoint. *Differential Phase Shift Keying.* https://www.tutorialspoint.com/digital_communication/digital_communication_frequency_shift_keying.htm. (Accessed on 03/23/2020).

[9] MIT 6.02 staff. *Convolutional Coding - Lecture 8 - MIT 6.02.* http://web.mit.edu/6.02/www/f2010/handouts/lectures/L8.pdf. (Accessed on 03/25/2020). Oct. 2010.

[10] Kais Mekki et al. "Overview of cellular LPWAN technologies for IoT deployment: Sigfox, LoRaWAN, and NB-IoT". In: *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops).* IEEE. 2018, pp. 197–202.

[11] Sigfox. *Secure Sigfox Ready devices - Recommendation guide.* https://www.aerea.nl/wp-content/uploads/2018/06/Secure-Sigfox-Ready-devices-recommendation-guide-II.pdf. (Accessed on 03/25/2020). June 2018.

[12] Sigfox. *Make things come alive in a secure way - Security white paper.* https://www.sigfox.com/sites/default/files/1701-SIGFOX-White_Paper_Security.pdf. (Accessed on 03/25/2020). Feb. 2017.

[13] Florian Laurentiu Coman et al. "Security issues in internet of things: Vulnerability analysis of LoRaWAN, sigfox and NB-IoT". In: *2019 Global IoT Summit (GIoTS).* IEEE. 2019, pp. 1–6.

[14] Eirini Karapistoli et al. "An overview of the IEEE 802.15. 4a standard". In: *IEEE Communications Magazine* 48.1 (2010), pp. 47–53.

[15] Mohamed S Haji Ali, Maan M Shaker, and Thair A Salih. "FREQUENCY OFFSET EFFECT ON THE PERFORMANCE OF CHAOTIC CHIRP SPREAD SPECTRUM SYSTEM". In: *AL-TAQANI* 24.3 (2011), pp. 38–46.

[16] Semtech Corporation. *SX1272/3/6/7/8: LoRa Modem Design Guide.* https://www.rs-online.com/designspark/rel-assets/ds-assets/uploads/knowledge-items/application-notes-for-the-internet-of-things/LoRa%20Design%20Guide.pdf. (Accessed on 10/12/2019). July 2013.

[17] Semtech Corporation. *AN1200.22 LoRa™ Modulation Basics.* http://wiki.lahoud.fr/lib/exe/fetch.php?media=an1200.22.pdf. (Accessed on 10/12/2019). May 2015.

[18] Semtech Corporation. *SX1261/2 Long Range, Low Power, sub-GHz RF Transceiver.* https://semtech--c.na98.content.force.com/sfc/dist/version/download/?oid=00DE0000000JelG&ids=0682R000006F5psQAC&d=%2Fa%2F2R000000HT7B%2F4cQ1B3JGOiKRo9DGRkjVuxclfwB.3tfSUcGr.

S_dPd4&operationContext=DELIVERY&viewId=05H2R000000Jk9xUAC&dpt=. (Accessed on 10/12/2019). June 2019.

[19] The LoRa Alliance Technical Committee. *LoRaWAN™ 1.1 Specification.* https://lora-alliance.org/sites/default/files/2018-04/lorawantm_specification_-v1.1.pdf. (Accessed on 10/12/2019). Oct. 2017.

[20] The Things Network. *LoRaWAN Architecture.* https://www.thethingsnetwork.org/docs/lorawan/LoRaWAN-Overview.png. (Accessed on 10/12/2019).

[21] Ismail Butun, Nuno Pereira, and Mikael Gidlund. "Analysis of LoRaWAN v1. 1 security". In: *Proceedings of the 4th ACM MobiHoc Workshop on Experiences with the Design and Implementation of Smart Objects.* 2018, pp. 1–6.

[22] Godfrey Anuga Akpakwu et al. "A survey on 5G networks for the Internet of Things: Communication technologies and challenges". In: *IEEE Access* 6 (2017), pp. 3619–3647.

[23] *Telenet Tinx van Telenet Business: Internet of Things voor uw bedrijf. - Telenet Business.* https://www2.telenet.be/nl/business/producten-diensten/internet-of-things/. (Accessed on 09/28/2019).

