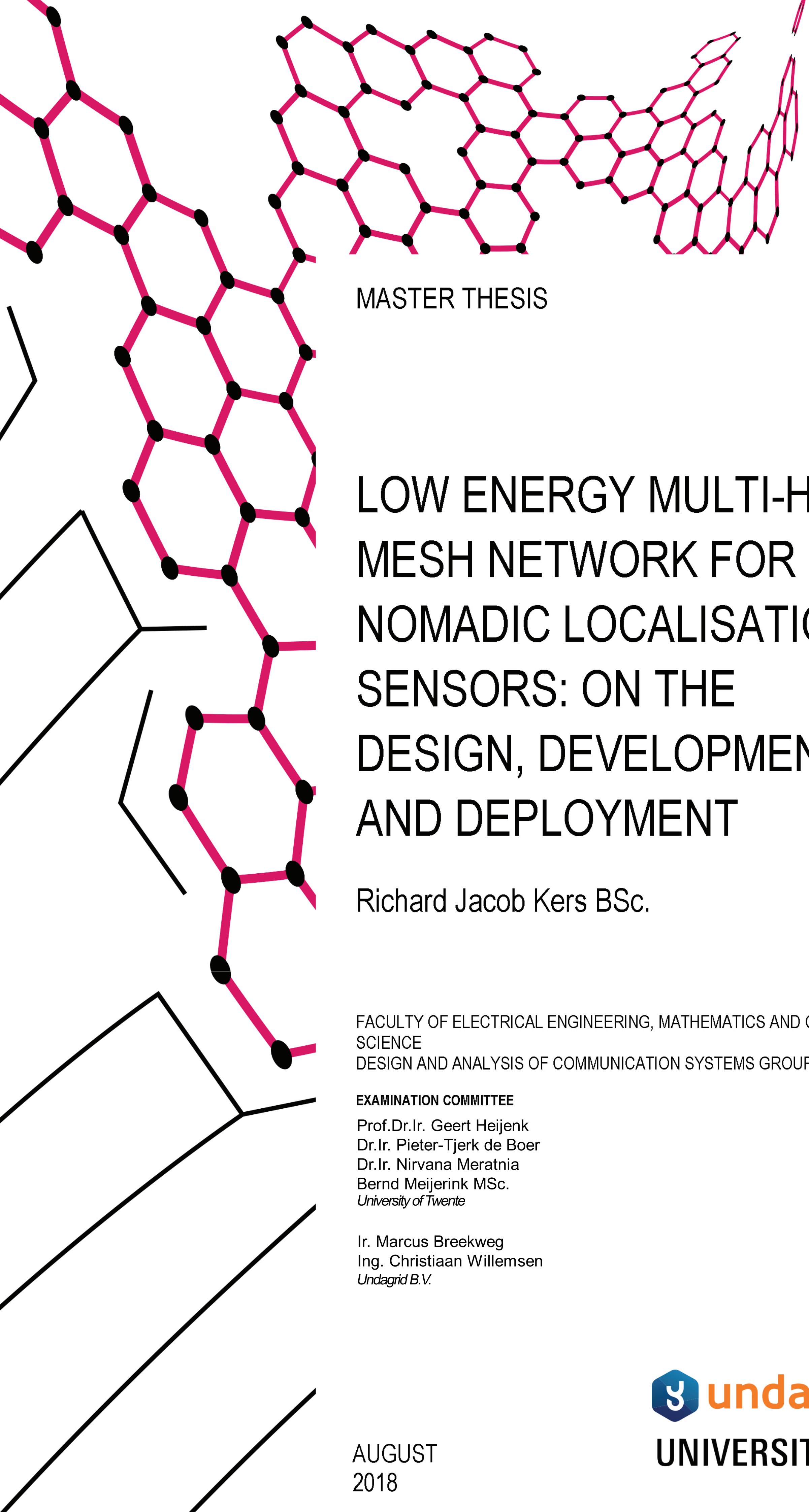


MASTER THESIS

LOW ENERGY MULTI-HOP MESH NETWORK FOR NOMADIC LOCALISATION SENSORS: ON THE DESIGN, DEVELOPMENT AND DEPLOYMENT

Richard Jacob Kers BSc.

FACULTY OF ELECTRICAL ENGINEERING, MATHEMATICS AND COMPUTER SCIENCE
DESIGN AND ANALYSIS OF COMMUNICATION SYSTEMS GROUP

undagrid
UNIVERSITY OF TWENTE.

AUGUST
2018

Abstract

The Internet of Things (IoT) is an ever growing market, and it will continue to grow in the coming years [1, 2, 3]. With wireless sensors becoming cheaper, more energy efficient, easier to integrate and more ubiquitous, the fields of application of IoT devices increases rapidly. One of the big challenges in IoT is scalability. The use of traditional internet access points to connect to sensors to the internet will quickly lead to bottlenecks [4]. One possible way of solving this is by using a so called multi hop mesh network. In such a mesh network, sensors do not have to connect directly to an internet access point (i.e. a so-called gateway), but rather use surrounding neighbours to deliver data packets via them to a gateway. This will not only solve bottleneck issues, but this network topology can also help out in situations where it is hard to get full site wireless connectivity coverage. However, since a sensor is not directly connected to a gateway, packet delivery to a gateway is harder in a mesh network. A routing algorithm should be implemented to correctly direct these messages.

The main question of this thesis is: what is a suitable approach for designing, implementing and testing a multi-hop mesh routing algorithm for a nomadic localisation sensor environment, keeping lower level behaviour in mind? As an example case for these nomadic sensors, data from Undagrid B.V. is used. The tools and related work that is being used in the process of designing this routing algorithm is described in this thesis. The thesis starts with constructing a data analysis framework to characterize sensor behaviour and to test network coverage of a mesh network. After that, related work is described using a literature study. Based on this, a routing design is proposed based on the Collection Tree Protocol (CTP) algorithm, and using the added routing metrics wireless link signal strength, the number of unique paths to a gateway and total transmitted packets. To preliminary test this design, an emulator environment in Python is constructed in which several routing implementations are tested. Then, with lessons learned from this validation step, the design is implemented in the network simulator NS3. This includes altering the MAC layer of the LR-WPAN module used by implementing an idle listening reduction system.

Using Undagrid's data, it was shown that nodes move 6 times per day and a network coverage between 96.96 % and 99.16% was achieved. The increased load in terms of transmitted packets relative to a star topology network increased by a factor of 2.48. With the use of an energy efficient MAC layer, average delay per hop was 2000 ms when a 4000 ms beacon period was used. These results are within the set requirements and therefore can be used in a real life mesh implementation.

Keywords

Internet of Things, wireless sensor networks, nomadic mobility, routing, Collection Tree Protocol, CTP, Expected Transmission Count, ETX, network simulation, NS3, LR-WPAN, localisation sensors, Python, NetworkX.

Acknowledgements

This thesis is not only my largest work so far, it is also the final chapter of my time as a student. It is the final project of my two year master program Embedded Systems at the Design and Analysis of Communication Systems (DACS) group in collaboration with Undagrid B.V. Although this thesis is the work of just one person, many supported me during the process of writing it. For that, I want to thank a few people in particular.

First of all I want to thank my daily supervisors Marcus and Christiaan of Undagrid. You did not only support me in my research, but also assisted me during the developed of my professional career. The meetings that we had were very helpful in times I got stuck and didn't know how to proceed. I learned a lot from you and I am looking forward to spend more time with you on future projects. Secondly, I would like to thank Prof.Dr.Ir. Geert Heijenk as my main supervisor from the exam committee. You helped me to start my research, structure my research and helped me in writing a better thesis. I also would like thank the other members of the exam committee, Dr.Ir Pieter-Tjerk de Boer, Dr.Ir. Nirvana Meratnia and Bernd Meijerink MSc. for the feedback I received on multiple versions of my report. Thanks to all my colleagues at Undagrid for supporting me in my daily activities, discussions during lunch times and fun we had during several projects. Thanks especially to Sjoerd, for helping me out in academic writing.

My thanks go out to all the people that personally are close to me. Special thanks go out to my parents and three sisters, for always believing in me and supporting me in whatever I did. I also would like to thank my second family from Huize Bokkepoot, for taking care of me after working hours. I also would like to express my gratitude for my friends Nick, Tom, Jort and Wout that I know already for at least 7 years. I couldn't have done it without you.

I could not state everybody by name here, but to all friends and family that helped me during my thesis project, master program or bachelor program, thank you so much. You are of great value to me.

Richard Jacob Kers,
Enschede, August 2018

Acronyms

Acronym	**stands for**
BER	Bit Error Rate
ETX	Expected Transmission Count
FSK	Frequency Shift Keying
IoT	Internet of Things
ISM	The industrial, scientific and medical radio bands
MAC	Medium Acces Controller
Manet	mobile ad hoc network
OSI	Open Systems Interconnection
PF	PathFactor
QPSK	Quadrature phase-shift keying
RSSI	received signal strenght
rxCount	Receiving Count
txCount	Transmission Count
txSelf	Transmission Count of self initiated packets

Contents

1 Introduction

The Internet of Things (IoT) is an ever growing market and will continue to grow in the coming years. Revenues for business-to-business IoT by 2020 are estimated at $300 billion by Bain [3] and $267 billion by Boston Consultancy Group (BCG) [1] in market analyses performed in 2017. This adds up to an approximated market size of around $457 billion, attaining a Compound Annual Growth Rate of 28.5%. Accenture projects a CAGR of 7.3% through 2020 [5]. Bain predicts that the most competitive areas of IoT will be in the enterprise and industrial segments. PwC estimates that $6 trillion will be spent between 2015 and 2020 [6]. Business investments will grow from $215 billion to $832 billion in that same period. Consumer spendings will rise from $72 billion to $236 billion. And there are more figures that indicate a rapid growth in IoT investment [1, 2, 3].

One of the big challenges in IoT is scalability. Statistics show that from 2015 to 2020 the number of IoT devices will double, from 15 billion to 30 billion [7]. Each of these devices needs to connect to the rest of the internet, to be able to share the data it collects. If many devices make that connection through one or a few central points (i.e. a so-called gateway), as is the traditional way of working, that quickly leads to bottlenecks [4].

One possible solution is a multi hop mesh network. In this approach, IoT devices do not directly connect to a central gateway, but rather to devices of the same type. Such an IoT device in a mesh network is called a node. If every node connects to its neighbour, an interconnected mesh arises. In a homogeneous mesh network, all nodes are equal in functionality. In an internet-connected mesh network, there are gateways that are the breakout points to the outside world. A visualisation of a mesh network can be found in Figure 1.

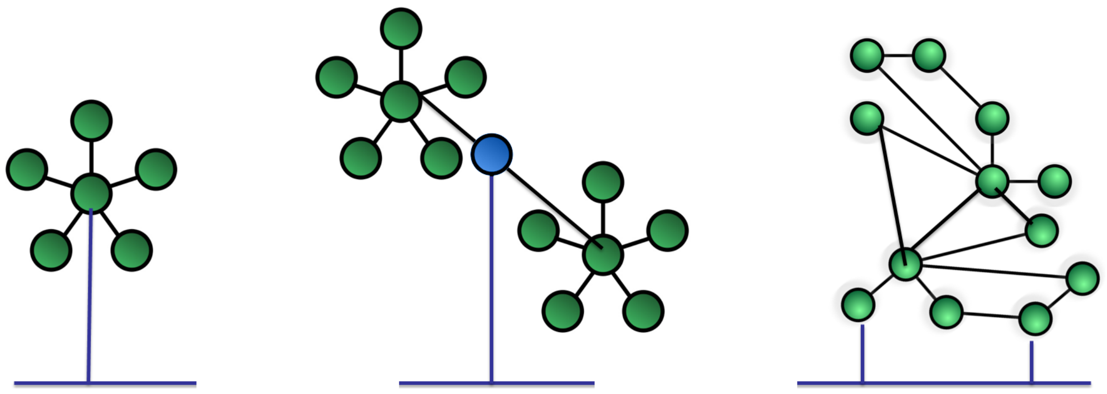


Figure 1: Network Topology Comparison; left a single star topology; middle a multi-star network; right a mesh network

Advantages of using a mesh network are the ease of deployment, the ability to transport information ”around” corners by hopping, the need of less infrastructure, more efficient spectrum utilisation and reuse of hardware [8, 9]. With this distributed network, routing packets from node to endpoint becomes a challenge since nodes do not inherently know how they can route their information to one of the gateways. This research proposes a routing protocol that solves this problem.

1.1 Research Context

Price reduction of smart sensors enables the use of smart sensors on many more things than before [10, 11]. One example of this is tracking of goods, vehicles and assets. This research will focus on developing a routing algorithm for an asset tracking environment with nomadic mobility behaviour.

To define the research context in more detail, this research will be performed in collaboration with a company in asset tracking solutions. This company is called Undagrid. Undagrid started its business based on the asset monitoring question of the S-P-S Group which supports airlines with ground support equipment (GSE). Examples of these carts and dollies are shown in Figure 2. A first pilot was implemented between January 2014 and September of 2014. One of the most important requirements for the S-P-S Group use case is the battery life of the sensor. Since carts are in maintenance only once a year, a battery should at least last for that period. The technology used for this solutions used a mesh network to transport its GPS location fixes to a back-end. To prolong battery lifetime, sensors sleep most of the time. They only wake up to forward messages of other sensors or if a movement is detected by the on-board accelerometer.

This solution did not perform as planned. The capabilities of the mesh network were not matching the requirements of the business case. Location updates could have a delay of hours because of network congestions. The congestions were caused by the way of routing of packets. This flaw made the solution useless and a star topology networking architecture using LoRa on 868MHz was implemented as a temporary alternative [12].

Nowadays, Undagrid provides tracking technologies to optimize logistical processes. At the time of writing, their main field of operation is airports, but business is expanding to ports and general logistics. Their main unique selling point is the tracking of unpowered vehicles in an efficient manner. The Undagrid trackers can run for years without battery replacement. Undagrid now delivers services to the five largest airports of Europe among others Frankfurt and Amsterdam Airport. Undagrid is also participating in projects in harbours and other logistical hubs [13].

At the moment, positioning of the trackers is done based on GPS fixes and Bluetooth Low Energy (BLE) Beacons. This localisation data is then sent over their proprietary LoRa network. This means that there needs to be a LoRa gateways installed on site. To prolong battery life, continuous tracking of assets is disabled in the current set-up. However, if an asset is moving, it is being used. Therefore, it is not available for other activities, and therefore the lack of continuous tracking is not a problem.

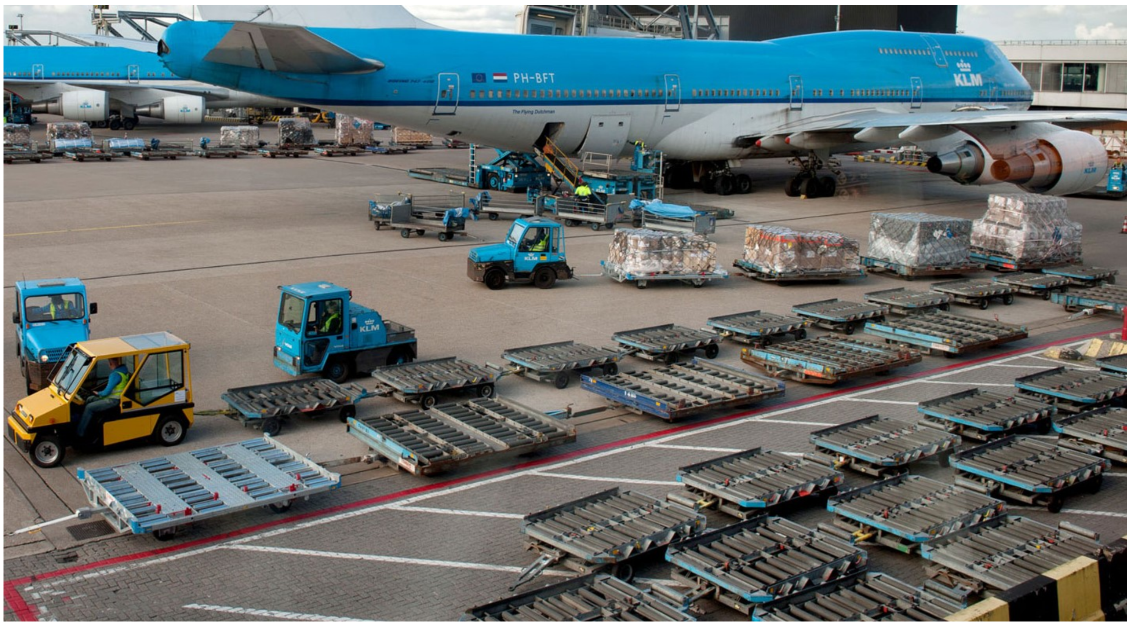


Figure 2: Ground Support Equipment at Schiphol with powered vehicles and unpowered carts [14]

1.2 Motivation and Problem Statement

The advantages of a mesh network stated before are general. However, for Undagrid some of these advantages are business critical. Undagrid's gateways are currently mounted on buildings or vehicles that are generally stationary. Getting approval for gateway installation is cumbersome and in some cases expensive. Besides placing of gateways, the deployment of this wireless infrastructure in the 868MHz band on airports is difficult due to regulations. On the one hand because the 868 MHz band is not a global ISM standard. And on the other hand because some airports fear for interference with air traffic control from communication in the 868 MHz band.

Using a higher frequency, and specifically 2.4 GHz, will circumvent this problem. It is a global standard that is widely accepted by airports. However, moving to a higher frequency sacrifices range. Next to that, 2.4 GHz is used intensively for wireless consumer devices using, for example, Bluetooth and Wifi. This can cause interference.

Yet another reason why a mesh network might be in Undagrid's benefit, is that in areas with loads of radio signal reflection and blocking, such as an airport hangar or shipping container logistic site, direct communication to a gateway might be blocked. Hopping around these obstacles could solve that problem. Another advantage of mesh networking for a nomadic localisation network is that nodes scan their neighbourhood anyway to determine their latitude and longitude coordinates. Therefore the impact on the battery for neighbour discovery solely for networking is limited since neighbour information is already present.

1.3 Research Questions

Because of the reasons presented in the previous section, Undagrid wants to develop a network that uses 2.4 GHz with smaller, more mobile gateways. This might be in the form of a mesh-network, if this is favourable in the case of a nomadic localisation sensor network. Whether or not a mesh network is beneficial, will depend on the mobility of sensors, radio communication characteristics, radio transmit and receive time slot, and routing algorithm approach. These elements have to be investigated in order to determine what mesh network, if any, is suitable for a nomadic localisation network. Because of time constraints, not every aspect can be investigated in this master thesis research. Therefore this research focuses on routing, and research into physical and medium access behaviour effects is kept minimal. This can be summarised in the main research question of this thesis, which is:

"What is a suitable approach for designing, implementing and testing a multi-hop mesh routing algorithm for a nomadic localisation sensor environment, keeping lower level behaviour in mind?"

To answer this main research question, sub research questions are formulated. These are stated below.

1. *To what extent do Undagrids sensors follow a nomadic mobility model?*
   (a) *Can the nodes be characterized as nomadic?*
      i. *How often does a node move?*
      ii. *For how long does a node move?*
      iii. *How far does a node travel?*
   (b) *What are the characteristics of a group of nodes that form one connected network, also referred to as a cluster?*
      i. *How many neighbours does a node have?*
      ii. *How many nodes are critical?* A critical node is a node that holds two smaller clusters together. If this node moves, a cluster splits in two. Load on this node is increased since all traffic of a cluster has to move through this node.
      iii. *What is the size of a cluster?*

iv. *What would be the coverage of a mesh network be, if a mesh network were to be deployed?*
v. *To what extent does a cluster change over time?*

2. *What Routing layer to design?* In previous research, a routing protocol comparison has been made including a proposed routing scheme [15]. In this thesis, the full design will be formalized. The routing layer has to work in a nomadic localisation sensor network, for example implemented at Undagrid.
3. *Which design is most suitable for the layers supporting the routing layer?* The routing layer will be build on top of a MAC layer, which is build on top of a physical layer. These layers influence the behaviour of the routing layer.
   (a) *What MAC layer for Routing Layer testing should be used?* Different MAC layers will be investigated and ranked. Criteria for this ranking are based on the total number of packets send, delay, and packet loss prevention and robustness
   (b) *What physical layer for Routing Layer testing should be used?* What constraints are there to the radio communication? Include but not limit to subjects like frequency band, modulation scheme, modulation efficiency, out of band emission, duty cycle and radio transmission regulations.
4. *What steps should be taken to verify system behaviour for these layers (PHY, MAC, Routing)?*
5. *What steps should be taken to simulate these layers?*
6. *What situations have to be tested?*

1.4 Approach

This project is divided into four parts. The first part is preliminary research, focussing on whether the mobility of Undagrid's nodes matches that of a multi-hop mesh network. Also, requirements for ad-hoc nomadic node routing will be presented. Based on these two aspects, a literature study is performed and several available mesh techniques and algorithms are investigated. This results in a list of concepts that give general directions for designing a routing algorithm. The second part is the design of a routing algorithm that should, based on results of the first part, fit a nomadic localisation sensor network. This design is tested in a preliminary design validation emulator, before continuing to a more elaborate simulator. The preliminary emulator only tests the routing layer and ignores lower layer behaviour. This means that delay, timing and packet collisions are not simulated. The third part describes a more elaborate simulation environment where the lower level behaviour is simulated as well. Then in the fourth and last part, several test cases for this routing algorithm is presented including their results.

1.5 Contribution

In this thesis, a successful multi-hop mesh network has been constructed and tested, containing a routing algorithm and a simple energy efficient MAC layer for use in a nomadic localisation sensor network. This has resulted in the following contributions:

- An overview of what multi-hop network requirements should be met for a nomadic sensor network.
- A framework for characterizing node mobility behaviour and cluster properties. One is able to form virtual clusters such that the applicability of network topologies can be tested. The output of this framework is in the form of an animation of cluster changes over time and graphing tools for plotting cluster statistics over time.
- An overview of what ready to market solutions are available for deploying a ad-hoc mesh network, as well as a literature study into different types of routing algorithms suitable for ad-hoc mesh networking.

