# Summary and Conclusion

Wireless sensors keep getting smaller and cheaper; however, most of them still run on batteries, so keeping their energy consumption low is as important as ever. There are numerous ways to achieve this, both on a sensor level and on a network level. As such, there is no standard network structure for wireless sensor networks.

In this thesis, a multi-sink, single-hop wireless sensor network is studied. This is a rather unconventional network, where sensors broadcast messages to all gateways in their vicinity, and the gateways forward messages between the sensors and the control center, which is also known as the cloud. Due to this design, the sensors may roam even when their physical location remains fixed, as external interference may alter which gateways they can reach.

This roaming can potentially lead to increased energy consumption in the sensors for several reasons. Some scenarios where this happens are identified in this thesis, and ways of reducing this increase are proposed. These scenarios are mainly related to address assignment and gateway selection, of which the latter was decided to look into.

The gateway selection problem is related to the fact that each sensor has an encrypted session with the cloud, and so the gateways are unable to read the content of any messages. In practice, this means that all replies must be created by the cloud and forwarded to a gateway, that in turn will send this reply to the sensor. Due to the high latency of the Internet, these replies need to be created and forwarded in advance.

If a sensor does not receive a reply in time, it will make several retransmission attempts, which naturally increases its energy consumption. Besides this, the sensors also have an operating mode called boost mode, in which they can communicate over longer distances at the expense of higher energy consumption. Seeing as the sensors may also roam, selecting which gateways to send replies via is not a trivial problem.

In this thesis, a transmission power control algorithm, ways of detecting retries and monitoring connection reliability, as well as four methods for selecting which gateway to forward replies to, are presented.

The simplest method selects gateways based on which gateway that has the best average signal strength for each sensor. This method is called the best average signal strength (BASS) method. The second one selects gateways based on how many retransmissions that occurred while they were responsible, and so it is called the lowest retry count (LRC) method. A further development of this method monitors the reliability of each individual connection and always selects the gateways with the highest connection reliability (HCR) to each sensor. The final method attempts to estimate the expected energy consumption for each sensor based on the reliability of the connections, and then it selects the gateway that leads to the lowest expected energy consumption (LEEC) for each sensor.

Due to the unconventionality of this wireless sensor network, no wireless network simulator that could be used as-is was found. Therefore a new simulator was implemented from scratch, and the rationale behind its design is discussed extensively in this thesis. The gateway selection methods were implemented in the simulator, and each of them were simulated five times on 12 different networks in order to test them under varying conditions.

The results show that transmission power control lead to a reduction of the energy consumption in the networks by more than 11%, despite causing a considerable increase in the number of retries and transmission in boost mode. It is therefore considered a success, and by improving the algorithm such that it causes fewer retries and transmissions in boost mode, the energy consumption could be reduced even further.

With boost mode enabled, LEEC was able to achieve the lowest energy consumption out of all the methods by minimizing the amount of boost transmissions. On average the energy consumption was just over 21% lower than for the closest competitor. For a wireless sensor, reducing its energy consumption by this much is very good, so this gateway selection method definitely works well. This was achieved by using a large window size for the calculation of average signal strength and connection reliability.

However, with a small window, and a huge amount of retransmissions, BASS consistently gave the lowest energy consumption. Not as low as LEEC with a large window, but on average a little over 19% lower than any other methods with a small window. It is therefore clear that the performance of these methods depends heavily on the window size, which should be adjusted according to the rate of change of the connection's properties. It also seems like having few transmissions in boost mode is more important than causing few retransmissions, if energy consumption is to be reduced.

Nevertheless, the proposed methods work quite well, and together with the simulator, they form a solid basis for future research on the issue of gateway selection in multi-sink, single-hop wireless sensor networks, where the sensors are prone to roaming.

# Sammendrag og konklusjon

Trådløse sensorer blir stadig mindre og billigere, men de fleste bruker fremdeles batterier, så å sørge for at de er energigjerrige er stadig like viktig. Dette kan gjøres både i selve sensorene, men også i nettverket rundt dem. Det finnes derfor ingen standard for hvilken nettverksstruktur som brukes i trådløse sensornettverk.

I denne masteroppgaven studeres trådløse sensornettverk med flere gatewayer og enhoppskommunikasjon. Dette er en noe uvanlig nettverkstype, hvor sensorene kringkaster meldinger til alle som er i nærheten, og hvor gatewayene videresender meldinger mellom sensorer og et kontrollsenter, som gjerne kalles for skyen. På grunn av dette kan sensorene roame. Ikke fordi de flytter på seg, men fordi forstyrrelser kan gjøre at hvilke gatewayer de når varierer.

Roaming kan føre til at sensorene bruker mer energi. Flere måter dette skjer på undersøkes og forklares i denne oppgaven, og noen metoder som kan redusere denne økningen foreslås og presenteres. Hovedgrunnene til at dette skjer er koblet til sensorenes adresser og valg av ansvarlig gateway. I oppgaven er det sistnevnte problem som har blitt forsøkt løst.

Problemet med valg av gateway skyldes at hver sensor har en kryptert sesjon med skyen. Da kan ikke gatewayene lese meldingene som sensorene sender, og derfor må alle svarene til sensorene lages av skyen og sendes til gatewayer, som så sender dem til sensorene. På grunn av de relativt høye forsinkelsene på internett må disse svarene lages og sendes til gatewayene før de faktisk skal brukes.

Dersom en sensor ikke får svar raskt nok vil den forsøke å sende meldingen på ny, noe som selvsagt fører til økt energiforbruk. Sensorene har også en modus som kalles for "boost" som lar sensorene sende meldinger over lengre avstander, men det krever mer energi. Siden sensorene i tillegg kan roame, er det ikke trivielt å velge hvilken gateway man skal sende svarene til på forhånd.

Fire metoder som kan brukes til å velge gateway blir presentert i denne oppgaven. Den enkleste velger den gatewayen som har best gjennomsnittlig signalstyrke på meldinger fra en sensor. Denne metoden kalles for "best average signal strength (BASS)" på engelsk. Den

andre metoden teller antall gjenforsøk som har blitt gjort mens hver gateway var ansvarlig, og velger den som har færrest. Denne metoden kalles for "lowest retry count (LCR)" på engelsk. En videreutvikling av denne estimerer påliteligheten til koblingen mellom hver gateway og sensor, og velger den mest pålitelige. På engelsk kalles denne "highest connection reliability (HCR)". Den siste metoden estimerer den forventede energibruken med hver gateway basert på pålitelighetene til koblingene, og velger så den med lavest estimat. Denne kalles for "lowest expected energy consumption (LEEC)" på engelsk.

Måter å detektere gjenforsøk på, og metoder for å estimere påliteligheten til koblinger blir også presentert i oppgaven, siden metodene for valg av gateway trenger dette. I tillegg presenteres en algoritme for å justere sendestyrken på radioen til sensorene, da forskning har vist at det er en effektiv måte å redusere energibruk i sensorer på.

I og med at dette sensornettverket er noe uvanlig oppbygd, var det vanskelig å finne simulatorer som kunne simulere det direkte. Derfor ble det besluttet å implementere en ny simulator fra bunn av. Designet på denne og begrunnelsen bak de ulike designvalgene diskuteres nøye i oppgaven. Metodene for valg av gateway ble implementert i simulatoren, og hver av dem ble simulert fem ganger på 12 ulike nettverk, slik at de ble testet under ulike forhold.

Resultatene viser at ved å bruke justering av sendestyrke ble energibruken gjennomsnittlig redusert med mer enn 11%, selv om den førte til et mye høyere antall gjenforsøk og sendinger i boost-modus. Dette må derfor anses som vellykket, og ved å forbedre denne algoritmen slik at den forårsaker færre gjenforsøk og sendinger i boost-modus, vil energibruken kunne reduserer ytterligere.

Total sett klarte LEEC å oppnå den laveste gjennomsnittlige energibruken, hovedsakelig ved å redusere antall sendinger i boost-modus. Energibruken var like over 21% lavere enn den nærmeste konkurrenten. For en trådløs sensor er en slik energibesparelse betydelig, så man kan helt klart si at LEEC fungerer bra. Disse resultatene ble oppnådd ved å bruke et stort vindu ved beregning av gjennomsnittlig signalstyrke og estimert koblingspålitelighet.

Ved å bruke lite vindu var det BASS som gav det laveste energiforbruket, selv med et meget høyt antall gjenforsøk. Ikke like lavt som LEEC med stort vindu, men 19% lavere enn det noen andre metoder klarte med lite vindu. Det kommer derfor tydelig frem at størrelsen på vinduet er en viktig parameter ved bruk av disse metodene. Dette bør justeres etter hvor raskt egenskapene til koblingene endres. Det virker også som om antall sendinger i boost-modus har større betydning for energibruken enn antall gjenforsøk.

Likevel kan det sies at de foreslåtte metodene fungerer bra, og at de sammen med simulatoren danner et godt grunnlag for videre forskning på valg av gateway i et trådløst sensornettverk med flere gatewayer og enhoppskommunikasjon, der sensorene er utsatt for roaming.

# Table of Contents

# List of Algorithms

# List of Figures

# List of Tables

# Preface

This thesis concludes a Master's degree in Engineering Cybernetics from the department of Engineering Cybernetics at NTNU. It was carried out during the spring semester of 2018.

The problem at hand was presented by Disruptive Technologies, a Norwegian company that operates in the wireless sensor market. Due to the fact that this is a highly competitive industry, I had to sign a non-disclosure agreement (NDA) with them. In addition to this, the publication of this thesis is delayed by three years.

As such, I was only allowed to discuss the research with my main supervisor, Tor Engebret Onshus, and the co-supervisor from Disruptive Technologies, Einar Yngve Aarflot. No other external assistance was received. I would like to thank Tor, as he was of great assistance. He helped me narrow down the problem and focus on the right aspects of it. He also helped me by discussing some technical details about the solutions that I proposed. I would also like to thank Einar for his great assistance. All information about how the specific wireless sensor network works was provided by him, and he answered numerous questions that I had about their sensor network. He also gave feedback regarding my prototypes and solutions.

All figures and pseudo code presented in this thesis was created by me alone, unless otherwise stated. Since I was not given any hardware to work with, and no simulator was readily available, I also implemented the wireless sensor network simulator from scratch, using Golang and its native packages. No other libraries were used. The plots were created using Matplotlib. [1]

Finally, I would like to thank the fellow students in my office for the countless fruit runs and cups of coffee, and last but not least, I also want to thank Lena for being so patient and awesome.

Ørjan Svendsen
Trondheim, Norway, June 2018

# Problem Statement

Millions of sensors communicate with the cloud through thousands of gateways. Keeping the energy consumption of these sensors at a minimum is of high importance. The sensors communicate using a local address, and if a sensor moves to a different gateway (roaming), the local address will no longer be valid. This is one of several scenarios, which leads to increased energy consumption. The assignment is to:

- Study previous research and existing literature on the subject.
- Identify scenarios where roaming could lead to increased energy consumption.
- Optimise the total energy consumption based on parameters related to these scenarios.
- Come up with improvements that could reduce the energy consumption.
- Implement some prototypes of these improvements.
- Test the prototypes in a simulator and compare their performance.

# Part I

# Introduction

# Chapter 1

# Introduction

## 1.1 Background and Motivation

Today, wireless sensors are located everywhere, from the harshest environments, monitoring volcanos and glaciers, to offices and shopping malls, measuring the quality of the air that people breathe. What they all have in common is that they need to be small, cheap and easy to deploy. A consequence of this is that more often than not, their energy demands are supplied by batteries to make them portable and small. This, on the other hand, means that in order to make them last for as long as possible, they should use as little of the precious energy stored in their batteries as possible.

In order to balance this demand for low energy consumption with the demand for flexibility, new and rather interesting ways of organizing the wireless sensors in networks have come up. In this thesis, a quite unconventional Wireless Sensor Network (WSN) presented by Disruptive Technologies, is investigated. This network reduces energy consumption in the sensors in many ways, but it also introduces new and interesting issues. Most importantly, it can cause the sensors to roam in the sense that which gateways they communicate with changes as time passes.

As this is a highly specific issue in a rather new kind of WSN, the motivation for this thesis is the opportunity to tackle a relatively novel issue, and to come up with new solutions and methods for controlling the WSN and reducing the energy consumption in its sensors.

## 1.2 Approach

The approach to this thesis was divided into different phases. First, a study into existing research on reducing energy consumptions in wireless sensors was conducted. Then the specific WSN was studied and documented, to clarify any ambiguities, and to set the groundwork for the rest of the thesis. Next the WSN was analysed for scenarios leading to increased energy consumption due to roaming.

When the problems had been identified and some of them had been chosen to be further looked into, ways of simulating WSNs was researched, such that proposed solutions could actually be verified. With this in place, the chosen problems were further studied, and solutions were proposed. In the end, these methods were tested under a variety of conditions in the simulator, with a specific emphasis on parameters that the author believed to be important for the performance of the methods.

Finally, the results were analysed and the performance of the methods was assessed based on the chosen parameters.

## 1.3 Contributions and Scope of this Thesis

The purpose of this thesis is to identify causes for increased energy consumption due to roaming in the given WSN, and to come up with methods to reduce this. Since no physical sensors were given, the methods had to be tested in a simulator. Due to the uniqueness of this WSN, the author did not find any suitable simulators, and therefore decided to implement a new simulator from scratch.

Since this task was not in the original scope of the thesis, some aspects of the simulator were simplified to give more time to solve the problems given in the problem description. Emphasis has mainly been put on causing roaming to happen for the sensors in the simulator, and on testing whether the proposed solutions actually do reduce the energy consumption or not.

The largest contribution in this thesis ended up being the WSN simulator. It is written in Golang, and can be customized quite a bit by using a configuration file. Its usage is explained in detail in appendix D. Besides this, a network grid generator was also implemented, in order to rapidly create new networks of sensors and gateways for testing purposes.

One critical aspect in this WSN was found to be selecting which gateway to make responsible for replying to each sensor. As such, the author came up with four methods for doing

this, which are presented in this thesis. All of them are implemented in the simulator. Furthermore, methods for detecting retries and monitoring the reliability of both uplink and downlink connections were also developed, as this unconventional WSN requires some unusual techniques for keeping track of the state of all the sensors. These methods are also presented in the thesis.

## 1.4 Outline of Chapters

The main contents of the thesis are the following.

- **Chapter 2:** Delves into related work and current research on the topic of reducing energy consumption in WSNs.
- **Chapter 3:** Presents the specific WSN that is studied in this thesis, and identifies scenarios caused by roaming that lead to increased energy consumption in the sensors.
- **Chapter 4:** Gives an overview of existing WSN simulators, explains how the custom simulator in this thesis was implemented and discusses various design choices that were made in the process.
- **Chapter 5:** Presents the proposed methods for selecting responsible gateways, and explains the rationale behind them. It also presents ways of monitoring the status of the WSN, which is needed in the selection methods.
- **Chapter 6:** Explains how the simulations are run to obtain the results that are used to test the gateway selection methods.
- **Chapter 7:** Presents the simulation results and gives an explanation of them.
- **Chapter 8:** Discusses and compares the simulation results.
- **Chapter 9:** Gives suggestions for future work related to this thesis and the problem of energy consumption due to roaming.

There are also several appendices in this thesis:

- **Appendix A:** Presents the derivation of several equations that are developed for and used in this thesis.
- **Appendix B:** Contains the configuration file used in the tests, along with all the network layouts.
- **Appendix C:** Presents tables with the full simulation results.
- **Appendix D:** Explains how to use the simulator and the configuration file.
- **Appendix E:** Discusses some technicalities regarding the implementation of the simulator.

# Chapter 2

# Related Work

As the name says, sensors in wireless sensor networks have no wires. Naturally, this means that they have to communicate wirelessly, but it also means that they cannot have remote energy sources. Therefore, they usually end up having a battery as their energy source. Since these kinds of networks normally have sensors of the order of hundreds or even thousands, manually charging them is not an option. [2] Of course it is possible to recharge them on-site by harvesting energy from the environment, but this is rather risky as the availability of energy from the environment is often non-continuous. [3]

For these reasons, wireless sensors should have a lifetime of months or ideally years. [2] In other words, keeping the energy consumption of these sensors low is of utmost importance. Furthermore, as they may be located in unavailable or even hostile environments, changing their batteries every now and then is not ideal either. [3] The goal is therefore to prolong their battery life for as long as possible. Due to this, there has been done extensive research on saving energy in the many different kinds of WSNs that exist.

Raghunathan et al. [2] split up the construction of a wireless sensor into four main modules, which can help when doing an energy consumption analysis:

1. Power Supply
2. Sensing
3. Processing
4. Communication

As Raghunathan et al. mention, in the power supply module, the energy consumption mainly depends on the efficiency of the DC-DC converter. The power consumption in the

sensing module depends entirely on what kind of sensor it is, and can be anything from almost negligible to a major part of the total energy consumption in the sensor. In the processing module, the energy consumption depends on the hardware, and the same goes for the communication module, but an interesting thing to note is that in general, radio communication consumes much more energy than the processing. In some cases, transmitting one bit consumes thousands of times as much energy as executing one instruction. [2]

As the problem to be looked into in this thesis is related to roaming sensors, the energy consumption in the power supply and sensing modules is not directly relevant, and will not be investigated further. Instead, the remaining parts of this chapter will mainly focus on energy consumption due to communication in a WSN, since the communication module generally consumes more energy than the processing unit, and because roaming can lead to increased communication.

## 2.1 Clustering

Clustering is a principle that has been researched thoroughly. [4–7] The idea is that in many kinds of WSNs, the sensors are located in clusters at specific locations, often far away from the nearest gateway. The sensors wirelessly transmit their data to the gateway; however, as mentioned in the previous section, compared to data processing, this is very expensive in terms of energy consumption. This is what is being exploited with clustering.

Instead of having all sensors transmit their data to the gateway, they send their data to one specific sensor in the cluster, usually referred to as the Cluster Head (CH). [5] The CH is much closer to the other sensors than the gateway, so the total energy used for transmission is significantly reduced. The CH then aggregates and compresses the data from all sensors in the same cluster, and transmits this to the nearest gateway. Since a significantly smaller amount of data is transmitted to the gateway, less energy is consumed.

Having the same sensor as the CH continuously would result in it draining its battery significantly faster than the remaining sensors. As the main goal of reducing energy consumption in wireless sensors is to reduce the frequency of having to replace their batteries, or to replace the sensors as a whole, the sensors should alternate in being subject to the extra energy load that comes with being the CH. [8, 9] A lot of research has focused on creating CH selection algorithms that are fair, but also efficient.

One of the most famous clustering algorithms is Low-Energy Adaptive Clustering Hierarchy (LEACH) [10], but many other proposals have been made, such as the one by Hoang et al. [4] or Nayak and Devulapalli. [11] In order to agree on a CH, the sensors need to communicate

internally in the cluster, which leads to increased energy consumption. Naturally, this new consumption must be less than the energy consumption that would be required if all sensors were to communicate directly with the gateway.

Some focus has also been given to further balance the load of being CH by taking each individual sensor's remaining battery capacity into account. [5] Some sensors in the cluster might be located significantly further away from the gateway than others, and so the energy they consume when being CH is greater. This is an intricate balancing issue that needs to take into account both internal distance between the sensors in the cluster, but also the distance from each sensor to the nearest gateway.

This research shows that clustering is an efficient way of increasing the lifetime of WSNs where the sensors communicate directly with a specific node, be it a gateway or another sensor. This is mainly due to the fact that processing is generally cheaper than communicating, and that transmission power must be quadrupled to transmit twice as far. [12]

## 2.2 Protocols

A lot of the research on WSNs is based on the protocols used by the sensors. Much of this focuses on clustering protocols, as explained in section 2.1, but there are also other ways to reduce the energy consumption with clever protocols.

While clustering exploits the fact that some sensors are located close to each other, also known as spatial correlation, it is also possible to exploit temporal correlation. [13] What this means is that the time-series data that a sensor sends can often be predicted, so instead of sending all the values continuously, the sensor can send the mathematical parameters needed to calculate the value. If this is shorter than the actual value, energy consumption is reduced.

In some WSNs the sensors do not necessarily communicate directly with the gateway, but indirectly via each other. So sensors will relay messages they receive from other sensors towards a gateway, which then forwards the messages to the cloud. For these kind of systems, making the routing of messages through the wireless network efficient can save quite a bit of energy. [14, 15]

A great deal of research has also been done on making Medium Access Control (MAC) protocols more energy efficient. [16–18] One important part of a MAC protocol's task is avoiding collisions between transmitting nodes. [17] Ye et al. identified several sources of energy waste in MAC protocols. [17] The most obvious one is *collisions*, because if

several nodes try to transmit at the same time, the data gets corrupted and needs to be re-transmitted. *Overhearing* is another one, which happens if a node receives a message that was not intended for it. Then it wastes energy on receiving a message that will just be discarded. Another interesting waste happens with *idle listening*, i.e. when a node listens for messages when no messages are due to arrive.

As a solution to this they proposed S-MAC, a new MAC protocol that, for instance, has periodic listen and sleep as a way of reducing *overhearing* and *idle listening*. This also reduces energy consumption in general, because, as previously noted, the radio communication module consumes much more energy than the processing module. In fact, even while in idle mode, the transceiver consumes almost as much energy as when sending or receiving. [19] Several other sleep scheduling schemes have been proposed, such as one by Paul et al. [20] and another one by bin Baharudin et al. [21], as general ways of reducing energy consumption in WSNs.

Besides scheduling, the issue of address assignment has also been looked into in relation to MAC protocols. The longer an address is, the more bits have to be transmitted per message, so keeping the address length short can reduce the energy consumption. The same goes for address collisions, where nodes will have to re-negotiate addresses. Tu et al. tackled the first problem in their paper *Mac Address Assignment In Wireless Sensor Networks: A Mixed Strategy Game Approach*, where they reduced the MAC address size and also used spatial re-use of the addresses. [18] Schurgers et al. also came up with a solution to this, where they encoded the addresses to make them even shorter. [22]

## 2.3 Transmission Power Control

The fact that transmitting and receiving data is generally more expensive than processing it, is used in many different ways to reduce energy consumption in the sensors. One approach is to reduce the transmission power as much as possible, while simultaneously keeping the connection stable. [23]. This is known as Transmission Power Control (TPC), and has been researched extensively. [24–26].

There are several ways to implement TPC, and many different algorithms have been proposed. A static TPC approach that calculates a common transmission power shared by all sensors in the network was proposed by Panichpapiboon et al. [27] This algorithm works well in static environments, but it might not work if new sources of interference are introduced. To cope with this, dynamic TPC implementations that adjust the transmission power continuously have been created.

Typically a sensor has discrete levels of transmission power that it can switch between. Depending on a value (which varies between algorithms) measured from incoming messages, it adjusts the transmission power to the lowest acceptable level. For instance, Du et al. proposed an algorithm named Adaptive Transmit Power Adjustment, which uses Packet Loss Rate (PLR) as a measure of the link quality. [23] In practice, it adjusts the transmission power to keep the PLR at an acceptable level. Kim et al. proposed a similar algorithm that uses the Packet Reception Rate (PRR) as an indicator. [28]

Kamarudin et al. used yet another measure of quality, namely Received Signal Strength Indicator (RSSI). [29] With this method, each sensor uses radio propagation models to estimate the received signal strength, and then it uses this to calculate the new transmission power that it is going to use.

Correia et al. proposed two other algorithms: one that adjusts the transmission power and checks if it works, and one that tries to calculate the correct transmission power from various measured parameters and thresholds, which, for instance, can be based on Signal to Noise Ratio (SNR). [30] These are only a few of the many methods around, but it shows that this is an important aspect of reducing energy consumption in wireless sensors.

## 2.4 Gateways

An interesting approach to reduce the energy consumption of the sensors, is the deployment of moving gateways. [31] Instead of having stationary gateways far away from the sensors, the gateways physically move around, and pause at strategical positions. This shortens the transmission distance, which reduces the energy required by the sensors for transmitting data to the gateways. It also ensures that the energy consumption for all the sensors is more uniform, so their lifetime is approximately the same. [32] A bonus is that this makes it possible to recharge the gateways. The downside of this approach is that it might significantly increase latency, but in some use-cases, this is perfectly fine.

Besides moving gateways, another option is to have several gateways in the proximity of the sensors. Cheng et al. looked into this in their paper *General Network Lifetime and Cost Models for Evaluating Sensor Network Deployment Strategies* and found that increasing the number of gateways improved the lifetime more than with one mobile gateway. [33] Which model to use naturally depends on the constraints for the specific deployment of each WSN.

## 2.5 Summary

The related work discussed in the previous sections only covers some selected topics from the ongoing research. WSNs include many different fields of research, each of which can help reduce energy consumption. Rault et al. came up with a useful classification of the various mechanisms that can lead to energy efficiency in WSNs, as shown in table 2.1. [34]

Table 2.1: Classification of energy-efficient mechanisms (adopted from [34]).

| Radio Optimisation | Data Reduction | Sleep/Wakeup Schemes | Energy-Efficient Routing | Battery Repletion |
|---|---|---|---|---|
| Transmission Power Control | Aggregation | Duty-Cycling | Cluster Architectures | Energy Harvesting |
| Modulation Optimisation | Adaptive Sampling | Passive Wake-up Radios | Energy as a Routing Metric | Wireless Charging |
| Cooperative Communication | Compression | Scheduled Based MAC Protocols | Multipath Routing | |
| Directional Antennas | Network Coding | Topology Control | Relay Node Placement | |
| Energy Efficient Cognitive Radio | | | Sink Mobility | |

Out of these, the ones deemed most relevant for roaming-related energy consumption have been discussed. An important thing to notice is that many of the solutions are made for a specific kind of WSN; multi-hop single-sink. Unfortunately, this is not the kind of WSN that is looked into in this thesis, as will be discussed further in chapter 3.

# Part II

# Methods

# Chapter 3

# Wireless Sensor Networks

WSNs are networks where data is gathered from multiple wireless sensors. In this thesis, the sensors will be illustrated as shown in figure 3.1. These sensors can measure anything from temperature, humidity and light intensity to touch, proximity, sound and more. This data is normally collected to a central place, where it is processed and analysed. How the data is transferred to the processing location varies, and mainly depends on the network structure, along with the sensor's power supply and physical location.

Figure 3.1: Wireless sensor.

If the processing location is sufficiently close to all sensors, the data can be transferred directly. However, in most cases the processing location will handle data from multiple physical clusters of sensors. These are usually not close enough to the processing centre to send the data directly, and even if they are, sending over such a long distance can be too energy demanding, unless the sensors have access to an abundant power supply.

Figure 3.2: Gateway.

In most cases, the sensors are battery powered and located in various locations that are far away from the processing centre. In that case, a common way to forward data between the sensors and the processing centre is through base stations. In network theory, these are often called data sinks, as they gather all the data from an area and send it to the right place. In this thesis, they will be referred to as gateways. A graphical illustration of a gateway is shown in figure 3.2.

As explained in chapter 2, there are many ways to reduce the energy consumption of WSNs. However, whether they are applicable to a given WSN or not often depends on its network structure. There are several ways to classify WSNs by network structure, and 3 of them will be explained here.

The first one is by gateway usage. As noted in chapter 2, it is possible to have one gateway that all sensors in an area forward their data to, but also to have several gateways in proximity of the same sensors. [33] Smart placement of these may reduce energy consumption. Besides this, a fairly new research area is that of moving gateways. So gateway-wise, the network structure can be either single or multi-sink, and static or dynamic sink.

To further complicate things, one can also distinguish a WSN depending on the number of hops required for a message to get from a sensor to a gateway. Sometimes all sensors might be close to a gateway, which makes transmitting data directly to the gateway viable. Since the data is only transmitted from one node to another, this is known as single-hop transmission. However, sometimes some sensors might be located much further away from the closest gateway than other sensors. In that case, they will have very uneven energy consumption if they use single-hop. A solution to this is to use multi-hop transmission. Practically this means that some sensors send their data to the gateway via other sensors, through several hops.

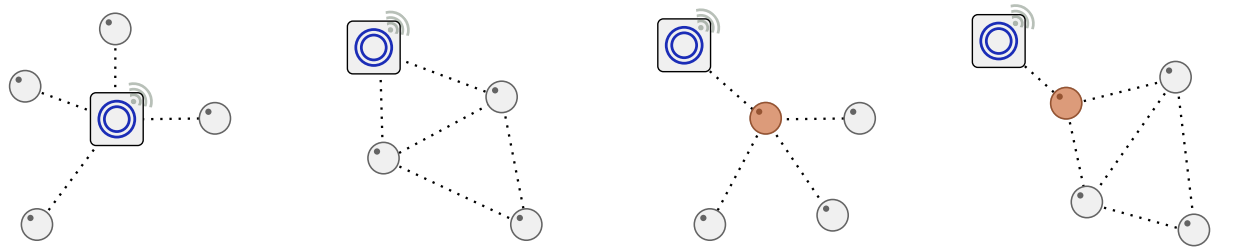
Combining the two classifications of hop-type and gateway type gives eight different network structures, as shown in table 3.1

Table 3.1: Network structure classified by gateway type and hop-type.

| | Gateway Type | | | |
|---|---|---|---|---|
| Hop-type | Static | | Dynamic | |
| | Single | Multi | Single | Multi |
| Single | A | B | C | D |
| Multi | E | F | G | H |

Clustering, a concept that was discussed in chapter 2, is so commonly used that it is useful to use it as a classifying parameter as well. Figure 3.3 illustrates the differences between clustering and no clustering for network structures A and E. So with clustering, there are 16 possible combinations of network structures for WSNs; however, since dynamic (i.e. moving) gateways are such a new concept, the majority of WSNs use structures A, B, E or F, which means eight common combinations if clustering is included.

Classifying WSNs by network structure is useful because the specific WSN that is studied in this thesis is quite different from the majority of WSNs, so many of the methods of reducing energy consumption are not applicable in this specific case. The details of the specific WSN are discussed in the next section.



(a) Single sink with single-hop.

(b) Single sink with multi-hop.

(c) Single sink with single-hop and one cluster head.

(d) Single sink with multi-hop and one cluster head.

Figure 3.3: Common network structures with single sink (adapted from [35]). Orange sensors are cluster heads.

## 3.1 Multiple Gateways with Single-hop Broadcast

As previously mentioned, the WSN that will be analysed in this thesis is a bit different from conventional WSNs. Instead of having rather autonomous sensors that organize themselves in the network, this system gathers data to and controls sensors from a central location. This does not necessarily mean one physical location, as it is most likely run in a distributed manner across multiple servers, but it can be thought of as one entity, referred to as the cloud in the rest of this thesis.

As in any WSN, the network exists of a large amount of wireless sensors, in addition to multiple gateways. The gateways are not sensors that act as a gateway to the cloud, rather they are a different piece of hardware, purposely built for being a gateway. All communication between the cloud and the sensors passes through one or several gateways. A simple example of this is shown below in figure 3.4

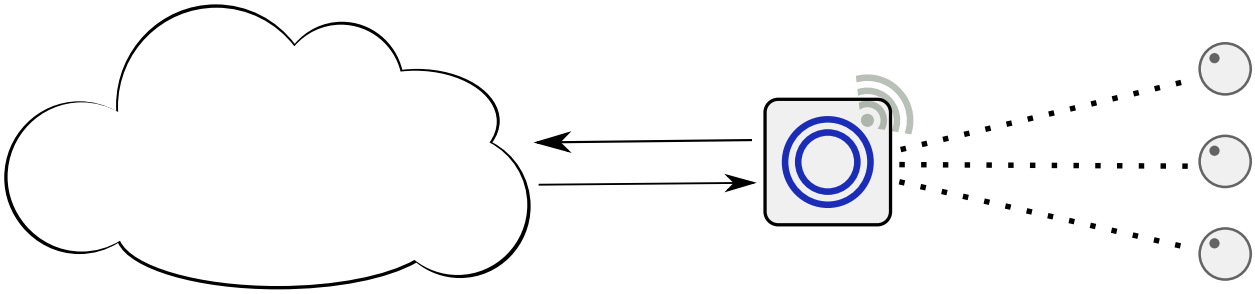


Figure 3.4: Example of centrally controlled WSN with cloud on the left, gateway in the middle and wireless sensors on the right.

Table 3.1 can be used to classify the WSN according to this network structure. As explained, there are multiple gateways. Furthermore, these gateways have fixed positions, and there is only single-hop transmission. This narrows it down to type F. What makes this WSN stand

out from other more conventional alternatives is the interesting design choice that sensors broadcast messages, so the cloud can get the same message from several gateways.

Another consequence of this design choice is that there is no concept of session between the sensors and the gateways. Instead, there is a session between the sensors and the cloud. Since the messages are encrypted, a consequence of this is that what the gateway can see is messages of the format in table 3.2.