[24] *Orange, partner voor uw Internet of Things-projecten.* https://business.orange.be/nl/business-oplossingen/internet-things. (Accessed on 09/28/2019).

[25] *Internet of Things, IoT, LoRaWAN en M2M — Proximus.* https://www.proximus.be/nl/id_cl_iot/bedrijven-en-overheden/oplossingen/verbonden-business/internet-of-things.html. (Accessed on 09/28/2019).

[26] Ericsson. *Mobile IoT In The 5G Future.* https://www.ericsson.com/4a8d35/assets/local/networks/documents/gsma-5g-mobile-iot.pdf. (Accessed on 05/22/2020).

[27] Sixfab. *LTE NB-IoT HAT for Raspberry Pi.* https://sixfab.com/product/raspberry-pi-nb-iot-shield/. (Accessed on 09/28/2019).

[28] Huawei. *Application of Firewalls in the LTE IPSec Solution.* https://support.huawei.com/enterprise/en/doc/EDOC1100087919. (Accessed on 05/20/2020).

[29] Jeffrey Cichonski, Joshua Franklin, and Michael Bartock. *Guide to LTE security.* Tech. rep. National Institute of Standards and Technology, 2016.

[30] Y-P Eric Wang et al. "A primer on 3GPP narrowband Internet of Things". In: *IEEE Communications Magazine* 55.3 (2017), pp. 117–123.

[31] Olabode Idowu-Bismark, FE Idachaba, and AA Atayero. "A Survey on Traffic Evacuation Techniques in Internet of Things Network Environment". In: *Indian Journal of Science and Technology,* 10.33 (2017).

[32] Collins Burton Mwakwata et al. "Narrowband Internet of Things (NB-IoT): From Physical (PHY) and Media Access Control (MAC) Layers Perspectives". In: *Sensors* 19.11 (2019), p. 2613.

[33] Raad Farhood Chisab and C Shukla. "Performance evaluation Of 4G-LTE-SCFDMA scheme under SUI And ITU channel models". In: *International Journal of Engineering & Technology IJET-IJENS* 14.1 (2014), pp. 58–69.

[34] GSMA. *Security Features of LTE-M and NB-IoT Networks.* https://www.gsma.com/iot/resources/security-features-of-ltem-nbiot/. (Accessed on 06/09/2020). Sept. 2019.

[35] Zigbee Alliance. *Why Standards.* https://zigbeealliance.org/why-standards/. (Accessed on 02/12/2020).

[36] Zigbee Alliance. *Why Certify.* https://zigbeealliance.org/certification/. (Accessed on 02/12/2020).

[37] *Supported devices — zigbee2mqtt.io.* https://www.zigbee2mqtt.io/information/supported_devices.html. (Accessed on 09/28/2019).

[38] Zigbee Alliance. *Zigbee Technical Presentation.* https://zigbeealliance.org/developer_resources/zigbee-technical-presentation/. (Accessed on 02/12/2020). Dec. 2019.

[39] IEEE Std 802.15 taskforce. *IEEE Std 802.15.4™-2015, IEEE Standard for Low-Rate Wireless Networks.* https://www.silabs.com/content/usergenerated/asi/cloud/attachments/siliconlabs/en/community/wireless/proprietary/forum/jcr:content/content/primary/qna/802_15_4_promiscuous-tbzR/hivukadin_vukadi-iTXQ/802.15.4-2015.pdf. (Accessed on 02/12/2020). Apr. 2016.

[40] Shaurov Dhar, Md Nayeem, and Hafiz Rahman. "Maximum Sun-Light Tracking and Optimum Battery Charge Monitoring for Solar Panel using ZigBee Wireless Sensor Network". In: *Proceedings of the 6th IASTED Asian Conference on Power and Energy Systems, AsiaPES 2013* (Apr. 2013). DOI: 10.2316/P.2013.800-122.

[41] Zigbee Alliance. *ZigBee Specification.* https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf. (Accessed on 02/12/2020). Aug. 2015.