- An emulator, written in Python, that can test routing algorithms. It omits timing properties of lower layers, but is able to compute path loss and bit errors.
- A NS3 environment that is able to simulate routing algorithms that operates in a 2.4 GHz environment. It includes the design, implementation and testing of an energy efficient MAC layer that is able to work with the NS3's LR-WPAN module. It also includes expansion of the NS3 LR-WPAN module with optimizations for speeding up simulations time in the case of a large amount of nodes, enabling beaconing, enabling 64-bit addressing mode and support for performing RSSI measurements. The environment also includes a set of tools written in Python and R to evaluate NS3 results, such as plotting and animating networks and network characteristics.
- The design, implementation and testing of a routing algorithm that is suitable for nomadic sensor networks. Using information from direct neighbours, routes can be constructed to the nearest gateway. In the case of multiple paths to a gateway, based on 4 metrics, the most optimal route will be chosen using a predefined scoring function. Aforementioned frameworks and environments are used to test the design. The four metrics are the number of hops to the nearest gateway, the number of unique paths from a node to the nearest gateway, the load of a node so far in terms of total transmitted packets and the link quality between nodes in terms of received signal strength indication (RSSI).

1.6 Thesis Outline

This report is structured in the following manner. First, a chapter is devoted to supplying background information. This is presented in Section 2. Then, in Section section 3, the requirements are specified, data analysis for sensor mobility characterisation is presented, related work is examined and a routing algorithm is proposed.

This proposal is worked out in more detail in Section 4. It starts with additional requirements on the design, followed by the basic principles and core parameters of the routing layer. The routing algorithm is also described using directed graphs. Since testing in a full-fledged network simulator takes a lot of time, a preliminary design verification will be done in an emulator. A summary of this work is presented in Section 4.6, but fully covered in Appendix D. This includes emulator simplifications and assumptions, design implementation, results and conclusion. Since that test was successful, the design implementation is finished.

The routing layer design will be tested in NS3. The way the NS3 environment is set-up, is described in Section 5. It contains parameter settings and alterations to existing NS3 modules to fit the environment of this research. Then in Section 6, the routing layer that will be designed is implemented and tested in NS3. This section will contain the implementation details and routing layer results. In this section, the routing layer will be tested using multiple simulation runs, each giving insight into a different aspect of the routing layer. It also covers the calculation of routing parameters that could not be determined analytically. Also all NS3 results are discussed and concluded in the final part of that section. Then, to conclude this thesis, Section 7 contains the overall conclusion of this thesis, including limitations and future work proposals.

2 Background Information

This section provides a theoretical background into mesh networking and the related technologies. Since this research is about networking, the basics of packet exchange between devices is explained in Section 2.1. Then, to give more insights in mesh, multi-hop and ad-hoc networking, Section 2.2 explains in detail what these type of networks are and what to take into account. Then, in Section 2.3 the wireless communication techniques Bluetooth, LoRa and FLRC are explained. These three techniques are candidates for being used at Undagrid.

2.1 Network Layers and OSI stack

The Open Systems Interconnection model (OSI model) distinguishes different layers in a telecommunication system and standardises functionalities associated with these layers [16, 17, 18]. Using this abstraction, implementations of layers can be interchanged which contributes to standardisation. An overview of the OSI stack can be seen in Figure 3.

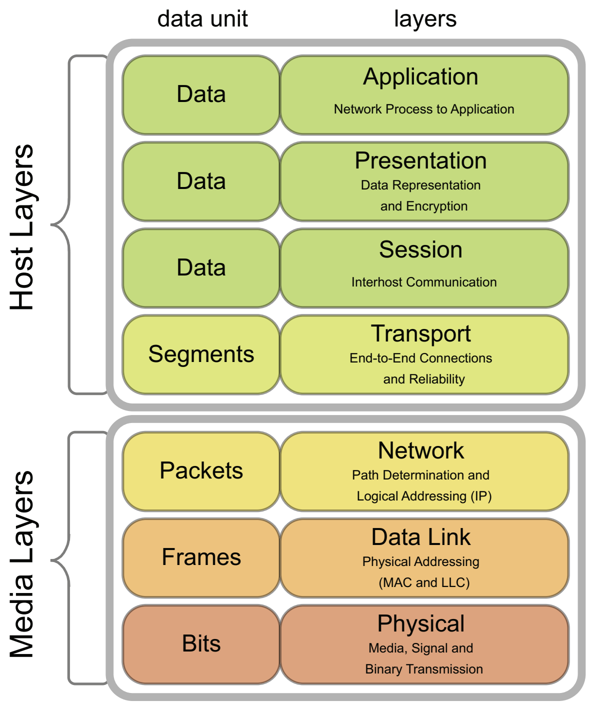

data unit
layers
Host Layers
Data
Application
Network Process to Application
Data
Presentation
Data Representation and Encryption
Data
Session
Interhost Communication
Segments
Transport
End-to-End Connections and Reliability
Media Layers
Packets
Network
Path Determination and Logical Addressing (IP)
Frames
Data Link
Physical Addressing (MAC and LLC)
Bits
Physical
Media, Signal and Binary Transmission

Figure 3: OSI model [19]

Wireless telecommunication is based on an electromagnetic (EM) waves. These waves can propagate through space. Using a radio, the transmission and receiving of these EM waves can be realised. EM waves, like waves in general, are characterized by their frequency, amplitude and phase.

To avoid interference, regulations state which frequencies are free to use for radio transmission and which ones are reserved. Available frequencies are bundled as a spectrum or bands. These bands describe a certain frequency range with pre-defined channels. An example of a free to use band is the 2.4 GHz band, which is used by Bluetooth, WiFi and other wireless links. Regulations prescribe what their maximum transmit power is and what duty cycle is allowed.

A simple wave travelling from one place to another does not necessarily carry information. To put information into a wave, modulation has to be applied. This technique changes one of the wave characteristics.

If the frequency of a wave changes, the modulation that is used is referred to as frequency modulation. The same holds for amplitude modulation (AM) and phase modulation (PM). Each modulation method has its own benefits and drawbacks, for example in terms of spectrum utilisation, out of band interference or energy efficiency.

The physical layer in the OSI stack describes the frequency and modulation to be used in a wireless link. It translates raw information packets from a string into bits (1s and 0s) and modulates this to a signal that can be transmitted. It also is responsible for decoding a received message from a modulated input signal and converting it back to characters [16, 17].

The data link layer consists mainly of the medium access controller (MAC) and logic link controller (LLC). The data link layer has the responsibility of reliably transmitting data from one device to another ordered by the network layer above it. If this fails it will callback to the upper layers. A MAC determines when a device has to transmit data, receive data or switch off the radio. It is also responsible for adding frame sequence numbers and error checking [16, 17, 18] .

The network layer, or routing layer, is responsible for routing. Routing data means that information from higher layers is sent to the right devices. Routing is by a routing algorithm that is based on routing tables or routing parameters. There are several routing algorithm categories: proactive routing, reactive routing, hybrid routing and hierarchical routing. An overview of these categories can be found in [15].

2.2 Mesh, Multi-Hop and Ad Hoc Networking

In networking, ad-hoc is generally used to indicate something temporary, provisional, or improvised methods to deal with a particular problem. In this research this specifically means that a network can form and break any time, anywhere without constraints. This behaviour has impact on the lower layers of the OSI stack, since information about connected devices might quickly become outdated.

As already shown in Figure 1, a mesh network is a network where devices connect to many neighbouring devices and thus form a network. Specifically, in a homogeneous network where all nodes have equal capabilities and in absence of dedicated control units, distributed routing algorithms should be deployed in order to have an efficient network. A multi hop network means that a packet can take multiple hops before it reaches its final destination. Forwarding a message from one device to another device is referred to as a hop.

2.3 Wireless Communication Technologies

A wireless localisation network should be able to communicate without wires. There are several standards that enable devices to connect via a radio. Following Undagrid's previous research and the constraint of 2.4 GHz, the list of communication technologies is limited to a smaller set. In the following section, Bluetooth, LoRa and FLRC communication will shortly be explained. This is done since network simulations will be influenced by these technologies in terms of packet collision, modulation and MAC timing characteristics. There are more wireless communication technologies that are focussed on energy efficient sensor networks, such as IEEE 802.15.4, Zigbee or ZWave. However, since these technologies are less integrated in Undagrid's systems, these are not considered further in this research. Changing to these techniques will be to much work.

Bluetooth is a full specification, including physical layer, networking layers and higher software layers. LoRa and FLRC however in principle modulation techniques, and hence only describe physical layer properties. It should be noted that LoRa and LoRaWAN are not the same.

2.3.1 Bluetooth Low Energy

Bluetooth is a wireless, short-range communication standard that was developed in Sweden by Ericsson in 1994. Initially, Bluetooth was as a low power, robust and low cost protocol and as an wireless alternative to a wired serial connection. This means that all sorts of applications should be able to use a Bluetooth connection. Bluetooth's range is typically between 10 and 100 meters and uses the 2.4 GHz frequency band within the industrial, scientific and medical (ISM) band.

Since it's first version, several versions Bluetooth have been released. The development is steered by the Bluetooth Special Interest Group (SIG), which publishes official documentation and standardisation for the different versions of Bluetooth. The major releases of Bluetooth are the Basic Rate/Enhanced Data Rate (BR/EDR) standard upto version 2.1, High Speed (HS) in version 3, and the Low Energy or Smart specification in version 4.0.

Bluetooth upto version 2.1 is Bluetooth's basic specification. It uses GFSK on the 79 defined channels with 1 MHz bandwidth (-20 dB). With this, it's peak data rate lies at 1 Mbps. Bluetooth uses frequency hopping to provide resistance against multipath effects and it acts as a multiple access system for neighbouring devices also communicating in the 2.4 GHz band. At connection initialisation, pseudo random sequence information is shared. This information is used for frequency hopping amongst connected devices[20]. Bluetooth version 3.0 adds, in addition to other enhancements, a mechanism for higher data rates, up to 24 Mbps. This involves the addition of an additional controller that is compliant to the IEEE 802.11 standard. Roughly speaking, this adds WiFi connection capabilities to the Bluetooth standard in case higher data rates are necessary. This controller is known as the Alternative MAC/PHY (AMP). It uses the default Bluetooth discovery of devices, pairing mechanism and configuration. This approach saves energy [20].

Bluetooth Smart or Bluetooth Low Energy (BLE) is a relativity new standard (introduced in 2010) and was quickly accepted and used by manufacturers in consumer goods [21]. The BLE standard comprises, at the time of writing, of a few versions that where developed over the years. These numbers are version 4.0, 4.1 and 4.2 [22, 23]. The first version of BLE, version 4.0, was introduced in June 2010. This version had to follow the paradigm shift of having many portable devices that are energy critical. Bluetooth Low Energy has a more efficient neighbour discovery system compared to it's previous releases. It also is more secure by using an improved pairing mechanism. BLE also adds the ability to have a changing public address. With this in place, unwanted tracking of a devices is harder and therefore enhances the privacy cabapilities of Bluetooth. Compared to previous Bluetooth versions, BLE uses 40 channels spaced 2 MHz apart instead of 79 channels spaced 1 MHz apart [20]. Bluetooth 4.2 uses bandwidth of 1.2 MHz and a bit rate of 1 Mbps using Gaussian Frequency Shift Keying (GFSK) [24]. Version 4.1 and 4.2 are incremental upgrades from version 4.0. The largest improvements in 4.1 is the improvement on coexistence with LTE 4G and the ability to be a BLE peripheral as well as a central device at the same time. Version 4.2 adds extra functionality to improve application in IoT use cases. By increasing maximum packet sizes from 27 bytes to 251 bytes and lowering latency, this version is capable of using IPv6 [23].

As of December 7, 2016 the Bluetooth SIG has officially adopted Bluetooth 5. This new version can get up to 4 times the range, 2 times the speed, 8 times the broadcasting message capacity, and improved coexistence with other cellular and wireless technologies [25]. It also introduces Bluetooth Mesh, a standard profile for devices to communicate in a multi-hop fashion.

2.3.2 LoRa and FLRC

LoRa is a wireless communication technology developed by Semtech that combines long range with low power and low energy consumption. It is a Direct Sequence Spread Spectrum (DSSS) system, which means that the transmitted radio signal is spread over a larger frequency spectrum. This process is generally achieved by multiplying the wanted data signal with a spreading code, also known as a chip sequence. An exclusive OR (XOR) operation can be used for this. The combined bit stream has the data rate of the spreading code sequence, and therefore uses a broader frequency spectrum. By using a much broader spectrum, the signal to noise figure can be way lower than in traditional radio systems. This can also be seen from Shannon-Hartley's theorem which states:

$$C = B \cdot \log_2(1 + \frac{S}{N}) \tag{1}$$

Where C is the channel capacity in bits/s, B the channel bandwith in Hz, S the average received signal power in Watts and N the average noise in Watts. using the relation of $ln = log_e$ and assuming $\frac{S}{N} \ll 1$, Equation 1 can be rewritten as [25]:

$$\begin{aligned} \frac{C}{B} &= \frac{1}{log(2)} \cdot log_e(1 + \frac{S}{N}) \\ \frac{C}{B} &\approx 1.433 \cdot \frac{S}{N} \end{aligned} \tag{2}$$

This means that for a system with a very low signal to noise ratio, error-less communication is possible if a sufficiently wide bandwidth is used for the wireless link.

In LoRa modulation the spreading of the spectrum is achieved by generating a chirp signal that continuously varies in frequency. Spread spectrum results in a higher resilience against distortions like signal jamming and Doppler shift. A visualisation of the spreading over a broader spectrum is shown in Figure 5 Typically, LoRa radio sensitivity has an 10 dB improvement compared to frequency shift keying (FSK) [25]. This is shown in Figure 4. At the moment, Undagrid uses a LoRa star network for their sensors.

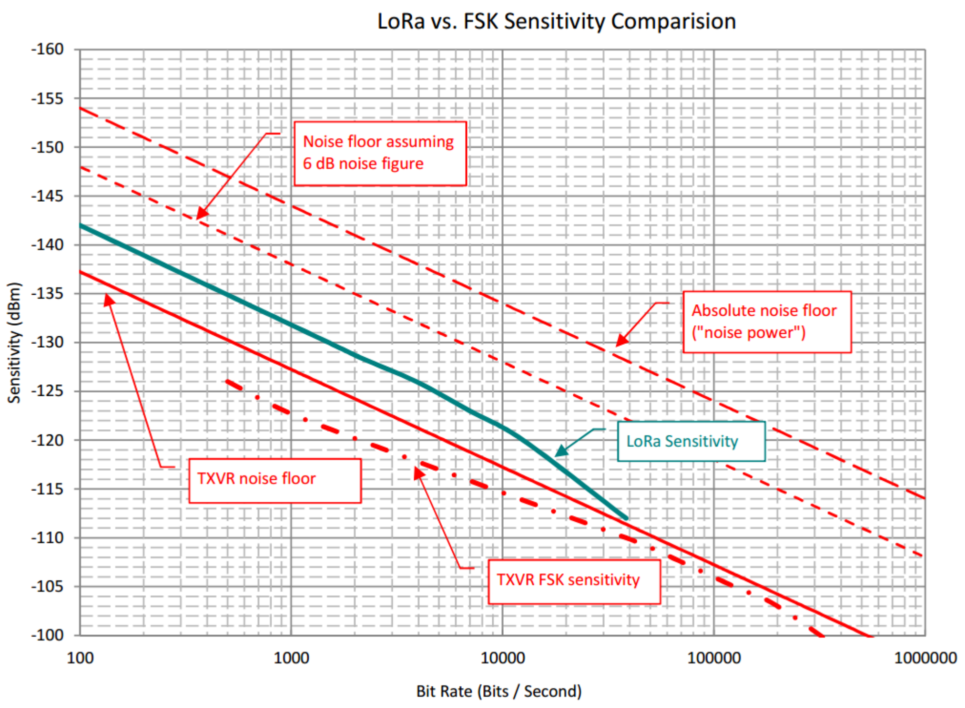


Figure 4: LoRa vs FSK radio sensitivity [25]

Fast Long Range Communication (FLRC) modulation uses Gausian Mean Shift Keying (GMSK) in combination with forward error correction to improve receiver sensitivity. Compared to FSK, FLRC can get 8 to 10 dB improvement at the same data rate. Data rates between 260 kbps and 1300 kbps are possible, using a bandwidth between 0.3 MHz and 1.2 MHz. There are radios available that support FLRC at 2.4 GHz, which would enable a nomadic sensor network to use FLRC physical layer for a mesh network.

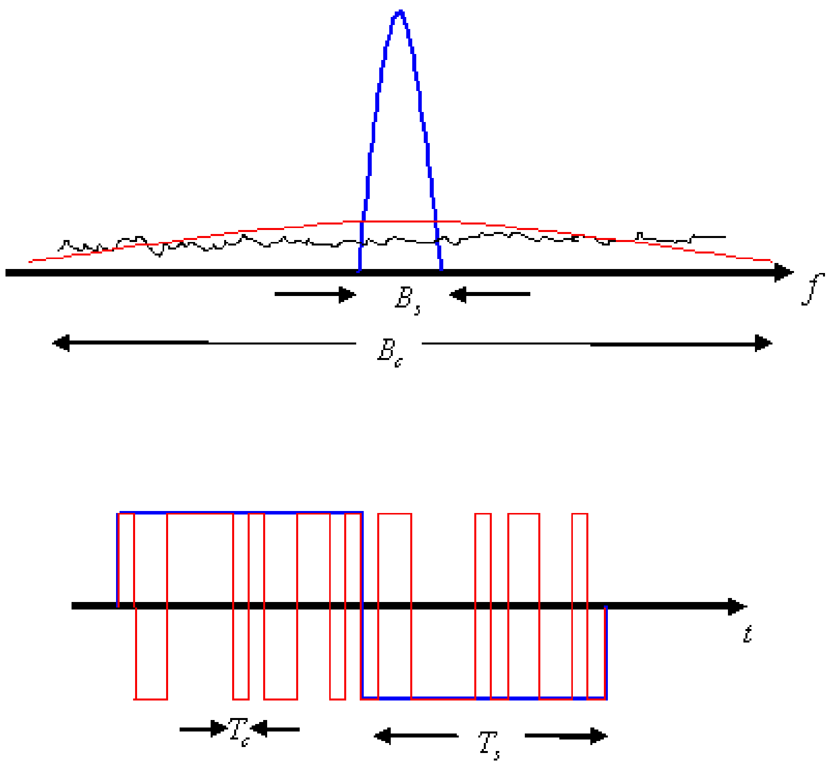


Figure 5: Spread Spectrum explained [26]. In blue the original bit signal in both frequency and time domain. In red the coding bit stream and the resulting frequency spectrum. As the pulses of the spread sequence get shorter, the resulting used frequency increases.

3 Prestudy on Requirements, Feasibility and Related Work

Prior to this report, a study on feasibility and related work has been performed. This research report can be found in [15]. In this study, the applicability of a mesh network to a nomadic node network is being tested and a literature study has been performed to investigate related work. This chapter will repeat the key aspects of this research papers.

In the first section of this chapter, requirements are put forward that give direction to a feasibility study and help to select interesting routing algorithms. Based on these requirements, a feasibility study is performed. This study will answer research question 1. It might be the case that Undagrid's nodes do not have a suitable mobility model or that nodes are spaced to far apart from each other to form a cluster.

3.1 Requirements

To evaluate both the feasibility of implementing a mesh network, as well as selecting routing candidates, it is important to know what the requirements in a mesh network are. Some requirements are set by the environment the mesh network has to operate in. For example, assets should not have to come to a repair shop to change batteries of the sensor. These requirements will help in selecting a routing scheme that fits the use case of a nomadic localisation network best. The requirements are split up in functional and non-functional requirements. Non-functional requirements are constraints upon system behaviour or quality attributes for a system [27]. These constraints can often be expressed quantitatively. Functional requirements specify something a system should do or how a system should behave.

The non-functional requirements are stated below.

- *Hardware:* Energy efficient hardware that can run from a battery for at least 2 years. In most cases, an asset has to be repaired or go for maintenance once every year.
- *Latency:* Packets should not have a delay greater than 1 minute. Higher delays can render the tracking solution invaluable for end customers that want to locate assets.
- *Cluster forming:* a node should be able to join a network and transmit its data within 30 seconds (see AppendixA.1). Quick cluster discovery lowers the delay and enables an isolated cluster to connect to a mobile gateway that drives by.
- *Bluetooth Low Energy mode:* for data offloading and integration, BLE should be supported. Since BLE is integrated into many mobile devices, such as smartphones, these devices can easily be deployed as mobile gateways. BLE is also lightweight enough to serve as a viable communication line between a sensor and a mobile gateway. This does not mean that node-to-node communication has to be BLE. It only means that nodes should be able to connect to a BLE breakout device. However, BLE supports beacon packets up to 32 bytes.
- *Bandwidth:* communication should be working with a bandwidth of 10 - 1000 kbps. Since FLRC and BLE have low data rates, transmission time could take longer than expected, compared to normal wireless networks.
- *Frequency:* 2.4 GHz has to be used for wireless communication.

Below, the functional requirements are stated. These requirements are system behaviour requirements and can help in differentiating routing algorithms.

- *No Packet Loss:* Since node states are updated, missing a packet means incorrect information to the customer.

- *Routing Overhead:* Sharing routing information should be kept to a minimum while maintaining the ability to be agile and determine optimum routes.
- *Load Balancing:* Load should be shared amongst nodes, preventing quick battery drainage of individual nodes.
- *Implementation:* The implementation of a routing layer should be such that it can be adopted by Undagrid's platform.
- *Gateway:* Nodes should forward messages in a way that a message should always end at a gateway.
- *Distributed:* Routing decisions should be made on a node basis and without help of a central controller.
- *Small memory footprint:* Not all routing information can be stored since RAM is limited on embedded devices.
- *Scalable:* The routing protocol should be able to handle cluster sizes that are large enough.
- *Packet buffer:* Isolated nodes should buffer their messages and offload them when connected to a cluster.
- *Oscillation and loop suppression:* Route information should not oscillate and the routing layer should avoid unnecessary packet loops.
- *Network changes:* Broken link detection and other network changes should be detected in a quick and efficient manner.
- *Matching Data Packet Flow:* Every node should be able to reach a gateway in an easy manner. A gateway does not have to be able to find every individual node.

3.2 Feasibility Study

First of all, the applicability of a mesh network should be proven. This is done by taking localisation sensor data from Undagrid's database and building a virtual mesh network using one or multiple graphs. This data contains latitudes and longitudes of sensors. Based on this, nodes can be placed in a graph. A graph is a collection of vertices or nodes with connections or edges between them. Graph analysis is formalised in graph theory. Based on graphs, node statistics can be deduced. This research has been done during the prestudy for this master thesis [15]. Details of this analysis can be found in that report. For convenience, the relevant chapter is included in the appendices under Appendix C. The keypoints of this research are repeated in this section.