Table 3.2: A message as seen from a gateway.

| Address | Encrypted data |
|---|---|

It is also important to mention that the sensors cannot communicate with each other. They merely broadcast their messages, hope that at least one gateway will receive it and forward it to the cloud, and wait for an answer. For this reason, many of the common energy saving concepts in WSNs, such as clustering or optimal routing through multi-hop transmission, is not possible to implement. This design choice is also why roaming can happen, even when the location of both the sensors and the gateways is fixed. It mainly occurs due to external influences that alter which gateways a sensor is able to reach.

This is the superficial explanation of the system. In order to explain in depth how the system works as a whole, and to identify the various sources of unnecessary energy consumption related to roaming, each individual component will be explained in the following sections. These components are the sensors, the gateways and the cloud.

### 3.1.1 The Sensors

The wireless sensors are all battery powered and used to measure various, physical parameters, be it temperature, lighting, humidity or proximity. Thus their exact functioning might vary slightly, such as their sending rate, length of messages etc., but the basic principles remain the same. Their goals are to monitor these parameters and send this data to the cloud at a set interval, and to extend their lifetime as much as possible.

The sending rate depends on what kind of parameter the sensors are measuring. Those measuring slowly changing parameters, such as temperature or humidity, only need to report on a regular, fixed interval. Say every 10 or 20 minutes. Other sensors are event driven, such as proximity or touch sensors. These will only send data when they are triggered. As this may not happen frequently, they send keep-alive signals to the cloud after a certain time has passed without any triggers.

In order to fully explain how the sensors function, a state diagram has been made. This is shown in figure 3.5, and will be explained in detail. Note that the circles represent states, while the squares are there to explain the modes represented by the colours.

When a sensor is first turned on, it starts in the discovery state in standard mode (lower right corner, bright blue with dark blue dots). In this state the sensor has no valid address and no negotiated keys with the cloud. As it cannot function without an address, it will pick a random address and send a message to the cloud.

The response from the cloud determines what happens next: if the sensor cannot read the response, it is encrypted, and so the address is probably taken by another sensor. The sensor will then try again with a new randomly selected address. Should the sensor receive a reply that it can understand, the address is available, and it will negotiate keys with the cloud and move over to the idle state. The process of key negotiation is outside of the scope of this thesis, and will not be further explained.

When idle, the sensor spends most of its time asleep. When an event happens, be it a touch sensor being pressed, or a temperature sensor being triggered by its fixed sending schedule, the sensor sends a message to the cloud and then it waits for a reply. If everything is normal, the reply is processed, nothing needs to be done and the sensor moves back to the idle state and goes to sleep. Should the reply require an action, the sensor will perform this and report back to the cloud.

If a reply is never received, the sensor will try re-sending the message several times. If no reply has been received after the retry limit has been reached, the sensor will move over to boost mode and try sending the message again. Boost mode is an operating mode where the sensor sends messages with a lower frequency, but with increased range. This way it might be able to reach a gateway far away instead of endlessly trying to reach a gateway in standard mode. Other than that, the behaviour in boost mode is mostly the same as in standard mode.

The sensor can also enter boost mode while in discovery. If it does not receive a reply while in discovery state with standard mode, it will enter discovery with boost mode. This also functions the same way as standard discovery, except of the transmission frequency and the range. On a successful connection to the cloud in this case, it enters the idle state in boost mode.

Notice that the sensor can also resort to the resetting state, in case it ends up in a state with nowhere else to go. This happens either if the sensor suddenly cannot decrypt the messages it receives from the cloud, or if it is in boost mode and runs out of re-sending attempts. The last resort then is to wait for a while, before starting over again in discovery state.

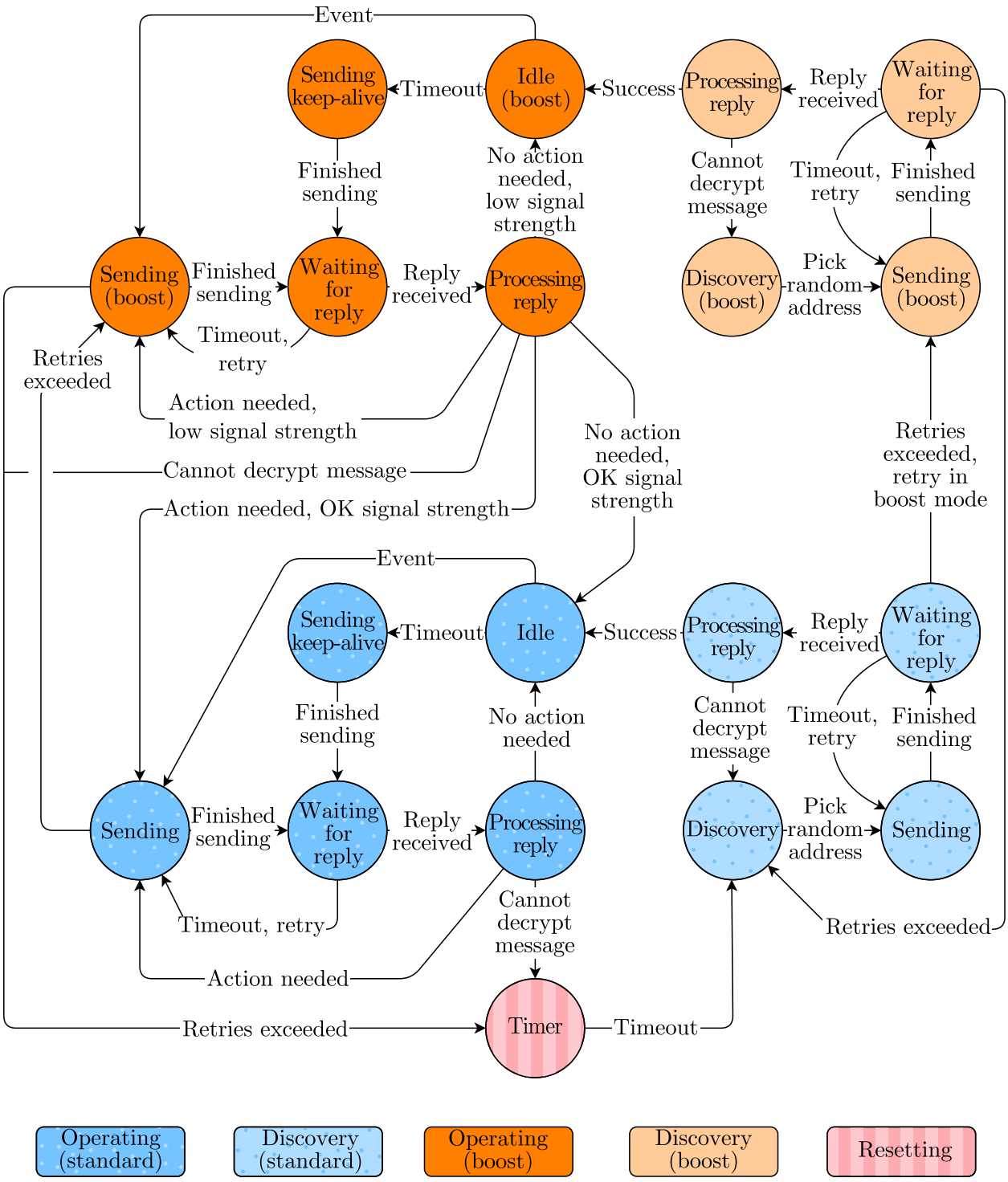



Figure 3.5: State diagram for sensors. The colours represent the modes, as listed at the bottom. The circles represent states. Start in discovery in the lower right.

### 3.1.2 The Gateways

For the scope of this thesis, the gateways can be seen as rather simple devices, whose main purpose is to forward messages from the sensors to the cloud, and vice versa. They have no limitations when it comes to power supply, so they constantly listen for new messages from the sensors. Communication with the cloud is done through the Internet, and this link can be considered stable.

When a gateway receives a message from a sensor, it forwards this to the cloud together with some appended information, such as the signal strength from that specific sensor and its own ID. This information is useful for the cloud when it sends messages to the sensors, as it must choose which gateway to send it through.

After a sensor has sent a message to a gateway, it expects a reply, and if it does not receive a reply sufficiently fast, it will resend its message. The problem is that the gateways cannot formulate replies themselves and send them to the sensors for two reasons. The first reason is caused by the fact that the sensors broadcast their messages. If the gateways were to send replies at will, transmission collisions could happen, and the sensors might receive several replies to the same message. This leads to unnecessary energy consumption, as they only need one reply. Secondly because the messages are encrypted from the sensors to the cloud, so the gateway has no idea what kind of reply to send to the sensors.

In other words, the replies need to be created in the cloud. As the gateway communicates with the cloud through the Internet, a channel that can have high latency, it needs to have these messages ready in advance, so that it can answer the sensors fast enough. In practice this means that each gateway might have replies that have already been prepared for the sensors ahead of time. As soon as it receives a message from a sensor, it takes one of these replies and sends it.

Also note that for this thesis, it is assumed that the gateways transmit messages with constant transmission power, as no other information has been given.

### 3.1.3 The Cloud

The whole system is controlled from the cloud, which can be thought of as a data collection centre, but also as a control centre for the sensors. As mentioned in the previous section, the cloud has to create all replies to the sensors in advance and forward these to the gateways. It must also gather all data from the sensors, and handle any problems that might happen with them.

The information available in the cloud is all the messages, along with information about the gateways that the messages came from, including details about latency and signal strength. So one message will typically contain the following information:

1. Message ID
2. Sensor ID
3. Sensor Address
4. List of Connections

As mentioned in the beginning of this chapter, the sensors broadcast their messages, which means that the cloud might receive the same message via multiple gateways. To make this information useful, other information is appended, so that the list of connections is made up of information about the gateways the message came from. This information is the following:

1. Gateway ID
2. Signal Strength
3. Latency

Based on this information, the cloud needs to make sure there are replies available for all sensors at appropriate gateways. There is no information about the physical location of each sensor, so information about which gateways are in proximity of which sensors must be gathered by monitoring the messages. The cloud must also be able to cope with sensors being added or removed, and changes to which gateways are available for each sensor.

In other words, the cloud must be able to make sure the network runs smoothly based on this limited information. This also includes detecting retries from individual sensors and handling them appropriately.

## 3.2 Sources of Increased Energy Consumption

As explained in the problem description, the problems to be solved in this thesis are all related to roaming sensors. Since the sensors broadcast their messages to any gateways in range, roaming may easily occur, and so the problems this can introduce should be looked into. Note that roaming in the context discussed here does not necessarily imply physical movement of the sensors, but a change in which gateways the sensors can reach.

An example is given in figure 3.6: a temperature sensor is located in a room where a gateway is also located (gateway A). Behind a wall in another room, another gateway is located (gateway B). As gateway A is the closest gateway and also has a free line of sight, it normally gives the strongest signal and should be chosen.

However, between the sensor and the gateway there is a metal door. When this door is open, the signal strength might be severely reduced, or even completely blocked, and gateway A is not the optimal choice. Instead, gateway B should be chosen, as the connection is stronger, so the probability of transmission errors is lower, which results in lower energy consumption by reducing the amount of retransmissions.

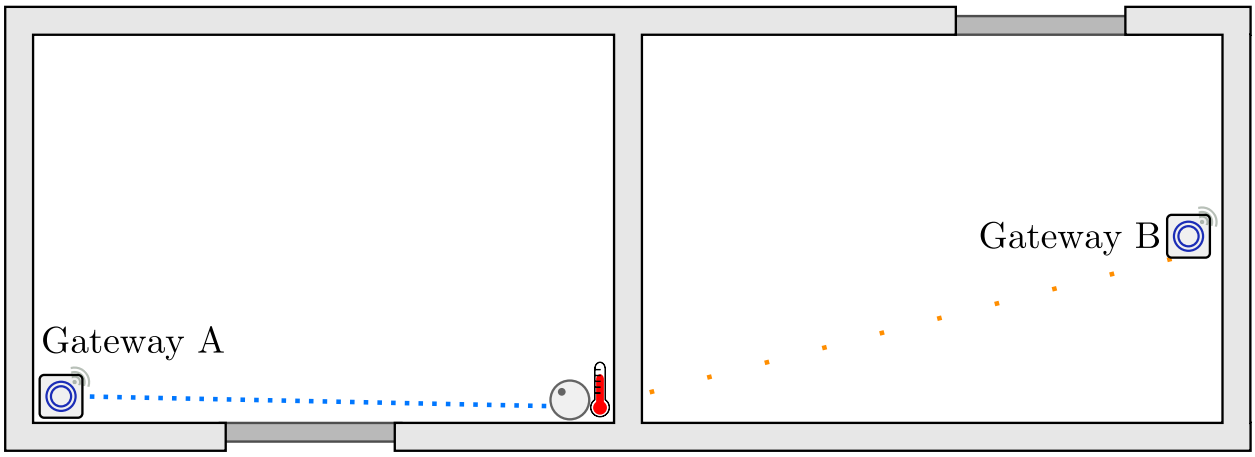
This is only one example of an external, physical disturbance that affects which gateway is optimal. The numbers of events that can lead to the same consequence are many, such as a meeting room being full of people at specific times, trolleys being moved in hallways at hospitals, external radio noise from other appliances operating in the same frequency range and so forth. Involuntary roaming can definitely be an issue in many cases.

The various identified scenarios caused by roaming that can lead to unnecessarily high energy consumption in the sensors will be introduced and discussed in the following section.

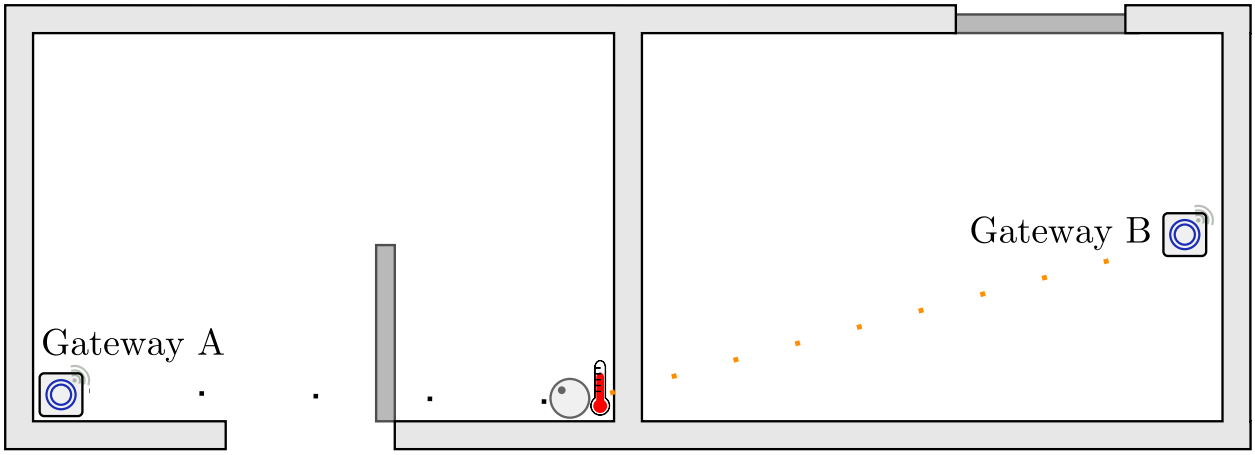
### 3.2.1 Missing Replies

If a sensor does not receive a reply to a message that it sent in a timely manner, it will, as explained in section 3.1.1, try to retransmit the message up to a fixed amount of times. Each re-transmission causes extra energy consumption and should therefore be avoided. This problem does not only happen when no gateways are in range, but also with multiple gateways in range. This is due to the way the system is designed.

As previously explained, the replies must be created in the cloud, because of the encrypted session that exists between the cloud and the sensors. Since the gateways communicate with the cloud through the Internet, a communication channel with high latency, the replies have to be forwarded to the gateways in advance.




(a) Room with closed door.




(b) Room with open door.

Figure 3.6: A scenario where the optimal gateway changes due to a door being opened.

When a gateway receives a message from a sensor, it will simply check if it has any pre-made replies to a sensor with that address, and if so, it will send this message. If it has no replies, some other gateway should have a pre-made reply readily available. Under perfect, stable operating conditions, selecting which gateway to forward the reply to is as trivial as selecting the gateway that has the strongest signal from the sensor. Unfortunately, due to roaming, the task of gateway selection is more complicated than that, and if the cloud does not handle it properly, the consequence is missing replies and increased energy consumption.

### 3.2.2 Address Collisions

The cloud identifies each sensor based on its address and which gateways it communicates through. If several sensors use the same address and send messages to the cloud through the same gateway, the cloud cannot directly know which sensor this message came from. As the messages are encrypted, the cloud must then attempt to decrypt the message with

each key for that address until it is able to successfully decrypt and read it. This leads to an increase in consumed Central Processing Unit (CPU) time, but most importantly to increased latency.

The same goes for messages received by the sensors. Since the gateway merely broadcasts the replies with the address, the consequence is that if several sensors use the same address, the sensors will have to attempt to decrypt messages before realizing that they were not meant for them. Naturally, this leads to increased energy consumption, which is what should be avoided.

Address collisions might occur due to numerous reasons. In figure 3.7 an example is shown. Sensor 1 is located in an area that is covered well by gateway B, while sensor 2 is mainly covered by gateway A, but also slightly by gateway B. In this scenario, sensor 2 might occasionally be able to reach gateway B, depending on external factors. If its address is the same as another sensor that is covered by gateway B, but not gateway A, for example sensor 1, address collisions will happen.

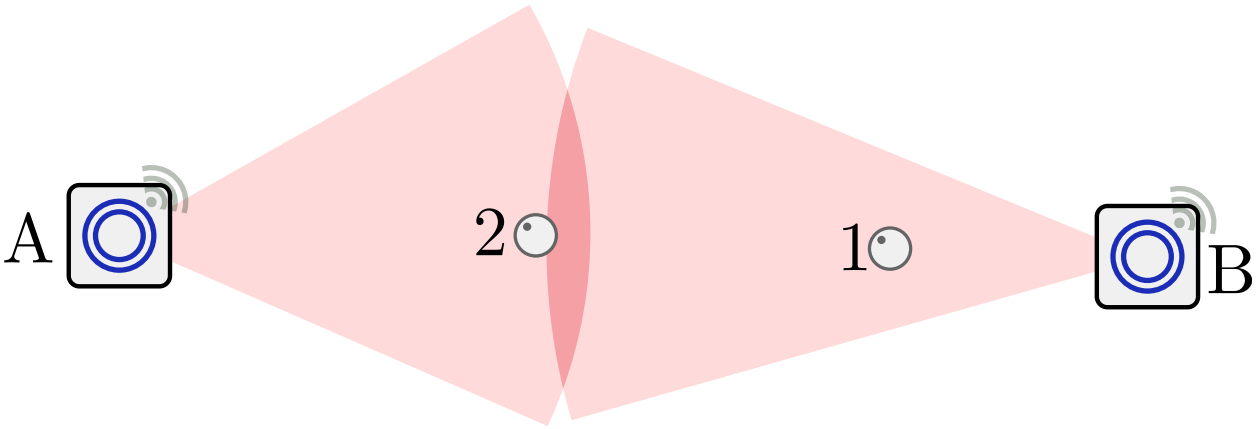



Figure 3.7: Two gateways with two sensors between them.

Another example is illustrated in figure 3.8. If a sensor is only in range of one gateway in standard mode, and that gateway suddenly goes off-line, the sensor will switch to boost mode. The consequence of this is that the sensor might reach other gateways where its address has already been assigned. In the given example gateway A goes off-line, so sensor 1 switches to boost mode. It is then able to reach gateway B, but since it has the same address as sensor 2, a collision happens.

The optimal scenario would be to ensure that there are no collisions at all. As of now, the sensors handle address collisions themselves, by choosing a new random address and then attempting to negotiate a new encrypted session with the cloud, but there are likely ways to reduce the amount of collisions by monitoring and controlling sensor addresses from the cloud.

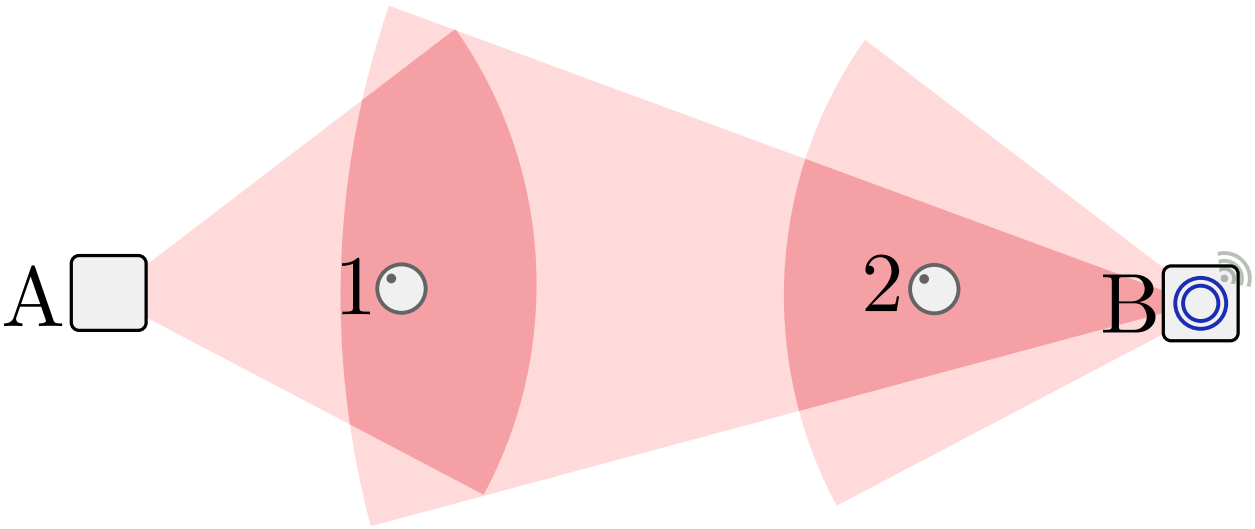



Figure 3.8: Example of address collision. Gateway A goes off-line, resulting in sensor 1 reaching gateway B due to switching to boost mode.

### 3.2.3 Summary

After investigating how roaming can lead to increased energy consumption in this specific WSN, it is evident that both address collisions and missing replies are major contributors. For address collisions, viable approaches might be to assign addresses to the sensors from the cloud, and to apply spatial re-use of addresses, as was done in by Tu et al. [18] Reducing the address length, like they also did, could also be worth a try.

It is also clear that the system in its current state has already implemented several common methods for reducing energy consumption in WSNs. For instance, the sensors spend most of their time asleep, which reduces overhearing. Furthermore, the sensors are normally in range of more than one gateway, which has also been found to be efficient for reducing energy consumption.

In the case of missing replies, it seems like the most efficient way of dealing with it is through selecting the optimal gateways for replying to each sensor. The author was unable to find any research regarding this, at it is a rather specific problem for this network type. Therefore it was decided to look into and attempt to solve this problem. As TPC has not been implemented in the system yet, and research shows that it is an efficient way of reducing energy consumption in wireless sensors, this will also be looked further into.

However, before attempting to find solutions, a WSN simulator must be chosen. This is looked into in the next chapter.

# Chapter 4

# Simulating the Network

As the system being analysed in this thesis has yet to be deployed in large scale, there is not enough real data available for analysis. Neither is the hardware available for real-life testing and experimentation.

From a research perspective, testing WSN related ideas on hardware requires substantial financial investments. [36] Since performing an analytic analysis on a WSN is also very complicated, the most common way to test these ideas is by simulating the WSN realistically, but also simplified for the purpose of testing. [37] This also makes it easier to test proposed solutions rapidly, as they can be implemented directly into the simulated cloud instead of pretending to use them on captured network traffic from a deployed system.

In order to test and improve the solutions quickly, the simulation must simulate faster than real-time. This can be done simply by converting actual time to faster time, for instance by converting one second in real-time to 15 minutes in simulated time. This might seem like a trivial thing to do, but it does set some limitations on the simulation.

It must be ensured that the processing time of all the components is fast enough, such that, for instance, a sensor is ready by the time it is supposed to send its next periodic message. To achieve this, the design of the simulator must be smart, and fast data types should be chosen.

For the reasons mentioned above, it is clear that simulation is the most viable way of testing the solutions to the problem of roaming. Before choosing a simulation method, be it using an existing simulator or implementing a custom one, the requirements for the simulator are listed:

- The simulator must be free.
- It should easily scale up to at least hundreds of sensors.
- Since the protocol in the actual system is proprietary, the simulator must be able to use a genetic, bare-bones protocol without much overhead.
- Semi-realistic or realistic signal strength loss must be implemented.
- Randomly blocked connections must be supported so that roaming will happen.
- One unit (i.e. the cloud) must be able to control all sensors, including assigning them addresses and creating replies for them ahead of time.
- The total energy consumption for the sensors must be estimated with realistic equations.
- Collision avoidance must be handled automatically.
- Sensors should be able to sleep.

This is quite an extensive list of requirements, so the next section looks into existing simulators, and concludes whether to use an existing one or not, and if so, which one.

## 4.1 Existing Simulators

Due to the continued growth of WSN in the Internet of Things (IoT) and the high cost of testing with hardware, the demand for WSN simulators is high. [36] This has lead to an enormous amount of simulators being available, both licensed ones and free ones.

Musznicki and Zwierzykowski did a survey of simulators for WSNs in 2012. [38] They attempted to include all the simulators they could find, and to classify them. In the survey they present the staggering amount of 36 simulators. Selecting the most suitable simulator is thus not a trivial task, and six years have passed since the survey, meaning that the number of available simulators is most likely even higher.

Although the number is high, the classification they propose helps narrow down which simulators are potential candidates, as several of the classes are not suitable. For instance, one class of simulators aims at realistically simulating specific hardware, while the hardware used for the sensors in this thesis is unknown. Another class focuses on topology and routing, which is also irrelevant for our use case, while yet another one has an emphasis on the environment surrounding the sensors.

What is needed in this thesis is a focus on the network, transportation of messages and energy consumption. Unfortunately, many of the simulators that focus on this are often rather limited in functionality. Some, like NetTopo [39] only provide certain WSN algorithms, while others, like Sinalgo [40], focus on visualization of the networks. Other famous simulators, such as Cooja, which comes with the Contiki Operating System (OS) [41], are made for simulating an actual WSN OS, so that the code can easily be exported to hardware afterwards.

The simulators that stand out as most relevant to this thesis are OMNeT++ [42] and NS-2 [43], because they support implementation of custom protocols and are widely used in research. Xiaodong Xian et al. did a comparison of OMNeT++ and NS2 in 2008. [44] They concluded that both simulators are of high quality, but that OMNeT++ has better performance and scales better.

McLoughlin et al. also compared NS-2 and OMNeT, but they also included NS-3 [45] in the comparison. [36] Like they noted, NS-2 has not been updated since 2011, which is also true at the time of writing this thesis. [43] NS-3 on the other hand, is in development and has lots of documentation online. Even still, they concluded that OMNeT++ was the best choice for testing WSN protocols and algorithms.

Even though using one of the simulators has its advantages, the possibility of implementing a custom simulator is considered. To get started with a simulator, one needs to install a whole new framework and possibly work with tools and programming languages that one is not familiar with. Furthermore, since the WSN in this thesis is rather unconventional, as noted in chapter 3, it would require much customization, and the cloud code would somehow need to be connected to the simulator in order to control it.

For these reasons, the author argues that creating a custom, bare-bones simulator is a more efficient way to test the methods in this thesis. As long as the experiments are repeatable, as explained by Tonneau et al. [46], the results obtained should be of value. This is done by open sourcing the code and using realistic equations for simulating energy consumption and loss of signal strength, as well as by fulfilling the requirements listed in the introduction of this chapter.

## 4.2 Choice of Programming Language: Golang

Selecting the right programming language for the task is important, as it sets limitations on the design possibilities, and also because it determines how fast the results are obtained. For this simulation, two criteria were of high importance, namely concurrency and how

comfortable the author felt with the language. Learning a new language was not on top of the priority list, unless it would make the end result significantly better.

A relatively new language that has simple and easy concurrency handling is Golang. [47] It meets the criteria of easy concurrency and the author has previous experience with it. Golang has its own lightweight thread called a goroutine, and starting one is as simple as writing *go* before calling a function. Communicating between these so-called goroutines is also very easy, and is discussed in the following section.

For these reasons, Golang was chosen as the language for implementing the simulator. It should ensure rapid development of a custom simulator that can simulate roaming and that can also be used for testing proposed solutions to the problem at hand.

### 4.2.1 Communication and Synchronization: Channels

Communication and synchronization between goroutines in Golang is done through channels. These make it possible to send various data types from one concurrent goroutine to another, and they come in two types: buffered and unbuffered.

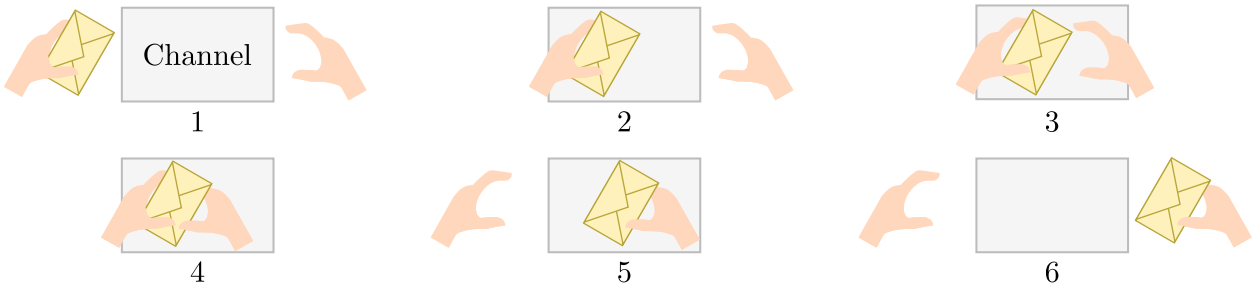



Figure 4.1: Unbuffered channel in Golang.

An unbuffered channel is shown in figure 4.1. This channel type is blocking, which means that if a goroutine attempts to send a message through an unbuffered channel, it is blocked until another goroutine is ready to receive it. The opposite is also true: a goroutine waiting to receive a message from an unbuffered channel is blocked until it receives something. This feature makes it possible to use channels for synchronization as well as for communication.

Buffered channels, on the other hand, are non-blocking, unless they are full when attempting to send, or empty when attempting to receive. How they work is shown in figure 4.2. They are useful when a goroutine just needs to dump a message before moving on, but not when synchronization is needed. In the simulator, it can be useful for receiving messages from gateways in the cloud, as it is supposed to simulate transmission over the Internet, in which case the gateways do not wait for a reply from the cloud.

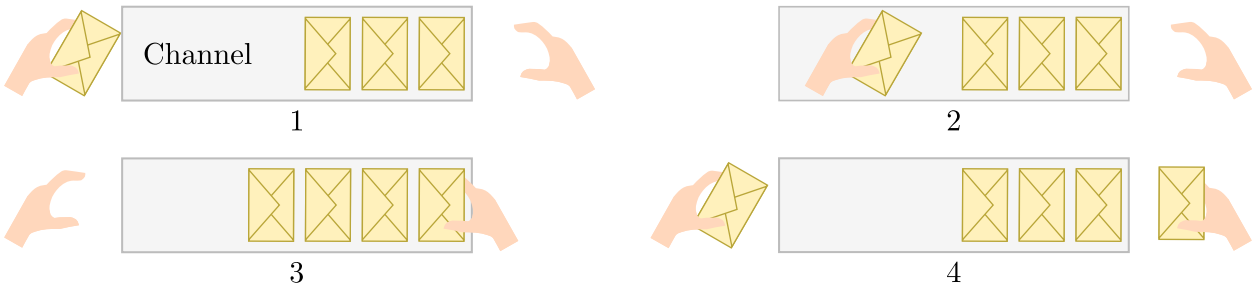



Figure 4.2: Buffered channel in Golang.

An important thing to note is that channels need to be distributed to a goroutine before it is started. It is of course possible to give a channel to several goroutines so that, for instance, both goroutines A and B can send messages to goroutine C. The main caveat for this is that if there are several goroutines at the receiving end, there is no guarantee for which of them will receive the message.

Say goroutines A and B are at the receiving end of a channel, and goroutine C sends one copy of a message on the channel. Then only the first of goroutine A and B will receive the message. On the other hand, if C sends two copies of the message, there is still no guarantee that both A and B will receive the message. If B is busy processing some big file, A will have time to receive the first copy, handle it and then receive the next copy.

This is important to think about, as it sets some design limitations with respect to the numbers of channels used and the distribution of them. Multiple goroutines at the receiving end can be problematic.

## 4.3 Communication

An essential part of the simulation is the communication between the cloud, the gateways and the sensors. It should behave like the actual system so that the results are realistic, but at the same time it should not be overly complex, so that the simulation can run sufficiently fast. As mentioned, Golang was chosen for its threading and synchronization capabilities. Since each component (i.e. cloud, gateway or sensor) will be run as individual, concurrent processes, they will use Golang's channels for communication.