[42] Zigbee. *ZigBee 3.0 Stack User Guide: JN-UG-3113.* https://www.nxp.com/docs/en/user-guide/JN-UG-3113.pdf. (Accessed on 05/23/2020). Sept. 2018.

[43] Zigbee. *UG103.2: ZigBee Fundamentals.* https://www.silabs.com/documents/public/user-guides/ug103-02-fundamentals-zigbee.pdf. (Accessed on 05/23/2020).

[44] Xueqi Fan et al. "Security analysis of zigbee". In: *Massachusetts Institute of Technology* (2017).

[45] Zigbee. *ZigBee Specifications 053474r06.* https://www3.nd.edu/~mhaenggi/ee67011/zigbee.pdf. (Accessed on 02/18/2020). June 2005.

[46] Niko Vidgren et al. "Security threats in ZigBee-enabled systems: vulnerability evaluation, practical experiments, countermeasures, and lessons learned". In: *2013 46th Hawaii International Conference on System Sciences.* IEEE. 2013, pp. 5132–5138.

[47] Z-Wave alliance. *Catalog of Certified Z-Wave Products.* https://products.z-wavealliance.org/. (Accessed on 03/29/2020).

[48] Christopher W Badenhop et al. "The Z-Wave routing protocol and its security implications". In: *Computers & Security* 68 (Apr. 2017), pp. 112–129. DOI: 10.1016/j.cose.2017.04.004.

[49] ITU. *ITU-T G.9959 - Short range narrow-band digital radiocommunication transceivers - PHY, MAC, SAR and LLC layer specifications.* https://ec.europa.eu/eip/ageing/standards/home/domotics-and-home-automation/itu-t-g9959_en. (Accessed on 04/01/2020). Jan. 2015.

[50] Tronika. *Z-Wave Technical Basics.* https://www.tronika.no/support/downloads/zwave/Z-Wave%20Technical%20Basics.pdf. (Accessed on 05/26/2020). June 2011.

[51] Behrang Fouladi and Sahand Ghanoun. "Security evaluation of the Z-Wave wireless protocol". In: *Black hat USA* 24 (2013), pp. 1–2.

[52] *OpenThread.* https://openthread.io/. (Accessed on 09/28/2019).

[53] Thread Group. *Thread Stack Fundamentals white paper.* https://www.threadgroup.org/Portals/0/documents/support/ThreadOverview_633_2.pdf. (Accessed on 05/05/2020). 2015.

[54] OpenTread. *OpenThread Primer.* https://openthread.io/guides/thread-primer. (Accessed on 05/04/2020). 2018.

[55] Silicon Labs. "AN1141: Thread Mesh Network Performance". In: (). (Accessed on 05/04/2020).

[56] Thread Group. *Thread Usage of 6LoWPAN white paper.* https://www.threadgroup.org/Portals/0/documents/support/6LoWPANUsage_632_2.pdf. (Accessed on 05/05/2020). 2015.

[57] Silabs. *UG103.11: Thread Fundamentals.* https://www.silabs.com/documents/public/user-guides/ug103-11-fundamentals-thread.pdf. (Accessed on 05/27/2020).

[58] Thread Group. *Thread Commissioning white paper.* https://www.threadgroup.org/Portals/0/documents/support/CommissioningWhitePaper_658_2.pdf. (Accessed on 05/27/2020). 2015.

[59] Yu Liu et al. "A taxonomy for the security assessment of IP-based building automation systems: The case of thread". In: *IEEE Transactions on Industrial Informatics* 14.9 (2018), pp. 4113–4123.

[60] The Things Network. *Duty Cycle.* https://www.thethingsnetwork.org/docs/lorawan/duty-cycle.html. (Accessed on 05/29/2020).

[61] Pavel Masek et al. "Implementation of 3GPP LTE Cat-M1 technology in NS-3: System simulation and performance". In: *2019 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT).* IEEE. 2019, pp. 1–7.

[62] Jean-Luc Aufranc (CNXSOFT). *A Look at LoRaWAN and NB-IoT Power Consumption.* https://www.cnx-software.com/2018/03/29/a-look-at-lorawan-and-nb-iot-power-consumption/. (Accessed on 05/29/2020). 2018.