To analyse if clustering is possible, GPS locations of nodes will be used to form virtual clusters. This data is provided via the Undagrid database. Forming clusters will be done using the DBSCAN function from the Python library sklearn. This algorithm will form clusters based on a predefined condition. In this case, this condition is that if two nodes are within 85m from each other, they can form a wireless link. This distance is based on Undagrids own research into the range of their nodes when communication over BLE. This data is included in Appendix B. This research shows that communication over 150 meter is still possible. However, to compensate for non-line of sight, 85 meter is used as a conservative distance. Tests using a distance up to 150 meter will be included in this section as well.

Multiple linked nodes can form a cluster. The minimum size of a cluster is 3 nodes. The result of the DBSCAN is fed into a Python graph theory library called NetworkX. This converts the cluster into a graph such that graph analysis can be performed on the clusters. Data of 17 days is used. The first 10 days severe as an initialisation phase such that an initial graph can be constructed. Changes in the network for the following 7 days are monitored and characterized.

During simulation, changes in clusters are written to a log file. This data will be used for determining cluster statistics. Also, the generated clusters are plotted and saved for post-processing.

3.2.1 Results

First, the results for individual node characteristics are shown. In Table 1, the travel times are presented. On average a node moves approximately 6 times a day. Some nodes show clock issues. This is not harmful in production, but these nodes are filtered out for calculation node characteristics.

Table 1: Time of Travel analysis results

Total number of movements:	40047
Total number of nodes before filtering:	1306
Total number of nodes after filtering:	846
Average number of movements of a node per day (global) :	6.76
Average number of movements of a node per day (grouped) :	5.78
Maximum travelled time (outlier) :	535718 [s]
Average travel time :	132.269 [s]

Then, distance characteristics are shown in Table 2. The number of movements differ from the number of movements presented in Table 1. This is because not every movement gets a proper GPS fix. This means that some data points have to be removed to get an accurate distance measurement.

Table 2: Distance of Travel analysis results

Total number of movements before filtering:	35756
Total number of movements after filtering:	35747
Average trip distance :	30.4m
Maximum travelled distance (outlier) :	4930 [m]
Average distance travelled of a node per day:	213m

The last data analysis results are shown in Table 3, and show different graph characteristics for different node-to-node distances.

In Table 3, critical nodes are nodes that connect two clusters together, where a cluster should be larger than 3 nodes in size. The out of cluster percentage represents how many nodes are not in a cluster. From Table 3 it can be seen that the connectivity of a cluster goes up if the range increases.

An overview of all nodes and their clusters is shown in Figure 6. It shows nodes on a map at the Schiphol site.

Now, let's zoom in on a couple of clusters and investigate what kind of clusters there can be found. Three examples are picked and displayed in Figure 7. The left figure shows a network with a central node. This node is critical to the system, since removing it would break the cluster. The middle image shows a well-connected cluster, and the most right an image of one of the smallest clusters.

3.2.2 Discussion and Conclusion

Nodes in the investigated network behave nomadically. They only move a couple of times a day, and only for short periods of time. Deploying a mesh network might be successful. A reasonable range of a node is between 80 and 150 meters. For these distances, out of coverage is between 3.04% and 0.84% and the number of critical nodes ranges between 7.24 and 0.51 on average. Also note that the average number of critical nodes at 150 meters is 0.51. This means that the clusters are highly interconnected. This conclusion can also be drawn from the average median connectivity.

Table 3: Overview of cluster characteristics with different maximum inter node distances.

	Maximum distance between nodes	**30m**	**85m**	**100m**	**150m**
Cluster					
	Mean number of critical Nodes	24.11	7.24	5.27	0.51
	Average Mean cluster size	13.44	50.88	67.88	132.55
	Average Median cluster size	5.42	17.02	18.41	53.29
	Average out of cluster percentage	20.69%	3.04%	1.84%	0.84%
	Average coverage of mesh network	79.31%	96.96%	98.16%	99.16%
	Total number of nodes in network	837	837	837	837
	Number of nodes that lose cluster connection at least once every 7 days	649	284	185	121
	or	77.54%	33.93%	22.10%	14.46%
	Number of nodes exceeding 3 hour out of cluster limit	480	112	67	24
	or	57.35%	13.38%	8.00%	2.87%
	Average median connectivity	3.19	7.25	8.11	23.09

3.3 Routing Algorithm Analysis

In this section, an overview of a literature study for nomadic routing algorithms will be presented. To structure the routing algorithms, a grouping of these algorithmns will be performed first. These algorithms can be grouped based on the way routes are found. One way to structure them in this manner is in four groups called proactive, reactive, hybrid or hierarchical [28, 29, 30, 31]. A full overview can be found in the accompanied research topics report [15].

Proactive routing shares routing information up front. If a packet needs to be routed, up to date information is available and a packet can be sent right away. This lowers latency but increases routing overhead when little packets have to be sent. An example of this type of routing is DSDV, which maintains a table of active links to other nodes [32, 33]. In general, this type of routing is too much overhead for a mesh network.

Reactive routing is the opposite of proactive routing. It only gets route information if a packet needs to be sent. This increases delay but reduces overhead if only a little number of packets has to be sent. DSR [34] is a scheme that is reactive and works by flooding a RouteRequest packet to direct and indirect neighbours. This packet carries information about the desired destination node. If the RouteRequest is received by the destination node, it will respond with a RouteReply containing link information and hop information such that data packets from source to destination can be transmitted. AODV [35] and OLSR [36, 37] are also reactive routing algorithms [33].

Hybrid routing combines reactive and proactive routing. This means, in the case of Zone Routing Protocol, a node is proactive in the neighbourhood of a node, and reactive outside this zone [38].

Hierarchical routing is a little different compared to proactive and reactive routing. In hierarchical routing, a node will belong to a certain level or category. Nodes can take responsibility depending on their hierarchy information, for example become a cluster head. Examples of this type of routing are Hierarchical State Routing (HSR) and Fisheye State Routing(FSR) [39, 30]. A fairly well established hierarchical routing algorithm is the Collection Tree Protocol [40]. Its first release was in 2009. Sometimes it is also referred to as gradient routing. It is a proactive routing protocol that uses one metric to create a hierarchical structure such that detailed routing information is not necessary.

The main metric in CTR is the expected transmissions count (ETX) for a message seen from a certain node to a gateway. This is calculated by knowing that a central destination node has a ETX of 0. Such a central destination node is also called a sink or a root. These two rules can be described by Equation 4 and

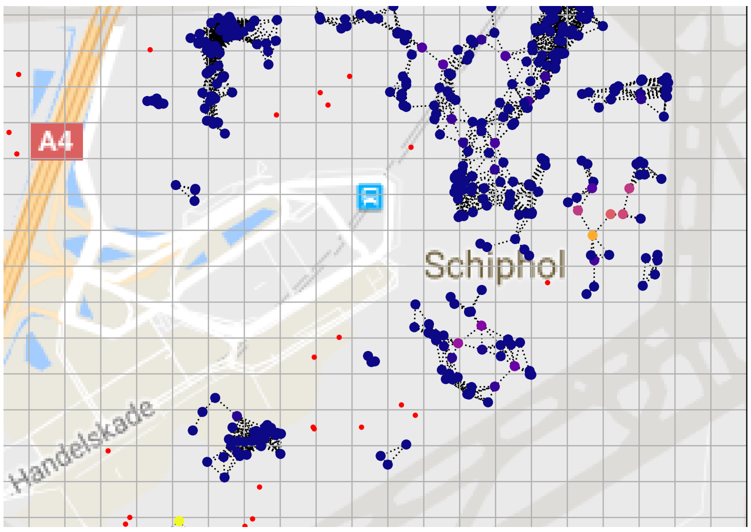
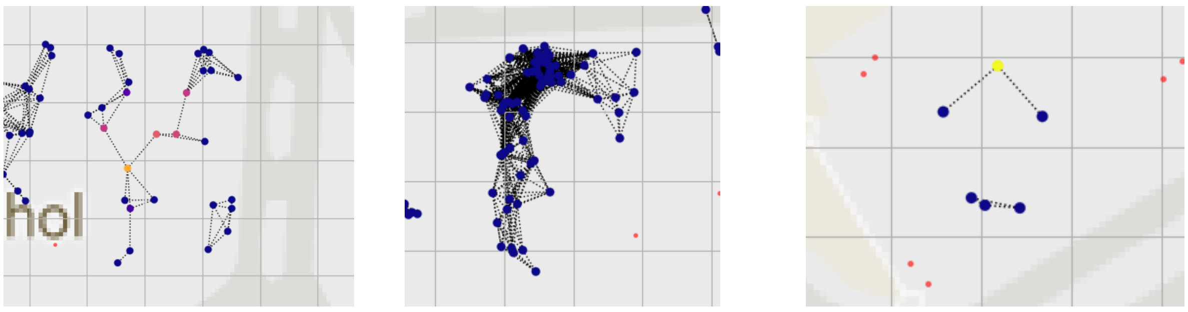


Figure 6: Clusters plotted on Schiphol. A full page version can be found in Appendix C



(a) a cluster with one critical node (yellow)

(b) A well-connected cluster

(c) Small cluster with outliers

Figure 7: Subset of clusters with different charcteristics

Equation 3 [41]. An illustration of ETX routing is shown in Figure 8a.

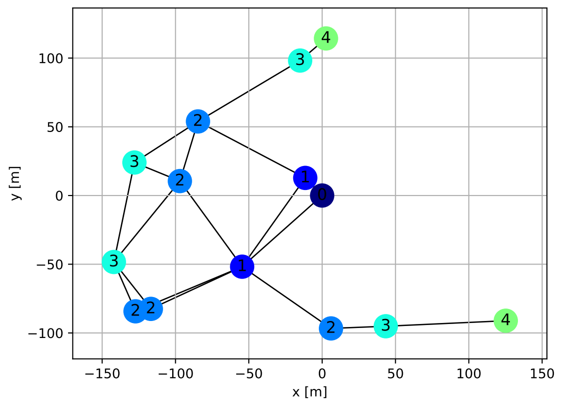
$$ETX_{root} = 0 \tag{3}$$

$$ETX_{node} = ETX_{parent} + ETX_{link} \tag{4}$$

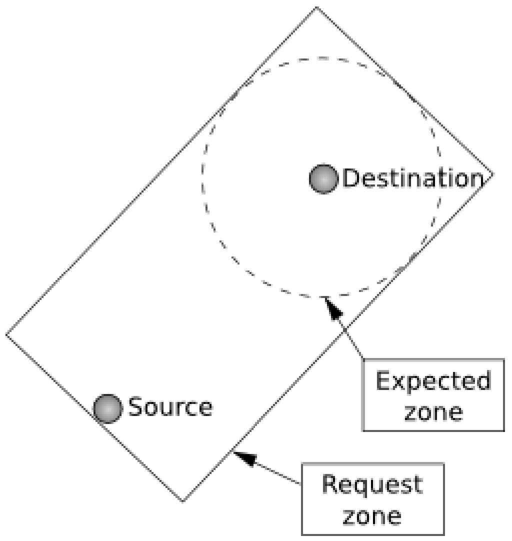
Besides these groups, that are based on how information is spread, grouping can also be done based on resources. This includes power-aware routing or location-aware routing. One possibility to implement location-aware routing is to construct a hierarchy using location data. Another option is geohash routing. In this approach, geohashes are used to simplify the grouping of nodes. If a packet needs to be forwarded, a node can look up its last known location and forward the packet in the right direction. This, however, has the thread of a packet becoming stuck at a dead end. These dead ends are encountered if there exists no path in de direction of the Euclidean shortest distance.

Location-aware routing or location aided routing was described by [34]. Since then, multiple derivatives have been made to suit special needs. It uses the basics of DSR [34] and AODV [42] but bounds the flooding of the RouteRequest packet by defining an expected zone and a request zone. The expected zone is the zone where the destination node should be located. The request zone is the zone on which RouteRequest packets should be forwarded. If a node outside the request zone receives a RouteRequest it won't forward

the packet. GeoAODV is another location-aware routing scheme [43]. An illustration these areas can be found in Figure 8b. In [44], LAR is compared with GeoAODV and plain AODV and show that LAR and GeoAODV performs similar and that both protocols perform better than plain AODV.
Another geo routing scheme is DREAM and was proposed in [45]. It uses the distance between nodes to indicate how quick neighbours should be updated with routing information. With a small distance, the update rate is high compared to far away nodes that get fewer updates. DREAM is also aware of the velocity of nodes, moving nodes update their location information more often.



(a) Example for CTR ETX values for nodes. Darkest node at (0,0) is the central sink. Edges are not weighted



(b) Location Aware Routing with a request box covering space between source and destination, and an expected region where the destination is expected to be. [31]

Figure 8: Routing candidates visualisation of the main principle

A totally different way of routing is flooded based routing [46]. With this type of routing, nodes forward their messages to all neighbours. In this way, a message is flooded through a network. In many cases, flooding is bounded in some way. Sometimes a node keeps a list of recently passed packets and stops messages of being forwarded if it already forwarded that packet before. Another technique to bound flooding is a time to live (TTL) counter registered in a packet [47, 48]. The TTL often indicates the number of hops a packet is allowed to make before dying out.

3.4 Commercial Mesh Network Solutions

Before implementing an academic design, or designing a routing layer from scratch, the possibility of integrating external party mesh technology is explored. See for more details the research papers report [15] or Appendix I about off-the-shelf solutions. The pros and cons of every solution are summarized in Table 4. Unfortunately, the most promising technologies are closed platforms and implementing a closed third-party stack is sub-optimal in some cases. Considering Undagrid specifically, the ability to have sufficient control over all layers is essential. Since Undagrid's communication stack, localisation and backend are in mature stage, it is advised to deploy an in-house mesh technology rather than depending on an external party.

Table 4: Comparison of ready for market mesh solutions

	Advantages	**Disadvantages**
BLE mesh [49, 50, 51]	BLE is very mature Integrated into many devices Undagrid indoor localisation uses BLE	BLE mesh is not mature Uses Flooding
Wirepas [52]	Supports Many Frequency bands Advanced routing mechanism Energy Efficient	Closed Platform
DASH7 [53, 54]	Multiple bands in sub 1GHz frequency range Energy Efficient Open Standard	Closed Platform Main focus on active RFID Max 2 hops
Serval [55]	Open Source	Only Wifi Focused on phone calls
Firechat [50]	Big base of users Feature rich Opportunistic routing	Closed source Main focus on cell phones, however TackR works
Filament [56]	Focus on distributed systems Big additional feature set, e.g. blockchain and encryption	Not lightweight enough data structure (JSON) No concrete products on market jet
DigiMesh [57]	Zigbee is a highly adopted standard	Closed platform
Thread [58]	Uses 802.15.4 standard	6LoWPAN not optimized enough Maximum of 250 nodes
FruityMesh [59]	Open source	No Support of external party Missing encryption

3.5 Routing Candidate Selection

Since it is not in the benefit of Undagrid to use external mesh solutions, a routing algorithm has to be designed. Because the direction of information flow is from node to one or multiple sinks, and not necessary node to node, a hierarchical routing structure might be beneficial. In addition to that, sensors in a localisation sensor network generally know their location. This information can be used for routing as well. With that, location aware routing protocols becomes interesting for a nomadic location sensor network.

From the routing algorithms presented in Section 3.3, two algorithms are selected for further investigation. Since ETX routing tries to reduce routing overhead by only sharing one metric and not full link states, this routing algorithm is selected for further investigation. Also, since location information is available at every sensor, LAR is selected as a routing candidate.

LAR and ETX routing are from different routing categories. The former is a resource aware routing algorithm,while the latter is a hierarchical routing algorithm. These two algorithms will be compared in a empirical comparison based on several metrics.

The first metric is the percentage overhead, which is the amount of information that is shared for routing purposes compared to the amount of data that has to be routed. This is shown in Equation 5, where $P_{overhead}$ is the percentage of overhead, $D_{overhead}$ the amount of data that is transmitted because of overhead and D_{data} the amount of data of packets that should be transmitted [60].

$$P_{overhead} = \frac{D_{overhead}}{D_{overhead} + D_{data}} \cdot 100 \tag{5}$$

Proactive information sharing performs well under relative high network load, and reactive well under low loads. This can be seen from Equation 5. If $D_{overhead} = 0$, the percentage of overhead is also 0. Proactive routing algorithms share information upfront and keep this list for long periods of time. Therefore, for proactive routing algorithms, if $D_{data} \gg D_{overhead}$ then $P_{overhead} \approx 0$. For reactive routing algorithms, where in many cases routing information is requested upon data sending requests, $D_{overhead}$ scales with D_{data}.

In every network, links break and new links form. Robustness against these network changes are network critical. Since ETX routing does not depend on links, but relies on local forwarding, sudden network changes are less when compared to LAR. Also, LAR relies on RouteRequests, which introduces additional overhead. The mobility of a nomadic sensor network changes relatively often, therefore ETX has a better fit to the sensor mobility model than LAR. LAR relies on calculations with GPS coordinates, which in general are computation intensive. Geohashes can reduce this impact, but it will always be more computational intensive than computing an ETX level.

A comparison of the two routing schemes is presented in Table 5. Both LAR and ETX are given a score on multiple metrics as discussed before.

Having investigated the advantages and disadvantages of ETX and LAR in [15], it is expected that ETX routing will perform better in a nomadic location sensor network. LAR has the advantage of having less proactive route information sharing, but the disadvantage of having too much overhead in higher load situations and the thread of having dead ends upon a network change. ETX routing only shares one value, the ETX parameter. The disadvantage of ETX routing is that links from gateway to node can be harder to establish. However, it is expected that this is not a problem in this localisation sensor network.

3.6 Conclusion

This section can be summarized with four conclusions. The first conclusion that can be drawn is that Undagrid's nodes do behave nomadic. They change position approximately 6 times per day and travel 213 meters per day on average. The second conclusion is that a mesh network would create between 96.96% and 99.16% coverage with average cluster sizes between 50.88 and 132.55 nodes per cluster. This indicates

Table 5: Candidate routing schemes compared. Score 0 is a bad score and 5 is a good score, all metrics weight equal.

Metric	**CTP**	**LAR**
Routing paradigm	proactive	reactive
Overhead by low load	2	4
Overhead by high load	4	1
Robustness against link changes	3	1
Fit to ideal mobility situation	4	2
Ease of implementation	4	3
Required memory	3	4
Processing Impact	5	3
Total	**25**	**18**

that a mesh network would be feasible to implement. The third conclusion is that, using the requirements of Section 3.1, there is no off-the-shelf solution that Undagrid can implement and therefore has to develop its own mesh implementation. There is no solution that is both open, efficient and flexible enough to suit Undagrid's needs. The fourth conclusion that can be drawn is that ETX routing is the most promising routing algorithm that could be used for Undagrid's mesh implementation.

4 Design of the Routing Layer

This chapter will cover the design of the routing layer. It will explain how a node gets information about neighbours, what information will be shared and how ETX routing will use this to determine the next hop neighbour. An example network and an overview of the impact of the routing layer on the embedded devices will be given. In the first section, additional requirements are listed. Then, in Section 4.2, the design of the ETX routing layer is explained. Section 4.6 a preliminary design verification is done using a Python Emulator.

4.1 Requirement Analysis

Following the analysis from Section 3.5, ETX routing is the most promising routing algorithm to use as a basis. To design a routing layer based on ETX routing, additional requirements have to be set. These requirements are design criteria for a routing protocol.

Since the ETX routing layer will be deployed in a Wireless Sensor Network (WSN) with constraints on computation power, energy consumption and storage, ETX routing should follow constraints given by this environment. Using the research context description from Section 1.1, requirements for an ad hoc nomadic localisation sensor network can be listed. Typically, data flows from node to gateway. This characteristic can be exploited for an efficient routing layer. Also, eventually, the to be designed routing layer should run on an embedded device. This puts constraints on resources like battery, CPU and RAM. Using this, the following requirements should be met by the routing layer. In addition to the requirements from Section 3.1, the following requirements are specified to meet for the routing design.

- *Packet buffer:* Isolated nodes should buffer their messages and offload them when connected to a cluster. From section 3 it is known that 14 % to 22% of all the nodes leave a cluster for some time.
- *Scalable:* The routing protocol should be able to handle cluster sizes of over a thousand nodes. As shown in the research topics results in Section 3, the total amount of nodes on one site is currently no more than 1500 nodes. With an average cluster size between approximately 50 and 150, the number of clusters on one site is around 10 to 30. Therefore, having support for 1000 per cluster should suffice.

4.2 Routing Design

This section will cover the design of components of the ETX routing scheme. It will cover how beacon messages work and how they spread routing information. After this, the ETX routing basics are explained, as well as additional parameters that can improve ETX routing.

4.2.1 Beaconing and ETX routing

A node receives neighbour information via beacon packets. These packets are sent by each node in a regular interval, typically between approximately 1 and 5 seconds. A beacon packet contains all the information a node needs to know about its neighbours. Using these packets, a node can build up a table with information about all the surrounding neighbours. This table serves as the basis of the routing layer. What information is shared and stored is up to the routing layer. The basis is a unique sensor ID. This ID is for every node unique. Assuming that there will be no more than 4 billion sensors deployed in the Undagrid network, 32 bit addressing should be sufficient.

The main metric that will be shared is ETX, but more metrics will help optimizing packet routes. As explained, ETX is the number of hops left until a gateway is reached. It follows two basic rules, depicted in Equation 3 and Equation 4. Here the root is the basis of the routing hierarchy and therefore a gateway. Following from this, the ETX of a node is calculated by getting the lowest ETX value from a node's neighbours and adding the ETX link value to this. ETX_{link} is in this case always 1. As stated in the requirements, over a thousand nodes per cluster should be supported. Therefore, the resolution of ETX is determined on 8 bits, resulting in 256 different levels of ETX values. In the case of a large line-shaped network with only one node per ETX level, assuming a 100-meter range per node, the length of this network can be a maximum of 25.6 km. The chance of finding a gateway along this line is considered higher than the probability of having such a network.

$$ETX_{root} = 0 \tag{3 revisited}$$

$$ETX_{node} = ETX_{parent} + ETX_{link} \tag{4 revisited}$$

4.2.2 Neighbour Selection Criteria

For nodes that have direct contact with a gateway (ETX = 1), forwarding is easy; simply forward to the gateway. For nodes at a higher level, i.e. ETX level 2, data has to be forwarded to nodes with ETX value 1. However, there is a possibility that a node has multiple level 1 neighbours. Which neighbour to forward to is not evident. This means that node selection criteria have to be added. Therefore, next to ETX values, other metrics are shared between nodes. First of all, there should be a parameter that indicates link quality between nodes. The worse the link, the higher the chance of a failed transmission, wasting energy. Secondly, a parameter should indicate how well a neighbour is connected to other nodes at lower levels. A higher connectivity increases the chance of a successful delivery to a gateway. There should also be a parameter that helps with load balancing. Nodes that have been loaded heavily should be avoided to save battery and share load equally.