Channels need to be distributed to the correct processes before they are started, to ensure that the processes communicate over the intended channels, and to avoid ending up with a deadlock. To reduce the complexity of this, using as few channels as possible is feasible. On the other hand, too few channels cannot be used, as that would give very unrealistic behaviour.

As mentioned, when a message is put on a channel, only one copy of it is sent. If several processes are listening to the same channel, it is first come, first serve, so only one of them will receive the message. Putting several copies of the same message on the channel does not solve the problem either, as there is no guarantee that one process might not take several of the copies from the channel.

In practice, this puts a lower limitation on the number of channels that must be used in the simulation. The simplest approach is saying that there should only be one process at the receiving end of a channel, but there does not need to be a limitation on the sending end. This means that there should be at least one channel for each device in the system, including the cloud.

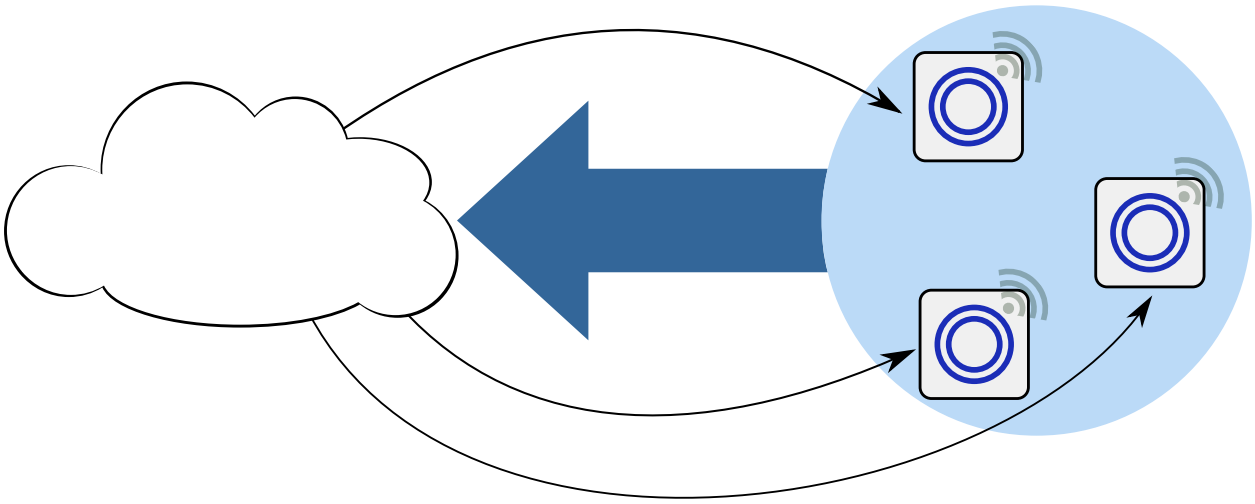


Figure 4.3: Channel distribution for communication between the cloud and the gateways.

Using this rule, the communication between the cloud and the gateways can be done fairly easily. In real life they communicate over the Internet. To reflect this in the simulation, all the gateways share the same, buffered channel for communicating with the cloud. Like when using the Internet, they dump the message, in this case in the channel, and continue doing whatever they were previously doing. There is no risk of any processes being blocked as long as the channel capacity is sufficiently large. This is illustrated in figure 4.3.

Communication from the cloud to the gateways is the other way around, with one channel dedicated for each gateway. This is done to work around the limitation mentioned above, but also to reflect the real-life behaviour, where the gateways only receive messages from the cloud that were intended for them. This way, the logic for routing the messages correctly is located in the cloud.

The same rules are used when designing the communication scheme between the gateways and the sensors. Each sensor and gateway has one channel each for receiving messages. When the system is initialized, each gateway gets the sensor channels for all sensors that are

located in its coverage area. These sensors must then also receive the gateway channels for all the gateways they can reach.

In this way, all sensors use the same channel for sending messages to the same gateway. When a gateway sends replies to a sensor, it uses the channel for that specific sensor. Figure 4.4 illustrates the channel usage. This way of communicating is not completely realistic, because in the deployed system, a sensor can receive messages not intended for it, which wastes energy. Fortunately, this is not part of the problem to be solved in this thesis, so this design is not regarded as an issue in this specific implementation.

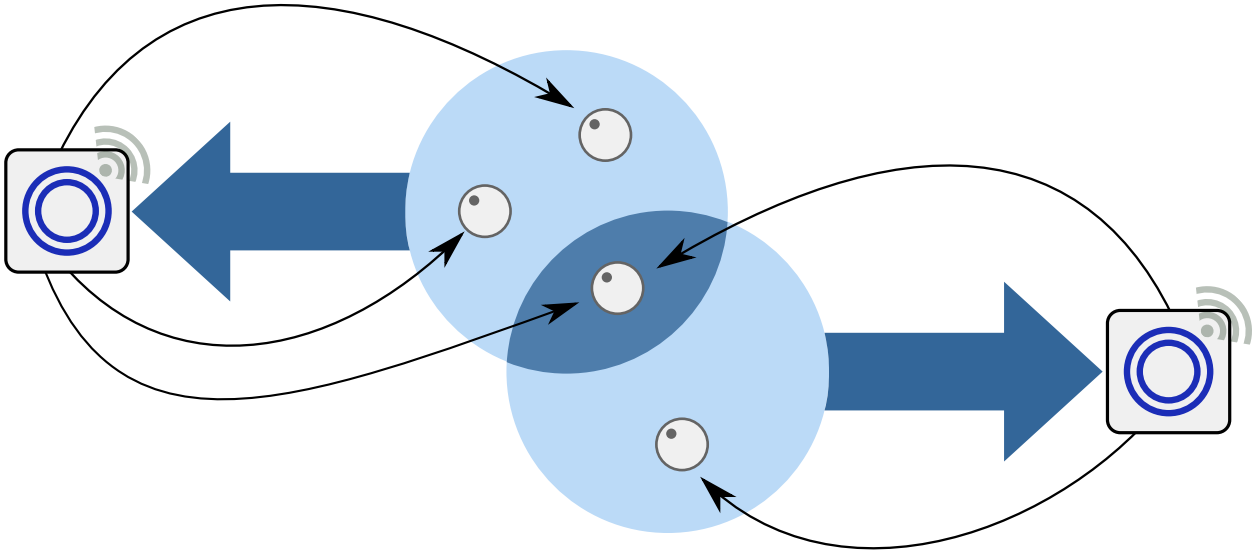


Figure 4.4: Channel distribution for communication between the sensors and the gateways.

Note that using channels in Golang also eliminates the problem of collisions, which can happen with wireless communication between sensors and gateways. The channels from sensors to gateways and vice versa are unbuffered, so if a sensor wants to send a message to a gateway while another sensor is also sending one, it will be blocked until the channel is free.

## 4.4 Wireless Connections

In order to test the proposed solutions for roaming sensors, roaming must happen in the simulator. In short, this means that the availability of gateways for some sensors must change during the simulation. This can be due to reduced signal strength, but also due to lack of any connection at all, such as when a gateway goes offline, or a connection gets blocked.

Kamarudin et al. argue that many simulations of WSNs oversimplify many energy related aspects, such as loss of signal strength over distance, or the fact that TPC is normally limited to discrete steps, and that the transmission power cannot be infinitely increased. [29] This was taken into account when deciding on how to simulate signal loss and unstable connections, as explained in the following subsections.

### 4.4.1 Transmission and Loss of Signal Strength

To keep the simulation realistic, the wireless transmission of messages between the gateways and the sensors should include loss of signal strength. Several methods for calculating the propagation of radio signals indoors exist, many of which are rather complicated, as the signal loss depends on the material of the obstructions, among other things. [48] However, since this thesis is looking into problems caused by roaming, the signal loss implementation is simplified to save time and place emphasis on the main task.

A well renowned formula for radio transmission that also include Free-Space Path Loss (FSPL) is Friis' transmission formula. [12] In its original form, it was formulated as in equation 4.1.

$$\frac{P_r}{P_t} = \frac{A_r A_t}{d^2 \lambda^2} \tag{4.1}$$

Here, $P_t$ is the power sent to the transmitting antenna, $P_r$ is the available power at the receiving antenna, $A_t$ and $A_r$ is the effective area of the transmitting and receiving antennas respectively, $d$ is the distance between them and $\lambda$ is the wavelength of the signal being sent.

For the sensors and gateways in this WSN, however, the effective areas of the antennas are not known. Fortunately, another way of expressing the equation exists. [49]

$$\frac{P_r}{P_t} = e_r e_t \left(\frac{\lambda}{4\pi d}\right)^2 D_r D_t = \left(\frac{\lambda}{4\pi d}\right)^2 G_r G_t \tag{4.2}$$

Equation 4.2 is based on the antenna's directivity rather than effective area. Here, $e$ is the efficiency of the antennas, $D$ is the directivity of the antennas relative to isotropic antennas and $G$ is the gain of the antennas, calculated as $G = eD$. Since it is very common to state the specifications of antennas in terms of their gain, the last expression of the equation is used. In radio theory, it is also common to express power, gains and losses in dBm and dB

respectively. Converting equation 4.2 to use dBm and dB results in equation 4.3

$$P_r = P_t + G_r + G_t + 20\log_{10}\left(\frac{\lambda}{4\pi d}\right) \tag{4.3}$$

This equation is very easy to implement and use in the simulator, and it uses common units for radio transmission. The last term in the equation is the FSPL. A more common way of expressing the FSPL in dB is written in *IEEE Standard for Definitions of Terms for Antennas (2013).* [50] It is shown in equation 4.4.

$$\text{FSPL} = 20\log\left(\frac{4\pi d}{\lambda}\right) \tag{4.4}$$

However, the only difference between them is the sign, meaning that in order to use FSPL as expressed in equation 4.4 in equation 4.3, it could simply be subtracted instead of added. The only remaining thing then is the fact that for radio transmission, it is common to operate with frequency rather than wavelength. Equation 4.5 relates the wavelength and the frequency ($f$) of an electromagnetic signal, where $c$ is the speed of light.

$$\lambda = \frac{c}{f} \tag{4.5}$$

Substituting this for $\lambda$ gives the final equation that is used in the simulator for simulating wireless transmission, as shown in equation 4.6.

$$P_r = P_t + G_r + G_t - 20\log_{10}\left(\frac{4\pi df}{c}\right) \tag{4.6}$$

With the calculation of received signal strength including losses in place, the only thing missing is to decide whether or not a transmission is successful. This depends on the sensor's and gateway's sensitivity. The sensitivity is the lowest signal strength that they can receive and still have reliable communication. It is quite common to give the sensitivity level in dBm, and so, given the sensitivity level of the receiver ($S_r$), a transmission is decided as successful if $P_r \geq S_r$.

### 4.4.2 Connection Types

Other than signal loss, it is also essential to simulate the different types of connections that can exist in an actual deployment of the system. Using the signal loss equation, it is possible

to simulate transmission in free-space, with no obstacles disturbing the signal. However, as previously mentioned, some connections might have heavily varying signal strength due to unknown, external obstacles disturbing it, while other connections might even be fully blocked. Using this information, the author has defined three general connection types:

1. Free-space connection (standard connection)
2. Binary connection
3. Dynamic connection

The **free-space** connection simply uses equation 4.4 to calculate the signal loss based on the distance between a sensor and a gateway. One could argue that this is also an over-simplification, as having constant signal strength with fixed distance is not possible, due to external influences. However, for the purpose of this simulation, the small variations to the signal strength from natural disturbances are ignored. This connection is referred to as the standard connection.

The **binary connection**, however, is quite different. It either acts as a normal free-space connection, or it completely blocks a transmission. In order to make it more realistic, the probability for it to block a transmission can be set in the configurations of the simulator. Also, as a blocked connection is often caused by an external obstacle, it makes sense that sometimes the connection should be blocked consistently for some duration. To simulate this, the binary connection can be set to block all retransmissions of a message from a sensor, if the initial transmission was randomly blocked.

The **dynamic connection** is the last connection type, which is something between the other two connections. It uses equation 4.4 to calculate the free-space path loss, but it also introduces some variation in the signal loss, both increased loss due to external obstacles, but sometimes also a slight reduction in the loss. When this happens is random, and how much extra it reduces the signal is also random, up to a set limit.

One could naturally use accurate equations to simulate increased loss from various materials, such as wooden doors and people, but in order to properly test the solution, the most important thing is that roaming happens in a semi-realistic manner, and not that the signal loss is completely accurate.

This connection can also be customized in the simulator configuration, by setting the maximum limit of how much the signal loss should vary in percents. For simplicity, it is also possible to enable adjustment of the FSPL. This means that the FSPL is adjusted such that for the maximum transmission distance in the configuration file, the path loss equals

the maximum acceptable path loss that can occur between a sensor and a gateway without them losing the connection. Details regarding configurations that can be set for the connections are discussed in section E.1.

### 4.4.3 Distributing the Channels

When it comes to distributing channels, each sensor should only have the channels to the gateways that it can reach, and vice versa. This is accomplished by ensuring that all sensors can communicate with all gateways within their maximum transmission range, and vice versa. If no maximum distance is given in the configuration file, the distance is determined by calculating the ideal transmission range for the given sensitivities and transmission powers.

The maximal transmission range that can be achieved is when the received power is equal to the sensitivity: $P_r = S_r$. Substituting this in equation 4.6 gives equation 4.7, which is used for calculating the range.

$$d = 10^{\left(\frac{P_t + G_t + G_r - S_r - 20\log_{10}\left(\frac{4\pi f}{c}\right)}{20}\right)} \tag{4.7}$$

Since the number of connections in a WSN quickly gets quite large, the connections are implemented in such a way that they do not need to be manually set. Instead, the simulator creates all the connections that are within the maximum distance. However, the connection types are set by the user prior to simulating, as a distribution. The connections are then randomly assigned a connection type according to this distribution. This is also discussed in section E.1.

## 4.5 The Cloud

In the simulator, the most complex component is the cloud. It has several tasks that it must do:

1. Receive and store messages from the gateways.
2. Make sure there are replies ready for all sensors at appropriate gateways.
3. Detect if a sensor does not receive replies from any gateways.
4. Handle address collisions.

So besides constantly listening for incoming messages, it must make sure that the system as a whole operates smoothly. When it comes to making sure there are replies ready for all sensors, this is done by monitoring the incoming messages. When the system starts, the cloud knows nothing about the sensors or the gateways. As soon as it receives a message from a sensor via a gateway for the first time, it will make this gateway responsible for replying to the sensor, so it sends a fixed amount of replies to that gateway. Each time it receives a new message from the same gateway, it sends a new reply to it, to make sure that it always has a minimum amount of replies available.

If a message arrives via another gateway that is evaluated as a better gateway for replying to this sensor, the cloud will tell the old gateway to delete all its replies, and then it forwards new replies to the new gateway. This is the basic functioning of the cloud. How it evaluates which gateways to select is the main problem that is looked into in this thesis. This is further discussed and explained in chapter 5.

## 4.6 The Gateway

For the purpose of this simulation, the gateway can be made quite simple. Its purpose is forwarding messages between the cloud and the sensors, and sending replies to the sensors, if it has any available. The way it does this is by continuously listening for messages from both the cloud and the sensors. If it receives a message from a sensor, it forwards it to the cloud and checks if it has any replies available. If so, it sends a reply to the sensor and continues listening for messages. If not, it simply continues listening. When replies are received from the cloud, it temporarily stores them for future use.

Once the communication of the system had been decided, a simple model was set up in the tool Labelled Transition System Analyser (LTSA). [51] The model was as simple as one sensor, one gateway and one cloud, communicating over two channels. However, due to the fact that the sensor can time out if it does not receive a reply sufficiently fast, a deadlock could potentially occur in the simulator.

Table 4.1: The gateway and sensor state transitions that result in a deadlock.

| **Step:** | **1** | **2** | **3** |
|---|---|---|---|
| **Sensor:** | Send Message | Receive Message | Timeout, Send Message |
| **Gateway:** | Receive Message | Handle Message | Send Message |

In table 4.1 the deadlock scenario is shown. First, the sensor sends a message to a gateway,

and the gateway is ready to receive it. Next, the sensor waits for a reply, while the gateway processes the message. If this processing takes longer than the sensor's timeout, the deadlock occurs, because then both the sensor and the gateway are attempting to send a message across the same channel, and they block until the other end receives it. Note that this is only an issue in the simulator, and not in the actual system, because in real life, messages are simply broadcast wirelessly, so no blocking can occur.

To solve this problem, the gateway was also implemented with a timer, so that when it attempts to send a reply to a sensor, it will time out after half as long as the timeout of the sensor. This ensures that there is no deadlock, and that the gateway will be ready to receive potential retries.

## 4.7 The Sensor

As in real life, the sensor is quite simple. It periodically sends a message, and as in the actual system with wireless communication, the message is broadcast to all gateways in its range. This is implemented by having the sensor iterate over all channels that belong to gateways within its standard mode range, and send the same message on all of them. It then waits for a reply. If it gets a time-out before a reply is received, it will retransmit the message a fixed amount of times.

If it reaches the retry limit without having received any replies, it will switch to boost mode and send the message to all gateways within its boost mode range. If the retry limit is reached in boost mode, the sensor gives up, stays in boost mode and goes to sleep.

In the actual system, the sensor switches back from boost mode to standard mode when the signal strength of the replies it receives is deemed sufficiently strong. In the simulator, this process has been simplified, so the sensor switches back to standard mode if it detects that a reply was received from a gateway that it knows is available in standard mode.

### 4.7.1 Energy Consumption

Energy consumption is one of the most important factors in this WSN, as it is the parameter used to evaluate the success of the suggested solutions. Therefore, a way to estimate the energy consumption in the sensors must be implemented. Lee and Kao presented two equations for energy consumption during wireless transmission and reception in *An Improved Three-Layer Low-Energy Adaptive Clustering Hierarchy for Wireless Sensor Networks.* [7] Their transmission equation is based on the number of bits transmitted, and it also takes

the distance into account. Hill et al. took another approach, combining the transmission power with the duration of the transmission to estimate energy consumption. [52].

In this simulator, a simplified energy consumption model is adapted as a combination of these two approaches. The energy consumed during transmission depends on the number of bytes transmitted, in addition to the transmission power, as the transmission power for the sensors might vary. It also depends on the duration of the transmission, which is calculated by dividing the message size by the data rate. The resulting model is shown in equation 4.8.

$$\mathrm{E_{Tx}}(n, R, P) = n \cdot 8 \cdot \frac{1}{R} \cdot P \tag{4.8}$$

Here, $n$ is the number of bytes transmitted, $R$ is the data-transfer rate and $P$ is the transmission power. The rationale behind this equation is explained in section A.1. This gives an energy consumption in mWs. A nice thing about this equation is that it also applies to boost mode, because the only differences between boost mode and standard mode is the data-transfer rates and the transmission power. In the simulator, however, calculation of energy consumption in boost mode is simplified to multiplying equation 4.8 with a constant that is given in the simulator configuration.

It must be noted that this is not a very accurate model for several reasons. Firstly, it assumes that the power of the transmitted signal is equal to the power required to transmit it. This would require the radio transmitter to have an efficiency of 100%, which is unrealistic. Furthermore, the time required to transmit the bits is likely longer than the amount of bits divided by the data rate.

However, for the scope of this thesis, this model is ok. Since the problem that is studied in this thesis is quite new, there are no results from other research to compare with. Therefore, the proposed solutions will all be tested in this simulator and compared to one another. This means that all results are obtained under equal conditions, so they can be compared directly without any issues.

In this chapter the design of the simulator has been explained. With this in place, the solutions to the problem can be tested. These solutions are proposed and explained in the next chapter.

# Chapter 5

# Proposed Methods for Limiting Energy Consumption in Sensors

As the issue to be looked into in this thesis is how to reduce energy consumption in sensors caused by roaming, proposed methods for solving this issue are presented in this chapter.

In chapter 2 it was shown that TPC has been extensively researched, and is frequently used as a method to reduce energy consumption in wireless sensors. In a network such as this one, where sensors are roaming, it makes sense to use TPC, as the transmission power required to successfully send a message from a sensor to a gateway might change at any time. Therefore, an implementation of TPC is suggested in section 5.1.

In section 5.2, methods for selecting the optimal responsible gateways for sending replies to sensors are presented. These methods use various parameters that are not directly given by the system, so they must be estimated. One of these parameters is the amount of retries caused by each gateway, so methods for detecting retries are discussed in section 5.3. Another parameter is the reliability of the connections between gateways and sensors, both uplink (sensor to gateway) and downlink (gateway to sensor). Ways of estimating this is presented in section 5.4.

## 5.1 Transmission Power Control (TPC)

There exists many different kinds of TPC algorithms, as explained in section 2.3. Many of them are quite complicated, but for this thesis, the TPC algorithm itself is not the goal. Instead, TPC is interesting because it introduces the possibility of balancing the energy consumption in sensors between two factors: retries and transmission power.

Without TPC the gateway selection problem is solved by selecting the gateways with the most reliable connections to each sensor, as the only unnecessary energy consumption is caused by retries. However, with TPC, gateways can be selected in such a way that the transmission power level of the sensors is lowered. This makes the whole problem even more interesting, as now there might exist scenarios where it makes sense to select a connection that is more unreliable, but that still gives a lower energy consumption due to TPC. Therefore the implementation of the TPC algorithm can be somewhat simple, as a lot of research has been done on TPC previously, but not on the gateway selection problem. Hopefully combining TPC with gateway selection methods can lead to a solution that achieves an even lower energy consumption than from using either TPC or the selection method alone.

The TPC algorithm that is presented here is inspired by the one presented by Kamarudin et al. [29] It takes advantage of the homogeneity of the components in the WSN, due to it being produced by a single company. Essentially, each sensor has very little information about the gateways. It does not know which gateways it communicates with, so it only knows that it received a message from some gateway, and the received signal strength of that particular message. As the sensor has very limited resources, it does not keep historical data of this signal strength, so the adjustment is merely based on the newest received message.

In order to adjust its transmission power correctly, the only measurable parameter that can be used is the received signal strength. This severely limits the usefulness of using TPC, as the path loss of the signal can vary a lot, and the distance to the gateways is unknown. However, by making certain parameters of the gateway known, it is possible to implement a simple TPC algorithm.

Assuming that the gateway transmits data with constant power, the sensor can use this to calculate the FSPL. Equation 4.6 relates the received power to the transmitted power in a very simplified way, which can be used to estimate the path loss as in equation 5.1:

$$\text{FSPL} = P_t + G_t + G_r - P_r \tag{5.1}$$

If the sensor also knows the sensitivity of the gateway, it can adjust its transmission power in such a way that, given the same FSPL during transmission as for the reception, the message should arrive. This theoretically works because for the gateway to receive the message, the signal strength received by the gateway ($P_{r_G}$) must be greater than its sensitivity ($S_G$). In other words, the sensor must attempt to make sure equation 5.2 holds (subscript G denotes gateway, and S denotes sensor).

$$P_{t_S} > S_G + \text{FSPL} - G_{t_S} - G_{r_G} \tag{5.2}$$

Wireless transmitters are normally limited to discrete levels of transmission power, rather than being able to continuously adjust them. To make sure the TPC algorithm works properly, it is implemented such that it stays one level above the minimum required to deliver the message, according to equation 5.2.

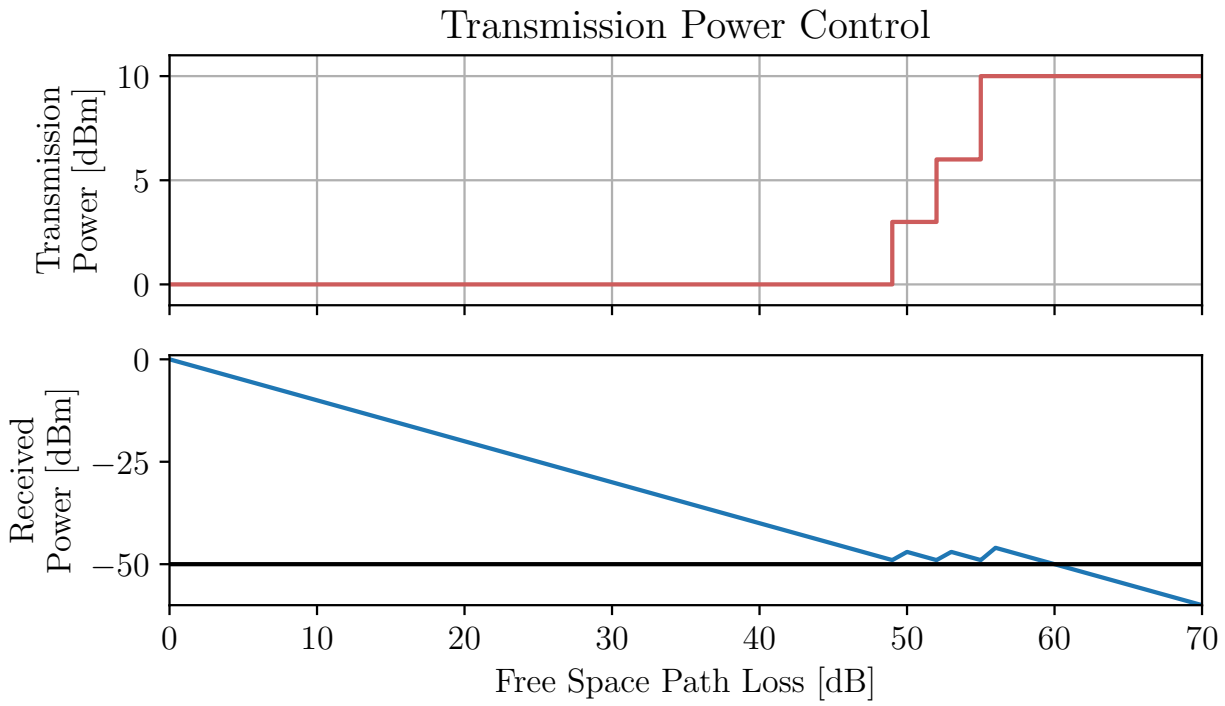



Figure 5.1: An example of the transmission power control algorithm, with gateway sensitivity at -50 dBm.

To illustrate how this works, an example is shown in figure 5.1. In this case, the sensor has four discrete levels of transmission power available: 0, 3, 6 and 10 dBm, and the gateway has a sensitivity of -50 dBm, while all gains are zero. Applying TPC to this simple example, it is shown that the sensor will stay at the lowest transmission power level for as long as possible, before gradually increasing it as the FSPL increases. As soon as the FSPL gets too high, it can be seen that the received power is below the sensitivity, and no data can be delivered. Note that this graph is merely an example, and not the results of a simulation.

## 5.2 Selecting Responsible Gateways

As previously explained, selecting which gateways to make responsible for replying to the sensors is an important aspect of reducing energy consumption in the sensors. In this section, four methods for selecting gateways are proposed, starting with some rather simple and naive ones, and ending with a more complex one. They are tested and compared in chapter 7.

One design choice made by the author that is valid for all the methods, is that the cloud only evaluates which gateway to choose upon reception of a message. This is due to the fact that the cloud does not know the status of the gateways and sensors at all times: all information that is known comes from the received messages. For instance, the gateway that was previously the optimal choice might now be offline. So the safest approach is to only evaluate a gateway as the optimal choice when it indirectly shows itself as a potential candidate by forwarding a message.

Another important design choice that was made by the author is that responsible gateways are only changed between unique messages, and not between retries of the same message. There are three reasons for making this decision. The first is that it is more risky to change responsible gateways between retries, as the time between them is shorter than the time between unique messages. A consequence of this is that the system might end up in a situation where no gateways have replies for a sensor, because the replies are in transit from the cloud to a gateway.

The second reason is that if the cloud is allowed to change gateways between retries, keeping track of the downlink connections gets quite complicated and requires a lot of overhead. The final reason is that changing optimal gateways between retries can cause too frequent switching. A short example of this follows in table 5.1.

Table 5.1: Message received via gateways A and B.

| Message ID | Gateway ID | Timestamp | Signal Strength | Attempt |
|---|---|---|---|---|
| 1 | A | 00:01:02 | -20.05 dB | 1 |
| 1 | B | 00:01:05 | -22.50 dB | 1 |
| 1 | B | 00:02:04 | -19.40 dB | 2 |
| 1 | A | 00:02:05 | -19.03 dB | 2 |

In this example, gateways are chosen based on having the highest signal strength. Initially, gateway A is the optimal choice, but a retry happens, and the next attempt is received with different signal strengths. This time, the message is received via gateway B first, and it has a better signal strength than what A had previously. If changing of optimal gateways is accepted between retries, the cloud would now instantly make gateway B the optimal gateway. However, one second later the message is received via gateway A, which has an even better signal strength, and yet another switching of the optimal gateway happens. Limiting the changing of responsible gateways to happen between unique messages and not between retries, will therefore lead to a more slowly changing system, which hopefully avoids leaving a sensor in a situation where sometimes several gateways have replies for it, and other

times no gateways have replies for it.

This design choice also means that for every message the cloud receives (even retries), the gateway that forwarded the message is reviewed for the position as responsible gateway. However, the gateway that ends up being the actual responsible gateway is only decided after all retries for the specific message have been received. Thus all the gateway evaluation algorithms only compare and nominate potential candidates, but the actual election of a new responsible gateway happens after all the potential retries have happened. Thus the selected gateway is the best one out of all the forwarded attempts that were received for a unique message (i.e. messages with unique IDs).

**Algorithm 5.1** Gateway Selection: General Procedure

```
1: current_responsible_gateway[...] ← [none, none, ...]
2: next_responsible_gateway[...] ← [none, none, ...]
3: t_retry, max_retries

4: procedure HANDLE MESSAGE(message msg, sensor s, gateway g)
5:     if msg.id is new from s then
6:         do current_responsible_gateway[s] ← next_responsible_gateway[s] in
7:             t_retry · (max_retries + 1) minutes
8:     end if

9:     EVALUATE GATEWAY(msg, s, g)
10: end procedure
```

The general procedure for selecting the new optimal gateway is written as pseudo code in algorithm 5.1. As seen in the algorithm, the author decided to use timing to ensure that gateways are only changed between transmissions of unique messages. For some sensors, this method might not work, but in the simulator, all the sensors are identical, so $t_{retry}$ and $max\_retries$ is known.

As long as the transmission interval between sending of unique messages is longer than the time it takes for a sensor to attempt $max\_retries$ retries, this method is reliable. This is also the easiest way to implement this, as the cloud has no way of knowing whether more retry attempts are on their way, or if a sensor will perform more retries.

The general procedure for selecting responsible gateways is therefore quite simple: for all attempts of a unique message that are received, evaluate the gateways that forwarded the attempts as candidates for being the responsible gateway. When the timer is out and no more retries will come, set the best candidate that was found as the new responsible gateway. Repeat this process indefinitely. In the next sections, the proposed methods for evaluating

gateways as candidates for being responsible gateway are presented.

### 5.2.1 Best Average Signal Strength (BASS)

In a WSN where all connections are stable (i.e. never blocked) and static (gateways never go offline), a valid method for selecting the optimal gateway is by choosing the one that gets the strongest signal strength from each sensor. This reduces the probability of transmission errors, and if the sensors use a TPC algorithm that relies on signal strength, it may also reduce their energy consumption.

Should the assumption that all connections are static not be valid, and a gateway suddenly goes offline, it will not be an issue, since, as mentioned in the introduction to this section, all algorithms only evaluate sensors that confirm that they are online by forwarding messages.

One weakness with this method occurs if only the newest signal strength from a connection is used to decide. Imagine that a sensor is located such that between it and the gateway, there is a thick door that only opens a few times every day. When the door is closed, the signal strength is quite low, but when it is open, this gateway has the best signal strength for that sensors out of all the gateways. Selecting gateways based on only the newest signal strength value will cause this gateway to be responsible when the door is open, but when it closes after a few minutes, it is no longer such a good choice, messages are missed and the cloud needs to change gateway again.

The apparent solution to this is to use the average signal strength for each connection rather than the newest signal strength. Averaging works fine, assuming that most patterns in the network stay the same, and no connections have some extreme signal strengths that will make the average unrealistically high, even though they are worse options most of the time. However, since constant patterns are unlikely, and since many connections will experience sudden changes due to external factors from time to time, a moving average is even better, as it calculates the average of the last $x$ signal strengths instead of the average of all signal strengths from the beginning of time. This moving average is explained in section E.2.

Algorithm 5.2 shows how the Best Average Signal Strength (BASS) method has been implemented in the simulator. In order to use the least naive method, it was decided to use a moving average with a window size that can be set in the simulator configuration file. Note that the responsible gateway is decided for each connection, so the method must monitor each (sensor, gateway) pair and choose gateways accordingly. Also note that the algorithm only selects a candidate for being the responsible gateway, and that it is not set as the actual responsible gateway until the timer in the cloud times out.