[63] Elodie Morin et al. "Comparison of the device lifetime in wireless networks for the internet of things". In: *IEEE Access* 5 (2017), pp. 7097–7114.

[64] Carles Gomez et al. "A Sigfox energy consumption model". In: *Sensors* 19.3 (2019), p. 681.

[65] *ATA8520E: Sigfox Partner Network: The IoT solution book.* URL: https://partners.sigfox.com/products/ata8520e.

[66] Semtech. *SX1276 — 137MHz to 1020MHz Long Range Low Power Transceiverh.* https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1276. (Accessed on 09/28/2019).

[67] *Quectel_BC95_NB-IoT_Specification_V1.2.pdf.* `https://www.quectel.com/UploadFile/Product/Quectel_BC95_NB-IoT_Specification_V1.2.pdf`. (Accessed on 09/28/2019).

[68] u blox. *SARA-R4 series.* `https://www.u-blox.com/en/product/sara-r4-series`. (Accessed on 06/08/2020).

[69] *AT86RF231 - Wireless Modules.* `https://www.microchip.com/wwwproducts/en/AT86RF231`. (Accessed on 09/28/2019).

[70] *nRF52840-product-brief.pdf.* `https://www.nordicsemi.com/-/media/Software-and-other-downloads/Product-Briefs/nRF52840-product-brief.pdf?la=en&hash=CE0FDDC1D5B83F5BB5DC5F031AF0A05` (Accessed on 09/28/2019).

[71] *Lora/GPS HAT.* URL: `https://wiki.dragino.com/index.php?title=Lora/GPS_HAT`.

[72] Pycom. *FiPy - Network Development Board with LTE-M, LoRa, Sigfox, WiFi and Bluetooth.* `https://pycom.io/product/fipy/`. (Accessed on 06/08/2020).

[73] *RaspBee premium module voor Raspberry Pi.* `https://www.ledstripxl.nl/raspbee-premium-module-voor-raspberry-pi.html`. (Accessed on 09/28/2019).

[74] *Z-Wave USB Dongle for ZIP Stack by Silicon Labs — PC-Adaptors/ USB-Dongles — Z-Wave Controller — Products — Z-Wave Europe Store.* `https://shop.zwave.eu/detail/index/sArticle/1796`. (Accessed on 09/28/2019).

[75] *nRF52840 DK - nordicsemi.com.* `https://www.nordicsemi.com/Software-and-Tools/Development-Kits/nRF52840-DK`. (Accessed on 09/28/2019).

[76] Marco Centenaro et al. "Long-range communications in unlicensed bands: The rising stars in the IoT and smart city scenarios". In: *IEEE Wireless Communications* 23.5 (2016), pp. 60–67.

[77] Adafruit. *SX1276-7-8.book.* `https://cdn-shop.adafruit.com/product-files/3179/sx1276_77_78_79.pdf`. (Accessed on 09/28/2019).

[78] Shadi Al-Sarawi et al. "Internet of Things (IoT) communication protocols". In: *2017 8th International conference on information technology (ICIT).* IEEE. 2017, pp. 685–690.

[79] Carles Gomez, Joaquim Oller, and Josep Paradells. "Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology". In: *Sensors* 12.9 (2012), pp. 11734–11753.

[80] Artem Dementyev et al. "Power consumption analysis of Bluetooth Low Energy, ZigBee and ANT sensor nodes in a cyclic sleep scenario". In: *2013 IEEE International Wireless Symposium (IWS).* IEEE. 2013, pp. 1–4.

[81] Ashkan Yousefpour et al. "All one needs to know about fog computing and related edge computing paradigms: A complete survey". In: *Journal of Systems Architecture* (2019).

[82] Shanhe Yi et al. "Fog computing: Platform and applications". In: *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb).* IEEE. 2015, pp. 73–78.

[83] Shanhe Yi, Cheng Li, and Qun Li. "A survey of fog computing: concepts, applications and issues". In: *Proceedings of the 2015 workshop on mobile big data.* 2015, pp. 37–42.