The first added metric is PathFactor(PF) which is calculated by determining the number of unique paths to the nearest gateway. The better connected a neighbour is, the higher the chance of a successful packet delivery at the gateway. Paths to neighbours with the same ETX level or higher level are ignored. In general, the PF for a node can be calculated by taking the sum over all PF values from nodes that have a lower ETX value. Let P be the set of PF values from neighbours with a lower ETX value and n be the number of elements in this set, then the PF of a node can be calculated according to:

$$PF_{node} = \sum_{i=1}^{n} P_i \tag{6}$$

A gateway has a PathFactor of 1, since there is only one way to the gateway. All nodes at ETX level 1 and higher follow Equation 6. Consider the following example. A node at ETX level 1 has a PF of 1 as well, since the gateway is the only neighbour in set P. The resolution of PF is set to 16 bits. Consider a network with an average connectivity of 10 nodes, which is an upper limit if 100-meter range is assumed and Table 3 is considered. Assuming that approximately a quarter of the nodes have a lower ETX value, the number of neighbours per node is 3. Then, assuming a 16 bit resolution, the number of supported ETX levels before PF gets an overflow, can be calculated via Equation 7.

$$2^{16} = 3^{N} \tag{7}$$

Which evaluates to $N \approx 10$. This is for now sufficient, but might be increased if larger networks are considered. Seen from another perspective, if a node has 65536 ways to reach a gateway, being able to count even more paths becomes irrelevant for the purpose of packet delivering.

Next to PathFactor, the total amount of transmitted packets is shared such that load balancing becomes feasible. By recording the total amount of transmitted data packets, the contribution of a node to a network is monitored. Using a 32 bit resolution for this, a node can transmit a message every second for 136 years before it has an integer overflow.

The last metric that will be recorded, is the link quality to a neighbour in terms of received signal strength (RSSI). Link quality indicators are relevant to neighbour selection since a bad link might result in packet loss which is a waste of resources. During neighbour discovery, neighbours may be at the boundary of being in range. In these situations, small beacon packets can be received correctly. However, larger data packets can become corrupt since there are more bits to go wrong. This can be deduced from Equation 8, where N is the number of bits and BER is the bit error rate. By measuring the RSSI of beacons, the probability of packet corruption can be estimated via an appropriate error model. This will be presented later. It is assumed that all nodes transmit with equal power. Therefore it is unnecessary to include transmit power in beacon

$$PER = 1 - (1 - BER)^N \tag{8}$$

How these metrics are weighted and used to calculate a score, will be determined later. The information contained in a beacon packet is shown in Figure 9.

32 bit Address	8 bit ETX	32 bit txCount	16 Bit PathFactor

Figure 9: Beacon Payload for routing information, total size is 11 Bytes or 88 bits

4.2.3 Routing Loops and Packet Duplication

Packet loops and packet duplication can occur in WSNs and is a waste of energy. Packet loops in an ETX routing network can only occur if a network is non-static and routing ETX levels are updated by link changes. This can happen, for example, if a link between nodes break. Packet duplication can happen if a message is sent in one direction, a link breaks, and a message has to return via the same path towards another path. In this situation a network is non-stationary. However, packet duplication can also happen if a network is static. In a busy network, acknowledgement packet can collide with other packets. This is generally not detected. When this happens, the node that initialized the packet can select another node to forward to since it did not receive an acknowledgement in time. Assume that the transmission to a second neighbour happens successfully. However, the node that sent out an ACK initially, did receive the packet correctly and will forward this packet to a gateway. At that moment, two nodes have the same packet and will both try to send this to a gateway. If these packet duplications happen to often, the network becomes congested. To prevent this, a node will record the last messages it received. This is done by recording the source address and packet count of every data packet a node forwards. This recording will be done using a first-in-first-out (FIFO) buffer.

4.3 ETX Example Network with Directed Graph

To illustrate ETX routing, a small example network will be constructed using graph theory. This network will be modelled as a directed graph with nodes representing sensors and edges representing a node score of the link between nodes. This score is based on routing information a node knows about its neighbours. These parameters are the aforementioned ETX, RSSI, PathFactor and total TX count.

Consider the situation in Figure 10 where circles represent nodes, lines directed edges and the number on the edges the score of a connection. The red number above each node represent the ETX level of each node.

The ETX routing scheme would work in the following way. If a message has to be sent from node 4 to node 0, node 4 will search for the best neighbour to forward to. This is node 3, since the link to node 3 has

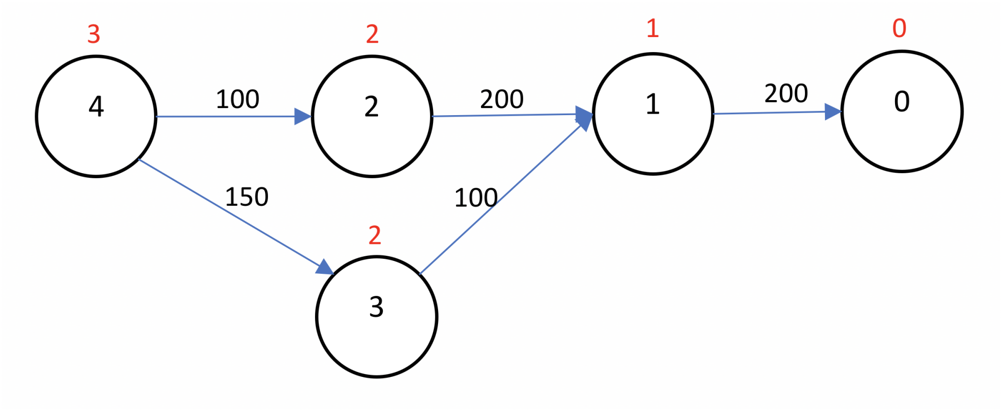


Figure 10: Directed graph with node score.

a higher score than the link to node 2. Node 3 will forward this packet to node 1, which in turn will deliver the packet to node 0. This algorithm searches for local optima, and uses these local optima for determining the final path.

However, for a global optimum, next hop neighbour selection should also be based on indirect neighbours. This problem will be different from local optima finding and can be solved for example by Dijkstra's algorithm. To see how this would work, consider again a packet that has to be sent from node 4 to node 0. In this situation, there are two possible paths: $p1 = [4, 2, 1, 0]$ and $p2 = [4, 3, 1, 0]$. Let $S(n_i, n_j)$ be the score of a single path, then these paths have the following commutative scores:

$$P(p_n) = \sum_{k=1}^{i} S(n_k, n_{k-1}) \tag{9}$$

Which in case of the example of Figure 10 means that

$$\begin{aligned} P(p1) &= S(n_4, n_2) + S(n_2, n_1) + S(n_1, n_0) = 100 + 200 + 200 = 500 \\ P(p2) &= S(n_4, n_3) + S(n_3, n_1) + S(n_1, n_0) = 150 + 100 + 200 = 450 \end{aligned} \tag{10}$$

From this, one can conclude that p1 is the better path. However, this conclusion can only be drawn if knowledge about every path is available at every node. Making routing decisions like that can be realised in two ways: either information about the whole network is present at every node or propagated node scores indicate total path score. However, nomadic networks change rapidly and information gets outdated quickly. Therefore, sharing only info of only direct neighbour values is simpler and propagated path scores might be investigated later on. For now, nodes will forward packets in ETX direction and use other parameters to deduce the best direct neighbour.

4.4 Neighbour List Construction and Refresh Criteria

An active nomadic ad-hoc sensor network is not static. Therefore, information about neighbours can get outdated and invalid. Therefore, the time a neighbour was recorded last should be saved. If a neighbour has not been detected for a certain amount of time, it should be discarded.

If a node has moved to a new location, all information about neighbours will be removed and a new neighbour list should be constructed.

4.5 Neighbour Selection Scoring System

The neighbour scoring metrics that are presented in Section 4.2.2 have to be combined into one single score to be able to compare neighbours. These metrics have different units, and comparing one parameter with another directly is impossible. Therefore, a scoring function is introduced. This function normalises the parameters and adds weights to these normalised parameters. First, every parameter is normalised to a value between 0 and 255. This is done such that a score fits into an 8 bit unsigned integer. This is useful during embedded systems integration. After normalisation, weights are added indicated by the parameters with a Scaler prefix in Equation 13, Equation 15 and Equation 16. After this, a weighted average is taken. This is done via Equation 11.

$$Score_{neighbour} = \frac{Score_{PathFactor} + Score_{RSSI} + Score_{txCount}}{Score_{MAXRSSI} + Score_{MAXtxCount} + Score_{MAXPathFactor}} \tag{11}$$

In Equation 13, the maximum values of each parameter, can be calculated via Equation 12. How the scaler values will be calculated, will be discussed in a later section.

$$\begin{aligned} Score_{MAXRSSI} &= 255 \cdot Scaler_{RSSI} \\ Score_{MAXtxCount} &= 255 \cdot Scaler_{txCount} \\ Score_{MAXPathFactor} &= 255 \cdot Scaler_{PathFactor} \end{aligned} \tag{12}$$

The PathFactor score is calculated via Equation 13. This formula maps all PathFactor scores to a linear line, where the maximum score is given to a neighbour with the highest PathFactor number. The lowest score is given to the neighbour with the lowest number.

$$Score_{PathFactor} = Scaler_{PathFactor} \cdot 255 \cdot \frac{PathFactor - PathFactorMin}{PathFactorMax - PathFactorMin} \tag{13}$$

The second parameter, the RSSI score, is computed via the relation shown in Equation 15. In scoring RSSI, three different cases are distinguished: the link between nodes is good, bad or a fair. To determine what a good link or a bad link is, RSSI is converted in to a signal to noise ratio (SNR). This is done using Equation 14. Based on SNR, the packet error rate (PER) can be computed via an error model. Since NS3 will be used later, the error model of NS3 is used. NS3 has a build in test that results in the error model [61]. The plot of this model using a 32 byte packet size is shown in Figure 11.

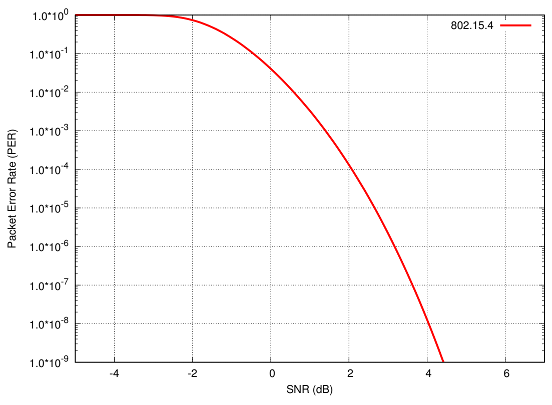
$$SNR = RSSI - RadioSensitivityAndNoise \tag{14}$$

For high RSSI figures, assuming constant noise levels, SNR figures are highs as well. With a high SNR the PER is low. It is assumed that all neighbours below an expected PER of $2 \cdot 10^{-7}$ can establish a reliable connection and get maximum score. This is because the chance of packet collision and fast fading effects have more influence than RSSI itself. A PER of $2 \cdot 10^{-7}$ corresponds to a SNR of 3.5 dB or higher.

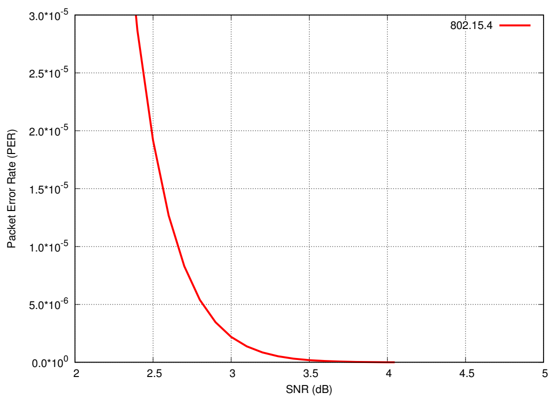
Low RSSI figures indicate a bad link between nodes. Using a neighbour with a low SNR might be the best choice, but there is a significant chance of packet corruption. All neighbours with an expected PER higher than $2 \cdot 10^{-5}$ are therefore scored with the lowest score. This PER corresponds to a SNR of 2.5 dB.

To compute a score of a fair RSSI measurement, a linear relation is used. A logarithmic scale might be more accurate but is harder to calculate on embedded devices. This mapping will run from $x = [2.5; 3.5]$ and $y = [2 \cdot 10^{-5}; 2 \cdot 10^{-7}]$ in Figure 11b to a normalised output of $y = [0; 1]$.

It should also be noted that a good RSSI does not have to be worse than the very best RSSI. A lower RSSI indicates a longer distance to travel, which might mean getting closer to a gateway. This is beneficial for packet delivery. Therefore, stable links with a good RSSI are treated equally compared to very good RSSI.



(a) Packet Error Rate



(b) Packet Error Rate zoomed and linear scale

Figure 11: Packet error rates (PER) for a range of SNR figures from the error model of NS3 LR-WPAN module. A packet size of 32 bytes is used at a frequency of 2.4 GHz and a QPSK modulation scheme

Using the aforementioned three cases, a relation can be constructed. Equation 15 describes this relation. In this formula, ThresholdLL is the lower limit for SNR and ThresholdUL is the upper limit. For the ETX routing layer in NS3 using the LR-WPAN module, these ThresholdUL is 3.5 dB and ThresholdLL is 2.5 dB.

It should be noted that there is only 1 dB difference between a good link and a bad link. This means that the scoreRSSI could end up in a binary relation returning only the best score or the worst possible score.

$$Score_{RSSI}(SNR) = \begin{cases} 0 & \text{if } SNR < ThresholdLL \\ Scaler_{RSSI} \cdot 255 \cdot (SNR - ThresholdLL) & \text{if } SNR < ThresholdUL \\ Scaler_{RSSI} \cdot 255 & \text{else} \end{cases} \tag{15}$$

The formula to calculate the txCount score is depicted in Equation 16. In this formula, txCount is the total amount or transmitted packets of a neighbour, txCountMin the lowest number of txCount of all neighbours and txCountMax the highest number of txCount amongst all neighbours. By doing this, the txCount score is normalized relative to all recorded neighbours. The drawback of this scoring method is that outliers are not filtered out. A very high or very low txCount can influence txScore for all neighbours. This might be solved by using outlier filtering such as inter quartile range filtering. However, to implement this is C and run in on embedded devices is for future work.

$$Score_{txCount} = Scaler_{txCount} \cdot 255 \cdot \frac{txCount - txCountMin}{txCountMax - txCountMin} \tag{16}$$

4.6 Summary of a Preliminary Design Validation in Python Emulator

To test ETX routing before implementing in a simulator, a simplified implementation was built in an emulator. Building, testing and evaluating a routing layer in NS3 will take a significant amount of time in a master thesis project. If ETX routing does not work in an emulator, it will not work in an advanced simulator. The emulator that was used is based on the data analysis framework from Section 3.2. The goals of this preliminary design test is to show that ETX routing works for a nomadic sensor network. How this emulator was designed, details of the ETX routing implementation, results and conclusion can be read in Appendix D.

The data analysis framework, that was build in Python using the NetworkX graph library, can be expanded to send messages across a graph. Using NetworkX, nodes are clustered and placed in a graph. Nodes can use this graph to get information from neighbours and to send data to these neighbours. This is in contrast to a real network that, uses beacon packets to spread routing information. In the Python emulator, routing information is therefore present at every node instantaneously. Or in other words, neighbour discovery is done by a lookup in a graph object.

Data transmission is done using send queues and receive queues. Every node in the NetworkX graph is an Python object that resembles a sensor, including properties and methods that a sensor has. If a node wants to send a message to a neighbour it fetches the node object of a neighbour and moves a message from its send queue into the receive queue of a neighbour.

Since timing properties are not simulated, packet collision and channel interference can't be simulated. However, the Python emulator has the ability to calculate packet error rates (PER). This is done by calculating the signal to noise ratio (SNR) using a log distance path loss model with a path loss exponent of 3.5. For a free space environment, this exponent is 2. A higher value is chosen to approximate a worst case bound on path loss. Using a random draw with PER as probability, packets are indicated as faulty or correct. If a packet is faulty, the whole packet will be dropped, i.e. there is no error correction. If a packet is dropped, the sender will try to send a packet again. This results in a higher and more accurate txCount per node. Packets will eventually arrive the selected neighbour.

The emulator flow is as depicted in Figure 12. The emulator uses a dataset from the Undagrid database. Nodes have a connection if they less than 85 meters apart from each other. Clusters with only 2 nodes ore less are kept as isolated nodes and not placed in a cluster. Also, gateways are defined such that clusters can offload their messages.

After the initialisation phase, messages from the database are fed trough the network as if there were a true multi hop network implementation. When a packet needs to be send, it is assumed that there was a change in the network. Therefore, the first thing that the emulator does is updating routing information for all nodes. This means that ETX values are computed from the gateway outwards, until all nodes have a stable ETX value. Routing information is gathered from the NetworkX graph and is present at a node without any delay. A change at one node forces the whole network to update routing information until routing information is stable. Since timing can not be modelled anyway, and the computational impact is minimal, this relative inefficient way of updating routing information is acceptable.

If all nodes have up-to-date routing information, a message can be send. A nodes that wants to transmit a message to a gateway finds its target neighbours via its neighbour list and selects the highest scoring node. After this, the node sends the packet to the designated destination neighbour. Packet exchange between nodes is done by means of moving a packet from a send queue at a node side to a receive queue at the neighbour side. This is a plain copy action, i.e. MAC and PHY effects are not modelled. Or in other words; packet collisions are not emulated, and radios are assumed to be always on and able to receive and transmit at the same time. The next step is that for all nodes the receive queue is processed. This means that all nodes are checked in a random order if their receive queue contains a message. If so, it is pulled from this queue and processed. A packet will only be forwarded if it hasn't been forwarded by a node before, as described in Section 4.2.3. Processing of packets is done until all receive queues and send queues are empty. If all queues are empty, the next message from the database is initiated and send trough the network. This process is continued until all messages in the data set are processed. At that point, the simulation stops.

4.6.1 Measurement Procedure

The Python Emulator is now ready to run packets through a network and deliver them to gateways. The list of data packets is fetched from the Undagrid database. The simulator will perform 5 different simulation runs, each with different simulator settings. Each run will show a certain characteristic of ETX routing or

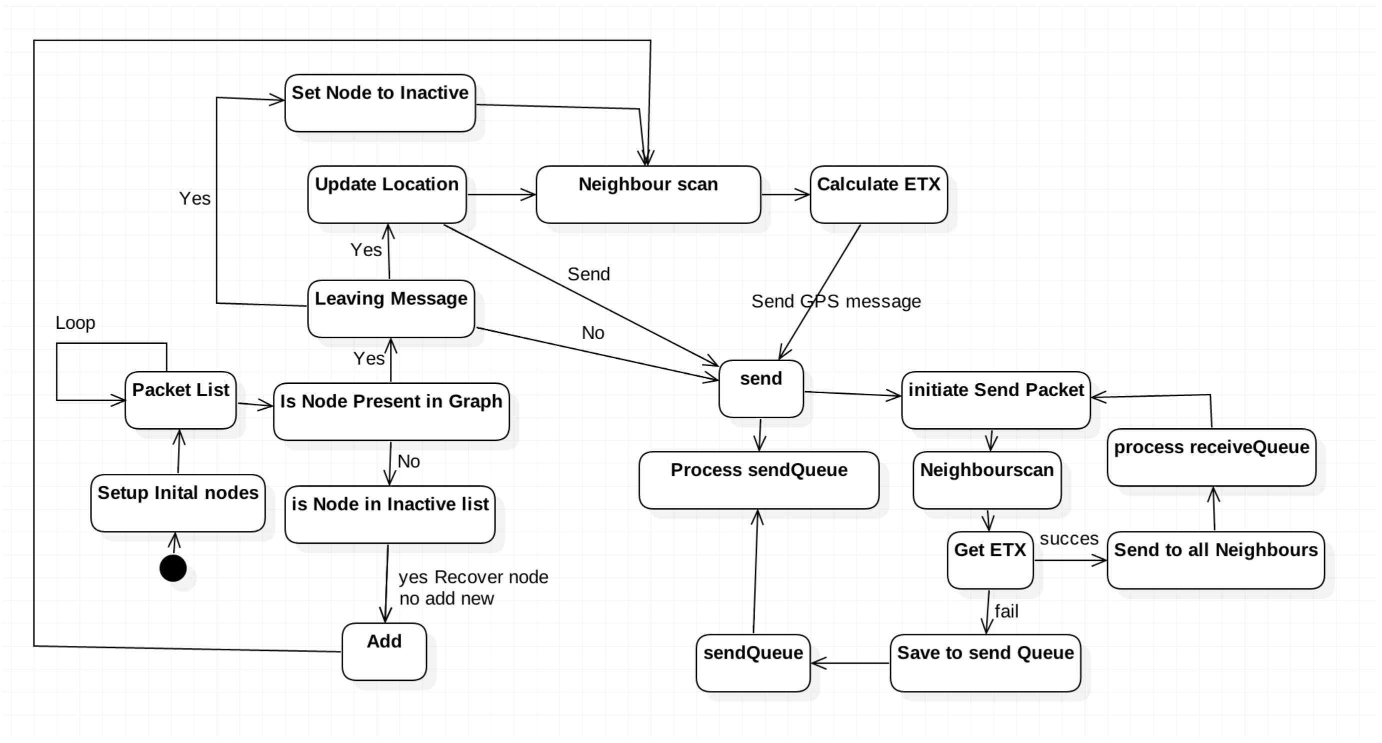


Figure 12: Information flow through the Undagrid Emulator

will help verifying ETX routing design choices. The necessity of PathFactor parameter will be tested, just as the possibility to send packets via a multitude of routes instead of one.

The first run is a bounded flooding test run. In this test, a packet that is received by a node is forwarded to all neighbours with a lower ETX count. Packets will be filtered on packet duplication. In this run, neighbour scoring is disabled, since a packet is sent to all neighbours. Pathloss computation is also disabled in order to get an idea of an upper-bound on total packet transmission count.

In the second run, packets are forwarded to just one neighbour using Equation 29. However, PathFactor will be disabled. This test will be compared with test number 4 to test the influence of PathFactor scoring. The influence of RSSI and txCount on routing performance will not be tested. This is done because PathFactor is a propagated scoring metric and might be unstable. Whether or not the PathFactor helps with routing can be investigated in this test.

Then, in the third run, the same settings as run 2 are used but the number of neighbours a node selects is increased from 1 to 2. This means that not the two best nodes according to Equation 29 are selected. This test will show weather or not redundant paths should be used for packet forwarding.

Test 4 focuses on the influence of PathFactor. As said before, this test will be compared to test 2. It might be that this factor is of no importance to finding an optimum route.