**Algorithm 5.2** Gateway Evaluation: Best Average Signal Strength

```
1: current_responsible_gateway[...], next_responsible_gateway[...]
2: d_min ← 3

3: procedure EVALUATE GATEWAY(message msg, sensor s, gateway g)
4:     update avg. signal str. for (s, g) with new signal strength from msg
5:     g_c ← current_responsible_gateway[s]
6:     g_n ← next_responsible_gateway[s]
7:     if avg. signal str. for (s, g) > avg. signal str. for (s, g_n) + d_min then
8:         next_responsible_gateway[s] ← g
9:     else if g == g_c then
10:        if avg. signal str. for (s, g) > avg. signal str. for (s, g_n) - d_min then
11:            next_responsible_gateway[s] ← g
12:        end if
13:    end if
14: end procedure
```

In the introduction to this section, the problem of too frequent switching of optimal gateways was mentioned. The factor $d_{min}$ in algorithm 5.2 is another adjustment factor that is added to avoid too frequent switching. The idea is that there is no point in switching gateways if one is only 0.01 dB better than the second best one, as this is too close and might change very quickly. The minimum difference factor $d_{min}$ is therefore included, and it can be set in the simulator configuration file. In the pseudo code it is set to 3 dB, as an increase in 3 dB is roughly equivalent to doubling the power of the signal.

Table 5.2: Message received via gateways A and B.

| Message ID | Gateway ID | Timestamp | Sig. Str. | Avg. Sig. Str. |
|---|---|---|---|---|
| 4 | B | 00:01:02 | -12 dB | -15 dB |
| 4 | A | 00:01:05 | -10 dB | -16 dB |

In the very simplified scenario shown in table 5.2, the responsible gateway is initially gateway A, with an average signal strength of -19 dB. Then a message is received from gateway B, which now has an average signal strength of -15 dB. Since this is the first time the message with ID 4 has been received, gateway B is automatically the next candidate for responsible gateway. Next, the same message is received from gateway A, which with this new message now has an average signal strength of -16 dB. This is not as good as for the current candidate (gateway B) with -15 dB. However, by changing gateway now, only an improvement of 1 dB

is gained, rather than 3 dB. To avoid frequent switching due to this, line 10 is used to let the current responsible gateway win as the candidate if its average signal strength is less than 3 dB lower than the new candidate.

### 5.2.2 Lowest Retry Count (LRC)

One weakness with the BASS method is that it does not monitor how reliable the connections between gateways and sensors are. This is a problem if a gateway that has good signal strength for a given sensor gets blocked from that sensor, and is unable to reply. This leads to retries, which again leads to increased energy consumption. To cope with this, the Lowest Retry Count (LRC) method selects gateways primarily based on the number of retries caused by a gateway for a given sensor. The pseudo code for this method is shown in algorithm 5.3.

**Algorithm 5.3** Gateway Evaluation: Lowest Retry Count

```
1:  current_responsible_gateway[...], next_responsible_gateway[...]

2:  procedure EVALUATE GATEWAY(message msg, sensor s, gateway g)
3:      update avg. signal str. for (s, g) with new signal strength from msg
4:      g_c ← current_responsible_gateway[s]
5:      g_n ← next_responsible_gateway[s]

6:      if msg is a new retry then
7:          count one more retry caused for s by g_c
8:      end if

9:      if retries caused for s by g < retries caused for s by g_n then
10:         next_responsible_gateway[s] ← g
11:     else if retries caused for s by g == retries caused for s by g_n then
12:         if g == g_c then
13:             next_responsible_gateway[s] ← g
14:         else if g_n ≠ g_c then
15:             if avg. signal str. for (s, g) > avg. signal str. for (s, g_n) then
16:                 next_responsible_gateway ← g
17:             end if
18:         end if
19:     end if
20: end procedure
```

As with BASS, in case of a tie between the next responsible gateway and the new candidate,

the new candidate wins if it is the current responsible gateway. This reduces the amount of unnecessary switching of responsible gateways. Also note that if there is a tie between the nominated candidate and the one being evaluated, and neither of them is the current responsible gateway, the one with the better average signal strength wins. The reasoning for this is that if two candidates are equally stable, using the one with the better signal strength should lead to fewer transmission errors, and so lower energy consumption.

An important thing about this algorithm is that it does not look at the uplink (sensor to gateway) and downlink (gateway to sensor) connections separately. It only counts retries that happened for a sensor, and blames them on the gateway that was responsible for replying to the sensor when the retries happened. The retry could have happened due to the sensor being unable to transmit its message to the gateway, but also due to the gateway being unable to deliver the reply to the sensor. It is therefore a quite simple, perhaps even a bit naive algorithm, but it might be an improvement from only using signal strength.

To be able to cope with changing environments, windowing is used to monitor the retry count. As with the signal strength method, the window size is set in the configuration file. The window size is set as the number of unique messages to count retries for, so if the window size is 10, only retries that happened during the last 10 unique messages will be counted. This is not a moving average, but a moving counting window, as further explained in section E.2.

### 5.2.3 Highest Connection Reliability (HCR)

Taking this one step further, the Highest Connection Reliability (HCR) method explained in this section looks at the uplink and downlink connections separately. In different systems, the importance of the uplink and downlink connections might be unequal, so a weighted average of them is used for evaluating new gateways. These weights can be set in the simulator configuration file, so for instance, one can say that the downlink should account for 75% of the decision, and the uplink for the rest. The pseudo code for this method is shown in algorithm 5.4

In the pseudo code the weights are identical for demonstration purposes; however, depending on the hardware, one might weigh uplink and downlink connections differently, as their contribution to the energy consumption could be different. The motivation behind this method is that counting retries is too simple, so monitoring both directions of the connections simultaneously makes the method more dynamic and customizable.

Like with the previous two methods, a threshold for when to switch gateways is also included here. This threshold, $K$, is set as how many percentage points more reliable a connection

**Algorithm 5.4** Gateway Evaluation: Highest Connection Reliability

```
1: current_responsible_gateway[...], next_responsible_gateway[...]
2: W_u ← 0.5, W_d ← 0.5, K ← 0.05

3: procedure EVALUATE GATEWAY(message msg, sensor s, gateway g)
4:     update reliability of uplink (r_u) and downlink (r_d) for (s, g) based on msg
5:     g_c ← current_responsible_gateway[s]
6:     g_n ← next_responsible_gateway[s]
7:     if W_u · r_u[s, g] + W_d · r_d[s, g] > W_u · r_u[s, g_n] + W_d · r_d[s, g_n] + K then
8:         next_responsible_gateway[s] ← g
9:     else if g == g_c then
10:        if W_u · r_u[s, g] + W_d · r_d[s, g] > W_u · r_u[s, , g_n] + W_d · r_d[s, g_n] − K then
11:            next_responsible_gateway[s] ← g
12:        end if
13:    end if
14: end procedure
```

has to be. In the pseudo code it has been set to 0.05, or 5 percentage points. This reduces the amount of gateway switches. Like with the signal strength method, a check is also added to ensure that the order of arrival of messages does not make any difference to the switching (line 9). If the current responsible gateway is evaluated, it wins over any better candidates, as long as they are not $K$ or more percentage points more reliable. Like with retries, uplink and downlink reliability also uses a moving window to cope with changes in the network.

Finally, it must be mentioned that this method does not differentiate between connections that are available in standard mode and boost mode, it simply looks at uplinks and downlinks. Also, downlinks can only be monitored for gateways that send messages to sensors, which are gateways that are or have been responsible gateways. The connection reliabilities are therefore only partial estimates of all the connections.

### 5.2.4 Lowest Expected Energy Consumption (LEEC)

So far, none of the proposed methods have used energy consumption as a direct way to choose a gateway, even though the goal is to reduce energy consumption. The main reason for this is that with only signal strength, number of retries and the reliability of uplink and downlink connections available as decision parameters, energy consumption in the sensors cannot be estimated.

However, if TPC is used by the sensors and the cloud knows the transmission power of each

sensor, the cloud can estimate the energy consumption for all sensors on all possible connections. This makes it possible to detect whether an unstable connection will actually lead to a lower energy consumption, as briefly discussed in section 5.1. Since the energy calculated here is an expected estimation of the energy consumption, and not an exact measurement, this method is referred to as the Lowest Expected Energy Consumption (LEEC) method.

In order to balance increased energy consumption introduced by an unstable connection, the estimation of the total energy consumption required to deliver each unique message must include the probability of retries. Equation 5.3 shows how the total energy consumption for transmitting one message can be estimated, where $P_r$ is the probability for a retry to happen, $a$ is the maximum number of allowed retries and $\mathrm{E_{Tx}}$ is the energy consumed when transmitting one message, as modelled in equation 4.8. The derivation of this equation is explained in section A.2.

$$\mathrm{E_{Total}} = \mathrm{E_{Tx}} \left( 1 + \sum_{n=1}^{a} {P_r}^n \right) \tag{5.3}$$

$P_r$ is calculated from knowing the reliability of both the uplink and downlink connections. Recall that a sensor retransmits a message if it does not receive a reply in time. This can happen both if the gateway does not receive the sensor's message, as it only replies to messages that it gets, and also if the sensor does not receive a reply that was in fact sent by a gateway. Since the reliability of these connections is estimated as $r_u$ and $r_d$ for the uplink and downlink connections respectively, they are used to calculate the probability for a retry to happen on a given connection, as shown in equation 5.4.

$$P_r = (1 - r_u) + (1 - r_d) \cdot r_u \tag{5.4}$$

If boost mode is enabled, equation 5.3 is not completely accurate. Boost mode must be included as an extra cost that happens if all retries in standard mode fail. This is shown in equation 5.5 and derived in section A.2. Subscript B and superscript Boost denotes boost mode. It is therefore clear that this method needs to distinguish between boost mode and standard mode when monitoring uplinks and downlinks.

$$\mathrm{E_{Total}} = \mathrm{E_{Tx}} \left( 1 + \sum_{n=1}^{a} {P_r}^n \right) + {P_r}^{a+1} \cdot \mathrm{E_{Tx}^{Boost}} \left( 1 + \sum_{m=1}^{a_B} {P_{r_B}}^m \right) \tag{5.5}$$

An example of a scenario where the more unstable gateway is favourable with boost mode disabled, follows. Assume that a sensor is in range of two gateways. The size of the message is the same regardless of which gateway is chosen. The first gateway is completely stable, so

$P_{r_1} = 0$, while the other gateway is stable 90% of the time, so $P_{r_2} = 0.1$. However, in order to reach the first gateway, the sensor must double its transmission power compared to when reaching the second gateway, so $p_1 = 2 \cdot p_2$.

Using equation 4.8, the result is that the energy required to reach gateway 1 is twice that of what is required to reach gateway 2: $\mathrm{E_{Tx1}} = 2\mathrm{E_{Tx2}}$. With the maximum amount of retries set to 3, this gives the following expected energy consumption for successfully transmitting one message to gateway 1 and gateway 2, respectively.

$$\mathrm{E_{Total1}} = \mathrm{E_{Tx1}} = 2 \cdot \mathrm{E_{Tx2}}$$

$$\mathrm{E_{Total2}} = \mathrm{E_{Tx2}} + 0.111 \cdot \mathrm{E_{Tx2}} = 1.111 \cdot \mathrm{E_{Tx2}}$$

It is thus evident that in some scenarios when TPC is used, selecting the responsible gateway is a compromise between stability and signal strength. However, for the cloud to be able to make this decision, it must know the sensor's current transmission power level. The only way to achieve this is to make the sensor include the transmission power that it uses in each message that it sends. Luckily, the number of transmission power levels are discrete, and thus final, so each message does not necessarily have to be much larger to include this. Furthermore, the system is made up of few different sensors, all with the same transmission unit, so the energy consumption for each transmission level is known in advance.

So instead of sending the actual transmission power, the sensor can merely send the current level as an integer, and the cloud can convert this to an actual transmission power in mW. Whether this method reduces energy consumption or not is highly dependent on how expensive it is for the sensor to transmit one extra byte. In some systems it might be beneficial, in others it might not. Using TPC in the sensors will also require more processing for them to determine which transmission power to use, however, the energy consumption of this is usually negligible compared to the cost of transmission. [2]

The pseudo code in algorithm 5.5 shows how the derived equation for expected energy consumption is used for selecting the responsible gateway. Here, the average energy consumption also uses an average window to keep up with recent changes, and it includes energy costs for standard mode and boost mode. To avoid frequent changes, just as with BASS and HCR, the average energy cost should be sufficiently lower before a change is made (as set by $K$). In the pseudo code it is required to be at least 10% lower, but any suitable threshold can be used. A check to ensure that the responsible gateway is only changed if the increase is sufficient, regardless of the order that the messages arrive in, is also implemented in this algorithm, as seen on line 12.

**Algorithm 5.5** Gateway Evaluation: Lowest Expected Energy Consumption

```
1: current_responsible_gateway[...], next_responsible_gateway[...]
2: K ← 1.1

3: procedure EVALUATE GATEWAY(message msg, sensor s, gateway g)
4:     update reliability of uplink (r_u) and downlink (r_d) for (s, g) based on msg
5:     update avg. retry probabilities (P_r) from connection reliabilities (r_u and r_d)
6:     update avg. energy consumption for (s, g) with P_r and tx power from msg
7:     g_c ← current_responsible_gateway[s]
8:     g_n ← next_responsible_gateway[s]
9:     if avg. energy cons. for (s, g) ·K < avg. energy cons. for (s, g_n) then
10:        next_responsible_gateway[s] ← g
11:    else if g == g_c then
12:        if avg. energy cons. for (s, g) < avg. energy cons for (s, g_n) ·K then
13:            next_responsible_gateway[s] ← g
14:        end if
15:    end if
16: end procedure
```

## 5.3 Detecting Retries

In order to save energy in the sensors, it has been made clear that reducing their amount of retries is essential. However, a sensor does not tell the cloud directly that it did not receive a reply. Instead, the cloud must detect this from the received messages and their meta-data. The author has identified three different ways to detect retries from this, all of which are explained in the following sections.

### 5.3.1 With Timestamps

As this WSN is made by one manufacturer, and it is made up of rather few different components, timestamps could be used as a reliable way to detect retries. The cloud will have the timestamp showing when each individual gateway received the message from the sensor. Since the retry time for the sensors ($t_{\text{retry}}$) is also known, this can be used directly to deduce whether a message was a retry or not. This approach does, however, require synchronized time between all gateways.

As all the gateways are located in reasonable vicinity of the sensors, the time delay from when the sensor has finished sending the message and when all the gateways have received it

is rather small. It is therefore reasonable to claim that any messages with a timestamp more than $t_{\text{retry}}$ seconds after the first timestamp belonging to the previous transmission attempt, must be a retry. Furthermore, this method can detect retries even when the retry is sent through a new gateway. To explain this more thoroughly, take the following scenario taking place in the network shown in figure 5.2.

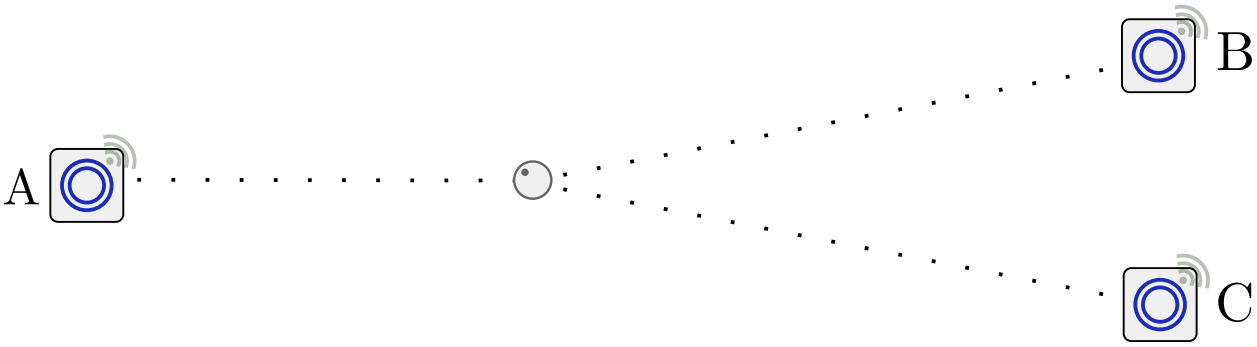



Figure 5.2: A sensor located in the coverage area of three gateways, with all connections being unstable.

Imagine that the sensor broadcasts message 1 at time 00:01:00, and that the cloud receives the message through gateways A and B, while the connection to gateway C is blocked. The result is as shown in table 5.3.

Table 5.3: Message received via gateways A and B.

| Message ID | Gateway ID | Timestamp |
|---|---|---|
| 1 | A | 00:01:02 |
| 1 | B | 00:01:05 |

The timestamp that will be used for comparison now is 00:01:02. Unfortunately, gateway C was the one responsible for sending a reply to the sensor, but since it was blocked, the sensor received no reply, so it attempts a retry after one minute ($t_{\text{retry}}$) at time 00:02:00. This time, however, both gateway A and B are blocked. Now the next message will only come from gateway C, as shown in table 5.4.

Table 5.4: The retry is only received via gateway C.

| Message ID | Gateway ID | Timestamp |
|---|---|---|
| 1 | C | 00:02:04 |

Using the proposed method, the new message's timestamp is one minute and two seconds after the previous earliest attempt. Since this is more than $t_{retry}$ seconds after, we count it as a new retry, and set 00:02:04 as the new comparison time. Notice that the retry is detected even though it came from a gateway that had previously not delivered that particular message. As long as $t_{\text{retry}}$ is known for all sensors and that the time-delay from when a sensor transmits a message until a gateway has received it is sufficiently small, this is a viable method for detecting retries.

**Algorithm 5.6** Detecting Retries: Timestamps

```
1: retry_time ← t_retry
2: last_time[...] ← [none, none, ...]
3: procedure CHECK FOR RETRY(message msg, sensor s, gateway g)
4:     connection c ← (s, g)
5:     if msg.id has been received via c previously then
6:         time_diff ← msg.timestamp - last_time[s, msg.id]
7:         if time_diff > 0 and time_diff ≥ retry_time then
8:             last_time[s, msg.id] ← msg.timestamp
9:             return true
10:        else if time_diff < 0 and time_diff > −t_retry then
11:            last_time[s, msg.id] ← msg.timestamp
12:        end if
13:        return false
14:    end if
15:    last_time[s, msg.id] ← msg.timestamp
16:    return false
17: end procedure
```

The pseudo code for this method is shown in algorithm 5.6. Notice that the algorithm also copes with messages coming out of order. Due to the varying latency of the Internet, the cloud might receive messages in a different order than by timestamps, so the cloud continuously attempts to detect the earliest timestamp belonging to a specific transmission attempt. If the messages in table 5.3 were received in the opposite order, the message from gateway A should still be the one used for comparing timestamps, as its timestamp was earlier than that of gateway B.

### 5.3.2 By Counting

The counting approach is rather different, and should only be used when timestamps are unreliable or unavailable. For instance, if the timestamps are made when the cloud receives

the message from the gateway, timing is unreliable due to the varying latency introduced from sending it through the Internet.

Take the same scenario, with the same network as in figure 5.2 above. Yet again, the sensor sends a message that is only received from gateways A and B, as displayed in table 5.3. The sensor does not receive a reply this time either, so it attempts to resend the message. This time the re-transmission is received by all the gateways, and the information in the cloud about this particular message is shown in table 5.5.

Table 5.5: The re-transmission is received by all gateways.

| Message ID | Gateway ID | Timestamp |
|---|---|---|
| 1 | A | 00:01:02 |
| 1 | B | 00:01:05 |
| 1 | A | 00:02:02 |
| 1 | C | 00:02:04 |
| 1 | B | 00:02:05 |

By counting, it is now evident that there has been a retry, since message 1 has been received two times from gateways A and B. However, if the re-transmission only came through from gateway C, as in the previous scenario, the case would be different.

Table 5.6: The re-transmission is only received via gateway C.

| Message ID | Gateway ID | Timestamp |
|---|---|---|
| 1 | A | 00:01:02 |
| 1 | B | 00:01:05 |
| 1 | C | 00:02:04 |

Recall that this method is based on counting alone, so it does not use the timestamps. Thus from the information in table 5.6, it can not be shown that a re-transmission has happened, as message 1 has only been received once from all gateways. This shows that the counting method has weaknesses in highly unstable environments, and that if the timestamps are reliable, the timestamp method should be preferred over the counting method.

Algorithm 5.7 shows the pseudo code for the counting method. It is rather simple, and sets the total times a message has been received as the highest amount of times it has been

received from a single connection. The author does not recommend using this method unless no other options are available.

**Algorithm 5.7** Detecting Retries: Counting

```
1: times_received_connection[...] ← [0, 0, ...]
2: times_received_total[...] ← [0, 0, ...]
3: procedure CHECK FOR RETRY(message msg, sensor s, gateway g)
4:     connection c ← (s, g)
5:     if msg.id has been received from c previously then
6:         times_received_connection[c, msg.id]++
7:         if times_received_connection[c, msg.id] > times_received_total[s, msg.id] then
8:             times_received_total[s, msg.id] ← times_received_connection[c, msg.id]
9:             return true
10:        end if
11:        return false
12:    end if
13:    times_received_connection[c, msg.id] ← 1
14:    return false
15: end procedure
```

### 5.3.3 From Unique Ciphertext

The final method is based on the assumption that the ciphertexts of all messages are unique, even for retries. An encryption scheme where encrypting the same plaintext results in the same ciphertext every time is known as a deterministic encryption scheme. [53] Unfortunately, deterministic encryption can leak information, because it does not possess semantic security. [54] For this reason, many encryption schemes use an initialization vector to obtain semantical security, where encrypting the same plaintext with the same key will yield different ciphertexts.

If this assumption holds true for the WSN in this thesis, the encrypted messages from the sensors will be different each time, even when they send the exact same message, such as in the event of a retry. Then the pseudo code shown in algorithm 5.8 can be used to detect retries.

Out of the three retry detection methods presented in this chapter, this one is the most reliable one. For this reason, this method is implemented and used in the simulator. However, if this does not hold true, and the retries simply resend the same encrypted message, the timing method should be used instead.

**Algorithm 5.8** Detecting Retries: Unique Ciphertext

```
1: procedure CHECK FOR RETRY(message msg, sensor s)
2:     if msg.id has not been received from s previously then
3:         return false
4:     else if msg.id with the same ciphertext has
                   not been received from s previously then
5:         return true
6:     end if
7:     return false
8: end procedure
```

## 5.4 Monitoring Reliability of Connections

In order to use some of the algorithms in section 5.2, the reliability of the connections between gateways and sensors must be estimated. The reliability of these connections is not necessarily symmetrical, so they must both be estimated individually. Furthermore, the reliability cannot be directly measured by counting successes and failures in transmissions, as the sensors do not share any information regarding this. Therefore this explanation is split into two sections, with section 5.4.1 and 5.4.2 explaining the reliability estimation of uplink and downlink connections, respectively.

### 5.4.1 Uplink Connections

The uplink reliability is measured as the probability that a message sent from a sensor is received by a gateway. Since some messages may not be received in the cloud at all (if a sensor is completely blocked), this probability is only an estimate. It is estimated as the amount of unique messages received from one sensor via one connection, out of the total amount of unique messages received from the same sensor via all its connections.

Algorithm 5.9 shows the pseudo code for estimating uplink reliability. In this algorithm, retries are also counted as unique messages, as every retry is unique due to its unique ciphertext. Note that this method assumes that unique messages are only received once from the same sensor via the same gateway, which is a reasonable assumption in the given WSN, because a sensor should only transmit the exact same message once to all gateways by broadcasting.

By counting messages separately for boost connections and standard connections, this algorithm can also be used to monitor uplinks separately based on their type.

**Algorithm 5.9** Monitoring Uplink Reliability

```
1: procedure CHECK MESSAGE(message, sensor, gateway)
2:     if message has not been received from sensor before then
3:         count one more message received from sensor
4:     end if
5:     count one more message received from sensor via gateway
6: end procedure

7: procedure GET UPLINK RELIABILITY(gateway, sensor)
8:     return (number of messages received from sensor via gateway) / (total number of messages received from sensor)
9: end procedure
```

### 5.4.2 Downlink Connections

The downlink connection is from gateways to sensors, so its reliability is measured as the probability that messages sent from gateways are received by the sensors. Unfortunately, this cannot be measured directly, as a sensor does not send a confirmation that it received a message from a gateway. What it does send is a retry if it did *not* receive a reply to its message. So to estimate the reliability of the downlink connection, one must count the number of messages each gateway sends to a sensor, and detect if any of these were not received by looking for retries.

Algorithm 5.10 shows how retries and the number of sent replies are used to estimate the downlink reliability. Since replies are sent from the cloud to the gateways before they are actually used, counting the number of replies the gateways have actually sent must also be done indirectly. This is done by noting that if a message from a sensor is received via the gateway that was responsible for replying to it, the gateway should have replied to the sensor, and one sent reply can be counted from this gateway to that sensor.

Luckily, the simulator is implemented with switching of responsible gateways to happen between unique messages, and not between retries. In this case, a simple solution such as setting a flag can be used to blame retries on the appropriate gateway. The rationale is that if a brand new message is received, one can assume that the responsible gateway did not yet reply to the sensor. As soon as the message is received via the responsible gateway, it should have replied, and flag can be set. If any retries occur and the flag shows that the responsible gateway did in fact reply, the reply most likely was not delivered, and one missed reply can be counted. This method works fine as long as the responsible gateway remains the same for each unique message ID, if so, it gives a reasonable estimate of the downlink reliability.

**Algorithm 5.10** Monitoring Downlink Reliability

```
1: did_reply[...] ← [false, false, ...]

2: procedure MONITOR DOWNLINK(message msg, sensor s, gateway g)
3:     g_o ← optimal gateway for s
4:     if msg has not been received from s before then
5:         did_reply[g_o] ← false
6:     else if msg is new a retry then
7:         if did_reply[g_o] == true then
8:             count one more reply not delivered for g_o
9:             did_reply[g_o] ← false
10:        end if
11:    end if

12:    if g == g_o then
13:        count one more reply sent from g_o to s
14:        did_reply[g_o] ← true
15:    end if
16: end procedure

17: procedure GET DOWNLINK RELIABILITY(sensor, gateway)
18:    return 1.0 - replies not delivered to sensor from gateway / replies sent to sensor from gateway
19: end procedure
```

This algorithm can naturally be used to monitor boost and standard downlinks separately, by having two flags, and setting the appropriate flags depending on which kind of connection the message was received from.

With all the proposed solutions in place, the next step is to test them. In the next chapter, the test setup is explained.

# Chapter 6

# Test Setup

When measuring the performance of the gateway selection methods, both the simulation settings and the network design affects the results. For instance, a network with few sensors per gateway might behave very differently from one with many sensors per gateway. The reliability of the connections may also be a major factor when it comes to energy consumption and the behaviour of the sensors.

To make sure that the results are as realistic as possible, the sensors, gateways and connections are tuned to behave in a realistic way. This is explained in section 6.1. Furthermore, to make sure the results are not too influenced by a specific network, each method is tested on 12 networks with different characteristics, such as the amount of gateways and sensors, and the distance between these. These networks are discussed in section 6.2.

To make sure there is enough data to properly analyse the results, each simulation is set to run for 7 days. In addition to this, each simulation is set to run 5 times with each network, but with a new seed to the random number generators each time. For consistency, each method is tested with the same seeds so that the results can be compared. In total, this means that each method is simulated 60 times, and the average results from these simulations are used.

## 6.1 Simulator Settings

The simulator has been made rather general, so for simulating traffic for the specific WSN in this thesis, the network and device specific settings need to be set. Since little information has been disclosed about the hardware, many assumptions must be made. These assumptions and the rationale behind them are presented in the following sections.

### 6.1.1 Sensor Settings

One of the important settings is the frequency that the sensors operate at. Unfortunately, the frequency that is actually used is unknown; however, the frequency range 433.05-434.79 MHz is, according to footnote 5.280 in the *ITU Radio Regulations (2016)*, defined to be the Industrial, Scientific and Medical (ISM) band. [55] As this band is quite common for the kind of wireless communication done in this system, this range will be used for calculating signal loss. More specifically, the center frequency of the chosen range (433.92 MHz) is used.

Furthermore, the gain in the sensors must be chosen. The gain depends on the antenna, more specifically the antenna's directivity and efficiency. This means that the direction that the antenna points matters. The sensors in this WSN broadcast their messages, and can be placed anywhere, be it on the ceiling, on windows or under desks. For this reason, the direction in which they point should not matter much, and they should have fairly good transmission power in most directions.

A common type of antennas is omnidirectional antennas, which are frequently used in Wireless Local Area Networks (WLANs). [56] Their advantage is that they have 360° coverage in the horizontal plane. However, there is no universal gain for omnidirectional antennas, as they can vary in construction. For this reason, the hypothetical, isotropic antenna with 0 dB gain (i.e. it radiates with equal strength in all directions) is used in the simulation. [57]

Other system settings that need to be set are the sensitivities and transmission powers of both the sensors and the gateways. According to *Forskrift om generelle tillatelser til bruk av frekvenser (fribruksforskriften), 2012, §8-11*, transmission in the 433.05-434.79 MHz range in Norway shall not exceed a transmission power of 10 mW. [58] Following this, since the author is based in Norway, the testing environment in the simulation uses 10 mW (10 dBm) as the maximum allowed transmission power for both sensors and gateways.

Sensitivity, on the other hand, does not depend on laws and regulations, but on hardware. A very common low energy transceiver that operates in the 433 MHz range is the nRF905 from Nordic Semiconductor. [1] At its best, its sensitivity is as good as -100 dBm. However, since the sensors in this WSN are very small and stingy with energy consumption, their sensitivity is probably not as good, so it is set to -60 dBm in the simulations. For setting the data rate, this transceiver is also used as inspiration, and the data rate is set to 50 kbps.

The transmission power levels must also be set, in cases where TPC is used by the sensors. It has already been made clear that the highest level is 10 mW, which corresponds to 10 dBm. The lowest level is set to 0 dBm, or 1 mW, while the intermediate steps are set to 3 and 6 dBm, as increasing by 3 dBm is roughly equivalent to doubling the power in mW.

[1] https://www.nordicsemi.com/eng/Products/Sub-1-GHz-RF/nRF905

The maximum amount of transmission attempts a sensor is allowed perform is set to 4, as the author was told that this is a common limit used in the WSN. These 4 attempts include the initial transmission, so only 3 retries are effectively allowed. The sending interval of the sensors is set to every 15 minutes, as is normal in the WSN, and the time a sensor waits for a reply before retransmitting is set to 1 minute, as this is frequently used in the actual WSN. This means that for each simulation, 672 messages are sent from each sensor.

Furthermore, the settings were chosen such that sending a message in boost mode is 8 times more expensive than in standard mode. This information was also given to the author by the co-supervisor. No further adjustments were made to the costs of transmitting messages, other than using the equation derived in section 4.7.1.

Ciphertext size has no direct consequences on the results of the simulations, so it is simply set to 2 bytes. Payload size and address size, on the other hand, matters when the LEEC consumption method is compared to the other methods. Recall that when using this method with TPC enabled, the transmission power level must be included in each message, making the messages 1 byte longer. Depending on how large the messages are, the impact of this extra byte will vary. Since addressing is not addressed in this thesis, address size is kept fixed at 2 bytes. To test the impact of the extra TPC byte, simulations are run with payload sizes of both 2 and 10 bytes. This makes the TPC byte account for 20% and approximately 7.7% of the message size, respectively.

### 6.1.2 Gateway Settings

In an actual WSN, one way to reduce the energy consumption in the sensors is by having a higher transmission power and better sensitivity in the gateways. This way, the gateways can receive transmissions with low power from the sensors, and send replies with higher transmission power to the sensors, so that they are able to receive them. However, for the simulation, the effect is the same if the gateways have the same sensitivity (-60 dB) and transmission power (10 mW) as the sensors. The result is then that if a gateway receives a message, the sensor is in its range, and it should be able to send messages to that sensor. Of course, the gateways also use the same frequency as the sensors.

### 6.1.3 Cloud Settings

Since all the gateway selection methods will be tested, the parameters for each method must be set in the cloud configurations. The BASS method requires a minimum signal strength improvement as the threshold for changing responsible gateways. This limit is set to 3 dB,

as this is roughly equivalent to a signal strength that is twice as strong. Other limits could also be used, but the most important thing is that this threshold remains constant for all the simulations, for comparison purposes.

The HCR method requires several parameters to be set. Two of them are the weights for calculating the weighted average of the uplink and downlink connections. Since the networks are randomly generated, the author has no profound knowledge of the behaviour of them, so the uplinks and downlinks are regarded as equally important with weights 0.5. It also requires a threshold in the form of a minimum reliability improvement, which is set to 5 percentage points in the simulation.