[84] EdgeX Wiki contributors. *EdgeX Foundry Project Wiki.* `https://wiki.edgexfoundry.org/`. (Accessed on 04/24/2020).

[85] Microsoft. *Documentatie voor Azure IoT Edge — Microsoft Docs.* `https://docs.microsoft.com/nl-nl/azure/iot-edge/`. (Accessed on 04/28/2020).

[86] EdgeX contributors. *EdgeX documentation.* `https://fuji-docs.edgexfoundry.org/`. (Accessed on 04/24/2020). 2019.

[87] EdgeX contributors. *device-rest-go library on GitHub.* `https://github.com/edgexfoundry/device-rest-go#device-rest-go`. (Accessed on 04/24/2020). 2019.

[88] Microsoft. *Azure IoT — Microsoft Azure.* `https://azure.microsoft.com/nl-nl/overview/iot/`. (Accessed on 04/28/2020).

[89] *Microsoft Azure Marketplace.* `https://azuremarketplace.microsoft.com/en-us/marketplace/apps/category/internet-of-things?page=1&subcategories=iot-edge-modules`. (Accessed on 04/28/2020).

[90] Balena. *balena - The complete IoT fleet management platform.* `https://www.balena.io/`. (Accessed on 04/27/2020).

[91] Balena. *Balena Documentation.* `https://www.balena.io/docs`. (Accessed on 04/27/2020). 2020.

[92] Bin Cheng. *Fogflow tutorial v2.0.* `https://readthedocs.org/projects/fogflow/downloads/pdf/latest/`. (Accessed on 05/11/2020). 2020.

[93] Bin Cheng et al. "Fog Function: Serverless Fog Computing for Data Intensive IoT Services". In: *2019 IEEE International Conference on Services Computing (SCC).* IEEE. 2019, pp. 28–35.

[94] Li Ping Qian et al. “Demand response management via real-time electricity price control in smart grids”. In: *IEEE Journal on Selected areas in Communications* 31.7 (2013), pp. 1268–1280.

[95] Fluvius. *Digitale meter — Fluvius.* `https://www.fluvius.be/nl/veelgestelde-vragen/digitale-meter`. (Accessed on 12/03/2019).

[96] Fluvius. *technische-specificaties-gebruikerspoorten-digitale-meters.pdf.* `https://www.fluvius.be/sites/fluvius/files/2019-10/technische-specificaties-gebruikerspoorten-digitale-meters.pdf`. (Accessed on 12/03/2019). Oct. 2019.

[97] Hector E Perez and Eric Burger. “Cloud Enabled Model Predictive Control of a Home Heating System”. In: (2014).

[98] Benjamin Biegel et al. “Electricity market optimization of heat pump portfolio”. In: *2013 IEEE International Conference on Control Applications (CCA).* IEEE. 2013, pp. 294–301.

[99] U.S. Energy Information Administration. *Annual Energy Outlook 2020.* `https://www.eia.gov/outlooks/aeo/pdf/AEO2020%20Full%20Report.pdf`. (Accessed on 03/16/2020). Jan. 2020.

[100] Miele. *Miele 3rd Party API.* `https://www.miele.com/developer/index.html`. (Accessed on 03/16/2020). Oct. 2019.

[101] Samsung. *Use SmartThings to control your Samsung Washer.* `https://www.samsung.com/us/support/answer/ANS00082522/`. (Accessed on 03/16/2020).

[102] Phoscon. *ConBee II Overview.* `https://www.phoscon.de/en/conbee2`. (Accessed on 03/16/2020).

[103] M Benghanem. “Optimization of tilt angle for solar panel: Case study for Madinah, Saudi Arabia”. In: *Applied Energy* 88.4 (2011), pp. 1427–1433.

[104] VREG Energie wijzer. *Samenstelling energiefactuur gezin.* `https://infogram.com/samenstelling-energiefactuur-gezin-1hxr4zn705no4yo?live`. (Accessed on 03/20/2020). Mar. 2020.

[105] Statista Research Department. *Electricity prices for households in Belgium 2010-2019.* `https://www.statista.com/statistics/418067/electricity-prices-for-households-in-belgium/`. (Accessed on 03/20/2020). Nov. 2019.