The last test, number 5, will test the influence of the RSSI metric by disabling the packet error rate (PER) calculation and assuming that every packet will always be delivered to the receiving neighbour.

For each run, 49999 packets were processed by the simulator, initiated by 726 nodes. However, some messages can not be processed due to wrong GPS fixes, timing issues or other anomalies. All messages from a node are deleted if an anomaly is detected. These failed messages add up to a total number of 5202. This results in a total number of initiated packets of 44797.

Table 6: ETX routing results from Python Emulator for different runs described in Section 4.6.1

Test Number	**1**	**2**	**3**	**4**	**5**
Description	Flooding	Base test	Dual Path	PF On	PER
Number of Paths	many	1	2	1	1
PathFactor Enabled	No	No	No	Yes	No
PER Enabled	No	Yes	Yes	Yes	No
Successfully Initiated Messages	44797	44797	44797	44797	44797
Message In Queue Left At Simulation end	1407	1407	1407	1407	1407
Total Transmitted Initial Packets	43390	43390	43390	43390	43390
Messages Received By Gateway	42815	43303	43237	43290	43312
Messages Lost	575	87	153	100	78
mean txCount	391	107.5	152	107	108
mean rxCount	3037	52	89.5	52	52
mean txSelf	55	55	55	55	55
mean txCount/txSelf	6.17	1.84	2.51	1.82	1.78
max txCount/txSelf	1288	288	707	295	234

4.6.2 Results

The result of the runs are presented in Table 6 and Figure 36 in Appendix D. Figure 36 shows the spread of several metrics from Table 6. In this table, txCount and rxCount are given. txSelf indicates the number of messages a node initiated, i.e. this node is the source node of the message. Then, the txCount/txSelf indicate how many packets a node had to forward compared to the number it initiated.

From Table 6, It can be seen that run one, where bounded flooding is emulated, performs the worst of all runs. It scores the worst on every metric. On average, the txCount of a node is between 2.6 and 3.7 times as high as other runs. The amount of messages lost, is between 3.8 and 7.4.

Run two is the baseline for the emulation runs. Comparing it to run three, it can be noted that mean txCount is 1.4 times as high. The number of messages lost is higher as well. It should also be noted that the maximum txCount/tsSelf is 2.5 times as high.

Comparing run two to run four show little difference. Run four performs little worse on messages lost, txCount, and txCount/txSelf.

When run two and five are put side by side, it can be seen that the number of lost messages drops, as well as the maximum txCount/txSelf .

4.6.3 Conclusion

The results from this study show that the average load on a node is acceptable. This is expressed in txCount/txSelf, and it represents the number of packets a node initiated relative to the number of packets it had to forward. The emulator results show a txCount/txSelf figure between 1.78 and 1.84. With this, average node load is not much higher than a star topology network, where this figure is equal to 1. However, this is on average. The txCount/txSelf spread indicate that certain nodes act as critical nodes. Some nodes have a txCount/txSelf of almost 300. It is expected that this figure will balance out when sensors are repositioned.

In the Python Emulator, the effect of RSSI and PathFactor scoring is tested by either enabling or disabling these parameters. Both parameters show to have an influence on routing, but further investigation is needed to test the significance of these parameters. Also, the ETX routing implementation in the emulator uses equal weights for all three parameters. Parameters are weighted full or are completely ignored in neighbour scoring. This does not have to result in an optimum routing setting. Therefore these weights have to be non-binary. These weights cannot be determined analytical and should therefore be calculated via other ways. How this can be achieved, will be explored later.

Due to the lack of timing properties in the Python emulator, route information retardation cannot be modelled. If a node has outdated route information, it might send packets to the wrong neighbour. Also, delay in updating can result in oscillations in update ETX levels. This should be included in an implementation where timing is modelled.

All things considered, the ETX routing design has been proven to work correctly in a routing emulator. To test routing algorithm stability and agility, a network simulator that is capable of simulating time influences should be used.

4.7 Routing Layer Summary

Summarizing the routing layer, ETX routing with neighbour scoring receives periodic beacons of neighbours which contain routing information. These messages contain the ETX, PathFactor and txCount values of a neighbour. By measuring the RSSI of a beacon packet, the link quality can be estimated.

With a beacon packet size of 11 bytes, the requirement of a maximum beacon size of 32 bytes is met. Also, the scalability requirement is met, however, the bit space for PathFactor might need to be increased in the future. The routing layer is fully distributed and with only little information, packets should correctly be forwarded to a gateway. Timing and response times are dependant on beacon interval and MAC layer. Therefore, these characteristics can not be described yet.

ETX routing is not compared to other routing layers. This is done because of time considerations. Also, based on earlier research, ETX routing should be more efficient than, for example, LAR.

5 NS3 Simulation Set-up

In previous sections, the working principles of ETX routing, neighbour selection and node balancing are explained. Previous tests have shown correct behaviour for this routing design. Since implementing and testing a routing layer in NS3 can cost a significant amount of time in a master thesis project, these tests were necessary to make sure this would not be a waste of time.

However, with positive preliminary results, a NS3 implementation is the next step in verifying the ETX routing design. A similar approach as Section 4.6 will be taken. First, the assumptions in NS3 simulation will be presented. This will cover simplifications and used models to describe real-life behaviour. After that, the simulation environment will be explained. Since energy critical wireless sensor networks might not be a trivial simulation mode in NS3, supported physical and MAC layers should be investigated.

It is outside of the scope of this research to validate physical layers in NS3. However, it is important to have the physical layer set-up correctly such that it will simulate the situation considered in this research. The packet error rate over distance is the most important result of tuning the NS3 physical layer.

It is also out of the scope of this research to design a full-fledged MAC layer. The reason for having a need for a MAC layer is the ETX routing layer that has to be tested. However, as will be explained later, there is no suitable MAC layer available in NS3 for the scenario at hand. Therefore, larger changes have to be made to the MAC layer to complete the measurement set-up for the ETX Routing layer. The performance of the routing layer will be discussed in the following section.

In Section 5.1, the basics of the NS3 simulator are presented. Section 5.2 is devoted to explaining the characteristics of the physical layer being used in the NS3 simulator. The next section, Section 5.3, the MAC layer with its improvements is presented. Then, Section 5.4 covers the general information flow in the NS3 simulator.

5.1 NS3; a Discrete Time Event Simulator

NS3 is a network simulator written in C/C++ and is open source. It is next to Omnet++ a tool that is often used to simulate networks. The advantage of programming in C/C++ in NS3 is that code developed in NS3 could run on embedded devices. To achieve this, the programming language of the embedded devices should match, proper abstraction layers should be defined and dependencies on software libraries that will not run on embedded devices should be avoided. Especially for larger networks where testing with test-beds becomes impractical, simulating software for embedded devices in one environment makes testing and debugging easier.

NS3 is a so-called discrete time event simulator (DTES). This means that every event is happening at a defined discrete time. A DTES has an event queue that is emptied in a chronological order. A DTES starts with the first event and continues until the event queue is empty. Every event that is executed might add a new event to the event queue. NS3 includes modules that help to simulate different types of networks. Examples of supported networks are Wifi, LTE, LR-WPAN, 802.15.4 and wired networks. IP stacks are supported as well. In general, NS3 follows the OSI stack abstraction layers. This results in being able to easily compare different types of layers and interchange them with the least effort. To validate ETX routing, a physical layer and MAC layer of NS3 should be selected and adjusted to simulate behaviour that matches the uses case of this research.

5.2 Physical layer modelling in NS3

The lowest layer that will be used in NS3 is the physical layer. There are several physical layer models available in NS3. To match physical layer characteristics to the requirements stated in Section 3, an evaluation of available physical layers is done. The best fit will be chosen and tuned to model real-world behaviour as close as possible. Since testing physical layers is outside the scope of this research, the goal of the evaluation is only to choose the right physical layer for routing layer testing.

5.2.1 Requirements

To select the right physical layer, selection criteria have to be defined. This will be done in this section.

As stated in the requirements in Section 3, the frequency that should be used has to be 2.4 GHz. This is to assure world wide compatibility and unlicensed usage. Next to that, the modulation that is used on that frequency should be as energy efficient as possible. For example, phase continuous modulation saves both energy and reduces spectral noise. Also, the modulation and frequency that are going to be used have to fit on real-life hardware. Current sensors of Undagrid use a Texas Instruments CC2650 radio for 2.4 GHz communication and SoC processor. The to be released sensor will have a Semtech SX1280 2.4 GHz radio and a STM32L443xC SoC. These radios support multiple modulation schemes. One of which is BLE. The newer radio, the SX1280, also supports FLRC modulation.

The goal of investigating the available modules and select the module that fits best to with research case of simulating an energy critical nomadic localisation sensor network. The purpose of this layer is to support MAC layer functionality. This means, that detailed behaviour is not important. The two important metrics of the physical layer are achieved communication distance, medium occupancy and the fit to an energy critical constraints.

5.2.2 Model selection and Parameter Settings

Models that fit the aforementioned criteria are presented in Table 7. In this table the module for LR-WPAN, WiFi and WiMax are compared. The scores in this table have equal weights to add up to a total score. Scores are given on how well a criterium matches requirements [62].

Table 7: Available NS3 physical layers and their fit to an energy efficient network [62].

Scoring	**LR-WPAN**	**WiFi**	**WiMax**
Supported Frequencies	868 MHz - 2.4 GHz	5-10 MHz, 2.4-5 GHz	2- 66 GHz
Modulation	BPSK, ASK, OQPSK	OFDM,DSSS	BPSK, QPSK, QAM
Modulation Score	4	2	3
Matching MAC layer	5	2	3
Overall Energy Efficiency	4	2	3
Total Score	**13**	**6**	**9**

From Table 7 it can be deduced that the LR-WPAN modules scores the highest, and is therefore the best matching NS3 module to use. Compared to LR-WPAN, WiFi is simply not energy efficient enough. The modulation schemes are not suited for low energy transmissions and the accompanied MAC layer is not focussed on saving energy. WiMax scores a little better but is still behind LR-WPAN on every point.

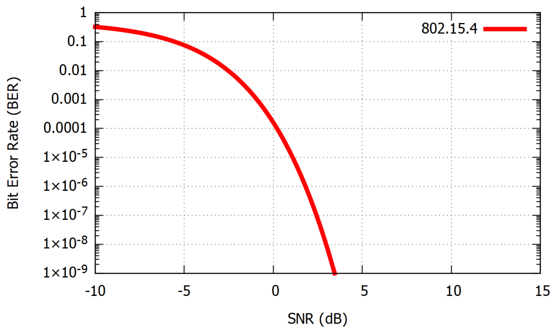
LR-WPAN, which stands for low-rate wireless personal area network, is based on the IEEE 802.15.4 standard. IEEE 802.15.4 is used in several wireless sensor networks such as Zigbee, 6LoWPAN, Thread and Wireless HART [63].

The properties of the model will be changed according to the hardware that will be used. The current hardware uses LoRa and Bluetooth Low Energy (BLE). These protocols do not use a modulation algorithm that is present in any of the NS3 physical layers. Although different modulation schemes behave differently, the most relevant metric is packet collision, the achievable transmission distance and transmission time. Therefore, using transmission power and receiver sensitivity, behaviour is matched to measurement data from Undagrid presented in Appendix B. From this data it is concluded that 150 meter is the maximum range where all packets are still received. However, this in line of sight environment but also with old hardware. It is expected that with tracker v3, 150 meter should also be feasible with non-LOS.

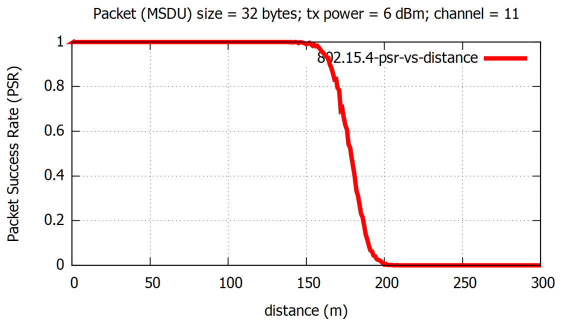
The LR-WPAN physical layer will be used with the following settings:

- *Modulation*: Orthogonal Quadrature Phase Shift Keying. OQPSK is the only supported modulation type on 2.4 GHz. It has double the amout of bits per symbol compared to FSK, which is used in BLE, and to Lora [64].
- *Base frequency*: 2.4 GHz (default)
- *Sensitivity*: -106.58 dBm (default and fixed)
- *Transmission power*: +6 dBm. To achieve the desired range of 150 meters, default output power is increased.
- *Packet Size*: 32 bytes. To simulate BLE beacons, the beacon size is set to the BLE maximum of 32 bytes.
- *Propagation Loss*: Log Distance model (default)
- *Propagation Delay*: Constant delay model (default)
- *Data Symbol Rate*: 250 kbps (default, and feasible for both BLE, LoRa and FLRC).

To verify correct transmission range, a test for LR-WPAN bit error rate (BER) and packet success rate (PSR) is run. The results of these tests are shown in Figure 13. The BER test is a statistical model test that uses radiosensitivity to determine the chance of a faulty bit. For a certain SNR, the physical layer of the LR-WPAN computes the BER. This is shown in Figure 13a. The PSR test uses the radiosensitivity model and propagation loss model. It positions two nodes a defined distance from each other and initiates 1000 packets from one node to the other. The test observes correct delivered packets. The PSR is then calculated by dividing the correctly received packets by 1000. The result of this is shown in Figure 13b.



(a) Bit Error Rate (BER)



(b) Packet Succes Rate.

Figure 13: Results of physical layer testing in NS3.

From these tests it can be observed that the PSR is approximately 1 at 150 meters which matches current sensor range as shown in Appendix B. This, together with the aforementioned physical layer settings, should model a BLE environment well enough to test and verify the ETX routing layer.

5.3 MAC layer modelling in NS3

Now that the physical layer is working and calibrated, the focus is shifted to the MAC layer. There are multiple MAC layers available for NS3. However, not all of them are suitable. Many of the available MAC layers are energy inefficient. The goal of this section is to explore the available NS3 modules and alter one such that it gives insights into delay, the number of packets transmitted and total transmission time.

5.3.1 Requirements

Before comparing different available MAC layers, MAC layer requirements should be known. The first requirement is that a node should discover and join a network as quickly as possible. From Appendix A.1 a total discovery and data packet transmission time is calculated to be approximately 34.64 seconds. Secondly, a MAC layer should be able to cope with a bound on resources. Since the MAC layer has to run on embedded devices, it should work on devices where RAM, CPU cycles and energy is limited. Thirdly, the MAC layer should model delay and timing properly such that the stability of the ETX routing layer can be validated. Lastly, the MAC layer should not depend on a central node that has a certain clock and every node has to synchronise to this clock. Since the network to be designed has to be as decentralised as possible, clock synchronisation will not work. Clock synchronisation in a decentralised manner is possible, but harder to achieve since consecutive synchronisation might become unstable.

5.3.2 Model Selection

One way of saving energy is duty cycle reduction. This means that a radio is switch off for some time and turn on only on designated windows such that nodes will still communicate with each other. Duty cycle reduction can be either static or dynamic. The advantage of dynamic reduction is that it sleeping periods can be shortened if activity on the network is detected and with that delay should be lowered. This however means a more complicated algorithm. Duty cycling reduction can also be characterized according to if network wakes up in synchrony or in asynchrony. In a synchronous network, all nodes wake up a the same time and send their data at the same time. This is sometimes easier to implement but results in inefficient use of the wireless medium.

In [65, 66] different energy efficient MAC layers are compared. Duty cycle reduction is one of the most widely used mechanisms for energy-efficient MAC protocols in sensor networks. Switching off parts of a radio has the drawback of increasing delay, slower neighbour discovery and the chance of missing packets. Adaptive duty cycling is a way to compensate for this. S-MAC is an example of such an adaptive duty cycle MAC layer [65, 66]. T-MAC is an improved version of S-MAC, and it mainly differs in how listening time periods are determined. B-MAC implements low power listening or channel sampling to set-up a link. WiseMac is a popular scheme that outperforms S-MAC in some cases. WiseMac uses CSMA with preamble sampling and thereby shortens the idle listening time of a node. These are all asynchronous static duty cycling mechanisms. Using adaptive duty cycle mechanisms can save more energy. MAC layers that are asynchronous might have too much overhead in finding neighbours if the frequency of neighbour scanning is too high. WiseMAC for instance is easy to implement, but has a higher overhead than S-MAC and T-Mac. Other efficient MAC layers are, amongst others, X-MAC, PW-MAC, AS-PW-MAC and explained in [67, 68, 69, 70]. The default LR-WPAN MAC implementation is unslotted CSMA/CA, similar to Contiki's NullMAC without sleeping features [61].

Pros and cons of these MAC layers relative to the use case of this research are displayed in Table 8. The most important metric is to lower the amount of idle listening time. Therefore it is given a weight of 4. On second place, the need to synchronise a clock between sensors is important. The next important metric, with a weight of two, is the ability to reduce latency. Latency is introduced by disabling radio for periods of time.

However, none of the mentioned MAC layers have an NS3 implementation, or is publicly available. The purpose of the MAC layer in this research is to investigate delay, transmission count and receiving and

Table 8: Overview of MAC layers

	weight	**S-MAC**	**T-MAC**	**WiseMac**	**LR-WPAN NullMac**
Ease of Implementation	1	4	4	4	5
Overhead	1	3	4	3	1
Idle rx time reduction	4	3	3	3	0
Delay reduction	2	2	2	2	5
Robustness	1	3	2	3	4
Need to synchronize	3	1	1	4	5
	total:	**29**	**29**	**38**	**35**

transmission time. An implementation that is suitable for usage on hardware will be developed outside the scope of this research. Therefore, LR-WPAN's default MAC will be altered to mimic the behaviour of an asynchronous MAC layer like WiseMac. The software architeture that this MAC layer has to be implemented on is explained in Appendix F.

5.3.3 Model adaptation and parameters: Asynchronous Radio Wake Up

There are two important constraints on improving the LR-WPAN MAC. The first is the need for time synchronisation. Since nodes are nomadic, two separate clusters can merge and split at any time, time synchronized clusters will not function correctly. This is because the propagation of synchronisation over multiple hops will be slow. Also, in larger clusters, synchronized MAC will result in underuse of the wireless medium. The second constraint is to sleep as much as possible while reducing delay.

To cut idle listening time, the design of this MAC layer, called UndaMAC, will use an asynchronous static wake up schedule. UndaMAC works in phases. To review these phases, let us take two nodes and call them A and B. B is a node that is stationary for a long period. Node A is a new node and just joins the network. Now, the first phase node A comes in, is the discovery phase, where it performs a neighbour discovery procedure for a multiple of beacon periods. If node A discovers a neighbour by receiving a beacon from B, it sends a beacon response back. This is shown in Figure 14a. If A knows all its neighbours, it goes to the running phase. This means that a node wakes up periodically, transmits a short beacon, listens for a preamble of a packet to receive, and goes to sleep again. This is shown in Figure 14b. Node B was in this phase all the time. The last phase is a backup phase, in which a node performs a neighbour discovery to compensate for missed neighbours. This is shown in Figure 14c. In Appendix E a state machine overview of this system is drawn. The implementation can be seen in sourcefile `model/UMNode` in function `void UMNode::changeMacState(uint16_t state)`

Performance of UndaMAC can be evaluated using probability theory and best-case execution time (BCET) and worst-case execution time (WCET) analysis. Assuming a random spread in timing offset X, the expected delay E for an amount of hops N_h is

$$\begin{aligned} E[X] &= \sum_{n=1}^{N_h} \left(\frac{T_{beaconPeriod}}{2} T_{overhead}\right) \\ &= N_h\left(\frac{T_{beaconPeriod}}{2} + T_{overhead}\right) \end{aligned} \tag{17}$$

This corresponds to the expected value of a random variable, which is the mean of an uniform distribution. For a distribution running from 0 to 4, the expected value is therefore 2. Therefore, for multiple hops, the expected delay is 2000 ms plus some radio delay overhead.

To calculate how quickly a message can be transmitted from one node to a gateway, the BCET is calculated. Let $T_{overhead}$ be the time needed to transmit, receive and process a message. Then the BCET is

Table 9: UndaMAC timing parameters

Parameters	**Description**	**time [us]**
TbeaconPeriod	Time of beacon period	4.000.000
Ttx	Time to send a packet	4.000
Trx	Listening Period. Sum of TrxPre, TrxPost and TrxSlot	250.200
TrxPre	Time buffer before packet	100
TrxPost	Time buffer after packet	100
TrxSlot	RX window for waiting of packet	250.000
TrxInit	Time to listen to the medium to discover nodes	2·TbeaconPeriod
TrxRepeat	Time of a repeat period	1·TbeaconPeriod
TrepeatPeriode	Repeat phase period (every 4hours)	14.400.000.000

simply

$$BCET = N_h T_{overhead} \tag{18}$$

Next up is the WCET. Consider three nodes A, B and C. A and B are in range and B and C are in range of each other. A and C do not have a connection. If C initiates a packet and sends it via B to A. If B receives C's packet right after A goes to sleep, B has to wait for the full $T_{BeaconPeriod}$. This is a worst case scenario. For

$$WCET = N_h(T_{overhead} + T_{BeaconPeriod}) \tag{19}$$

Default timing values are depicted in Table 9. These are based on energy estimations done by Undagrid. They can be optimized to save energy if UndaMAC turns out to work as expected. This could be done by i.e. shortening the rx window.

5.3.4 MAC Layer Validation

Three tests are being performed to verify correct behaviour of the UndaMAC layer. The first test will focus on the time a node needs to discover neighbours. The second test will test the node-to-node delay packets have using UndaMAC. Lastly, a test is run to verify that no packets are getting lost using the UndaMAC layer.

Consider a node in a network with multiple clusters. If a node is moved from one cluster to a new cluster with new neighbours, the moved node should detect these neighbours as quickly as possible. Every neighbour is sending out beacons according to a fixed beaconing period, mentioned in Table 9. Assuming a random distribution of node beacon offsets, the expected time for a node to find its first neighbour is between 0 and 4 seconds. This behaviour will be verified in this test. The test set-up contains initially only one gateway. Then, a node is added to the simulator. After 10 seconds, the next neighbour is added behind the first node, such that it can only detect the first node. After another 10 seconds, a second node is introduced that can only connect to the first node. The first node now sees node two and a gateway. Then, after another 10 seconds, a third node is added. This procedure continues until there are 20 nodes. During simulation, the time it takes to detect its first neighbour is registered. This simulation is run 19 times, resulting in 380 measurements.