Finally, the LEEC method requires a minimum energy improvement that it uses to decide whether to change responsible gateways or not. During the performed tests, this limit is set to 5% improvement, as it is not feasible to change responsible gateways too often, but also because in the long lifetime of a sensor, a reduction of 5% in energy consumption is substantial.

### 6.1.4 Connection Settings

Using sensible and realistic settings for the connections is important, as it is desirable to test whether the suggested methods can work in a real WSN or not. An important decision is the distribution of the three connection types: standard, binary and dynamic. In an actual WSN, it can be expected that most sensors are placed with some care, so not too many of them will experience full blocks on connections. As such, the amount of binary connections are set quite low in the simulation, to 10%. Similarly, very few connections can be expected to have a consistent path loss with almost no variance, so the amount of standard connections is set to a mere 30%. The remaining 60% of the connections are dynamic connections, where full blocks are rare, but varying path loss is noticeable. All connections, including boost mode connections, have this distribution.

In the same manner, symmetric connections (i.e. path loss on uplinks and downlink is consistently equal) is hardly realistic, so it is disabled in the simulation. The likelihood that a binary connection is blocked must also be set. In order to test the robustness of the methods, the binary connections will be set to block for quite some time, so their probability of blocking is set as high as 40%. Next, the maximum change of signal loss in dynamic connections is set to $\pm 10\%$.

Finally, the maximum transmission distance is set for both standard and boost transmissions. Since the simulation is set to be indoors, perhaps in an office or a school, where lots of walls and people block the the transmission path, which reduces signal strength, the maximum

distance for standard transmissions is set to 15 meters. Boost mode normally has twice the range of standard mode, so it is set to 30 meters. Finally, adjustment of FSPL, as explained in section 4.4.2, is enabled, to make the simulation behave in such a way that the maximum range is in fact the maximum range.

### 6.1.5 Parameters to be Tested

Some parameters are not fixed in all the simulations, as their effects are going to be studied in the results. One of them is the window size, because the author suspects that different window sizes will yield different results. As such, simulations are set to run with a window size corresponding to 12 hours (48 messages) and to the full simulation length (672 messages), i.e. no window at all, but a full average.

Boost mode is also enabled for some simulations and disabled for others, to see how it affects the gateway selection methods. Whether TPC is enabled or not, is also an interesting parameter, so simulations are set to run with and without TPC. As mentioned in section 6.1.1, the size of the payload is also tested with 2 bytes and 10 bytes.

To test how the methods perform in extreme networks, two variants of that will also be tested, by introducing new connection distributions. One of them is an unstable one, where 60% of the connections are binary with a blocking probability of 50%, and the remaining 40% are dynamic with a dynamic loss of ±10%. The opposite extreme is a stable network, where 80% of connections are standard (i.e. constant signal strength loss) and the remaining 20% are dynamic with a dynamic loss of ±10%.

All the final test setups are listed in table 6.1 below, and the corresponding JavaScript Object Notation (JSON) file is shown in appendix B.

Table 6.1: Test setups with the tested parameters.

| Test | Boost | TPC | Window Size | Payload Size | Connections |
|---|---|---|---|---|---|
| 1 | No | No | 48 | 2 | Normal |
| 2 | Yes | No | 48/672 | 2 | Normal |
| 3 | Yes | Yes | 48/672 | 2 | Normal |
| 4 | Yes | Yes | 48/672 | 10 | Normal |
| 5 | Yes | Yes | 672 | 2 | Stable |
| 6 | Yes | Yes | 672 | 2 | Unstable |

## 6.2 Network Design

With all the simulation settings in place, the only missing parameters are the networks with the placement of sensors and gateways. This can be varied indefinitely, so the author created a network generator that creates random networks based on the following parameters:

- Grid width and height.
- Number of sensors and gateways.
- Minimum distance between gateways.
- Minimum distance between sensors.
- Minimum distance between sensors and gateways.
- Maximum distance between sensors and gateways.
- Minimum number of sensors per gateway.
- Maximum number of sensors per gateway.
- Minimum number of gateways per sensor.
- Number of grids to generate.

Most of these parameters are only set once. For instance, the size of the grid is not important, because the density of gateways and sensors can be adjusted, and so can the FSPL. For this reason, the grid is set to be 50 by 50 meters. The maximum distance between a sensor and its available gateways is set to 15 meters, to comply with the previous settings. The minimum number of gateways per sensor is set to 2, because if it were 1, the gateway selection problem would be trivial.

Similarly, the minimum amount of sensors per gateway is set to 1, because otherwise, some gateways might end up being pointless. The minimum distance between sensors and gateways is set to 2 meters in order to limit their positioning as little as possible. The same goes for the minimum distance between sensors, which is set to 2 meters. Maximum number of sensors per gateway is only set to avoid network layouts where almost all sensors are located around a few gateways. This parameter, as well as the minimum distance between gateways, is adjusted according to the density of gateways. The different parameters used for creating the final networks are listed in table 6.2.

Table 6.2: Parameters used for generating the testing networks.

| Layout | Sensors | Gateways | Min. dist. between gateways | Max. sensors per gateway |
|---|---|---|---|---|
| 1 | 50 | 5 | 15 | 25 |
| 2 | 80 | 15 | 10 | 20 |
| 3 | 25 | 5 | 15 | 10 |

The goal of the layouts is to test networks with different densities of sensors and gateways, and with different amounts of sensor per gateway and gateways per sensor. The least dense network has 0.01 sensors per $m^2$, and only 5 sensors per gateway on average. The densest network has 0.032 sensors per $m^2$, and roughly 5.3 sensor per gateway, while the one in between has 0.02 sensors per $m^2$, and 10 sensors per gateway. 4 networks of each layout were generated with the network generator, and the test results are averaged with the performance on all of these.

All of the networks that were simulated are shown in appendix B. The figures show the connections and their types when using the normal connection distribution. They also show the large amount of connections that are obtained when boost mode is enabled, which illustrates that the simulator is able to handle quite a lot of connections. The connection types are not shown for the boost connections due to the high amount of connections.

With the test setup summed up, the corresponding simulation results are presented in the next chapter.

# Part III

# Results and Discussion

# Chapter 7

# Simulation Results

All simulation results are presented and discussed in this chapter, but the full details, including standard deviations, means and relative standard deviations, are shown in the tables in appendix C.

## 7.1 Test 1: No Boost and no TCP

As this test was run with boost mode disabled and without TPC, all measured energy consumption is directly related to the number of messages sent, as the message length is constant, so all transmissions consume the same amount of energy. As seen in figure 7.1, BASS caused a significantly higher amount of retries than the other methods. In fact, it caused just over 31% more retries than HCR, which caused the least amount of retries.

Since the sensors are set to send the same number of messages in all the test simulations, more retries should result in higher energy consumption. This is verified in the results, where HCR resulted in about 5% lower energy consumption than BASS. BASS gave the highest energy consumption out of all the methods, which is not surprising, as BASS selects gateways solely based on signal strength, with no notion of retries or energy consumption.

It should also be noted that the difference in the amount of retries caused by HCR and LRC is negligible. This makes sense, because they both monitor the reliabilities of connections, and choose gateways based on this. However, since HCR monitors all the uplinks, in addition to the downlinks of responsible gateways, while LRC only monitors the downlinks of responsible gateways, it is a bit surprising that HCR only caused 0.07% less energy to be consumed, compared to LRC. In practice, their performance in networks without boost mode and TPC seems to be almost identical.

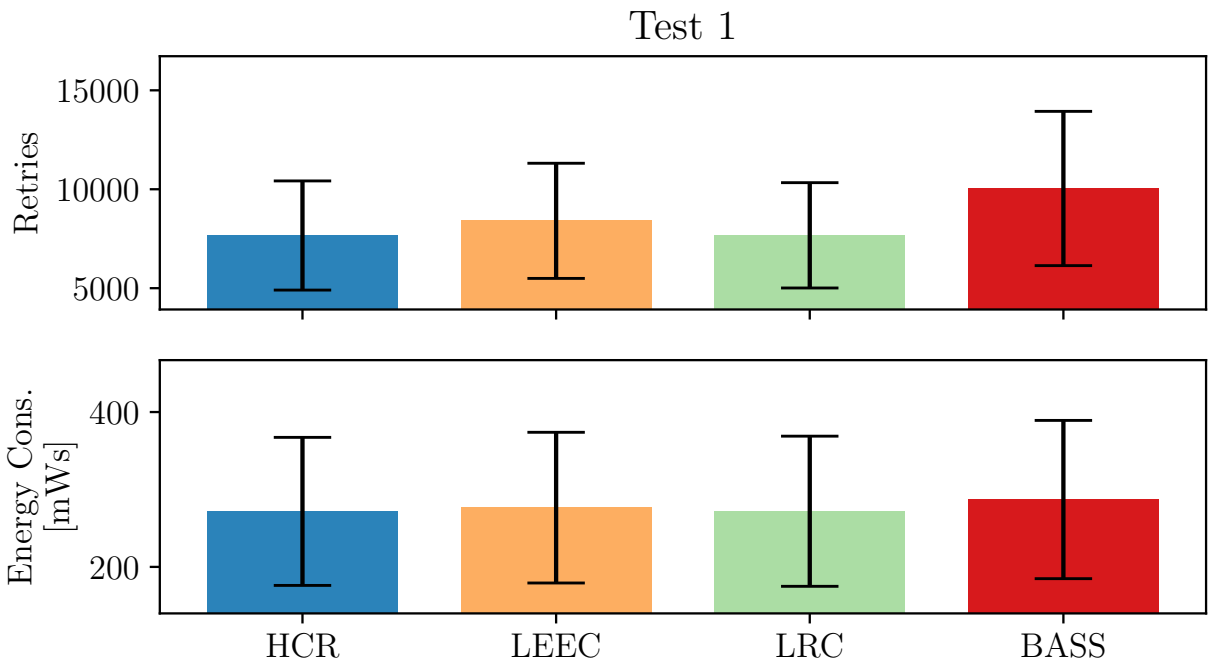



Figure 7.1: Results from test 1.

The fact that LEEC caused more retries than both HCR and LRC is also a bit surprising, as it uses the same method for monitoring uplink and downlink connections as HCR. The most significant difference between them when boost mode and TPC is disabled is the switching threshold in algorithm 5.5, which only changes responsible gateways if the expected energy consumption is 5% lower. For detailed results from this test, see table C.1.

## 7.2 Test 2: Boost without TPC

The goal of this test is twofold: firstly, to be able to measure the impact on energy consumption caused by introducing TPC in the next test, and secondly, to investigate the importance of the window size. Figure 7.2 shows the results obtained when the window size was set to 12 hours. Here it is clear that BASS outperformed all the other methods by far, with an energy consumption 29.5% lower than HCR (the highest), and approximately 16.5% lower than its closest competitor, LRC.

This was achieved with a significantly higher amount of retries: BASS caused more than twice as many retries as HCR, and almost 1.9 times as many as LEEC. However, it hardly caused any messages to be sent in boost mode compared to the other methods, in fact, it resulted in 57% fewer messages sent in boost mode than LRC, which caused the second fewest boost transmissions. This is mainly why it lead to a significantly lower energy consumption.

One possible reason for the low amount of boost transmissions for BASS, is that while in boost mode, the sensors can reach both the gateways that are in range when in standard mode, and the gateways that are further away, up to the maximum boost mode range. Unless obstructed, the closer a gateway is, the stronger the signal it will receive, due to the FSPL. BASS will then often select a closer gateway, which will respond in boost mode, but as it is close enough to be reached in standard mode, the sensor will revert back to standard mode when it receives a reply, which reduces the amount of boost transmissions.

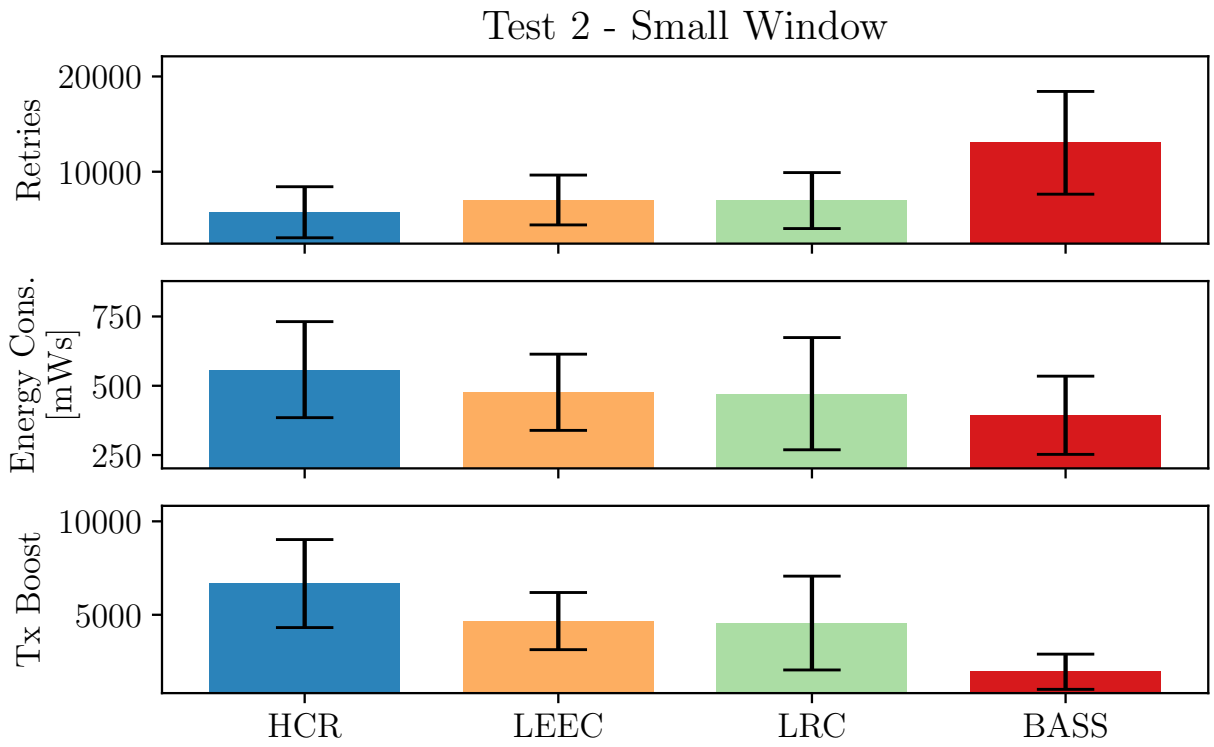



Figure 7.2: Results from test 2 with small window.

The fact that HCR caused many boost transmissions compared to the other methods also makes sense, as it only tries to use the most reliable connections, and since the distribution of connection types was equal for both standard mode connections and boost mode connections, the likelihood for it to select boost mode connections and standard mode connections was the same. This naturally lead to a quite high energy consumption compared to the other methods, which used boost transmissions less frequently.

Also, compared to the previous test, the energy consumption increased for all methods. This is as expected, as the introduction of boost mode allows each sensor to make more transmission attempts than previously (retries both in normal mode and boost mode), and because boost transmissions require more energy. As expected, the amount of retries was reduced for all methods, except BASS. This happens because with boost mode, the gateway selection methods get more connections to choose from, and due to the identical connection

type distribution, they get more reliable connections to choose from than without boost mode. So all the methods that use reliability should be able to reduce the amount of retries with boost mode enabled.

In figure 7.3, the results with a window as long as the simulation are shown. This window length means that in practice, there is no window, and a running average throughout the whole simulation is used. With this window size, LEEC resulted in the lowest energy consumption, more than 28% lower than BASS. This was obtained as a result of both fewer retries (44.6% fewer than BASS) and fewer boost transmissions (70.3% fewer than BASS).

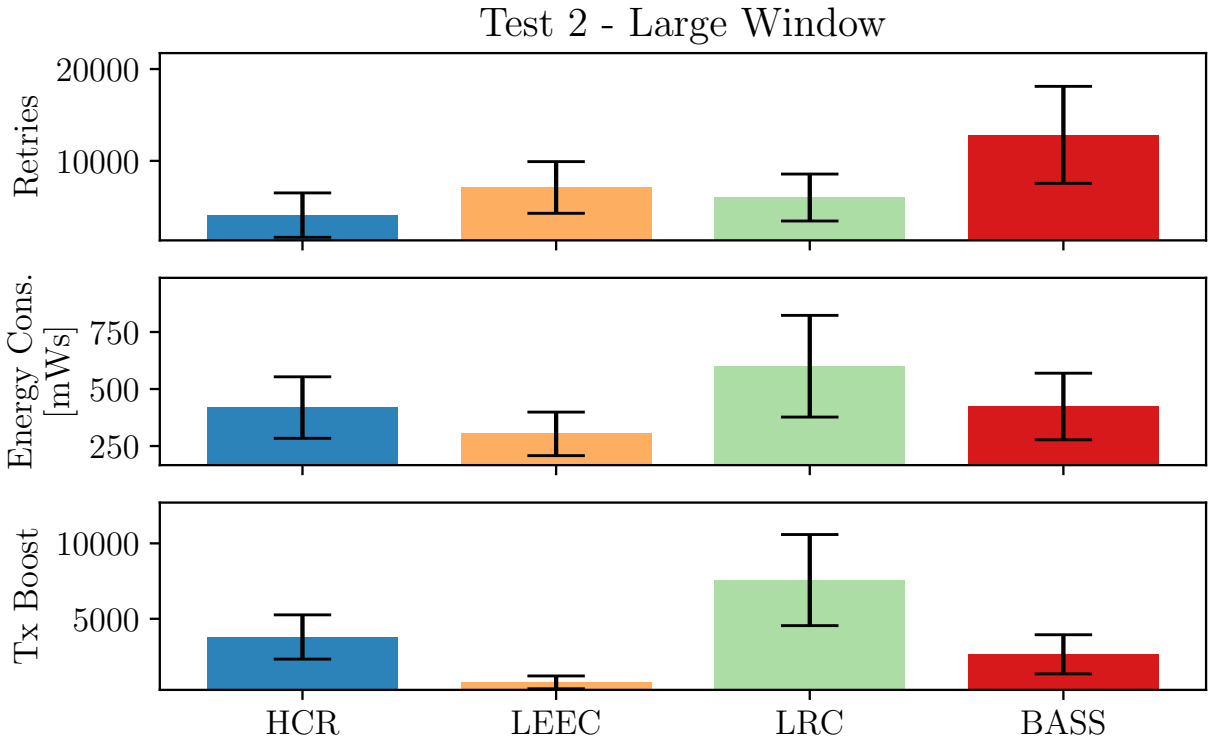



Figure 7.3: Results from test 2 with large window.

Compared to the smaller window, LEEC now lead to 36% lower energy consumption, and HCR lead to 25% lower energy consumption. HCR caused 28.7% fewer retries, and 43.1% fewer boost transmissions, so it makes sense that it lead to lower energy consumption. LEEC's amount of caused retries was almost unchanged (1% increase), but the amount of boost transmissions was reduced by 83.2%.

The fact that LEEC achieved lower energy consumption than HCR is expected, and it shows how these two algorithms work in different ways, where HCR only attempts to reduce the usage of unreliable connections, whereas LEEC also takes the extra energy consumption caused by boost transmissions into account, and balances the energy consumption based on both retries and boost transmissions.

LRC and BASS also lead to a lower amount of retries with this new window size, but they lead to a noticeable increase in the amount of boost transmissions. This made them both cause higher energy consumption: 27.3% more for LRC and 7.6% more for BASS. For LRC, this makes sense, as it attempts to reduce the amount of retries at all costs. For BASS, on the other hand, this might have happened because the average signal strength of most connections varied a lot, as they were dynamic, and so the boost mode connections turned out to be good options for many of the sensors.

Even so, BASS still performed quite well, with a resulting energy consumption only marginally larger than for HCR, even though it caused twice as many retries, and only 30% fewer boost transmissions. It therefore seems likely that the amount of boost transmissions is more important than the amount of retries when it comes to energy consumption for the sensors. For detailed results from test 2, refer to tables C.2 and C.3.

## 7.3 Test 3: Boost with TPC

In this test, all sensors used TPC, and boost mode was enabled. When using a small window of 12 hours, the results are as displayed in figure 7.4. As shown, BASS resulted in the lowest energy consumption by far. The energy consumption was 20.4% lower than when using the runner-up, LRC. Yet again, this seems to be related to the amount of boost transmissions, as BASS caused almost twice as many retries as the other three methods, but half as many boost transmissions as LEEC and LRC.

Compared to test 2 (boost mode without TPC) with small window, introducing TPC in this test lead to an average reduction in energy consumption of 17.9% for all methods except LEEC, which had an increase of 1.4%. The reduction is due to the fact that TPC allows the sensors to transmit messages with a lower power level. LEEC, however, requires the transmission level to be sent with each message, so each message consists of 5 bytes instead of 4, which should lead to an increase of 25% in energy consumption. Luckily, it did not lead to 25% higher energy consumption, as LEEC attempts to keep the energy consumption low, but it still gave the highest energy consumption, 54.9% higher than BASS.

The fact that the larger message size impacts the energy consumption for LEEC can also be seen by comparing it with LRC. LEEC only caused 0.3% more retries and 1% fewer boost transmissions than LRC, but it resulted in an energy consumption that was 23.3% higher. Since the main difference between them is the message size, this is most likely the cause of the higher energy consumption caused by LEEC.

Another contributing factor might be that the TPC algorithm is based on signal strength, so BASS could have an advantage by always choosing the connections with the best average signal strengths, as this could drive the transmission power levels down. LRC could also get an advantage from this, as it uses signal strength to decide in a draw. This could naturally also happen with the other methods by accident, but not as consistently as with the two aforementioned methods.

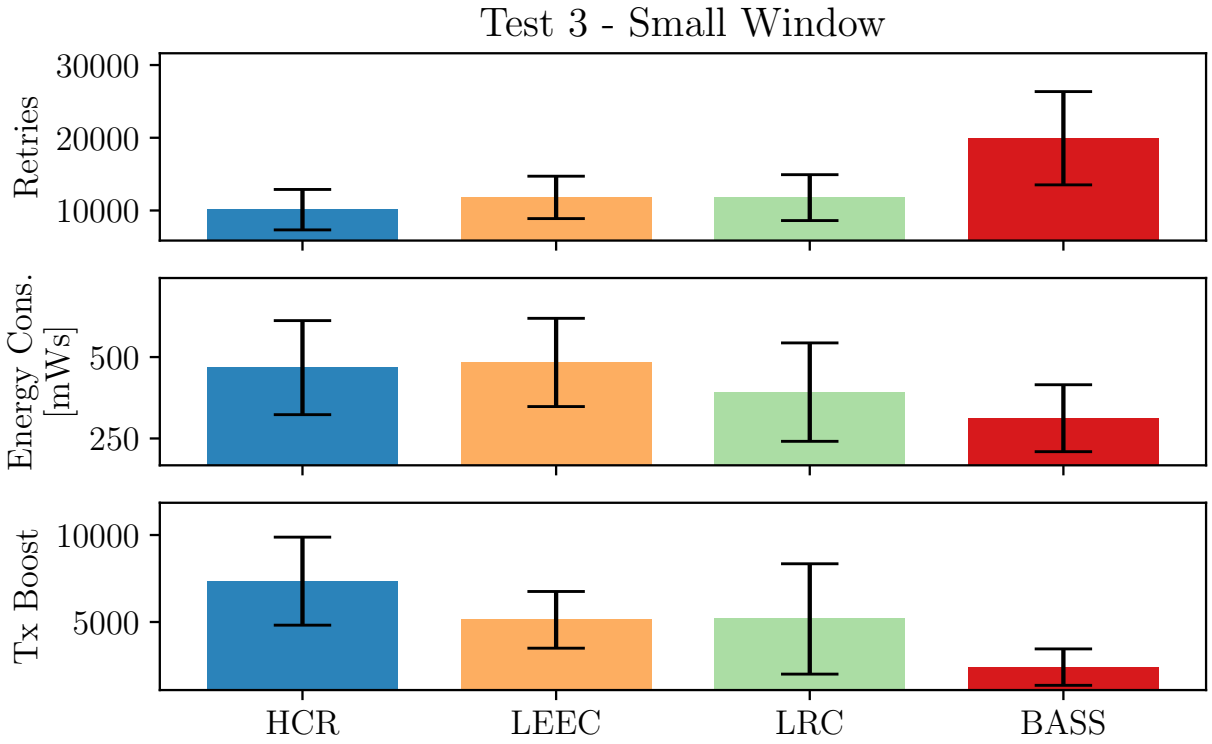



Figure 7.4: Results from test 3 with small window.

The results with the long window are shown in figure 7.5. LEEC resulted in the decidedly lowest energy consumption, 12.8% lower than HCR. Once again, increasing the window size improved the performance of both HCR and LEEC and reduced the performance of BASS and LRC.

This is an example of when LEEC really shines, as it caused quite a high amount of retries, but few boost transmissions. This resulted in it causing the lowest amount of energy to be consumed. LRC, on the other hand, caused few retries, but a tremendous amount of boost transmissions, which lead to an energy consumption that was more than twice that of LEEC.

Out of both the window sizes, HCR caused the lowest amount of retries. Unfortunately it did not lead to the lowest energy consumption. This was achieved by LEEC with a large window, with BASS and HCR quite a bit behind. LEEC also caused the lowest amount of boost transmissions, which was likely the reason for the low energy consumption. For detailed results from this test, refer to tables C.4 and C.5.

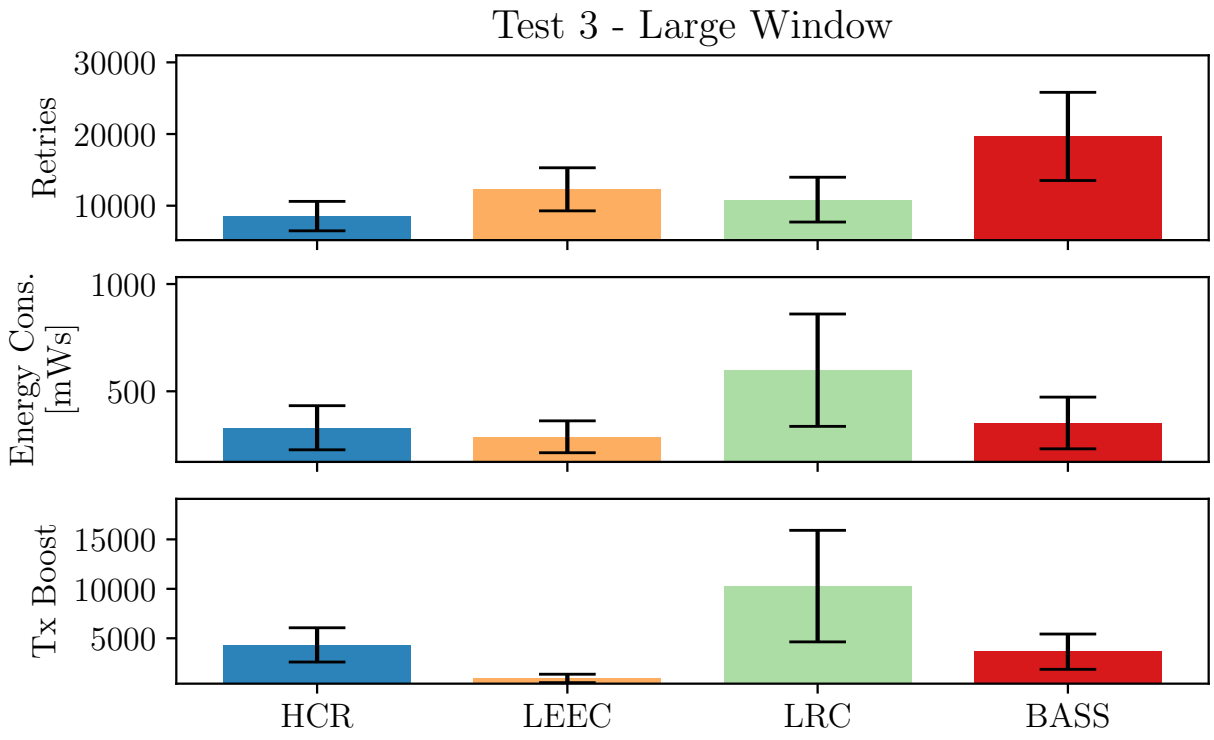



Figure 7.5: Results from test 3 with large window.

## 7.4 Test 4: TPC with Long Messages

In this test, the setup was the same as in the previous one, except that the payload of each message was longer. On average, this almost doubled the energy consumption for all the methods when using a small window, which is not too bad, considering that the message size was tripled, from 4 bytes to 12 bytes.

The results obtained with the small window size are shown in figure 7.6. They show the same pattern as previously, where BASS caused the lowest energy consumption, 20.7% lower than the closest method, LRC, and 33.2% lower than HCR, which had the worst performance.

Yet again, this is connected to BASS having the lowest amount of boost transmissions, 51.8% fewer than the closest method (LEEC). The fact that it by far caused the highest amount of retries did not impact the energy consumption significantly. In fact, HCR lead to the highest energy consumption even though it caused the lowest amount of retries. Unfortunately it also caused the highest amount of boost transmissions, so again, the amount of boost transmissions seems to be the deciding factor.

Another interesting thing to notice is that now that the extra TPC byte had less of an impact on the message size, LEEC no longer caused the highest energy consumption, even with the small window.

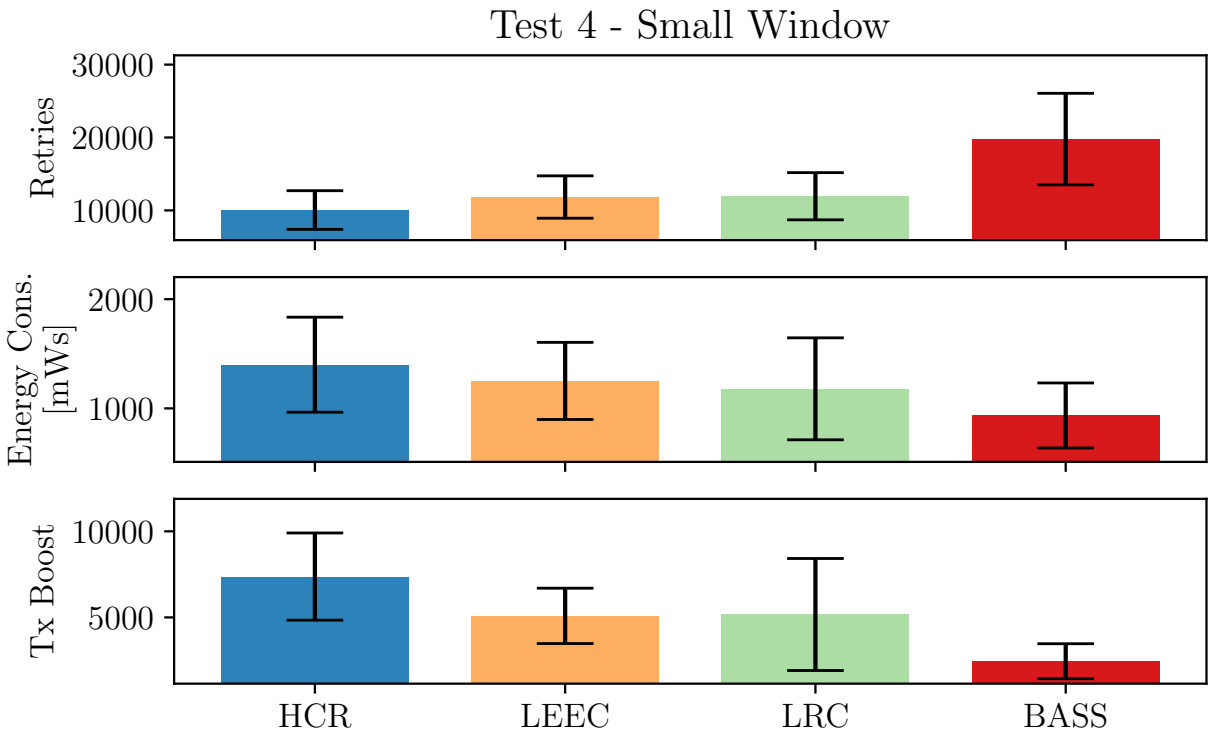



Figure 7.6: Results from test 4 with small window.

As seen in figure 7.7, when increasing the window size drastically, LEEC's performance increased, while LRC's performance got severely degraded. LEEC caused an energy consumption that was 39.8% lower than with the small window, and 22.9% lower than HCR with large window. Again, LEEC achieved this by causing extremely few boost transmissions and some retries. In fact, it caused less than a third as many boost transmissions as BASS.

It is also interesting that HCR lead to lower energy consumption than BASS by causing fewer retries, but more boost transmissions. LEEC was the only algorithm that managed to find a middle point where the total energy consumption was lowered even further. This shows that finding the optimal balance of retries and boost transmissions is the way to achieve the lowest energy consumption. For full details for test 4, see tables C.6 and C.7.

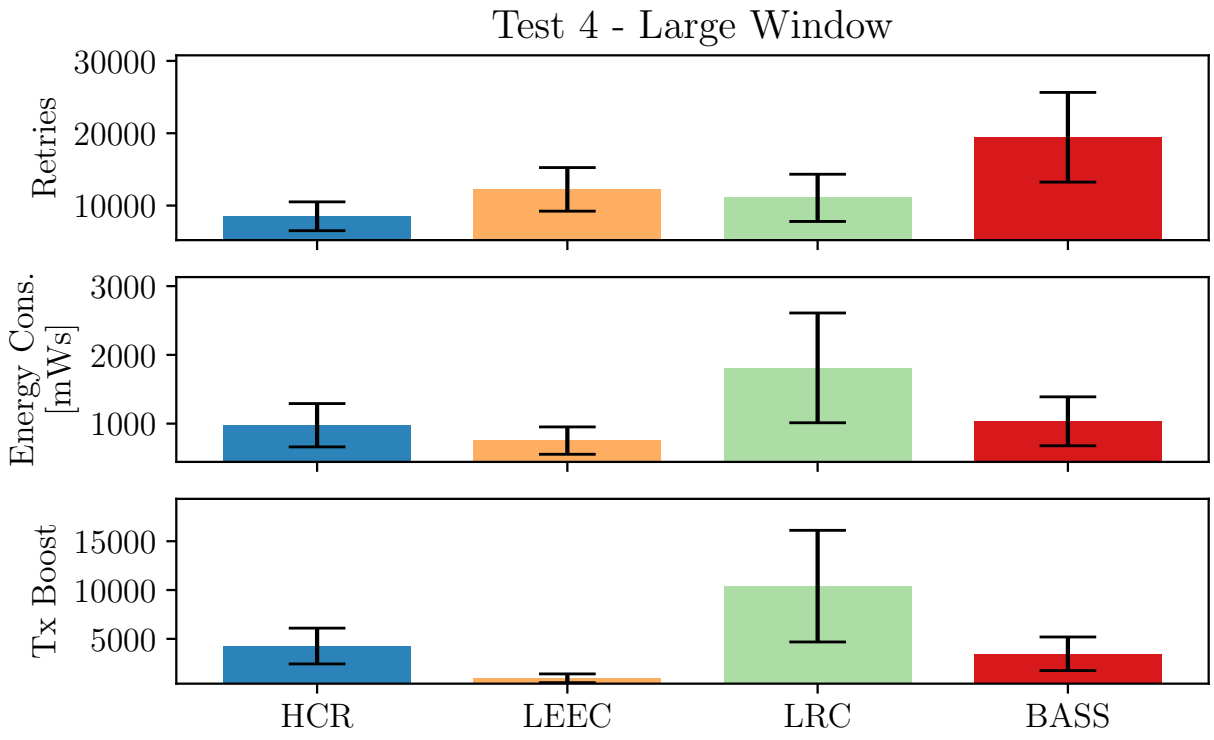



Figure 7.7: Results from test 4 with large window.

## 7.5 Test 5: TPC and Boost with Stable Network

In this test, the network was highly stable, with 80% of the connections having constant signal loss, and 20% having a dynamic signal loss of ±10%. The test was only run with a window size of the full simulation duration. Unlike previously, LEEC did not achieve the lowest energy consumption in this test. In fact, this was achieved by HCR, with BASS just behind, as seen in figure 7.8.

Since the connections were highly stable, the total energy consumption for all methods was very low, even lower than in test 1. This is as expected, as the amount of retries gets dramatically lower because of this. Also note that BASS gave almost as low energy consumption as HCR, even though it caused almost 8 times as many retries and more than twice as many boost transmissions. This most likely happened because the TPC algorithm is based on signal strength, which probably gave BASS an advantage.

Perhaps the strangest thing with these results is that LEEC caused the highest energy consumption despite having 98% fewer boost transmissions and 56.1% more retries than LRC. Either the amount of retries are much more important in a stable network, or LRC got an advantage by the fact that it uses signal strength as the deciding factor in case of a draw in amount of retries, which works in a positive way with the TPC algorithm. For

detailed results from test 5, see table C.8.

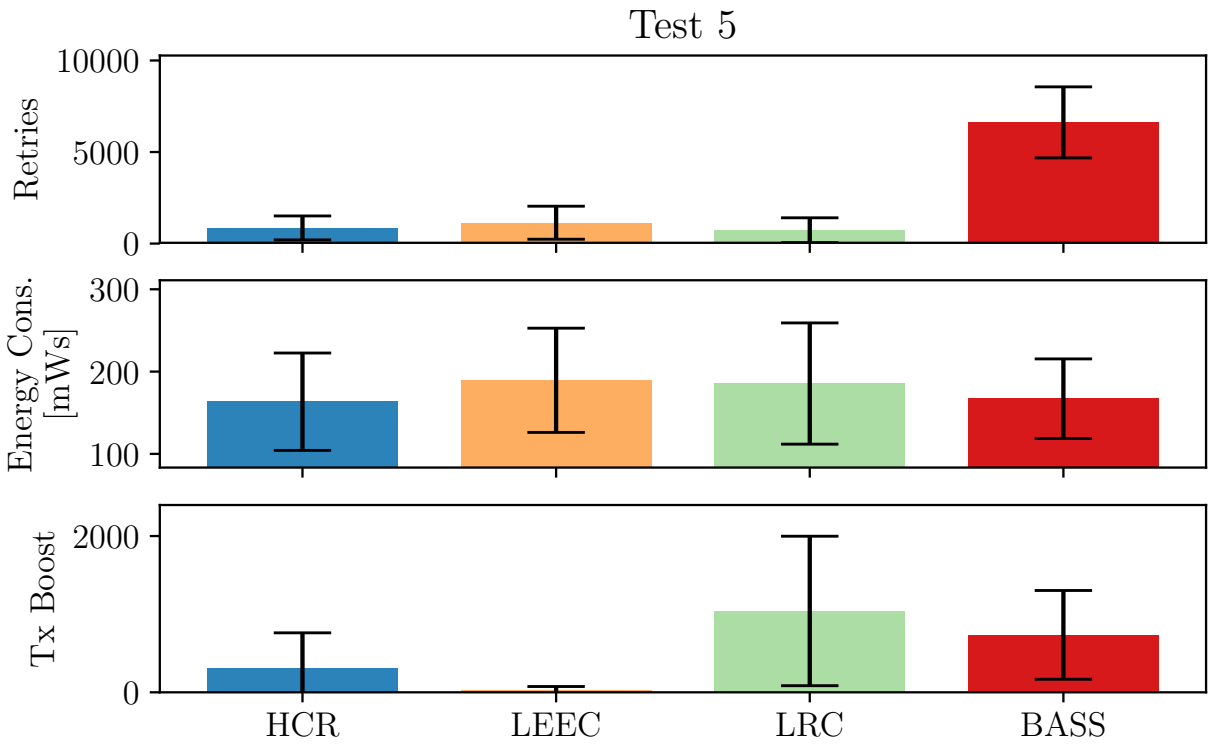



Figure 7.8: Results from test 5.

## 7.6 Test 6: TPC and Boost with Unstable Network

This test was the opposite of the previous one, with a highly unstable network, where 60% of the connections blocked with a probability of 50%, and the remaining connections had a dynamic loss of ±10%. The amount of retries and boost transmissions in this test was naturally quite high, due to the unstable connections. When it comes to energy consumption, HCR, LEEC and BASS performed quite similarly, but HCR lead to a marginally lower energy consumption, 9.5% lower than LEEC and 15.2% lower than BASS.

LRC performed very bad compared to the other methods. This is not surprising, as it only uses retries as a means of selecting gateways. This way, it only monitors the gateways that are responsible for replying at any instant, while the other gateways and connections are not monitored. As such, it has very little information about the system compared to HCR and LEEC. However, it even lead to 25.5% more retries than BASS, effectively making it the method that caused the most retries, which is a bit ironic for a method that is only based on keeping the number of retries low. It is therefore quite unsuitable in an unstable network.

In fact, it seems like the most important factor in keeping the energy consumption low in a highly unstable network is by reducing the amount of retries, rather than having few boost transmissions. However, LEEC most likely caused a higher energy consumption than HCR because of the extra byte it requires in each message, so with longer payloads, it might have lead to the lowest energy consumption. Detailed results for this test are found in table C.9.

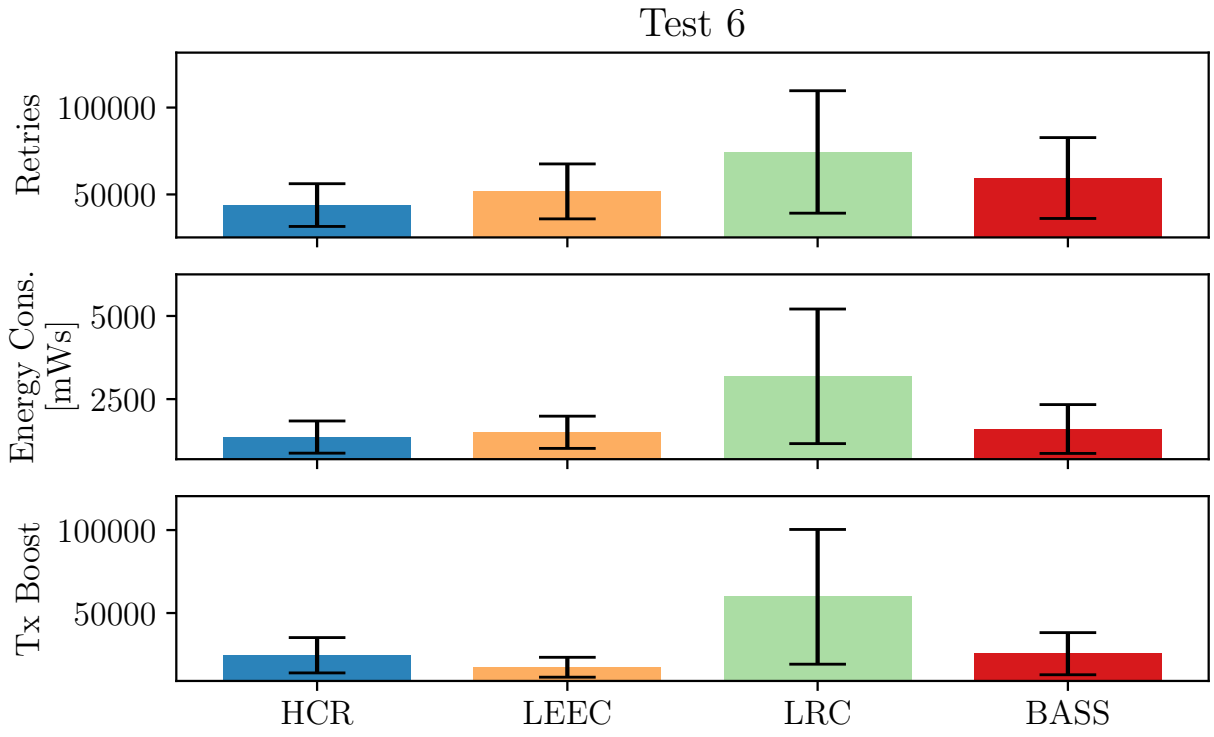



Figure 7.9: Results from test 6.

# Chapter 8

# Discussion

With both boost mode and TPC disabled in the sensors, the total energy consumption for all gateway selection methods remained low, and the worst performing method only caused 5.6% more energy to be consumed compared to the other methods. Without boost mode and TPC, only the amount of transmissions affect the energy consumption, which means that the most effective strategy is to keep the amount of retries at a minimum. It was therefore no surprise that HCR gave the lowest energy consumption, and that BASS, which does not consider retries or stability of connections, gave the highest.

After enabling boost mode in test 2, the energy consumption for all methods increased. On average it was increased 1.6 times, regardless of window size. This comes as no surprise, as with boost mode, each sensor is allowed to attempt 3 retries in standard mode, before switching to boost mode, where it can perform 3 new retries if the initial boost transmission fails. The amount of allowed transmission per message thus doubles, and half of these transmissions can be in boost mode, which is 8 times more energy consuming than standard mode.

On the other hand, retries were reduced by 7% by enabling boost mode. Knowing that the boost mode connections have the same distribution of connection types as the standard ones, this is expected, as it means that the selection methods have a higher amount of reliable connections to choose from when boost mode is enabled. The increased energy consumption is therefore not caused by increased retries, but by transmissions in boost mode.

Introducing TPC in the sensors in addition to boost mode further reduced the energy consumption by 11.6% on average, compared to only using boost mode. Since the sensors send all messages with the highest available transmission power without TPC, while the TPC algorithm allows the sensors to operate with lower transmission power levels in a smart way, it was expected that it would reduce the energy consumption.

Unfortunately, it did come at the expense of an increase in retries of 67.1%, which is significant. This was most likely caused by the TPC algorithm sometimes setting the transmission power too low, resulting in missing replies, which causes retries. Even the amount of boost transmissions was, on average, increased by 20.3%, which means that the reduced energy consumption must solely come from the TPC algorithm. It therefore seems likely that by improving the TPC algorithm, and making the gateway selection methods aware of how it works, the total energy consumption could be further reduced. This way, the selection algorithms could manipulate the sensors' TPC algorithm to reduce the energy consumption, while also avoiding retries and boost mode.

In order to take full advantage of the LEEC method, the transmission power level needs to be appended to each message sent from the sensors. To investigate the impact of this byte, test 4 was run in exactly the same way as test 3, except that the size of the payload in each message was increased from 2 to 10 bytes. Naturally the energy consumption was increased for all methods, in fact it was almost doubled. However, the interesting thing to investigate was whether LEEC's energy consumption relative to the other methods was improved by increasing the size of the payload, as increasing the payload reduced the message size increase caused by the power level byte from 25% to 8.3%.

As LEEC resulted in the lowest energy consumption with the large window, it is used for comparison. With the smaller payload, the energy consumption of LEEC was 12.8% lower than its closest competitor, HCR. With the larger payload, however, the resulting energy consumption was 22.9% lower than HCR, which yet again was its closest competitor. This indicates that even though LEEC increases the size of the messages, it can actually lead to the lowest energy consumption of all the methods, and that if the transmission power level can be sent as a small part of each message, the increased message size is worth the extra energy consumption during transmission.

In the simulator, the transmission power level was appended to each message as a full extra byte. However, in most hardware, the transmission power levels of a transmitter are discrete, and not continuous. So there is a finite amount of transmission power levels, and for instance, by using a nibble rather than a byte, it is still possible to represent 16 transmission power levels, while halving the amount of bits that must be transmitted. This therefore seems like an interesting area to investigate further.

Some interesting patterns were present in all the tests on the normal network, one of which was the effect of changing the window size. Increasing the window size from 12 hours to the full simulation length consistently reduced the energy consumption with HCR and LEEC, and increased it for LRC and BASS. In fact, this pattern was so apparent, that in test 2 - 4, BASS always went from causing the lowest energy consumption to causing the second highest one, LRC always went from causing the second lowest one to causing the highest one

and LEEC always ended up causing the lowest one.

Since this pattern was present both with and without TPC, one can suspect that TPC is not the cause of this, rather, it is caused by the window size itself. When looking at how the algorithms work, it makes sense that HCR and LEEC are impacted similarly by the change of window size, as they both use the estimated connection reliabilities for selecting gateways, in one way or another.

The reliability of the connections is estimated as out of all messages received from one sensor, how many of them (including retries) were received on each specific connection, and out of all replies sent to the sensor, how many of them caused a retry (i.e. were not delivered). It therefore seems like the accuracy of the estimated connection reliability is better when the window is longer. This shows that choosing an appropriate window size is important to ensure good performance in algorithms that use average values, so that they can get reasonable estimates, while also being able to cope with changes in the reliability of the connections.

Unfortunately, this also means that the fact that the energy consumption was reduced for HCR and LEEC with a longer window is only shown to be true for these networks with their specific connection distribution. It is likely that the optimal window size is not one of the extremes that were used in the tests, but one somewhere in between. An interesting approach could be to implement a method that dynamically finds and adjusts the window size to the optimal one, which is probably related to the rate of change of the connections.

It should also be noted that the connections in this simulator are very simplified. For instance, the standard connection with constant FSPL, albeit not frequently used in the tests, is not realistic. Likewise the binary connections have a fixed probability of blocking, and the dynamic connections have a fixed upper and lower limit for how much the FSPL can be randomly increased or reduced. All of this is based on a pseudo-random number generator. This could affect the results by either being too random, such that no patterns are found, and finding the most reliable connection becomes too hard, or by introducing patterns that perhaps should not be there.

In real life, these connections would probably behave very differently depending on unknown external parameters, such as people, doors, wireless transmitters etc. Perhaps there would be less blocking and lower FSPL during the weekends when the office is empty etc. It is thus evident that selecting appropriate window sizes is important, and that the results here in no way show that having long windows for HCR and LEEC is ideal, only that choosing a suitable window size is very important for the performance of these algorithms.

The same can be said for BASS and LRC. BASS uses an average window with the given window size for calculating the average signal strength. How accurately this average represents the current status of a connection depends on the window size, which needs to be adjusted according to how quickly the connection's properties change. For this specific network, it seems like the pseudo-random signal loss that happens on the connections changes rather quickly, so a small window is favourable compared to a large one.

LRC also uses a moving window to count the amount of retries caused by a gateway for a specific sensor. Naturally, this is also linked to the connections, so the window size must be adjusted according to the rate of change for the connections. To further emphasise the importance of the window size: after increasing the window size, HCR experienced an average reduction in energy consumption of 28.2%, LEEC experienced an average reduction of 38.9%, BASS experienced an average increase of 10.3% and LRC experienced an average increase of 44.5%.

The two last tests were only run with the large window size, as their goal was mainly to test the gateway selection methods under two opposite extremes, namely an incredibly reliable network, and a highly unstable one. In the stable one, the average energy consumption was lower than in all the other tests, even the one without boost, of which it was 36.3% lower. This is good, as it shows how all of the methods manage to avoid using the unstable connections too frequently.

In the unstable network, the energy consumption was a lot higher, which is to be expected, as the number of retries and boost transmissions was much higher than in any previous tests. One interesting finding in this test, was that the difference in the energy consumption of the various methods increased. This shows that they do not always perform similarly. In fact, in both the stable and unstable networks, HCR was the best performing method, which it was not in any of the other tests.

In the other tests with a normal network, the patterns clearly show that with the small window size, BASS consistently resulted in the lowest energy consumption, on average 19.2% lower than the second lowest. However, the TPC algorithm in the simulations uses signal strength to decide whether to change transmission power, so BASS might have obtained especially good results with TPC by bringing down the transmission power for the sensors more than the other methods.

With the large window, LEEC always gave the lowest energy consumption, on average 21.1% lower than the closest candidate. In total, the lowest achieved energy consumption for these tests regardless of window size was always obtained with LEEC, which on average resulted in 16.7% lower energy consumption than the second lowest achieved energy consumptions. So in these tests, LEEC was the best performing method, but in general, this will depend on the window size.

A final remark must be made regarding the relative standard deviations of the tests. They are located in appendix C, and by inspecting them, it is clear that the test results highly depend on the underlying networks. Recall that 12 different networks, where 4 and 4 were of the same types, were used in the simulations. The high relative standard deviations illustrate that the energy consumption, boost transmissions and retries vary quite a lot depending on the network. This is why it was decided to use an average of many simulations to verify the methods, so that the results show trends of the methods rather than how they behave in one specific case.

# Chapter 9

# Future Work

For future work, it would be interesting to run simulations with different window sizes to further investigate how it impacts the performance of the various algorithms. Another thing worth looking into is to reduce the number of bits used to send the transmission power level for LEEC, and then investigate how it affects the energy consumption of LEEC relative to the other methods.

The TPC algorithm implemented in the simulator is very simple, yet it still managed to reduce the energy consumption quite a bit. As there has already been conducted a lot of research on TPC, as explained in section 2.3, implementing more complex TPC algorithms is also a possibility.

## 9.1 Improving the Gateway Selection Methods

As the window size was found to be of high importance for the performance of the methods, finding a way to dynamically scale the window for the best results, perhaps based on the rate of change of the measured values (signal strength, retries and connection reliabilities), could be an interesting approach to make the methods reduce the energy consumption further, but also to make their performance more consistent in different network environments.

Designing and testing a new gateway selection method is also an interesting thought. Recall that LEEC selects gateways based on an estimated energy consumption, while BASS simply selects based on the best average signal strength. One surprising strength of the BASS method was that it performed particularly well due to its effect on the TPC algorithm.

A further development of LEEC could be a method that also estimates if using a different

connection could make a sensor lower its transmission power, and then checks whether doing this would reduce or increase the expected energy consumption. The author believes this extra information could lead to even better decisions regarding gateway selection, as well as better performance from the TPC algorithm.

## 9.2 Further Development of the Simulator

Since simulation is a far cheaper way of testing the gateway selection methods than doing it on actual sensors and gateways, further developing the simulator in order to make the results even more realistic is a smart way to go.

Some things can be implemented to make the simulator even more configurable than it is right now. For instance, it is currently not possible to set a separate frequency for boost mode. In the actual system, boost mode operates on another frequency, but this is currently simulated by increasing the energy consumption by a factor when using boost mode. Implementing radio gains is also something that could be done. This is currently not implemented due to a gain of 0 dB being used.

Energy consumption in the sensors is also something that can be made much more realistic. As of now, the only energy consumption that is calculated is the one used for transmitting messages. The calculation of this could naturally be made more realistic, although the version that is already implemented is sufficient for investigating the performance of the various gateway selection methods. It is likely more viable to implement energy consumption that occurs when receiving messages, and perhaps also energy consumption related to data processing on the sensors, for instance for TPC.

Lastly, the simulation of the wireless connections can be improved. For instance, it could be interesting to implement connections that simulate patterns, such as varying signal loss during weekends and weekdays. Furthermore, the dynamic connections should be made even more dynamic, rather than being limited to a fixed limit of increased or reduced signal loss. This also goes for binary connections, which should have a more varying behaviour throughout the simulation.

The path loss on the connections can also be made more realistic. Right now it is implemented as free space path loss, however, loss of signal strength indoors is a bit different. For more realistic signal strength loss, it could be interesting to see other models implemented, but they are many, and there is no unified standard. [29] Thus using the FSPL model for testing is ok, and it might be better to test in a real system rather than making the simulator too complex.

# Part IV

# Appendices

# Appendix A

# Equations

## A.1 Energy Consumption During Transmission

The rationale behind the energy consumption equation used in section 4.7.1 is based on the fact that for most radio transmitters, the power they use during transmission is given. To calculate how much energy they use to transmit a message, the unknown factor is for how long this power is used. Since the data rate for transmission is also normally given, this can be used to calculate the duration of the transmission, and from that the total energy consumption can be calculated, as shown in equation A.1.

$$\mathrm{E_{Tx}}(n, R, P) = n \cdot 8 \cdot \frac{1}{R} \cdot P \tag{A.1}$$

Here $n$ is the number of bytes to be transmitted, $R$ is the transmission rate given in bit/s and $P$ is the power required to transmit in mW. Since the simulator operates with whole bytes for messages, $n$ is simply multiplied by 8. Using the units, it can be seen that the corresponding energy is in the form of mWs (see equation A.2).

$$\mathrm{byte} \cdot \frac{\mathrm{bit}}{\mathrm{byte}} \cdot \frac{\mathrm{s}}{\mathrm{bit}} \cdot \mathrm{mW} = \mathrm{mWs} \tag{A.2}$$

This can also easily be converted to mWh by dividing with 3600 seconds/hour.

## A.2 Estimating Expected Energy Consumption

When estimating the energy consumption needed for a sensor to successfully send a message, one must take retries into account. The energy consumption for one attempt is found with equation 4.8. Letting $P_r$ denote the probability that a sensor must perform a retry, the expected energy consumption for sending one message can be estimated as in equation A.3

$$\mathrm{E_{Total}} = \mathrm{E_{Tx}} + P_r \cdot \mathrm{E_{Tx}} \tag{A.3}$$

where $\mathrm{E_{Tx}}$ is the cost of transmitting one time. However, each transmission comes with the possibility of not being delivered, so the chance that a retry leads to a new retry must be included. This leads to equation A.4

$$\mathrm{E_{Total}} = \mathrm{E_{Tx}} + P_r \cdot (\mathrm{E_{Tx}} + P_r \cdot (\mathrm{E_{Tx}} + P_r \cdot (...))) \tag{A.4}$$

This equation can be generalized as shown in equation A.5 by using a summation, where $a$ denotes the maximum amount of allowed retry attempts.

$$\mathrm{E_{Total}} = \mathrm{E_{Tx}} + \mathrm{E_{Tx}} \sum_{n=1}^{a} {P_r}^n \tag{A.5}$$

However, this equation does not include the risk that a sensor will enter boost mode. If a sensor reaches its limit of allowed retries without getting a reply, it will switch to boost mode and keep on retrying until it reaches the retry limit in boost mode. Using the same derivation as before with boost mode enabled gives equation A.6

$$\mathrm{E_{Total}} = \mathrm{E_{Tx}} \left(1 + \sum_{n=1}^{a} {P_r}^n\right) + P_r^{a+1} \cdot \mathrm{E_{Tx}^{Boost}} \left(1 + \sum_{m=1}^{a_B} {P_{r_B}}^m\right) \tag{A.6}$$

where $\mathrm{E_{Tx}^{Boost}}$ denotes the energy required to transmit one full message in boost mode, $P_{r_B}$ denotes the probability that a retry must be attempted in boost mode, and $a_B$ denotes the maximum amount of allowed retry attempts in boost mode.

# Appendix B

# Configuration File and Networks

The JSON file format that was used for running the tests is shown in this appendix. Some of the values were changed a bit for the different tests, as explained in section 6.1.5. These are the distributions of normal mode and boost mode connections, TPC, payload size, blocking probability, dynamic loss and window size. The JSON file below is the one that was used for test 1.

For test 1, with 50 sensors, 5 gateways and 4 layouts, the number of coordinates adds up to 220. The author deemed all of that information to be excess in this appendix, but it can be provided on request.

The network figures are shown after the JSON file.

Listing B.1: test1.json

```
{
  "sensors": {
    "coordinates": [],
    "multi_coordinates": [[{"X":17,"Y":18}, ...], ...],
    "transmission_power": [0, 3, 6, 10],
    "tpc": false,
    "send_tpc": false,
    "sensitivity": -60,
    "max_attempts": 4,
    "payload_size": 2,
    "address_size": 2,
    "ciphertext_size": 2,
    "start_interval": 0,
    "send_interval": 15,
```

```
      "data_rate": 50000,
      "waiting_time": 1
    },
    "gateways": {
      "coordinates": [],
      "multi_coordinates": [[{"X":41,"Y":34}, ...], ...],
      "transmission_power": 10,
      "sensitivity": -60
    },
    "cloud": {
      "minimum_energy_improvement": 1.05,
      "minimum_signal_strength_difference": 3,
      "minimum_reliability_improvement": 0.05,
      "selection_method": "highest_signal_strength",
      "weight_downlink": 0.5,
      "weight_uplink": 0.5
    },
    "connections": {
      "distributions": {
        "standard": 0.3,
        "binary": 0.1,
        "dynamic": 0.6
      },
      "block_prob": 40,
      "dynamic_loss": 10,
      "symmetric_connection": false,
      "normal_range": 15,
      "adjust_fspl": true,
      "boost_range": 30,
      "boost_cost": 8
    },
    "simulation": {
      "simulation_speed": 60000,
      "frequency": 434920000,
      "seed": "Never Cry Shitwolf",
      "duration": {
        "days": 7,
        "hours": 0,
        "minutes": 0,
```

```
        "seconds": 0
      },
      "window_size": 48,
      "title": "Test 1",
      "boost": false,
      "bit_cost": 1
    }
  }
```

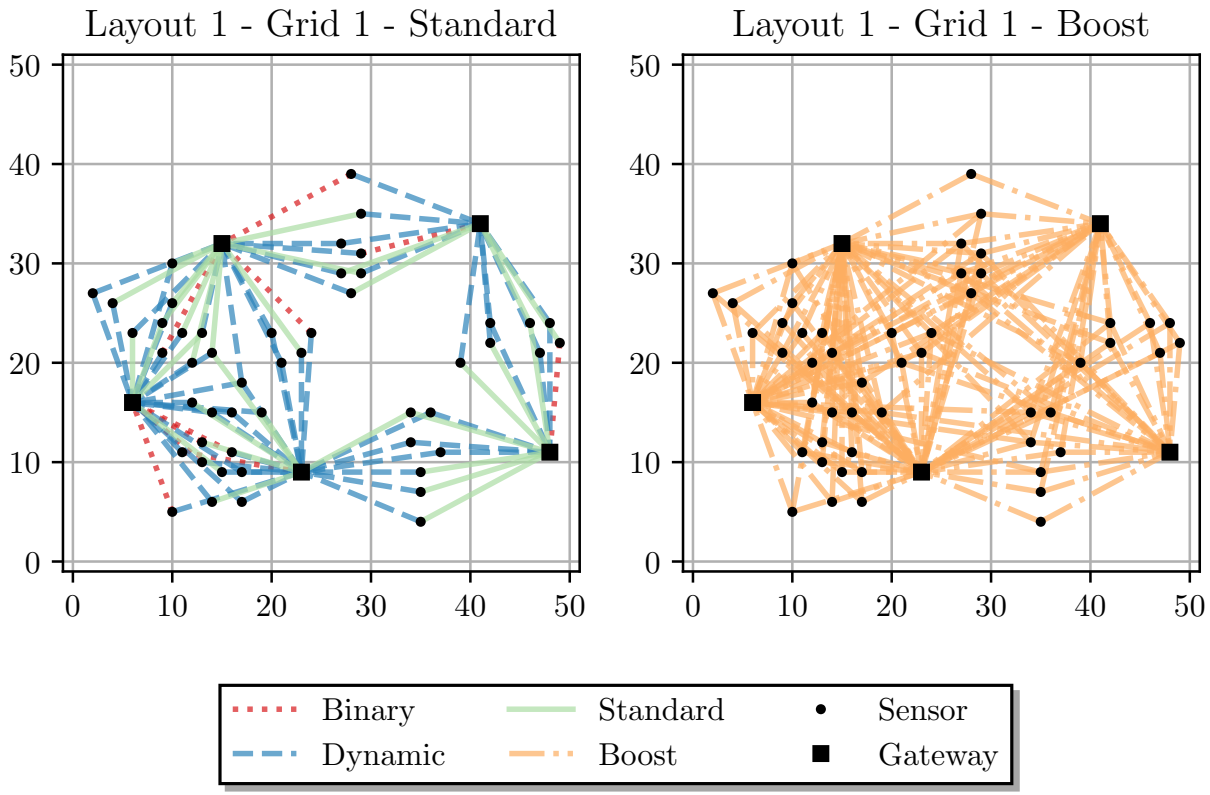



Figure B.1: Connections in network 1 with layout 1.

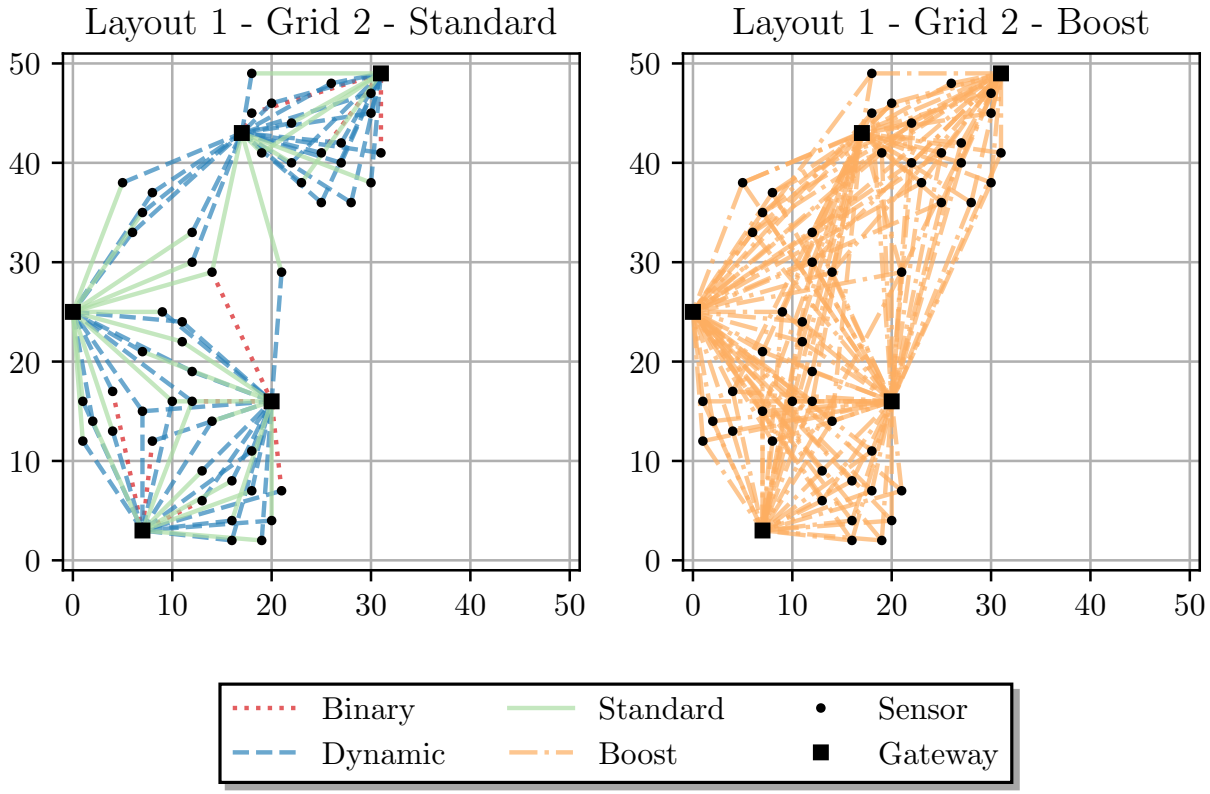



Figure B.2: Connections in network 2 with layout 1.