Table 10: Results of packet loss analysis for UndaMAC

Total number of nodes	63
Total initial messages	103
Total tx count	214
Messages queued	31
because of isolation	31
Messages received by Gateway	72
Messages lost	0

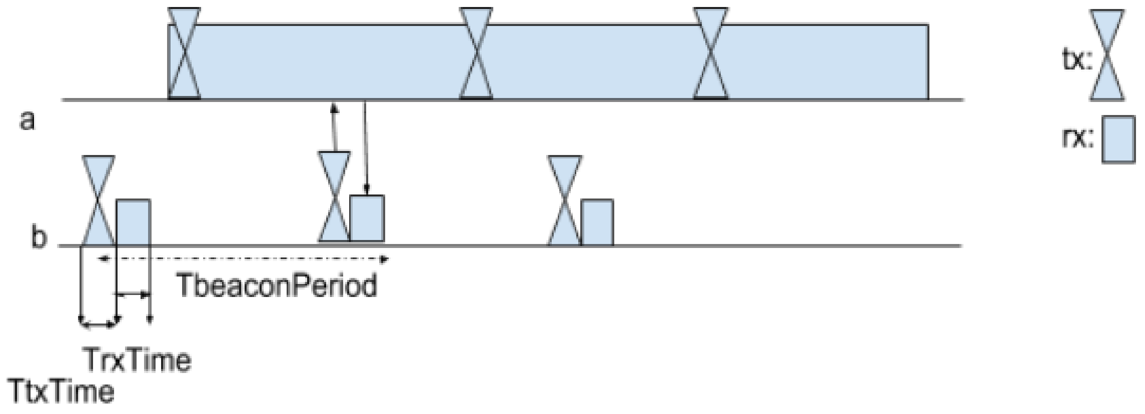
The timing results are shown in Figure 15. Of these 380 delay points, 13 are longer than 4000 ms. 10 of these points originate from the first node after the gateway.

A second test verifies message sending delay. Timing properties of transmitting data packets are described in Figure 14b. According to Equation 17 and using data from Table 9, the expected delay is on average 2000 ms. To test packet delay, a simple ETX routing scheme is implemented. In this set-up, 20 nodes are placed in a line, where every node can only communicate with one ancestor and one predecessor. The simulator is run 25 times with 2 packets per simulation per node, which results in 50 packets per node in total. This adds up to 1000 packets sent in total.

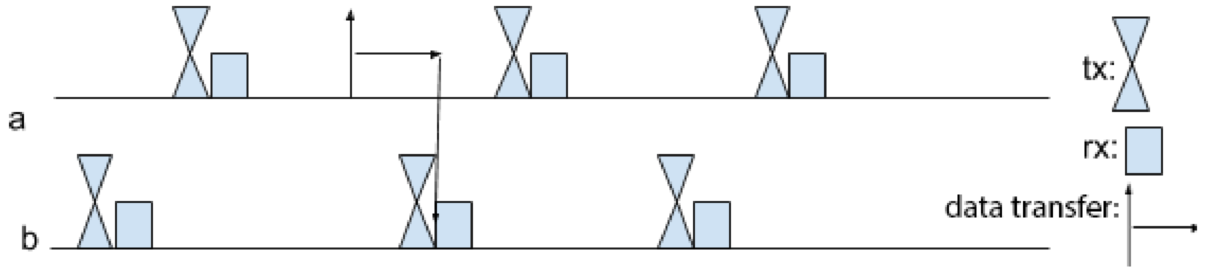
After running this test, the results show an average delay of a node-to-node transmission of 2099 milliseconds. Results of the measurement series is included in Table 20 in Appendix G.

The last test checks if there are any lost packets caused by UndaMAC. A subset of data available from Schiphol will be used for testing packet loss. Based on this data set, a network will be formed and a list of data packets that should transverse this network is constructed. The network consists of multiple clusters. There is one major cluster with a gateway placed in the middle. Next to this cluster, there are smaller isolated clusters to test buffering. The exact distribution of the initial graph is displayed in Figure 39 in Appendix H.

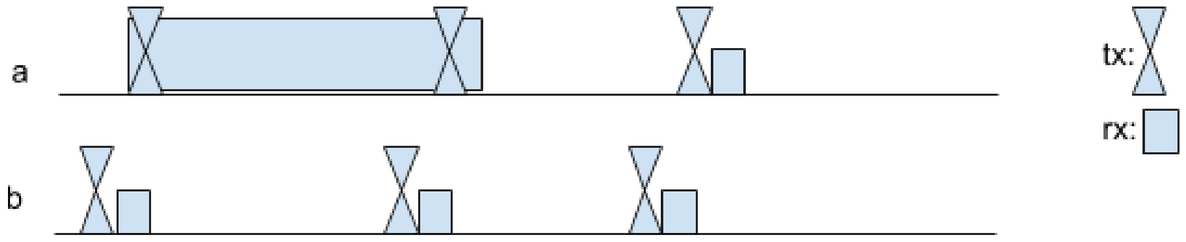
After network initialisation in NS3, the network has 60 seconds to become stable. At this point, the network is ready to transport messages. To help testing, a simple ETX routing scheme is implemented to route packets to a gateway that is at the centre of the main cluster. The list of packets is true to life and originates from the Undagrid database. However, the maximum time between two messages is shortened to 20 seconds. This is done to lower simulation time under the assumption that after 20 seconds, packets will not interfere with each other any more. However, the time between consecutive messages might be shorter than 20 seconds, simulating real network requirements. The results of this test are summarized in Table 10. This MAC layer will be used to test the ETX routing design.



(a) Initialisation phase where node A performs a neighbour scan.



(b) Running phase with a packet that has to be sent from A to B. A has to wait till B wakes up and sends a beacon. After having received the beacon of B, A sends its packet in the rx window of B



(c) Repeat phase

Figure 14: UndaMAC phases

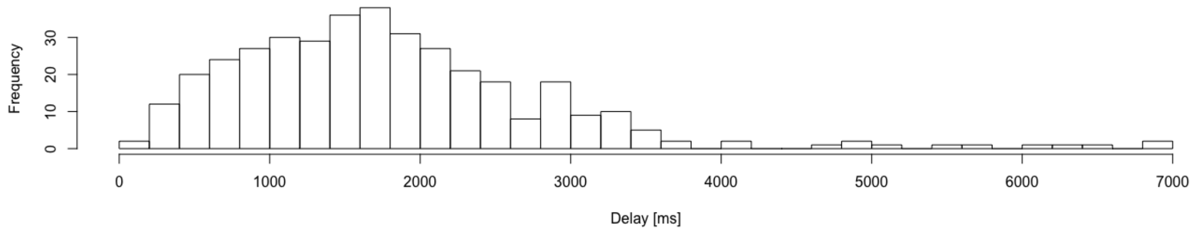


Figure 15: Neighbour discovery delay in milliseconds for 20 nodes

5.4 NS3 Simulator Flow and Data Structures

This section will focus on explaining how the simulator uses the PHY and MAC layer and what the information flow is for a simulation. The software design information flow can be seen in Figure 16. From this figure it can be seen that there are three types of packets that can be received by a node: a beacon packet, a data packet and an acknowledgement packet. In the case of a beacon packet, the list of neighbours should be updated with new information. This also triggers updating routing information. If a received packet is a data packet, packet forwarding will be analysed and suitable neighbours will be selected. If a packet is sent to a neighbour, a node will wait for an acknowledgement packet.

To simulate a node in NS3, node objects are constructed. A node in this simulator run will be called an UMNode and is an inherited class based on the native NS3 Node class. It introduces UndaMesh specific properties to a Node, such as energy models, GPS to Cartesian conversion functions, beaconing support and other helper classes. However, the most important added property is the UMNodeStruct. This C structure holds all parameters that should be globally available in an embedded device.
A neighbour receives a neighbour score, based on three parameters. Every parameter ranges from 0 to a maximum of 255. Selection of neighbours and their parameters is done in the following order.

1. Filter all neighbours that have a lower ETX, and therefore force a packet flow towards a gateway.
2. Calculate the smallest and largest parameter values of txCount and PathFactor for these neighbours.
3. Calculate neighbour scores
4. Select neighbour with the best weighted score

The three individual parameters are weighted. This is done using the Scaler values shown in Equation 13, Equation 15 and Equation 16. What weight belongs to which parameter will be calculated in a later section. For now, it is sufficient to know that each scaler ranges from 0 to maximum of 255. Multiplying two 8 bit unsigned integers results in a 16 bit integer. Then, using Equation 11, a weighted average is calculated. By dividing by an 8 bit unsigned integer, the result is an 8 bit value again.

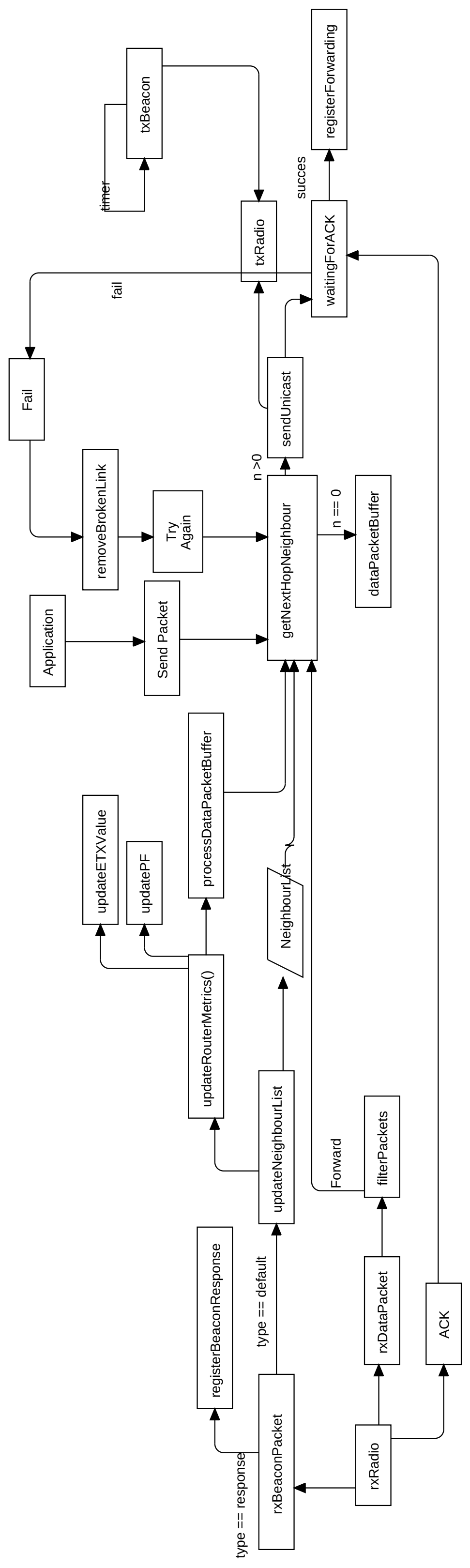


Figure 16: NS3 UndaMesh information Flow

The packet types that are used in this research are structured in the following manner. There are two types of messages that can be send. These are beacon packets and data packets. Beacon packets spread information from nodes to surrounding neighbours. This information indicates when a nodes wakes up and it contains routing information. This beacon packet consists of two parts. The first part is the NS3 native beacon packet, and the second part is the beacon payload. NS3 packets is structured as shown in Figure 17. For beacon packets the sequence number, destination PAN and destination address are not used. Therefore, a beacon packet includes frame control, sequence number, source PAN identifier, source address and frame check sequence. With that, the total size excluding payload of a beacon packet is therefore 13 bytes if an 8 octet source addressing is used.

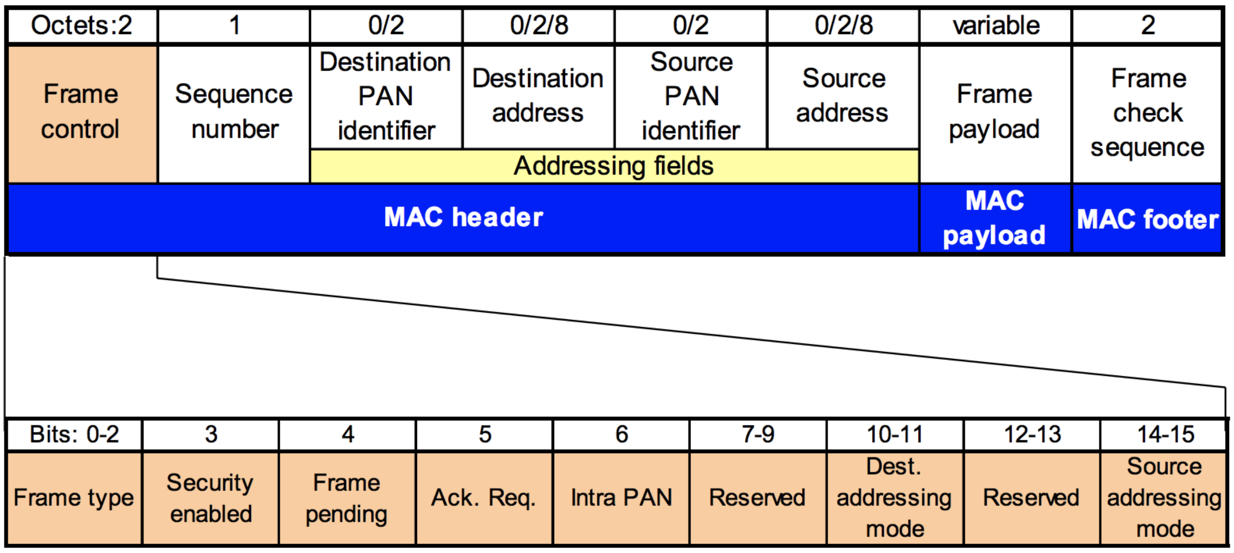


Figure 17: General 802.15.4 Frame structure used by NS3 [71]

Using the requirement of having a maximum of 32 bytes of beacon packet size, the maximum payload in a beacon is 19 bytes. The beacon payload is shown in the listing below. The total size of the beacon payload is 12 bytes. The message type indicates what kind of beacon message is transmitted. From Figure 16 it can be seen that beacon messages are handled differently than beacon response messages. Next, the three routing metrics ETX, txCount and PathFactor are included. Note that RSSI is a measured value and does not have to be included into beacon packets. The last field, named nextWakeUpUs, is the number of micro seconds that a node will wake-up. This field is used for the MAC layer.

```
typedef struct {
        uint8_t messageType;
        uint8_t ETX;
        uint32_t txCount;
        uint16_t pathFactor;
        uint32_t nextWakeUpUs;
} beaconPayloadPacket;
```

For data packets, the packet structure from Figure 17 is used as well. However, in this case PAN ids are set and destination address is included as well. The payload of a NS3 data packet is filled with serialized data listed below.The source ID is the address of the node that initiated the packet. This is therefore not the same as the source address in Figure 17, which is the source address of the forwarding node. The message type contains information what kind of packet it is. The payload is a custom field that can be used for applications. The messageType field prescribes the size of this field. The destination ID field is an optional field. The timestamps is the epoch time stamp at which the message was initiated.

```
typedef struct {
        uint8_t sourceId[8];
        uint8_t messageType;
        uint8_t packetCount;
        char payload[DATA_PACKET_PAYLOAD_BYTE];
        uint8_t destinationId[8];
        uint32_t timestamp;
} beaconPayloadPacket;
```

As stated before, NS3 keeps a node container which holds all Node objects that are generated during a simulation. Each Node has the ability to store data in its data structure. One part of this data is the list of detected neigbhours and contains information about MAC timing as well as routing metrics. This data is used when a node has to send a data packet. When a Node receives a data packet, the node analyses the data packet and decides to forward it. If forwarding is necessary, it will use the neighbour list to determine the best neighbour to forward to. Pseudo C code for this implementation is shown in the code listing below. In this function,p_{um} is a pointer to the UMNodeStruct that keeps all data for one node. This structure contains an attribute called router which holds router settings, such as the amount of recorded neighbours, ETX values, and neighbour information. Then, the function goes through all neighbours recorded and calculates the best neighbour. It filters neighbours that have been inactive, or that have a higher ETX value than itself. Then, the implementation of scoring functions from Section 4.5 is shown on lines 10 till 22. Lines 23 till 26 will keep track of the highest scoring neighbour. Eventually, via line 28, the highest scoring neighbour is returned. If there is no neighbour, the NULL pointer will be returned.

```
1  neighbour * extRouterCalculateNodeScore(UMNodeStruct *p_um){
2  ..
3  extRouterCalculatedMaximumPFTx(..);
4  neighbour * bestNb = NULL;
5  for(int i = 0; i < p_um->neighbourListMaxItems; i++)
6  {
7  if(p_um->neighbourList[i].lastActive >0 ){
8          int16_t deltaETX = p_um->router.ETX - p_um->neighbourList[i].ETX;
9          if(deltaETX >0){
10                 uint16_t RSSIscore =
11                                 p_um->router.scalerRSSI * etxRouterRSSIScore(..);
12                 uint16_t pathFactorScore =
13                                 p_um->router.scalerPathFactor * etxRouterPathFactorScore(..);
14                 uint16_t txCountScore =
15                                 p_um->router.scalerTxCount *etxRouterTxCountScore(..);
16
17                 sumScore = RSSIscore + pathFactorScore + txCountScore;
18                 sumScaler =     p_um->router.scalerRSSI +
19                                                         p_um->router.scalerPathFactor +
20                                                         p_um->router.scalerTxCount;
21                 score = (uint8_t) (sumScore/sumScaler);
22                 p_um->neighbourList[i].nodeScore = score;
23                 if(score >= maxScore){
24                         maxScore = score;
25                         bestNb = &p_um->neighbourList[i];
26 }}}};
27 return bestNb;
28 }
```

5.4.1 Discussion and Conclusion

The physical layer of NS3 is modelled used the LR-WPAN module. This module is intended to simulate IEEE 802.15.4 networks. However, it is also the best fit for simulating ad hoc energy critical nomadic sensor networks. By altering the transmission power of a node, transmission range of a node is matched to the transmission range of a real sensor, see Appendix B. With energy consumption of the radio not simulated, any difference in power consumption because of a different modulation scheme can be ignored. All other parameters of the physical layer are kept at default settings. The path loss model that is used in LR-WPAN is a log-distance model with an exponent value of 3. If this is a valid approximation is not validated. However, a log distance model is simple and not always applicable. Therefore RSSI measurements in NS3 might not match with real world experiments. All things considered, the physical layer matches its requirements and fulfils its purpose. The focus was a model that simulates packet transmission considering transmission range and medium occupancy. Therefore it can be concluded that the physical layer is sufficiently accurate modelled to meet the requirements of this simulation study.

Because there was no MAC layer available that would simulate an asynchronous MAC layer, this layer had to be designed and implemented. The goal of this layer was to test the designed routing layer. One aspect of this is the ability to adapt to network changes in time. Another aspect is the packet delivery delay and packet loss.

Neighbour discovery delay is tested in the first part of Section 5.3.4. Looking at the points between 0ms and 4000ms in Figure 15 the distribution is not completely uniform. This is caused by nodes that are in the MAC initialisation phase while a new neighbour is introduced. These nodes will respond with beacon response messages if a new neighbour is detected and are therefore detected earlier. Discovery after 4000 ms, which is the beacon interval of a node, is caused by missed beacons. Beacons don't have acknowledgement enabled and are therefore not always received by all neighbours at first try. However, in all cases, a first neighbour was discovered within 7000 milliseconds, which lies within the requirement of finding and sending a packet within 34.64 seconds (see Section A.1).

With a near-uniform delay pattern, the expected average delay to send a packet from one node to the other should be 2000 milliseconds plus overhead. Overhead figures of NS3 could not be deduced and are therefore not calculated. For a real-life integration, these timings should be calculated to get a reliable and efficient MAC layer. Using the second MAC test, the average delay per hop is 2099 milliseconds. This result follows autorefeq:macExpectedDelay, which states that the expected delay is 2000 ms plus overhead. However, looking at the raw data in Table 20 in Appendix G, delay per hop is sometimes below 2000 ms. This might be because of a test set that is too small and therefore randomness is not spread. The exact cause of this could not be determined.

Previous tests might be biased if sent packets get lost. Therefore, a test is run to verify correct packet delivery. Results from Table 10 show that there is no packet loss in that test run. However, proving that packet loss is not possible by running tests is not possible. One would need to simulate every possible network with every combination of packets. Guarantees about packet delivery should follow from MAC layer process flow using formal methods. However, since this MAC implementation cannot be ported to Undagrid's embedded software stack, this effort will be saved for a final MAC implementation. Future work might include a test that uses a bigger data set. With this, the accuracy of the claim of not packet loss can be increased.

With the aforementioned test results, it can be concluded that the MAC layer performs according to the set goal for this MAC layer. Delay is introduced as expected, neighbour discovery is within requirements and packet loss is low enough to get reliable results for testing the routing layer. Testing the routing layer is the main subject of the next section.

6 NS3 Simulation: Routing Layer testing

In Section 5, the workings of NS3 is explained, as well as the settings of the simulator and changes made to the LR-WPAN module of NS3. This means that the simulation environment is now ready to use for ETX routing testing. The goal of this section is validating routing behaviour and determining the scaler values of Equation 13, Equation 15 and Equation 16. With that, the neighbour scoring system is completed using Equation 11.

$$Score_{neighbour} = \frac{Score_{PathFactor} + Score_{RSSI} + Score_{txCount}}{Score_{MAXRSSI} + Score_{MAXtxCount} + Score_{MAXPathFactor}} \tag{11 revisited}$$

$$Score_{PathFactor} = Scaler_{PathFactor} \cdot 255 \cdot \frac{PathFactor - PathFactorMin}{PathFactorMax - PathFactorMin} \tag{13 revisited}$$

$$Score_{RSSI}(SNR) = \begin{cases} 0 & \text{if } SNR < ThresholdLL \\ Scaler_{RSSI} \cdot 255 \cdot (SNR - ThresholdLL) & \text{if } SNR < ThresholdUL \\ Scaler_{RSSI} \cdot 255 & \text{else} \end{cases} \tag{15 revisited}$$

$$Score_{txCount} = Scaler_{txCount} \cdot 255 \cdot \frac{txCount - txCountMin}{txCountMax - txCountMin} \tag{16 revisited}$$

To achieve the goal of validating correct routing behaviour and calibrating parameters, five tests are performed. The first test is a simple test where each routing parameter is tested individually. This means that PF, txCount and RSSI are used as sole metrics. It picks one parameter and using three neighbours, with different scores on that parameter, the best neighbour is selected. Then in test 2, neighbour score weighting and averaging is tested using Equation 11. Test 3 is an integration test where a random network is being used to verify routing behaviour and to verify how the NS3 implementation compares to the Python emulator presented in Section 4.6. The fourth test is a large test that characterizes the weights of each parameter in Equation 11. It uses a linear regression study to determine optimal parameter weight settings. Each parameter will have 6 different values, ranging from 0 till 255, resulting in $6^3 = 216$ possible settings. Then, based on transmission count and lost packets an optimal setting is deduced. The last test, test number 5, tests the developed network against the most dynamic situation possible; a drive by gateway. This test requires the network to detect the gateway in a quick manner, connect to it, spread ETX values, offload all packets that are contained in the network and finally also detect that the gateway is out of range and change ETX values accordingly.