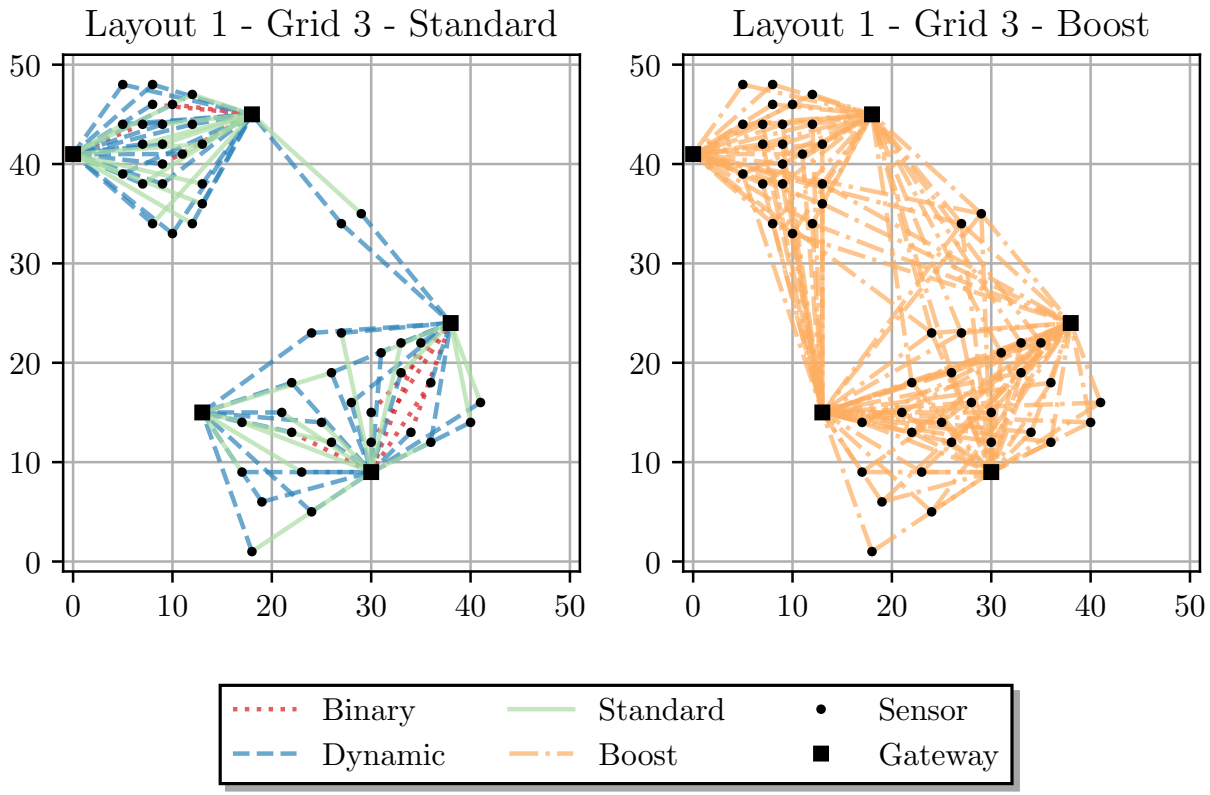



Figure B.3: Connections in network 3 with layout 1.

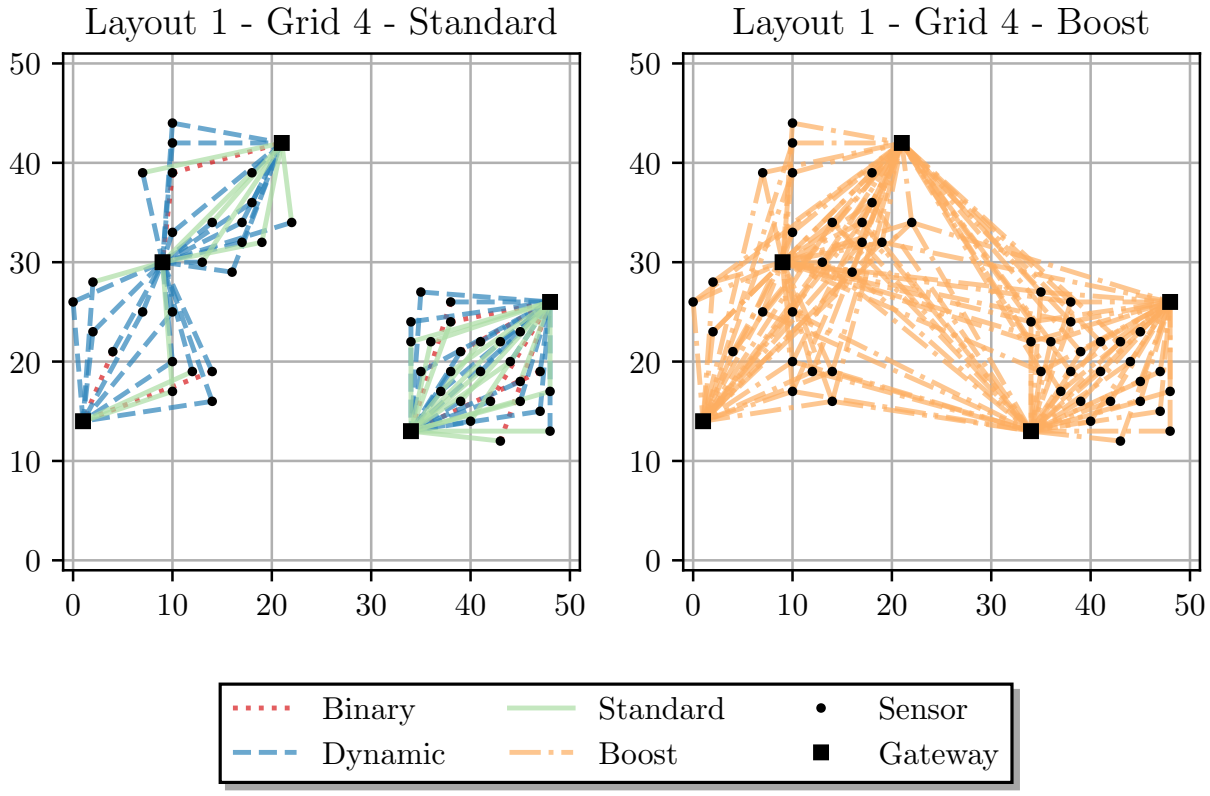



Figure B.4: Connections in network 4 with layout 1.

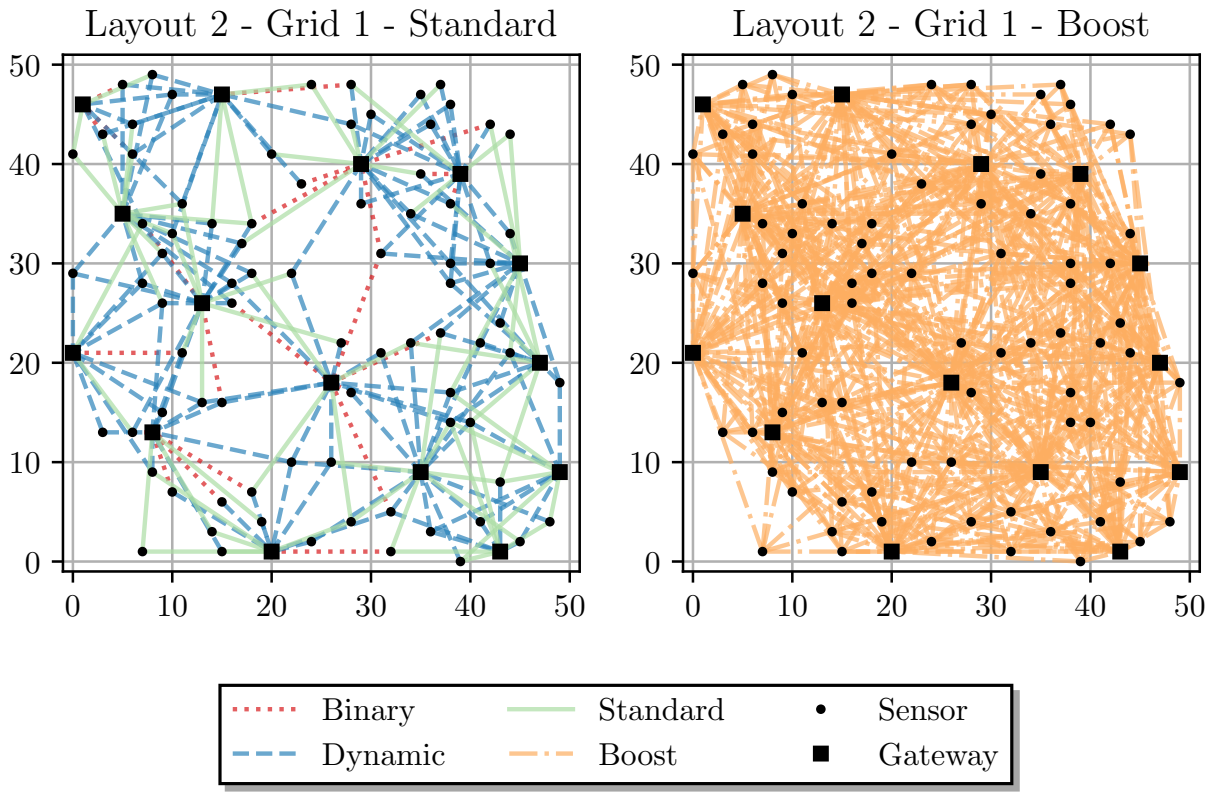



Figure B.5: Connections in network 1 with layout 2.

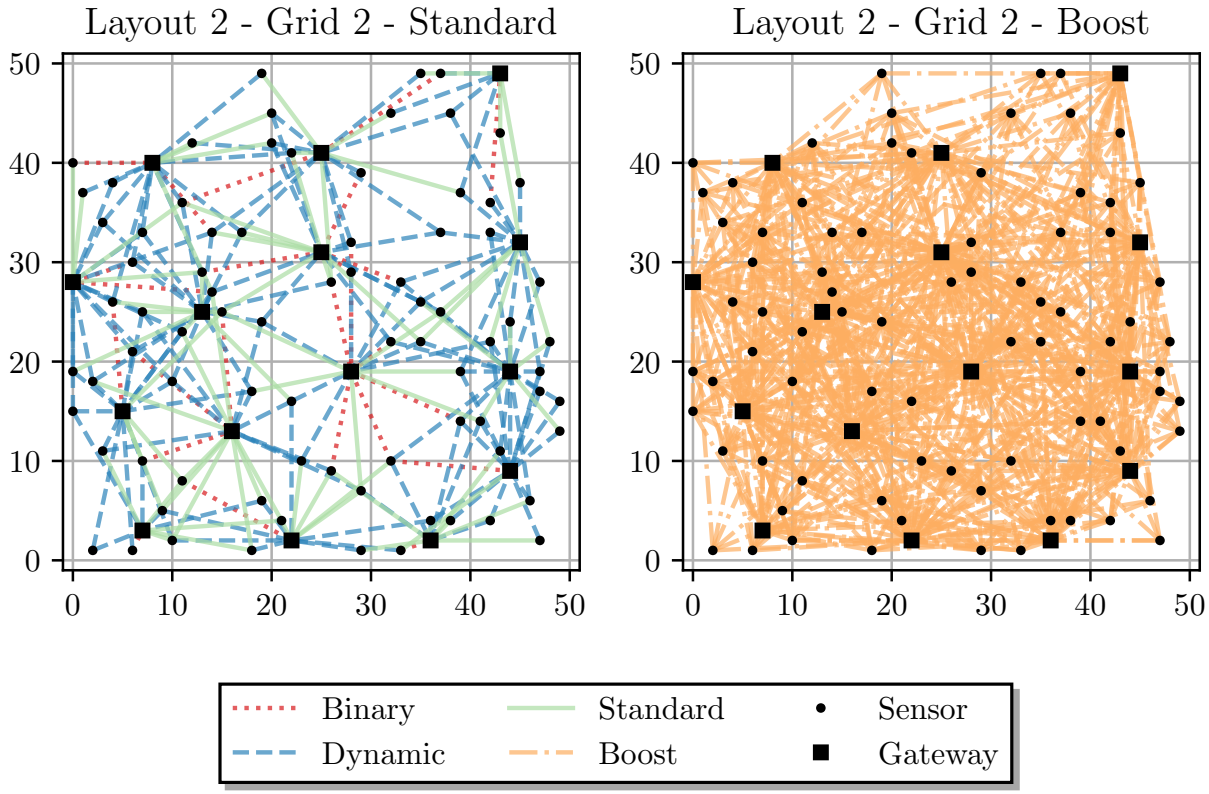



Figure B.6: Connections in network 2 with layout 2.

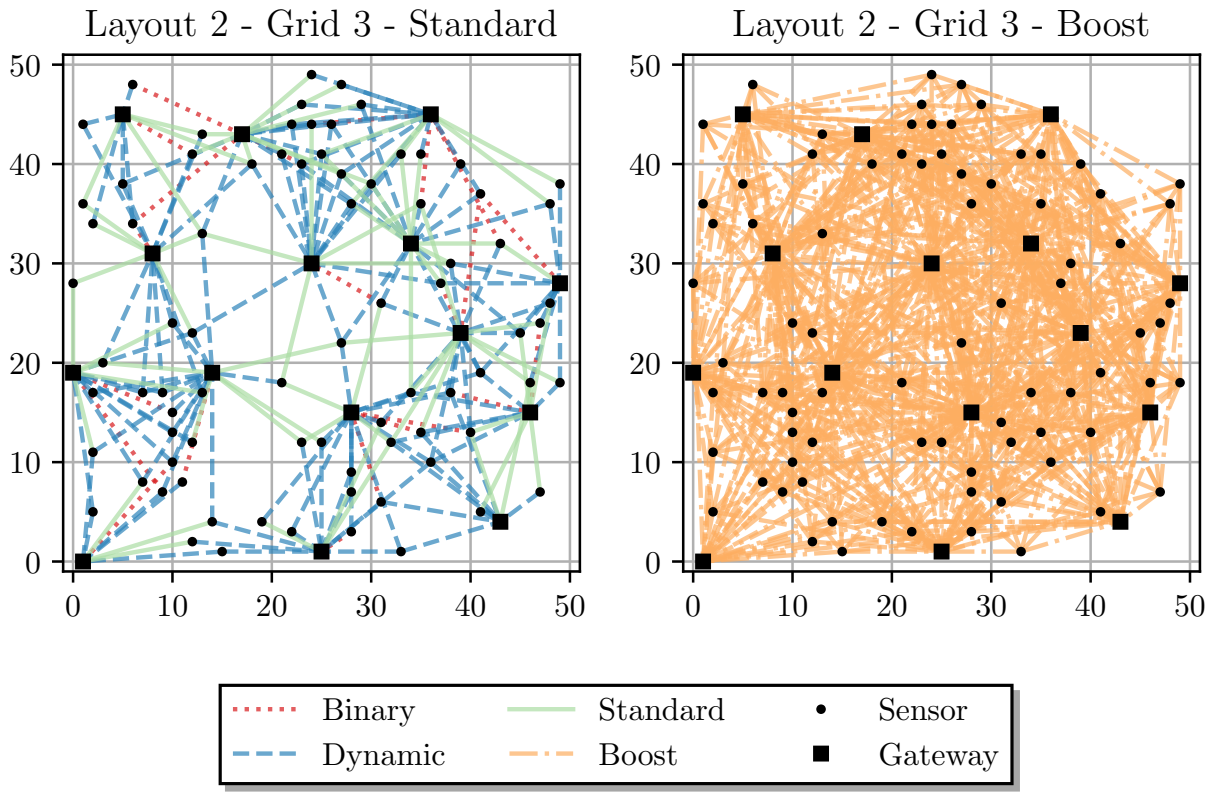



Figure B.7: Connections in network 3 with layout 2.

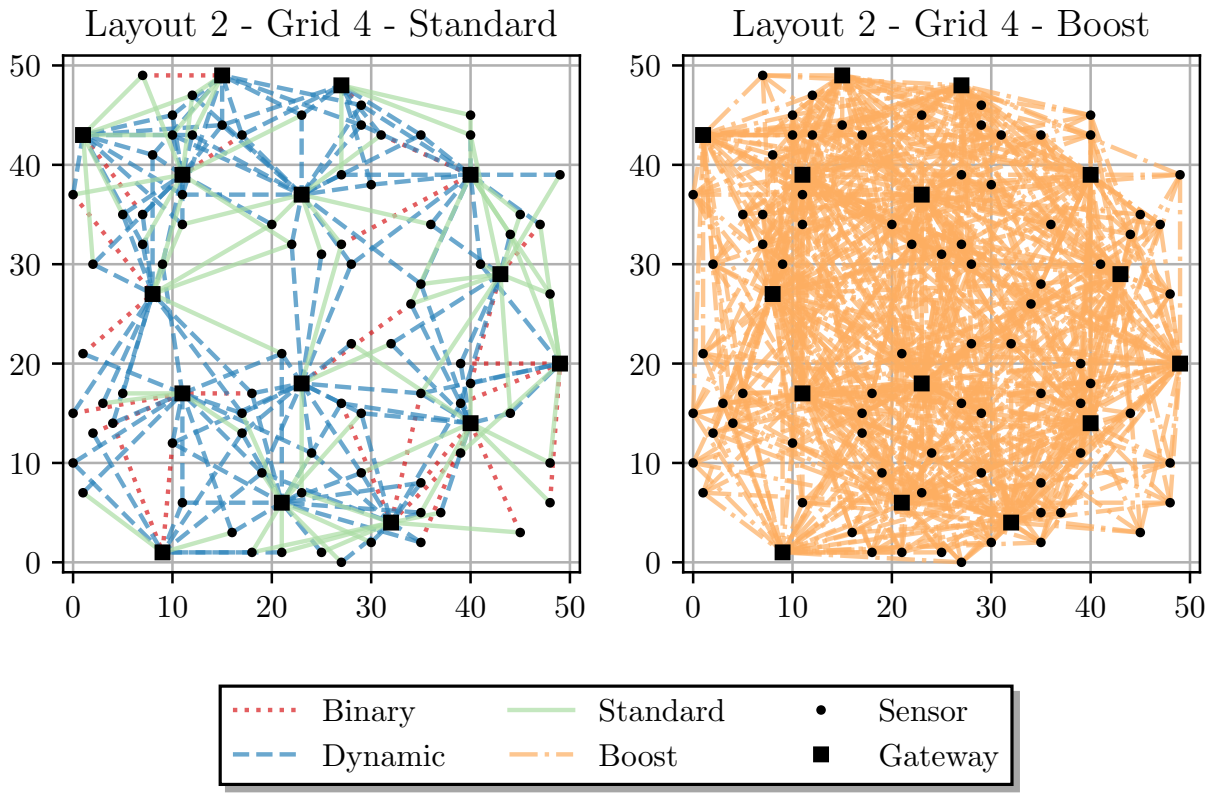



Figure B.8: Connections in network 4 with layout 2.

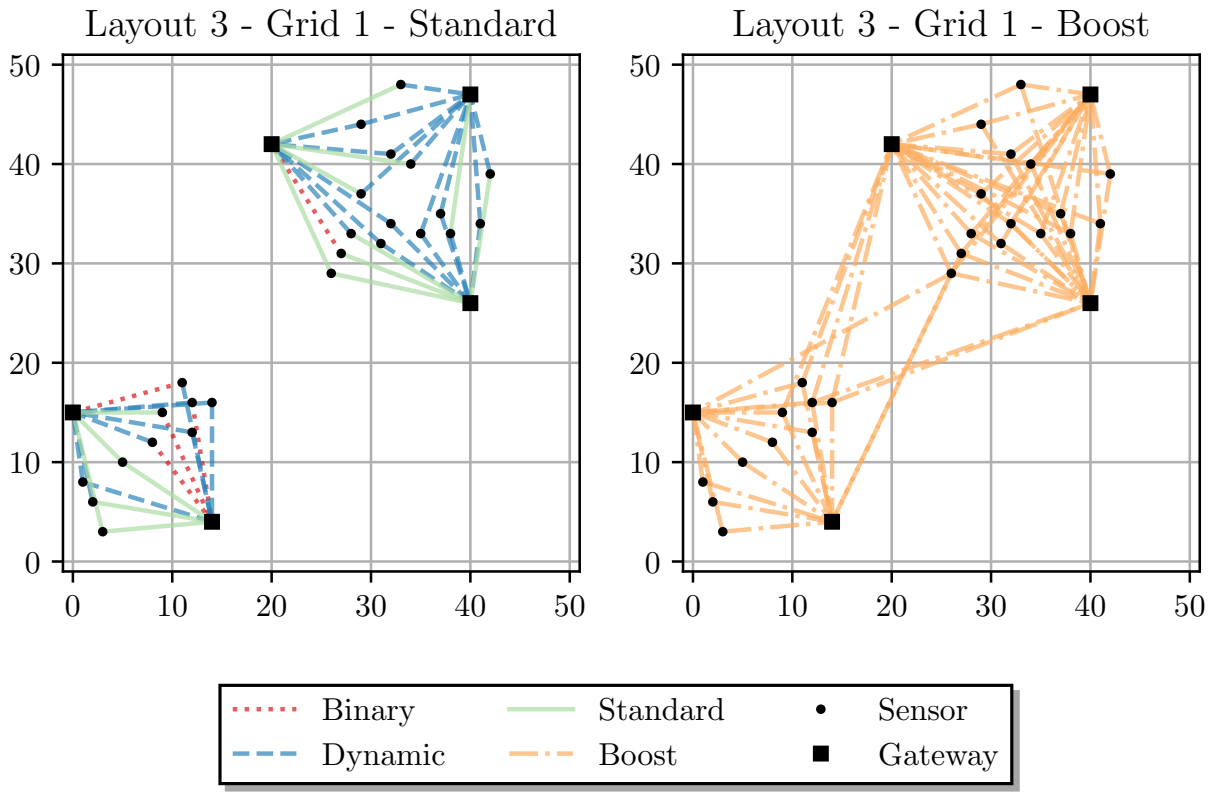



Figure B.9: Connections in network 1 with layout 3.

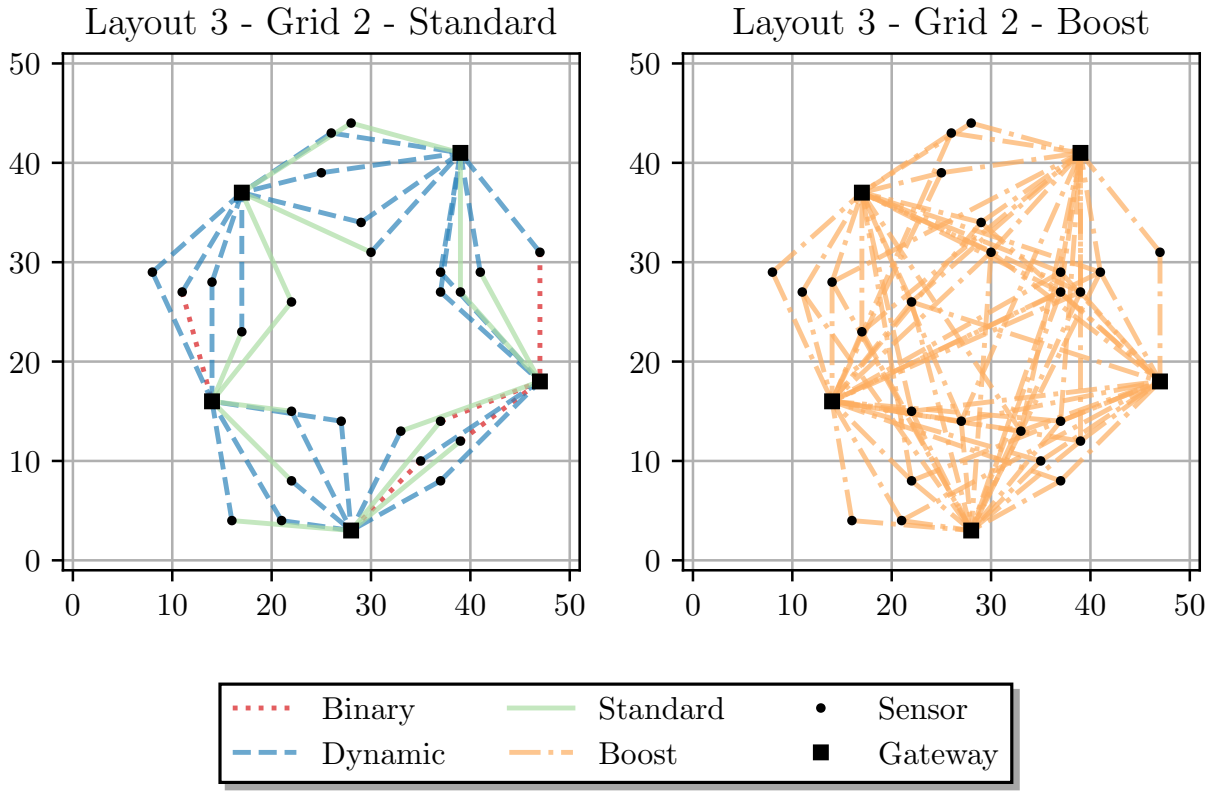



Figure B.10: Connections in network 2 with layout 3.

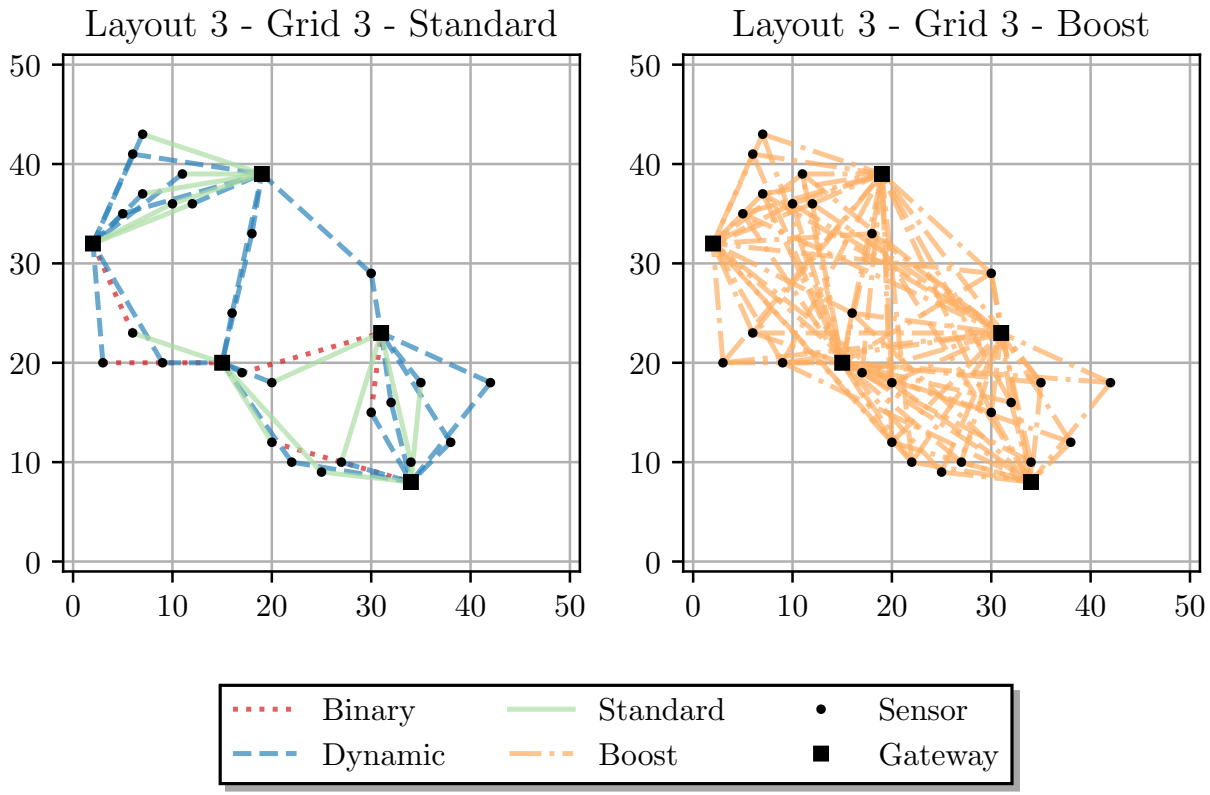



Figure B.11: Connections in network 3 with layout 3.

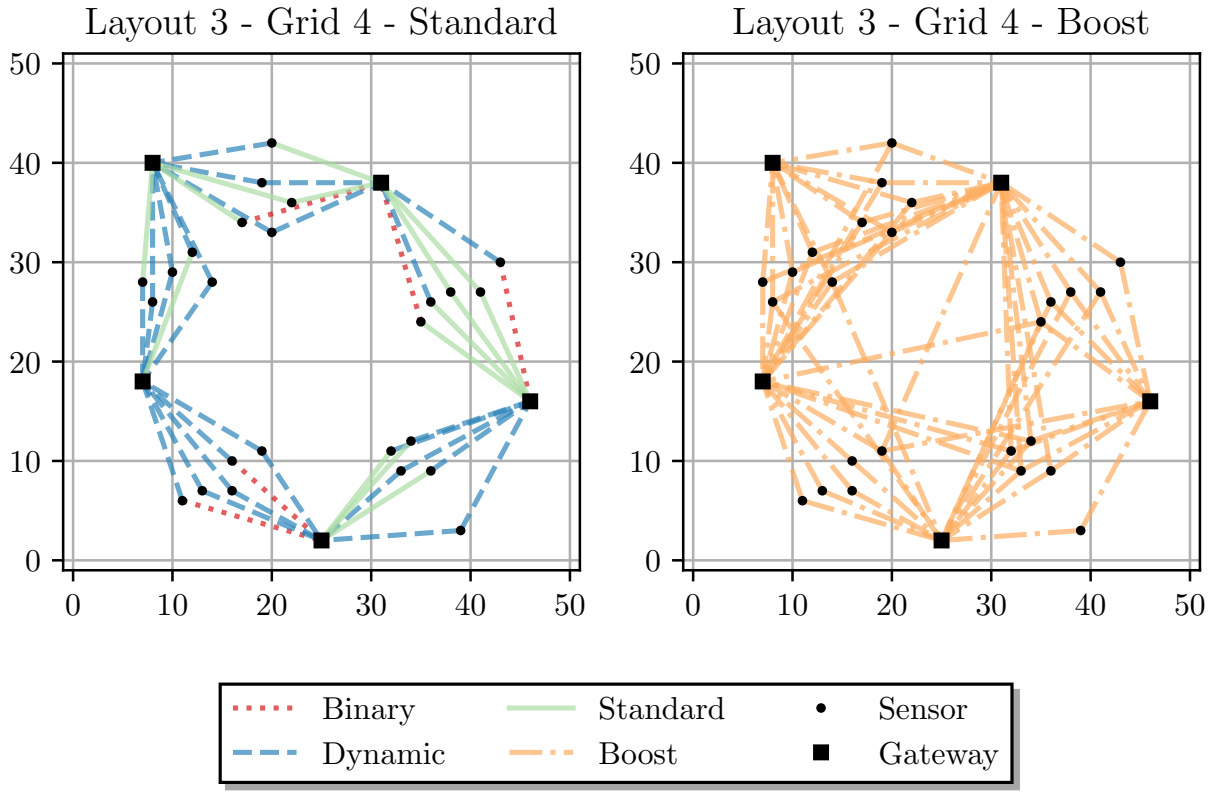



Figure B.12: Connections in network 4 with layout 3.

# Appendix C

# Simulation Results

Table C.1: Results from Test 1.

| Parameter | Value | HCR | LEEC | LRC | BASS |
|---|---|---|---|---|---|
| Retries | $\mu$ | 7662.72 | 8404.07 | 7671.42 | 10038.53 |
| | $\sigma$ | 2758.20 | 2909.88 | 2661.57 | 3898.20 |
| | $\frac{\sigma}{\mu}$ | 0.36 | 0.35 | 0.35 | 0.39 |
| Energy Consumption [mWs] | $\mu$ | 271.69 | 276.53 | 271.89 | 287.00 |
| | $\sigma$ | 95.60 | 97.31 | 96.96 | 102.20 |
| | $\frac{\sigma}{\mu}$ | 0.35 | 0.35 | 0.36 | 0.36 |

Table C.2: Results from Test 2 with Small Window

| Parameter | Value | HCR | LEEC | LRC | BASS |
|---|---|---|---|---|---|
| Retries | $\mu$ | 5744.75 | 7032.48 | 6971.53 | 13034.82 |
| | $\sigma$ | 2680.13 | 2616.87 | 2938.19 | 5393.46 |
| | $\frac{\sigma}{\mu}$ | 0.47 | 0.37 | 0.42 | 0.41 |
| Energy Consumption [mWs] | $\mu$ | 558.18 | 476.52 | 471.46 | 393.56 |
| | $\sigma$ | 173.22 | 137.48 | 202.44 | 141.1 |
| | $\frac{\sigma}{\mu}$ | 0.31 | 0.29 | 0.43 | 0.36 |
| Tx Boost | $\mu$ | 6670.37 | 4663.32 | 4557.98 | 1952.85 |
| | $\sigma$ | 2354.27 | 1528.93 | 2510.03 | 941.96 |
| | $\frac{\sigma}{\mu}$ | 0.35 | 0.33 | 0.55 | 0.48 |

Table C.3: Results from Test 2 with Large Window.

| Parameter | Value | HCR | LEEC | LRC | BASS |
|---|---|---|---|---|---|
| Retries | $\mu$ | 4094.92 | 7105.72 | 6009.58 | 12835.97 |
| | $\sigma$ | 2416.24 | 2813.57 | 2556.17 | 5278.49 |
| | $\frac{\sigma}{\mu}$ | 0.59 | 0.4 | 0.43 | 0.41 |
| Energy Consumption [mWs] | $\mu$ | 418.75 | 303.31 | 600.01 | 423.34 |
| | $\sigma$ | 134.81 | 95.52 | 223.0 | 146.07 |
| | $\frac{\sigma}{\mu}$ | 0.32 | 0.31 | 0.37 | 0.35 |
| Tx Boost | $\mu$ | 3793.55 | 785.45 | 7565.7 | 2645.47 |
| | $\sigma$ | 1466.83 | 426.05 | 3019.65 | 1296.57 |
| | $\frac{\sigma}{\mu}$ | 0.39 | 0.54 | 0.4 | 0.49 |

Table C.4: Results from Test 3 with Small Window.

| Parameter | Value | HCR | LEEC | LRC | BASS |
|---|---|---|---|---|---|
| Retries | $\mu$ | 10110.73 | 11799.85 | 11768.48 | 19935.22 |
| | $\sigma$ | 2780.43 | 2914.5 | 3155.48 | 6413.31 |
| | $\frac{\sigma}{\mu}$ | 0.27 | 0.25 | 0.27 | 0.32 |
| Energy Consumption [mWs] | $\mu$ | 467.55 | 483.49 | 392.35 | 312.22 |
| | $\sigma$ | 144.44 | 135.51 | 151.14 | 102.79 |
| | $\frac{\sigma}{\mu}$ | 0.31 | 0.28 | 0.39 | 0.33 |
| Tx Boost | $\mu$ | 7348.12 | 5124.58 | 5175.78 | 2406.33 |
| | $\sigma$ | 2530.02 | 1629.9 | 3171.21 | 1046.85 |
| | $\frac{\sigma}{\mu}$ | 0.34 | 0.32 | 0.61 | 0.44 |

Table C.5: Results from Test 3 with Large Window.

| Parameter | Value | HCR | LEEC | LRC | BASS |
|---|---|---|---|---|---|
| Retries | $\mu$ | 8548.37 | 12286.08 | 10847.02 | 19666.05 |
| | $\sigma$ | 2053.79 | 3005.49 | 3125.91 | 6151.54 |
| | $\frac{\sigma}{\mu}$ | 0.24 | 0.24 | 0.29 | 0.31 |
| Energy Consumption [mWs] | $\mu$ | 330.01 | 287.81 | 598.36 | 352.26 |
| | $\sigma$ | 102.84 | 74.33 | 261.93 | 120.53 |
| | $\frac{\sigma}{\mu}$ | 0.31 | 0.26 | 0.44 | 0.34 |
| Tx Boost | $\mu$ | 4330.6 | 940.72 | 10274.63 | 3644.83 |
| | $\sigma$ | 1731.9 | 425.77 | 5638.34 | 1784.66 |
| | $\frac{\sigma}{\mu}$ | 0.4 | 0.45 | 0.55 | 0.49 |

Table C.6: Results from Test 4 with Small Window.

| Parameter | Value | HCR | LEEC | LRC | BASS |
|---|---|---|---|---|---|
| Retries | $\mu$ | 10051.95 | 11832.88 | 11943.55 | 19782.93 |
| | $\sigma$ | 2650.84 | 2904.72 | 3237.9 | 6277.85 |
| | $\frac{\sigma}{\mu}$ | 0.26 | 0.25 | 0.27 | 0.32 |
| Energy Consumption [mWs] | $\mu$ | 1400.11 | 1251.51 | 1179.07 | 935.11 |
| | $\sigma$ | 435.03 | 353.06 | 466.82 | 297.74 |
| | $\frac{\sigma}{\mu}$ | 0.31 | 0.28 | 0.4 | 0.32 |
| Tx Boost | $\mu$ | 7372.52 | 5088.78 | 5168.98 | 2455.28 |
| | $\sigma$ | 2532.45 | 1608.09 | 3253.38 | 1015.81 |
| | $\frac{\sigma}{\mu}$ | 0.34 | 0.32 | 0.63 | 0.41 |

Table C.7: Results from Test 4 with Large Window.

| Parameter | Value | HCR | LEEC | LRC | BASS |
|---|---|---|---|---|---|
| Retries | $\mu$ | 8517.08 | 12238.97 | 11066.35 | 19443.02 |
| | $\sigma$ | 1992.39 | 3013.82 | 3264.18 | 6199.87 |
| | $\frac{\sigma}{\mu}$ | 0.23 | 0.25 | 0.29 | 0.32 |
| Energy Consumption [mWs] | $\mu$ | 976.47 | 753.02 | 1810.99 | 1033.55 |
| | $\sigma$ | 315.25 | 199.05 | 798.52 | 355.89 |
| | $\frac{\sigma}{\mu}$ | 0.32 | 0.26 | 0.44 | 0.34 |
| Tx Boost | $\mu$ | 4265.82 | 963.33 | 10399.55 | 3472.47 |
| | $\sigma$ | 1832.54 | 448.05 | 5715.76 | 1720.54 |
| | $\frac{\sigma}{\mu}$ | 0.43 | 0.47 | 0.55 | 0.5 |

Table C.8: Results from Test 5 with Large Window.

| Parameter | Value | HCR | LEEC | LRC | BASS |
|---|---|---|---|---|---|
| Retries | $\mu$ | 854.35 | 1138.77 | 729.43 | 6619.68 |
| | $\sigma$ | 654.43 | 901.85 | 677.68 | 1939.96 |
| | $\frac{\sigma}{\mu}$ | 0.77 | 0.79 | 0.93 | 0.29 |
| Energy Consumption [mWs] | $\mu$ | 163.41 | 189.42 | 185.49 | 167.06 |
| | $\sigma$ | 59.16 | 63.32 | 73.64 | 48.35 |
| | $\frac{\sigma}{\mu}$ | 0.36 | 0.33 | 0.4 | 0.29 |
| Tx Boost | $\mu$ | 310.87 | 21.62 | 1041.02 | 735.25 |
| | $\sigma$ | 450.27 | 52.59 | 956.46 | 567.9 |
| | $\frac{\sigma}{\mu}$ | 1.45 | 2.43 | 0.92 | 0.77 |

Table C.9: Results from Test 6 with Large Window.

| Parameter | Value | HCR | LEEC | LRC | BASS |
|---|---|---|---|---|---|
| Retries | $\mu$ | 43873.45 | 51747.48 | 74449.6 | 59443.12 |
| | $\sigma$ | 12298.04 | 15811.5 | 35234.87 | 23271.38 |
| | $\frac{\sigma}{\mu}$ | 0.28 | 0.31 | 0.47 | 0.39 |
| Energy Consumption [mWs] | $\mu$ | 1358.45 | 1502.4 | 3185.21 | 1600.76 |
| | $\sigma$ | 483.83 | 484.68 | 2023.86 | 734.25 |
| | $\frac{\sigma}{\mu}$ | 0.36 | 0.32 | 0.64 | 0.46 |
| Tx Boost | $\mu$ | 24562.33 | 17332.13 | 59768.8 | 25499.22 |
| | $\sigma$ | 10647.93 | 5994.11 | 40598.03 | 12696.02 |
| | $\frac{\sigma}{\mu}$ | 0.43 | 0.35 | 0.68 | 0.5 |

# Appendix D

# Using the Simulator

To use the simulator, one must provide a configuration file in JSON format (see appendix B for an example). There are numerous settings that can be adjusted in this configuration file, and it is used by giving it as an input argument to the executable. Several optional arguments can be given to the simulator, for instance -b for running a benchmark that simulates all given grids for all the gateway selection methods. For all optional arguments and help, use -h as an optional argment.

```
$ ./simulate configuration.json
```

The settings can be split into five major parts: sensors, gateways, cloud, connections and simulation. The parts and their various options are discussed in the following list. Note that some settings have default values that are used if they are missing in the configuration file, while others are explicitly needed.

## D.1 Sensors

- **coordinates:** a list of (x, y) pairs where the sensors should be located.
- **multi_coordinates:** a list of lists of (x, y) pairs where the sensors should be located. This is used for simulating multiple grids in one run during benchmarking.
- **transmission_power:** a list of available transmission powers in dBm. Defaults to 0, 3, 6 and 10 dBm. 10 dBm if TPC is not used.
- **tpc:** true if the sensors should use TPC. If missing, TPC is not used.
- **send_tpc:** makes the sensor send its current transmission power level. Only needed for lowest energy consumption, but it can be forced for other methods with this option.

- **sensitivity:** the lowest signal power a sensor can receive. Defaults to -60 dBm.
- **max_attempts:** the maximum number of attempts a sensor is allowed to make for transmitting one single message. Defaults to 4.
- **payload_size:** the size of the payload of the messages in bytes. Defaults to 2 bytes.
- **address_size:** the size of the addresses in bytes. Defaults to 2 bytes.
- **ciphertext_size:** the size of the ciphertext of each message in bytes. Defaults to 2 bytes. Currently not used for anything.
- **start_interval:** the amount of seconds to wait between starting each sensor. Defaults to the sending interval divided by the number of sensors for even spacing.
- **send_interval:** how many minutes to wait between each normal transmission. Defaults to 15 minutes.
- **data_rate:** the transmission rate in bits per second. Defaults to 500 kb/s.
- **waiting_time:** the number of minutes a sensor waits for a reply to a message it sent. Defaults to 1 minute.

## D.2 Gateways

- **coordinates:** a list of (x, y) pairs where the gateways should be located.
- **multi_coordinates:** a list of lists of (x, y) pairs where the gateways should be located. This is used for simulating multiple grids in one run during benchmarking.
- **sensitivity:** the lowest signal power a gateway can receive. Defaults to -60 dBm.
- **transmission_power:** the transmission power in dBm. Defaults to 10 dBm.

## D.3 Cloud

- **selection_method:** the gateway selection method from section 5.2. It defaults to strongest_signal, and it has the following options: *strongest_signal, lowest_retry_count, highest_reliability* and *lowest_energy_consumption*.
- **minimum_signal_strength_difference:** the threshold for changing gateways in algorithm 5.2. Defaults to 3.0 dB.

- **minimum_reliability_improvement:** the threshold for changing gateways in algorithm 5.4. Defaults to 0.05.
- **minimum_energy_improvement:** the threshold for changing gateways in algorithm 5.5. Defaults to 1.1.
- **weight_downlink:** the weight on the downlink in algorithm 5.4. Should be between 0 and 1. Defaults to 0.5.
- **weight_uplink:** the weight on the uplink in algorithm 5.4. Should be between 0 and 1. Defaults to 0.5.

## D.4 Connections

- **distributions:** the distribution of the different connection types. Should add up to 1.0. Defaults to standard (free space connection). Has the following types that can be set.
  - standard
  - dynamic
  - binary
- **symmetric_connection:** true if signal loss on uplinks and downlinks should be symmetric. Defaults to false.
- **block_prob:** the probability that binary connections should block. Defaults to 20%.
- **dynamic_loss:** the maximum signal loss increase and decrease that occurs on dynamic connections in percents. Defaults to 10%.
- **normal_range:** the longest distance a sensor can communicate with a gateway in standard mode. Defaults to calculated threshold if missing.
- **boost_range:** the longest distance a sensor can communicate with a gateway in boost mode. Defaults to two times **normal_range**.
- **adjust_fspl:** if true, free space path loss is adjusted such that at the longest distances with the highest transmission power, the signal strength received equals the sensitivity. Defaults to false.
- **boost_cost:** how much more expensive transmission in boost mode should be compared to normal mode. Defaults to 8.

## D.5 Simulation

- **simulation_speed:** number of seconds to simulate per real time second. Defaults to 60 simulated seconds per second.
- **frequency:** the frequency that the sensors and gateways communicate with in Hz. Defaults to 434.92 MHz.
- **seed:** the seed for the random number generator. Defaults to 0.
- **window_size:** the window size to use in moving average and moving windows. Defaults to 10.
- **duration:** the duration of the simulation. If missing, it runs forever. Is set with the following parameters.
  - Days
  - Hours
  - Minutes
  - Seconds
- **title:** the title that will be used in the plots. Defaults to the filename of the configuration file.
- **boost:** true to enable boost mode. Defaults to true.
- **bit_cost:** can be used to increase the cost of transmitting one bit. Defaults to 1.

# Appendix E

# Simulator Implementation

Since the implementation of the simulator turned out to be a major part of the job that needed to be done in order to solve the problem in this thesis, a deeper look into some selected parts of the implementation are discussed in this appendix. The aim is to give a general overview, so that others might be able to take over the simulator and further develop it in the future.

## E.1 Connections

The connections that were discussed in chapter 4 have a couple of design choices about them that are worth mentioning. The most important one is that when a simulation is started, the connections are not explicitly set. As the simulation grows large, the amount of connections between gateways and sensors grows at a very high rate, so it is not feasible to manually set the connections and their type. Instead the simulator is implemented such that one sets the distribution of the various connection types. For instance one can say that 10% of the connections are binary, 30% are dynamic and the rest are simply free space connections. The simulator will then randomly assign connections based on this distribution and the given maximum transmission distances.

Furthermore, the connections are not implemented as separate processes. The way they are used is that each sensor has one connection that corresponds to each gateway channel that it has. When it wants to send a message on a channel, it uses this connection to calculate the signal strength that should be in the receiving end (i.e. the gateway). If it gets blocked, it simply does not send the message. This means that the sensor also calculates the signal strength for messages that it receives from gateways.

Ideally this should be separated as a wireless transmission module that calculates signal strengths for all transmissions in the simulation. However, due to the available time scope, it was chosen to implement the connections in this rather simple way, so that more focus could be given to the problem of roaming. Nevertheless, this does successfully simulate wireless roaming, so it is a reasonable solution.

As the FSPL is calculated based on frequency and distance, and successful reception of transmissions depends on the receiver's sensitivity, setting all these parameters such that the maximum transmission range is as wanted is not straight forward. To simplify this, the option of adjusting the FSPL to a set maximum range was implemented in the simulator. It works by calculating the maximum distance that can be reached from the sensitivities and transmission powers, and then this is used to scale the distance in the connections such that the maximum range given in the configuration file results in the maximum acceptable signal loss.

## E.2 Moving Average and Moving Window

Moving average and moving window is used in several of the gateway selection methods. These were implemented from scratch. The moving average is merely a slice of a given size. For each received message, a value is inserted, and the average value of all the values in the slice is used in the methods. When the slice is full, it starts replacing old values with new values from the beginning, and this process is continuously repeated.

The moving window was implemented for when a sum was needed rather than an average value. One example is for monitoring uplink connections. In an actual system, it makes no sense to monitor how many messages have been received from a sensor during the last year, as the external environments affecting each uplink connection is likely to change during that timespan. To cope with this, the number of attempts received for the last $x$ unique message IDs is monitored, and the sum of these attempts makes up the total amount of attempts received. As with the moving average, when the slice is full, it starts from the beginning, and overwrites old values as it receives messages with new IDs.

# Glossary

| | |
|---|---|
| **channel** | Pipes that connect concurrent goroutines. |
| **ciphertext** | The result after encrypting a plaintext. |
| **dBm** | Power ratio in dB referenced to 1 mW: $10 \log_{10} \left( \frac{\text{power in mW}}{1 \text{ mW}} \right)$. |
| **deterministic encryption** | An encryption scheme that always produces the same ciphertext given the same key and plaintext. |
| **directivity** | The degree of how much of the radiation that is focused in one direction. |
| **downlink** | The connection from the gateway to the sensor. |
| **goroutine** | A lightweight thread of execution in Golang. |
| **initialization vector** | A number that is mixed with the plaintext in the first step of an encryption scheme to ensure semantic security. |
| **isotropic antenna** | A hypothetical antenna that radiates with the same intensity in all directions. |
| **plaintext** | Name used for unencrypted information in cryptography. |
| **relative standard deviation** | The ratio of the standard deviation ($\sigma$) to the mean ($\mu$). |
| **slice** | The name of arrays with dynamic size in Golang. |
| **transceiver** | A device that has both a transmitter and a receiver. |
| **uplink** | The connection from the sensor to the gateway. |

# Acronyms

| | |
|---|---|
| **BASS** | Best Average Signal Strength. |
| **CH** | Cluster Head. |
| **CPU** | Central Processing Unit. |
| **FSPL** | Free-Space Path Loss. |
| **HCR** | Highest Connection Reliability. |
| **IoT** | Internet of Things. |
| **ISM** | Industrial, Scientific and Medical. |
| **JSON** | JavaScript Object Notation. |
| **LEACH** | Low-Energy Adaptive Clustering Hierarchy. |
| **LEEC** | Lowest Expected Energy Consumption. |
| **LRC** | Lowest Retry Count. |
| **LTSA** | Labelled Transition System Analyser. |
| **MAC** | Medium Access Control. |
| **OS** | Operating System. |
| **PLR** | Packet Loss Rate. |
| **PRR** | Packet Reception Rate. |
| **RSSI** | Received Signal Strength Indicator. |

| | |
|---|---|
| **SNR** | Signal to Noise Ratio. |
| **TPC** | Transmission Power Control. |
| **WLAN** | Wireless Local Area Network. |
| **WSN** | Wireless Sensor Network. |

# References


[1] J. D. Hunter. 'Matplotlib: A 2D graphics environment'. In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[2] V. Raghunathan et al. 'Energy-aware wireless microsensor networks'. In: *IEEE Signal Processing Magazine* 19.2 (Mar. 2002), pp. 40–50. DOI: 10.1109/79.985679.

[3] Giuseppe Anastasi et al. 'Energy conservation in wireless sensor networks: A survey'. In: *Ad Hoc Networks* 7.3 (May 2009), pp. 537–568. DOI: 10.1016/j.adhoc.2008.06.003.

[4] Duc Chinh Hoang et al. 'Real-Time Implementation of a Harmony Search Algorithm-Based Clustering Protocol for Energy-Efficient Wireless Sensor Networks'. In: *IEEE Transactions on Industrial Informatics* 10.1 (Feb. 2014), pp. 774–783. DOI: 10.1109/tii.2013.2273739.

[5] Noritaka Shigei et al. 'Centralized and Distributed Clustering Methods for Energy Efficient Wireless Sensor Network'. In: *Proceedings of the International MultiConference of Engineers and Computer Scientists 2009*. Vol. 1. Newswood Limited, Mar. 2009, pp. 423–427. ISBN: 978-988-17012-2-0.

[6] Changjiang Jiang et al. 'Low-Energy Consumption Uneven Clustering Routing Protocol for Wireless Sensor Networks'. In: *2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*. IEEE, Aug. 2016. DOI: 10.1109/ihmsc.2016.214.

[7] Jin-Shyan Lee and Tsung-Yi Kao. 'An Improved Three-Layer Low-Energy Adaptive Clustering Hierarchy for Wireless Sensor Networks'. In: *IEEE Internet of Things Journal* 3.6 (Dec. 2016), pp. 951–958. DOI: 10.1109/jiot.2016.2530682.

[8] Ankit Thakkar. 'An improved advanced low energy adaptive clustering hierarchy for a dense wireless sensor network'. In: *2016 International Conference on Communication and Electronics Systems (ICCES)*. IEEE, Oct. 2016. DOI: 10.1109/cesys.2016.7889806.

[9] Tianqi Yu et al. 'Energy-Efficient Scheduling Mechanism for Indoor Wireless Sensor Networks'. In: *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*. IEEE, May 2015. DOI: 10.1109/vtcspring.2015.7145818.

[10] W.R. Heinzelman, A. Chandrakasan and H. Balakrishnan. 'Energy-efficient communication protocol for wireless microsensor networks'. In: *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*. IEEE Comput. Soc. DOI: 10.1109/hicss.2000.926982.

[11] Padmalaya Nayak and Anurag Devulapalli. 'A Fuzzy Logic-Based Clustering Algorithm for WSN to Extend the Network Lifetime'. In: *IEEE Sensors Journal* 16.1 (Jan. 2016), pp. 137–144. DOI: 10.1109/jsen.2015.2472970.

[12] H.T. Friis. 'A Note on a Simple Transmission Formula'. In: *Proceedings of the IRE* 34.5 (May 1946), pp. 254–256. DOI: 10.1109/jrproc.1946.234568.

[13] Chong Liu, Kui Wu and Jian Pei. 'An Energy-Efficient Data Collection Framework for Wireless Sensor Networks by Exploiting Spatiotemporal Correlation'. In: *IEEE Transactions on Parallel and Distributed Systems* 18.7 (July 2007), pp. 1010–1023. DOI: 10.1109/tpds.2007.1046.

[14] Saleh Bouarafa, Rachid Saadane and Moulay Rahmani. 'Inspired from Ants Colony: Smart Routing Algorithm of Wireless Sensor Network'. In: *Information* 9.1 (Jan. 2018), p. 23. DOI: 10.3390/info9010023.

[15] Juan Luo et al. 'Opportunistic Routing Algorithm for Relay Node Selection in Wireless Sensor Networks'. In: *IEEE Transactions on Industrial Informatics* 11.1 (Feb. 2015), pp. 112–121. DOI: 10.1109/tii.2014.2374071.

[16] Abdelmalek Bengheni, Fedoua Didi and Ilyas Bambrik. 'Energy-saving comparison of asynchronous MAC protocols for wireless sensor networks'. In: *2017 International Conference on Mathematics and Information Technology (ICMIT)*. IEEE, Dec. 2017. DOI: 10.1109/mathit.2017.8259727.

[17] Wei Ye, J. Heidemann and D. Estrin. 'An energy-efficient MAC protocol for wireless sensor networks'. In: *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*. IEEE. DOI: 10.1109/infcom.2002.1019408.

[18] Kun Tu et al. 'Mac Address Assignment In Wireless Sensor Networks: A Mixed Strategy Game Approach'. In: *2006 International Conference on Systems and Networks Communications (ICSNC'06)*. IEEE, 2006. DOI: 10.1109/icsnc.2006.52.

[19] V.C. Gungor and G.P. Hancke. 'Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches'. In: *IEEE Transactions on Industrial Electronics* 56.10 (Oct. 2009), pp. 4258–4265. DOI: 10.1109/tie.2009.2015754.

[20] Sounak Paul, Sukumar Nandi and Indrajeet Singh. 'A dynamic balanced-energy sleep scheduling scheme in heterogeneous wireless sensor network'. In: *2008 16th IEEE International Conference on Networks*. IEEE, 2008. DOI: 10.1109/icon.2008.4772568.

[21] Ahmad Muzaffar bin Baharudin et al. 'Low-energy algorithm for self-controlled Wireless Sensor Nodes'. In: *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*. IEEE, Oct. 2016. DOI: 10.1109/wincom.2016.7777188.

[22] C. Schurgers, G. Kulkarni and M.B. Srivastava. 'Distributed on-demand address assignment in wireless sensor networks'. In: *IEEE Transactions on Parallel and Distributed Systems* 13.10 (Oct. 2002), pp. 1056–1065. DOI: 10.1109/tpds.2002.1041881.

[23] Tianyu Du et al. 'Adaptive Transmit Power Adjustment Technique for ZigBee Network under Wi-Fi Interference'. In: *Ad Hoc Networks*. Ed. by Natalie Mitton et al. Cham: Springer International Publishing, 2014, pp. 146–157. ISBN: 978-3-319-13329-4.

[24] Huseyin Cotuk et al. 'The Impact of Transmission Power Control Strategies on Lifetime of Wireless Sensor Networks'. In: *IEEE Transactions on Computers* 63.11 (Nov. 2014), pp. 2866–2879. DOI: 10.1109/tc.2013.151.

[25] Shan Lin et al. 'ATPC: Adaptive Transmission Power Control for Wireless Sensor Networks'. In: *SenSys'06: Proceedings of the Fourth International Conference on Embedded Networked Sensor Systems*. Vol. 1. Jan. 2006, pp. 223–236. DOI: 10.1145/1182807.1182830.

[26] Qian Tan et al. 'Energy harvesting aware topology control with power adaptation in wireless sensor networks'. In: *2014 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, Apr. 2014. DOI: 10.1109/wcnc.2014.6952859.

[27] S. Panichpapiboon, G. Ferrari and O.K. Tonguz. 'Optimal Transmit Power in Wireless Sensor Networks'. In: *IEEE Transactions on Mobile Computing* 5.10 (Oct. 2006), pp. 1432–1447. DOI: 10.1109/tmc.2006.155.

[28] Dae-Young Kim et al. 'Transmission Power Control with the Guaranteed Communication Reliability in WSN'. In: *International Journal of Distributed Sensor Networks* 2015 (2015), pp. 1–12. DOI: 10.1155/2015/632590.

[29] L.M. Kamarudin et al. 'Modeling and Simulation of WSNs for Agriculture Applications Using Dynamic Transmit Power Control Algorithm'. In: *2012 Third International Conference on Intelligent Systems Modelling and Simulation*. IEEE, Feb. 2012. DOI: 10.1109/isms.2012.109.

[30] Luiz H.A. Correia et al. 'Transmission power control techniques for wireless sensor networks'. In: *Computer Networks* 51.17 (Dec. 2007), pp. 4765–4779. DOI: 10.1016/j.comnet.2007.07.008.

[31] Miao Zhao, Dawei Gong and Yuanyuan Yang. 'Network Cost Minimization for Mobile Data Gathering in Wireless Sensor Networks'. In: *IEEE Transactions on Communications* 63.11 (Nov. 2015), pp. 4418–4432. DOI: 10.1109/tcomm.2015.2480088.

[32] Miao Zhao, Dawei Gong and Yuanyuan Yang. 'A cost minimization algorithm for mobile data gathering in wireless sensor networks'. In: *The 7th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (IEEE MASS 2010)*. IEEE, Nov. 2010. DOI: 10.1109/mass.2010.5664020.

[33] Zhao Cheng, M. Perillo and W.B. Heinzelman. 'General Network Lifetime and Cost Models for Evaluating Sensor Network Deployment Strategies'. In: *IEEE Transactions on Mobile Computing* 7.4 (Apr. 2008), pp. 484–497. DOI: 10.1109/tmc.2007.70784.

[34] Tifenn Rault, Abdelmadjid Bouabdallah and Yacine Challal. 'Energy efficiency in wireless sensor networks: A top-down survey'. In: *Computer Networks* 67 (July 2014), pp. 104–122. DOI: 10.1016/j.comnet.2014.03.027.

[35] Sidra Aslam, Farrah Farooq and Shahzad Sarwar. 'Power consumption in wireless sensor networks'. In: *Proceedings of the 6th International Conference on Frontiers of Information Technology - FIT '09*. ACM Press, 2009. DOI: 10.1145/1838002.1838017.

[36] D. McLoughlin et al. 'Review and evaluation of WSN simulation tools in a cloud based environment'. In: *2016 10th International Conference on Sensing Technology (ICST)*. Nov. 2016, pp. 1–6. DOI: 10.1109/ICSensT.2016.7796270.

[37] E. Egea-López et al. 'Simulation Tools for Wireless Sensor Networks'. In: *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS05)*. 2005. ISBN: 1-56555-300-4.

[38] Bartosz Musznicki and Piotr Zwierzykowski. 'Survey of Simulators for Wireless Sensor Networks'. In: 5 (Sept. 2012), pp. 23–50. ISSN: 2005-4262.

[39] Lei Shu et al. 'NetTopo: Beyond Simulator and Visualizer for Wireless Sensor Networks'. In: *2008 Second International Conference on Future Generation Communication and Networking*. IEEE, Dec. 2008. DOI: 10.1109/fgcn.2008.18.

[40] *Sinalgo*. URL: https://sourceforge.net/projects/sinalgo/ (visited on 17/04/2018).

[41] *Contiki: The Open Source OS for the Internet of Things*. URL: http://www.contiki-os.org/ (visited on 17/04/2018).

[42] *OMNeT++ Discrete Event Simulator*. URL: https://omnetpp.org/ (visited on 17/04/2018).

[43] *The Network Simulator - ns-2*. URL: https://www.isi.edu/nsnam/ns/ (visited on 17/04/2018).

[44] Xiaodong Xian, Weiren Shi and He Huang. 'Comparison of OMNET++ and Other Simulator for WSN Simulation'. In: *2008 3rd IEEE Conference on Industrial Electronics and Applications*. IEEE, June 2008, pp. 1439–1443. DOI: 10.1109/ICIEA.2008.4582757.

[45] *ns-3*. URL: https://www.nsnam.org/ (visited on 17/04/2018).

[46] Anne-Sophie Tonneau, Nathalie Mitton and Julien Vandaele. 'How to choose an experimentation platform for wireless sensor networks? A survey on static and mobile wireless sensor network experimentation facilities'. In: *Ad Hoc Networks* 30 (July 2015), pp. 115–127. DOI: 10.1016/j.adhoc.2015.03.002.

[47] *The Go Programming Language*. URL: https://golang.org/ (visited on 17/04/2018).

[48] 'Indoor Radio Propagation'. In: *Radio Propagation and Adaptive Antennas for Wireless Communication Networks*. John Wiley & Sons, Inc., Apr. 2014, pp. 179–215. DOI: 10.1002/9781118816707.ch6. URL: https://doi.org/10.1002%2F9781118816707.ch6.

[49] Joseph A. Shaw. 'Radiometry and the Friis transmission equation'. In: *American Journal of Physics* 81.1 (Jan. 2013), pp. 33–37. DOI: 10.1119/1.4755780.

[50] *IEEE Standard for Definitions of Terms for Antennas*. DOI: 10.1109/ieeestd.2014.6758443.

[51] *Labelled Transition System Analyser*. URL: https://www.doc.ic.ac.uk/ltsa/ (visited on 24/04/2018).

[52] Jason Hill et al. 'System architecture directions for networked sensors'. In: *ACM SIGARCH Computer Architecture News* 28.5 (Dec. 2000), pp. 93–104. DOI: 10.1145/378995.379006.

[53] Mihir Bellare et al. 'Deterministic Encryption: Definitional Equivalences and Constructions without Random Oracles'. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 360–378. DOI: 10.1007/978-3-540-85174-5_20.

[54] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography: Principles and Protocols*. Boca Raton: Chapman & Hall/CRC, 2008. ISBN: 9781584885511.

[55] *ITU Radio Regulations*. URL: https://www.itu.int/pub/R-REG-RR-2016 (visited on 16/04/2018).

[56] Xu Yang, Ying Liu and Shuxi Gong. 'Design of Wideband Omnidirectional Antenna with Characteristic Mode Analysis'. In: *IEEE Antennas and Wireless Propagation Letters* (2018), pp. 1–1. DOI: 10.1109/lawp.2018.2828883.

[57] Haim Matzner and Kirk T. McDonald. *Isotropic Radiators*. 2003. arXiv: physics/0312023.

[58] *Forskrift om generelle tillatelser til bruk av frekvenser (fribruksforskriften)*. URL: `https://lovdata.no/dokument/SF/forskrift/2012-01-19-77#KAPITTEL_2` (visited on 30/04/2018).