6.1 Measurement Procedure

As explained, this section will cover 5 tests. Network performances are measured based on the amount of packets transmitted, the delay of a packet and the number of dropped packets. In order to measure these metrics, every events that occur at nodes is recorded for analysis. NS3 has a build in logging module that records .pcap files that can be inspected using WireShark. However, this module is hard to expand and not suited for overall network performance analysis. Therefore, a separate module is written that is able to write events to a comma separated value (CSV) file on hard drive. Each test will result in four CSV files. The first is a packet tracing file that records all transmitted and received packets. With this, packet delay and packet loss can be calculated. The second file is the neighbour tracing file. This file contains all entries of received beacon packets by all nodes. This gives insights into neighbour discovery times. The third file is the queue tracing file. This file is generated at the end of a simulation run and contains all data packets that were queued for transmission but due to cluster isolation could not be delivered to a gateway. The last file is the node statistics file, which is also generated at the end of a simulation run. Every entry in this file belongs to a node and it gives insights in node statistics. It contains transmission counters, receive counters, last node location, last ETX level, energy levels, routing parameter settings and active time figures.

These four files will be analysed using R, a statistical programming language. Since every packet recorded in each csv file has its own unique ID, packets can be traced through the network. Using this data, measurements on RSSI, packet loss, txCount values and delay can be computed.

6.2 Test 1: Individual parameter influence on neighbour Scoring

A first scoring test is to test whether each component of the routing scoring system individually functions as expected. By isolating one scoring component and eliminating the two remaining ones, correct behaviour is tested. In this set-up, 4 nodes and one gateway are used and placed in a kite shape configuration shown in Figure 18. Each node tries to send 20 messages to the gateway, located at $[-995, -819]$ with node id 4. The result of each run in shown in Table 11. In the first run, txCount is given all priority and other parameters are ignored. This results in load balancing between node 0,1 and 2 since they forward the 20 messages of node 3 in collaboration. The second run focusses on RSSI importance. This can be seen by the fact that node 1 forwards all messages from node 3, since these are closest together. This results in a better RSSI measurement. The last run will test how PathFactoring works, if all PathFactor scores are equal. As can be seen, it picks one node and uses this as forwarder. This happens because the order that neighbours are registered at nodes happen at different times if an other simulation run is executed. If all nodes have equal PathFactor scoring, the neighbour that was registered first will be selected.

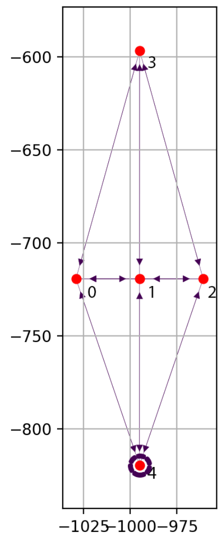


Figure 18: Node configuration for scoring test 1. Node 4 (encircled) is a gateway with ETX of 0

Table 11: Result of scoring test 1. Values of RSSI measurements could not be included due to NS3 limitations.

Set-up	ID	ETX	PF	Scaler-txCount	Scaler-RSSI	Scaler-PathFactor	txCount	txCount/ txSelf
1	4	0	1	255	0	0	0	0
1	0	1	1	255	0	0	27	1.35
1	1	1	1	255	0	0	27	1.35
1	2	1	1	255	0	0	26	1.3
1	3	2	3	255	0	0	20	1
2	4	0	1	0	255	0	0	0
2	0	1	1	0	255	0	20	1
2	1	1	1	0	255	0	40	2
2	2	1	1	0	255	0	20	1
2	3	2	3	0	255	0	20	1
3	4	0	1	0	0	255	0	0
3	0	1	1	0	0	255	40	2
3	1	1	1	0	0	255	20	1
3	2	1	1	0	0	255	20	1
3	3	2	3	0	0	255	20	1
4	4	0	1	0	0	255	0	0
4	0	1	1	0	0	255	20	1
4	1	1	1	0	0	255	20	1
4	2	1	1	0	0	255	40	2
4	3	2	3	0	0	255	20	1

6.3 Test 2: Combined parameter influence on neighbour Scoring

The second scoring test comprises of a line set-up of 20 nodes, spaced 50 meters apart, where every nodes sends two messages. In this line set-up, each node can make a connection to three nodes that have a lower ETX value. These neighbours are at a distance of 50 meters, 100 meters and 150 meters respectively. The previous test showed that each individual parameter works as they should, if full priority is given to one particular parameter. This test will combine these separate scorings and determine the best neighbour based on this weighted score.

To weight separate scores, weighs have to be calculated. These weighs can not be calculated analytically. Therefore, these weighs will be determined using linear regression in the fourth test in Section 6.5. For now, a smaller test is run to determine a rough estimate on scaler settings. After running 10 tests, each with a alternating scaler settings, the best performing settings are chosen based on total txCount. This is no proof of an optimum, nor is it necessary to be at an optimum. The goal of this test is to see whether the weighting system works as predicted. The scaler values of Equation 13, Equation 15 and Equation 16 are given in Equation 20.

$$\begin{aligned} Scaler_{PathFactor} &= 0.1 \cdot 255 = 25 \\ Scaler_{RSSI} &= 0.3 \cdot 255 = 76 \\ Scaler_{txCount} &= 0.8 \cdot 255 = 204 \end{aligned} \tag{20}$$

Using theses scalers, the maximum Score becomes

$$\begin{aligned} Score_{MAXRSSI} &= 255 \cdot 25 = 6375 \\ Score_{MAXtxCount} &= 255 \cdot 76 = 19380 \\ Score_{MAXPathFactor} &= 255 \cdot 204 = 52020 \end{aligned} \tag{21}$$

Then, using Equation 20 and Equation 21, the combined scoring can be calculated via Equation 11.

$$Score_{neighbour} = \frac{Score_{PathFactor} + Score_{RSSI} + Score_{txCount}}{Score_{MAXRSSI} + Score_{MAXtxCount} + Score_{MAXPathFactor}} \tag{11 revisited}$$

Using these values and the described set-up, a simulation is run. A subset of the results are shown in Table 12. In this table, three neighbour score calculations are shown, each from a different node. Each run shows a different scoring characteristic.

For each node, neighbours are found at 50 meters, 100 meters and 150 meters. This also means that the RSSI score should be the same. This can also be seen from the column RSSI - Score. Then, looking at PathFactor, the neighbour with the lowest PathFactor value also gets a PathFactor score of 0. Equally, the neighbour with the highest PathFactor value get the highest PathFactor score. With a PathFactor scaler of 25 and a maximum PathFactor scale, the highest PathFactor score is $25 \cdot 255 = 6,375$. A PathFactor exactly in between the lowest and the highest PathFactor value, receives a score of $127 \cdot 25 = 3175$ as can be seen from node A - neighbour L. The same weight principle is applied to txCount scoring.

Table 12: Subset of results of scoring test run using parameter values from Equation 20. Abbriviations used; PF stands for PathFactor, NB for neighbour node.

ID		RSSI			PF				txCount				Scaled
node	NB	RSSI	snr	score	PF	max	min	score	txCount	max	min	score	
a	k	-92	14	19380	1	3	1	0	0	1	0	52020	234
a	l	-101	5	19380	2	3	1	3175	1	1	0	0	73
a	m	-106	0	0	3	3	1	6375	0	1	0	52020	191
b	x	-101	5	19380	4	10	1	2125	0	1	0	52020	241
b	y	-92	14	19380	1	10	1	0	0	1	0	52020	234
b	z	-106	0	0	10	10	1	6375	1	1	0	0	20
c	e	-101	5	19380	5	15	1	1800	2	5	1	39168	197
c	f	-92	14	19380	1	15	1	0	5	5	1	0	63
c	g	-106	0	0	15	15	1	6375	1	5	1	52020	191

6.4 Test 3: Packet Delay and txCount for a random generated network

Since the routing scaler functions work correctly, integration testing can be done. To do integration testing, a random distributed node network will be used. This network is constructed by spreading 50 nodes in an area of 400 by 400 meters. One gateway will be place in the centre of the network, i.e. at location (0,0). First, an initial network is generated and run till it the network stable. Once the network is stable, every 5 to 15 seconds a message is send. 997 messages will be send through the network.

The result of this test are shown in Table 13 and Table 14 . In Table 13, one can find statistics about at what ETX level how many messages were send. Messages send at 255 indicate data transmission at an isolated node, but are delivered later when a connection to a cluster could be made defined. It indicates that a packet has been queued. Therefore, median delays are higher compared to other ETX level groups. Median delays are a little under 2000ms per hop.

From Table 14 one can see that there was no packet loss, and that the load per node is comparable to the load simulated in the Python Emulator shown in Figure 36c and Table 6. The txCount/txSelf excluding retries is 1.91 and including retries is 2.48. The former is similar to the Python simulation where, depending on the range of a node, the txCoun/txSelf was between 1.78 and 1.84.

Table 13: Delay results of random network test

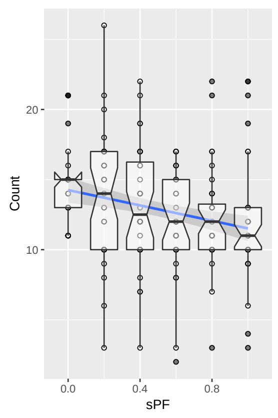
ETX level	Packet	Delay [ms]		
	count	minimum	median	maximum
1	394	16	1730	3991
2	276	323	3962	433266
3	69	2683	5884	412159
4	1	4307	4307	4307
255	256	500	19895	485792

Table 14: Node load results of random network test.

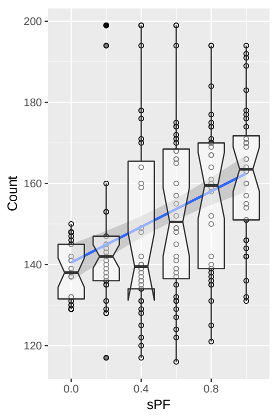
Initial Packets	997
Successful received by Gateway	996
Packets in queue	1
Lost packets	0
txCount excluding retries	1901
txCount	2472
average txCount/txSelf	2.48

6.5 Test 4: Parameter Scaling Test

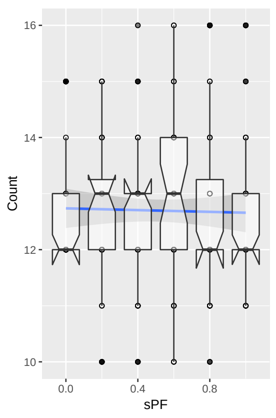
As said before, scaler parameters will be determined using a linear regression analysis. That regression study will be dealt with in this section. For this test, every parameter will be altered from 0 to 1 with steps of 0.2 resulting in 6 values per parameter. Making every unique combination results in $6^3 = 216$ simulation runs. Run 1, where all parameters are zero, is omitted. With this test, each parameter's influence will be tested based on cumulative txCount, maximum individual txCount, median txCount and packet loss.
As input data, a subset of Schiphol data will be used. This is the same as used during the Packet Loss test of the MAC layer and details are presented in Figure 39 in Appendix H. In total, 1500 messages per run will be processed by the network.



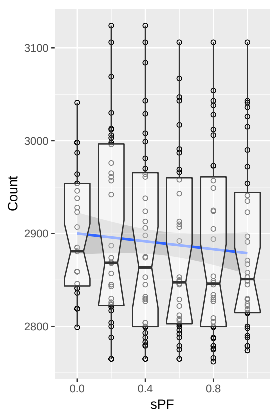
(a) Packets Lost



(b) Max txCount

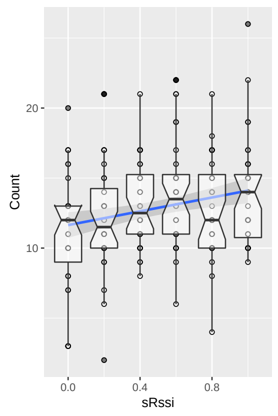


(c) Median txCount

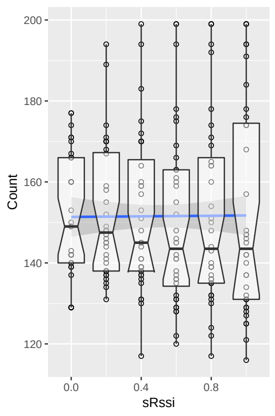


(d) Total txCount

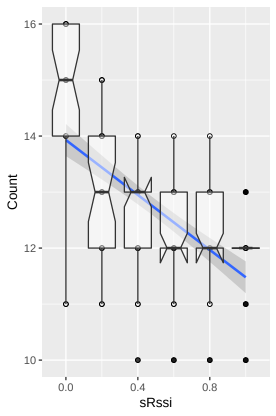
Figure 19: PathFactor Scaler influence



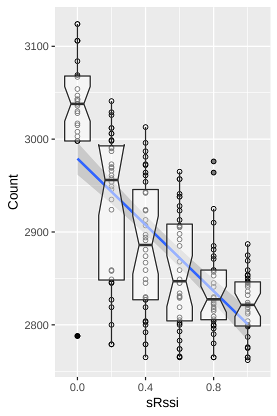
(a) Packets Lost



(b) Max txCount

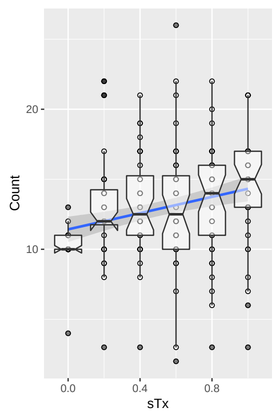


(c) Median txCount

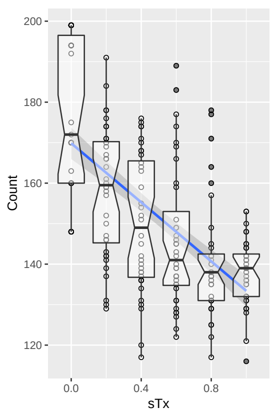


(d) Total txCount

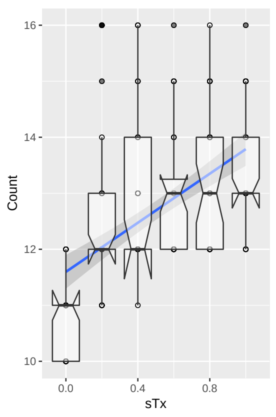
Figure 20: RSSI Scaler influence



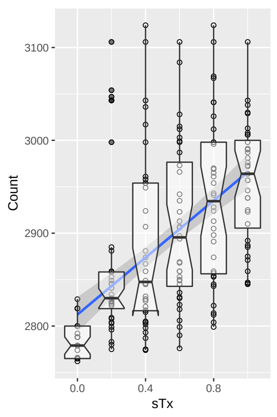
(a) Packets Lost



(b) Max txCount



(c) Median txCount



(d) Total txCount

Figure 21: txCount Scaler influence

The plots in Figure 19, Figure 20 and Figure 21 are scatter-plots with a linear regression fit line and box plots per group. The blue line is the linear fit with a grey area indicating the 95% confidence interval. The boxes of the box plot show the interquartile range (IQR). The IQR is the 25 to 75 percentile, and are also known as Q1 and Q3. The IQR is where the centre 50% of data points will fall. The whiskers add 1.5 times the IQR to the 75 percentile (Q3) and subtract 1.5 times the IQR from the 25 percentile (Q1). If these lie beyond the maximum c.q. mininum of the data set, the minimum or maximum are used as whisker. The line in the middle of the box indicate the median. The notch displays the confidence interval around the median. This is based on the $median \pm \frac{1.57IQR}{\sqrt{n}}$. According to Graphical Methods for Data Analysis [72] although not a formal test, if two boxes' notches do not overlap, it can be stated that median differs with a confidence of 95%.

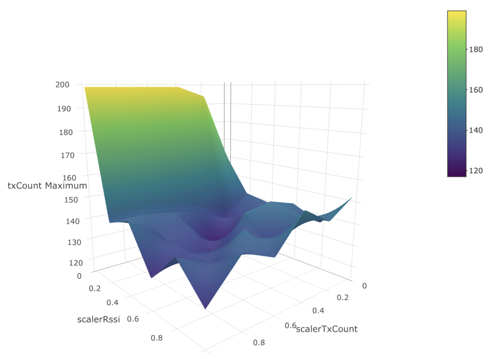
Starting with Figure 19, one can deduce that PathFactor has little effect on Packet Loss, median txCount and Total txCount. It does have a negative correlation with the Max txCount. This can be seen from the fact that the higher the PathFactor scale value, the higher the Max txCount.

Looking at Figure 20, one can deduce that there is little correlation between RSSI Scaler and Packet Loss and max txCount. However, there is a strong correlation between Median txCount and Total txCount. Looking at Figure 21, one can deduce that there is little correlation between RSSI scaler. However, there is a negative correlation between Tx Scaler and Max txCount. This is logical since this scaler steers towards equally divided workload. Also, there is a correlation between median txCount and total txCount. What should be noted is that txScaler and rssiScaler have inverse effect on median txCount and Total txCount. This means a trade-off between these two parameters. It is also interesting to see that a high txCount scaler will enhance load balancing but also increases the total txCount.

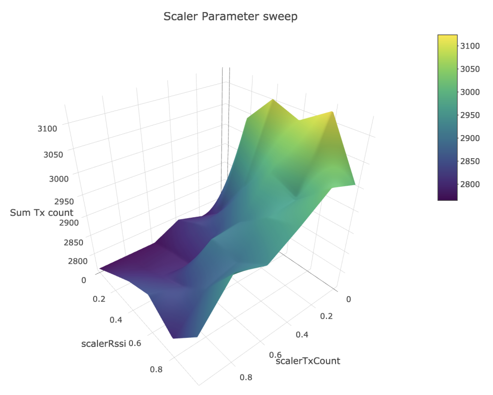
To determine an optimum, further investigation is needed. To simplify the problem, influences of scaling parameters on packet loss and median txCount are discarded. Also, since a lower PF scaler is favourable, PF scaler will be taken constant for the next analyses. Figure 19a indicate that a higher PathFactor weight reduces the number of lost packets. However, this effect is only marginal. Looking at Figure 19b the effect on maximum txCount is significant. Therefore, the PF weight is limited to a low value, in this case between 0 and 0.4.

In Figure 22, the visualisation of relation between the scaler of RSSI and the txCount scaler are shown against the result in maximum txCount per node, the total txCount and the median txCount. To make visualisation easier, the scaler of PathFactor is kept constant at 0.2. From Figure 22a it can be seen that if the maximum txCount per node increases significantly if the scalerRSSI is set to 0. A lower maximium txCount per node is preferred. Hence, the RSSI parameter has a significant contribution to the neighbour scoring system. However, from this figure it is not exactly clear where an optimum is at. Therefore, Figure 22b is considered. In this picture, the total amount of transmitted messages is shown against a range of RSSI and txCount scalers. A lower total txCount is preferred. Therefore, according to this metric, the scaler of txCount should be as high as possible since the total txCount decreases with an increasing scalerTxCount value. From the same figure, it seems that a lower scaler RSSI lowers the total txCount. In Figure 22c a similar relation can be seen. The median txCount should be as low as possible. From this, it can be seen that a higher txCount scaler is preferred. Also, the RSSI scaler should be as low as possible to get the best score. From these observations it is still hard to deduce an optimal setting. Therefore a filtered list of the raw data from the test is presented in Table 15. The list is filtered such that if an entry scores too bad on one of the parameters, it is filtered out. To be precise, the following conditions are applied. The maximum txCount per node might not exceed 158 packets (range 117 - 199), the median txCount not 14 (10 -16), the total txCount should be lower than 2872 (2765 - 3124) and the number of packets lost should not exceed 15 (8 - 26). Please note that due to a linear scoring function, every multitude of scoring parameters yields the same node scoring. This can be seen for example from the second and third entry in the table.

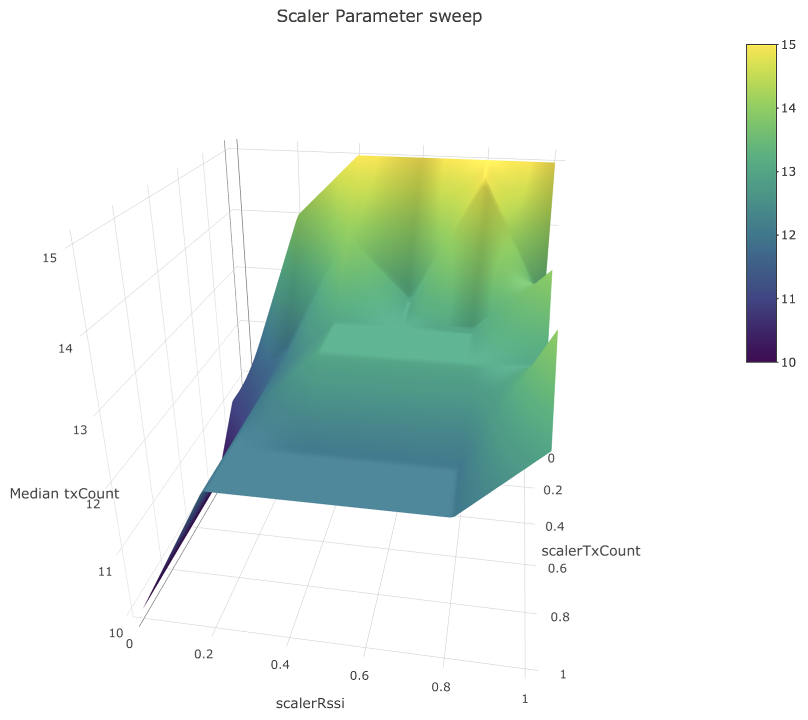
Based on this analysis, the optimum for all parameters are determined and presented in Equation 22.



(a) Maximum txCount Per Node



(b) Total txCount



(c) Median txCount

Figure 22: The influence of Scaler Tx and Scaler Rssi on Maximum txCount Per Node, Total txCount and median txCount for a PathFactor scaling of 0.2

For these parameters, the total txCount is 2804, 14 packets were lost, average txCount/txSelf is 1.909 and median txCount/txSelf is 1.

$$\begin{aligned} Scaler_{PathFactor} &= 0.4 \cdot 255 = 102 \\ Scaler_{RSSI} &= 1.0 \cdot 255 = 255 \\ Scaler_{txCount} &= 0.8 \cdot 255 = 204 \end{aligned} \tag{22}$$

Table 15: txCount and packet loss relation to scaler values. List is filtered based on worst performing settings.

Scaler			**Tx**				**Packets**		**txOver-**	**TxSelf**
sPF	**sTx**	**sRssi**	**Max**	**Mean**	**Med.**	**Sum**	**Queue**	**Lost**	**Max**	**Mean**
0.4	0.8	1	125	22.79675	12	2804	46	14	23	1.909786
0.2	0.2	0.2	131	23.13008	12	2845	50	10	24.66667	1.858101
0.4	0.4	0.4	131	23.13008	12	2845	50	10	24.66667	1.858101
0.4	0.4	1	134	23.02439	12	2832	51	12	25.66667	1.937287
0.2	0.6	0.8	135	22.95122	12	2823	50	14	25	1.922321
0.2	0.4	0.6	136	23.13008	13	2845	51	14	26.33333	1.943385
0.2	0.2	0.4	139	22.79675	12	2804	50	8	27	1.90886
0.4	0.4	0.8	139	22.79675	12	2804	50	8	27	1.90886
0.2	0.2	0.6	141	23.00813	12	2830	51	13	26	1.94219
0.2	0.2	1	142	22.95935	12	2824	51	10	25.33333	1.939761
0.2	0.8	1	142	23.11382	12	2843	46	15	25.66667	1.95569
0.2	0.2	0.8	143	22.97561	12	2826	51	12	26	1.94329

6.6 Test 5: Network Response Time to a Drive-By Gateway

A routing layer should be able to respond quickly to network changes. The introduction of a gateway, or a gateway leaving a cluster should be handled in an efficient manner. In this test, a gateway will drive by an isolated cluster with a velocity of 9 m/s, or 32.5 km/h. The cluster consists of 30 nodes, randomly positioned on a 200 meter by 200 meter grid. Each node has a message to deliver to the gateway, which adds up to 30 messages. All messages are sent when there is no connection to a gateway, simulating a worst case scenario. At t=120,000 ms, the first message is initiated and the last one is sent at 315,000 ms.

In this test, most of the default MAC settings are used as shown in Table 9 from Section 5.3.3. Only the frequency of a node scanning its surroundings, defined as the repeat phase in 5.3.3, is increased to once every 60 seconds. This is because timings in the MAC layer could not fully be determined. In a test it is shown that using an interval of 10000 seconds, 26 messages were correctly delivered to the gateway. In this test, all nodes should deliver their message. This could be achieved in several ways. One way is to improve the timing model of the MAC layer. This would mean measuring radio overhead and delay. Another way is to decrease the repeat interval period. Since improving the MAC layer would take too much time, the latter option is chosen. Using 60 seconds repeat interval, all messages are delivered.

The result of this test is shown in Figure 23. In this figure, five different stages of the test are depicted. The first image shows the beginning of the test where a gateway is outside a cluster. At 400,000 ms the gateway starts to move and comes in range of the cluster. The nodes of the cluster and the gateway set-up a connection between each other and data packets are offloaded. This happens in Figure 23b and Figure 23c. Then, in Figure 23d, the gateway is on the boundary of the cluster's range. It will take some time before the cluster registers the loss of the gateway. A neighbour is discarded only if it hasn't been recorded for more than 72 seconds. This means that a neighbour is removed after a repeat scan.

As can be seen in Figure 23b, nodes that are in range of the gateway make a connection. Then in Figure 23c it can be seen that these nodes propagate their ETX value to their neighbours in the cluster, i.e. indirectly connected nodes get a ETX value. Then, in Figure 23e, it can be seen that nodes that first notice a link break, also indicate first that they are isolated.

In this test, all 30 packets were delivered to the gateway. At approximately 408,000 ms the gateway comes in range of the first nodes in the cluster. Within three seconds, at 411,000 , the gateway makes its first connection to a node of the cluster. At 419,000 ms, all nodes in the cluster have an updated ETX value and packets are being transported to the gateway.

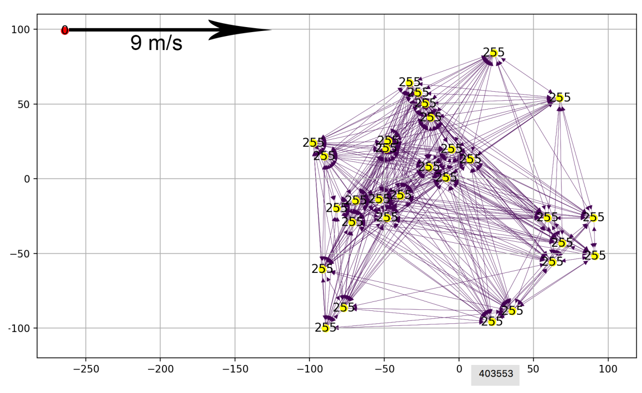
6.7 Discussion and Conclusion

With the five tests completed, results will be discussed in this section.

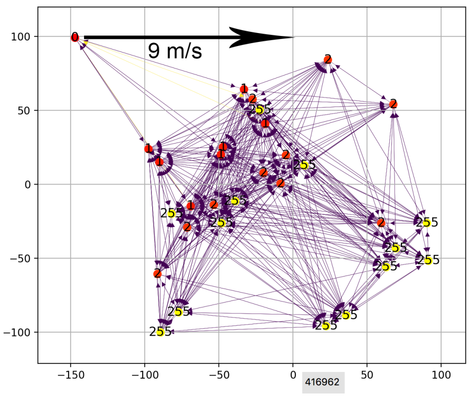
From test 1, in Section 6.2, it can be seen that each scoring component scores as desired. A high RSSI is preferred above a low RSSI, a low TX count is preferred above a high txCount and a higher PathFactor means a higher score. With this first test, the first requirements set at the beginning in Section 3.1 are met. The routing layer is capable of forming a cluster and sharing load over surrounding neighbours. It can do this without exceeding the beacon limit size of 32 bytes.

From test 2, in Section 6.3, it can be seen that combining parameter scores results in selecting a neighbour with the highest score. This does not mean this score is representative for a optimal scoring system. Looking at the scoring formula given by Equation 11, the total score of neighbours can be verified. With this test, it is shown that the implementation of the scoring algorithm is done correctly. Whether this algorithm also produces optimal results were tested in other tests.

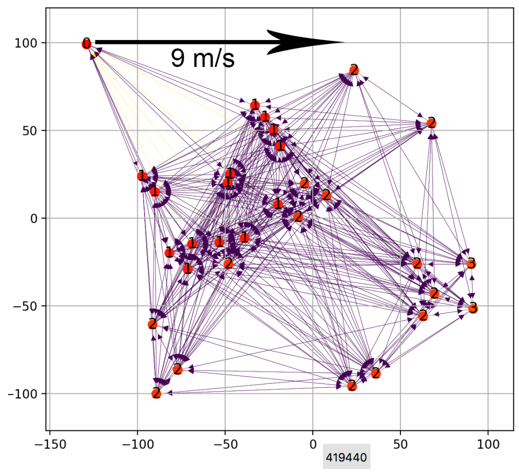
From test 3, in Section 6.4, it can be seen that for default settings, there is no packet loss for a dataset of 1000 messages. The average txCount/txSelf is 2.48. Comparing this to the preliminary design verification, where average txCount/txSelf is between 1.78 and 1.84, reveals that the randomly scattered network from



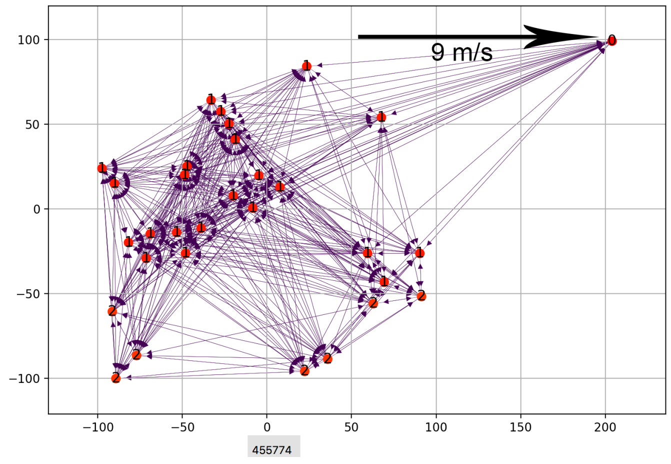
(a) Gateway starts to move towards cluster, at 403,535 ms



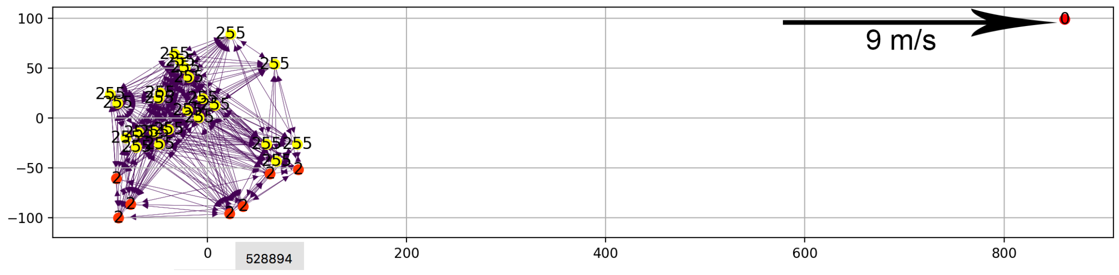
(b) Gateway connected to first nodes at 416,962 ms



(c) All nodes have an ETX value that makes data transfer possible. time is 419,440 ms



(d) Gateway moves away from cluster, nodes still (think they have) a connection to the gateway, time is 455,774 ms.



(e) Nodes removed the gateway from their neighbour list. Cluster is isolated again at 528,894 ms

Figure 23: Drive by mobile gateway and response of a cluster to this gateway. Velocity of gateway is 9 m/s (or 32.4 km/h) and touches the north side of the cluster. Cluster consists of 30 nodes, spread randomly across a 200 meter by 200 meter perimeter. Yellow edges mean a one-way connection, purple edges indicate a two-way connection. The number inside a red node indicate ETX value. Nodes without a route to a gateway have ETX level 255

this test, put a higher load on messages. However, in this test, the txCount/txSelf is calculated using the total number of transmitted data packets, including retries. The LR-WPAN module has a default retry count of 3. Calculating this ratio using the txCount without retries will yield a txCount/txSelf of $\frac{1901}{996} = 1.91$. This is closer to the 1.78 or 1.84 calculated previously. For analysis however, the number of retries should be taken into account, especially for RSSI scoring purposes.

The fourth test uses a big simulation to calculate the influence of RSSI, txCount and PathFactor and to determine the optimum scaler settings for these parameters. From Figure 19, Figure 20 and Figure 21 the influence of each parameter can be determined.

As indicated in the preliminary design verification, the influence of the PathFactor parameter is discussable. The influence of PathFactor on packet loss, the median txCount and total txCount is not significant, since most of the notches of the box plots are overlapping. The influence of PathFactor on the maximum txCount is significant and correlates positively. However, the maximum txCount should be as low as possible. Therefore, the overall performance will be best if the PathFactor scaler is set to 0. This means turning off PathFactor completely. Combining the PathFactor with other parameters, the optimum for PathFactor scaling is between 0.2 an 0.4.

From Figure 20 it can be concluded that the only significant influence of the RSSI parameter is on the total transmitted packets, called total txCount.

From Figure 21 it can be concluded that the only significant influence of the txCount parameter is on maximum txCount and total txCount. By increasing the txCount scaler, the maximum txCount per node is decreased. This is logical, since the txCount should steer on dividing load over nodes. This influence levels out at a txCount scaler of 0.8 at 1.0. However, increasing the txCount scaler also increases the total txCount.

To determine scaler values, a 3d plot in Figure 22 and Table 15 are presented to give more insights in the different simulation runs. It should be noted that weights to each scoring metric can be changed. For example, if the total txCount of a network should be lowered, a different optimum would probably be found. Eventually, scalar values were found.

It should be noted that in this test, packet loss was not expected. Earlier tests did not have packet loss. Why packet loss is the case here is unknown. It is expected that this is a MAC layer shortcoming. It could also be the packet duplicate filter that filters packets to aggressively. In any case, the percentage packet loss is not significant compared to the total amount of packets send.

The fifth test servers as a check for all network settings and test the agility of the network. Using a drive-by gateway, response times are measured and packet offloading is observed. This test shows that the network behaves correctly and that routing information spreads correct and quick through a network. With a gateway that has a larger drive-by distance, performance could decrease.

7 Conclusion and Future Work

In this research, a routing algorithm for a nomadic sensor network has been designed, implemented and tested. From this research a number of conclusions can be drawn as well as recommendations for future work. This will be presented in this chapter.

7.1 Conclusion

With performing this research, the main research question it is tried to answer, which is:

"How to design, implement and test a multi-hop mesh routing algorithm in a nomadic localisation sensor environment, keeping lower level behaviour in mind?"

Sub research questions have been defined before in Section 1.3. To formulate conclusions for this research, the structure of these research questions is followed. The research questions are restated below.

"What is a suitable approach for designing, implementing and testing a multi-hop mesh routing algorithm for a nomadic localisation sensor environment, keeping lower level behaviour in mind?"

To answer this main research question, sub research questions are formulated. These are stated below.

1. *To what extent do Undagrids sensors follow a nomadic mobility model?*
   (a) *Can the nodes be characterized as nomadic?*
      i. *How often does a node move?*
      ii. *For how long does a node move?*
      iii. *How far does a node travel?*

   (b) *What are the characteristics of a group of nodes that form one connected network, also referred to as a cluster?*
      i. *How many neighbours does a node have?*
      ii. *How many nodes are critical?* A critical node is a node that holds two smaller clusters together. If this node moves, a cluster splits in two. Load on this node is increased since all traffic of a cluster has to move through this node.
      iii. *What is the size of a cluster?*
      iv. *What would be the coverage of a mesh network be, if a mesh network were to be deployed?*
      v. *To what extent does a cluster change over time?*

2. *What Routing layer to design?* In previous research, a routing protocol comparison has been made including a proposed routing scheme [15]. In this thesis, the full design will be formalized. The routing layer has to work in a nomadic localisation sensor network, for example implemented at Undagrid.

3. *Which design is most suitable for the layers supporting the routing layer?* The routing layer will be build on top of a MAC layer, which is build on top of a physical layer. These layers influence the behaviour of the routing layer.
   (a) *What MAC layer for Routing Layer testing should be used?* Different MAC layers will be investigated and ranked. Criteria for this ranking are based on the total number of packets send, delay, and packet loss prevention and robustness

   (b) *What physical layer for Routing Layer testing should be used?* What constraints are there to the radio communication? Include but not limit to subjects like frequency band, modulation scheme, modulation efficiency, out of band emission, duty cycle and radio transmission regulations.

4. *What steps should be taken to verify system behaviour for these layers (PHY, MAC, Routing)?*
5. *What steps should be taken to simulate these layers?*
6. *What situations have to be tested?*

In the first part of this research, presented in Section 3, a prestudy on a multi-hop mesh network for nomadic localisation sensors is presented. This contains requirements, a feasibility study and related work. The first research question is answered in that section. From the feasibility study it can be seen that nodes move approximately on average 6 times per day. An event, on average is 30 meters and on average takes 132 seconds. This includes sensors that are mostly standing still, or being used during offloading. In anyway, it shows that Undagrid's sensor mobility is nomadic. Looking at cluster characteristics, it can be seen that node range has a impact on cluster characteristics. A reasonable range of a node is between 80 and 150 meters. For these distances, the percentage of nodes that are out of coverage is between 3.04% and 0.84%. The total number of critical nodes ranges between 7.24 and 0.51 on average. The mean cluster size is between 50.88 and 132.55 and the median cluster size is between 17.02 and 53.29, both averaged over time. From this, it can be concluded that the Undagrid's deployment environment is structured such that well connected mesh networks can be formed.

The second research question can be answered using the requirement analysis and related work part of Section 3, using the design procedure from Section 4 and the tests run in Section 6.

Regarding the requirements, it should be clear that Undagrid operates in an special environment. Wireless sensors should comply to strict radio communication regulations, the deployment of infrastructure is harder than normal and sensors can not be accessed at any given moment. For the routing layer this means that it should be resource efficient, have as little overhead as possible, be able to balance load amongst nodes and respond in a quick manner to network changes. These requirements are measured against existing ready to market solutions. As it turns out, there is no suitable partner for Undagrid that can deliver a multi-hop mesh network. Platforms are not flexible enough, are too closed in terms of source code,too expensive or simply do not match the desired topology. This proved the need for the design and implementation of a custom routing algorithm.

The literature study compared several routing algorithms and presented Location Aware Routing (LAR) and Collection Tree Protocol (CTP) or Expected Transmission Count (ETX) routing as candidates for a custom routing design basis. From an empirical comparison it is concluded that ETX would probably work better than CTP. However, proof that ETX performs better than LAR was not given. Based on this assumption, ETX routing was expanded to improve performance. At this point the routing parameters txCount score, RSSI score and PathFactor score were introduced. This design was tested using a Python emulator. Results from this emulator showed that RSSI scoring has significance to routing, that the effect of PathFactor is disputable and that route redundancy does not increase network performance. It was concluded that ETX routing outperforms flooding and that further simulation was needed for full design verification. From this test, it is also known that the average txCount/txSelf is between 1.78 and 1.84. This means that a node transmits 1.84 times as much packets than it would have done in a star topology network. If overall energy consumption is 1.84 lower than star topology, battery life will be similar to a star topology.

Using a NS3 environment with a representative MAC and PHY layer, the routing layer was tested based on 5 different tests. These tests showed that routing based on individual and on combined scoring parameters worked as desired. The first two tests proved that parameter scoring worked, and that combining these scores results in a total score that is representative for the weights given to these parameters. The third test used a random network, containing 50 nodes, to test packet delay and txCount performance. txCount/txSelf was increased to 1.98 if retries are ignored. A realistic figure for txCoun/txSelf, including retries, is 2.48. This means a higher load on nodes compared to the Python emulation. The fourth test is a characterisation test. Since weights for scoring different parameters could not be computed algebraically, it had to be computed numerically. This is done using a linear regression model where, considering packet loss, maximum txCount and total txCount, global optimal parameters were found. Finally, the network is tested in a challenging situation: an isolated network with a drive by gateway. This involves quick gateway discovery, routing information propagation, packet offloading, broken link detection, cluster isolation and packet buffering. The test succeeded with all packets delivered correct, in time and correct routing behaviour. One of the requirements for the network was that packet loss should be absent. This was not realized. Why packets got lost has to be investigated.

For answering the third research question, lower layer characteristics have been investigated. To be able to simulate the routing layer in NS3, it was decided to use the LR-WPAN module which uses IEEE 802.15.4. Based on this, PHY and MAC layer parameters have been set.

On the PHY layer, the model was adapted such that transmission range is up to 150 meters with beacon packets of 32 bytes and using QPSK. This should be feasible with current sensor technology of Undagrid. The MAC layer of LR-WPAN has been altered to mimic duty cycle reduction. With that, delay of packets have a more realistic figure and packet loss is more realistic .Median packet delay was observed to be around 2000 ms per hop, using a 4000 ms beacon interval. This was as expected and will save energy by cutting idle listening time. However, timing is not perfect yet and should be improved for a better energy efficient MAC layer.

Regarding the fourth research question, several tool sets have been used to be able to describe the three different network layers. For the routing layer, graph theory has been used to describe a network and the flow of packets. Graph theory has also been used to perform data analysis for characterizing clusters and emulating an early ETX routing implementation. With this, it was possible to proof that Undagrid's network is indeed nomadic and clusters were sufficiently connected to form a mesh. It also gives insights in optimal route finding. Worst case execution time (WCET) analysis has been used to estimate packet delay. This proofed that a packet can be delivered in less than 60 seconds, assuming a network no larger than 30 hops. To predict physical layer behaviour, only the BER relation had to be constructed. Other physical layer properties did not have to be changed. These only had to be simulated which was taken care of by NS3.

The fifth question is answered by the implementation of the three network layers in NS3. Routing designs have been verified using a Python emulator. This proved the concept of ETX routing. True validation and a useful implementation of ETX routing, that can be ported to real life hardware, has been accomplished in NS3. By expanding default NS3 classes, it was possible to model a node and with that a full mesh network. Routing methods are implemented in pure C such that direct porting to an embedded platform is made easy. The MAC layer has been simulated using the LR-WPAN MAC with additions. A full PHY layer was already implemented in NS3 and no significant contribution to this layers has been made.

The last question can be answered by looking at the last test, test number 5. This test checked whether the network is dynamic enough to connect to a gateway that travelled with more than 30 km/h. With this test, it has been proven that the network is able to respond to network changes, that it can detect new nodes quickly enough and it is able to offload packets to a gateway even if had been isolated for some time.

All in all, a routing layer has been proposed and tested that would function in a nomadic localisation sensor network. Having proven that Undagrid's sensors follow nomadic behaviour, this routing layer can be used by Undagrid. The layers below this routing layer, the MAC and PHY layer, are advised to be as efficient as possible. For both PHY and MAC recommendations are made. As a MAC layer, it has been shown that an idle listening time reduction system would introduce acceptable delays. Rough receive and transmit window estimations show that energy could be save, although further investigation is needed.

7.2 Limitations

This section will discuss limitations of the current research and issues that could not have been resolved during this research.

One of the issues of the NS3 implementation is that packet loss is present. This could have been caused by packet duplication filtering or by a faulty ACK system. A bug in NS3 would a node allow to receive two exactly the same packets right after eachother. It was impossible to intervene in between the reception of these two packets. Therefore, faulty ACK filtering could not be realized.

Another improvement that should be made to the simulation environment is MAC layer timing. Due to packet scheduling and delays in NS3, it is difficult to include node wake up times in data packets. Altering MAC and PHY layers to include this information would solve these problems. Also, since accurate timing in this implementation was impossible, there is still a significant amount of idle listing time that could be reduced if accurate neighbour timing tracking is realised.

The LR-WPAN module in NS3 has to simulate each receiving event on every node, even though it is clear that a message will not be received correctly based e.g. its location. Filtering out these events will reduce computation complexity and speed up the simulation time. This indirectly means that larger networks can be simulated. This could make it possible to simulate a full sensor site. This would mean that a simulation can be done on energy consumption. This was impossible before, since nodes typically leave a cluster and move to another cluster. This means that two clusters have to be simulated, which is too computational expensive.

During simulation, QPSK is being used by the physical layer as modulation scheme. It does not exactly match the FSK from BLE or FLRC, since it has double the bits per symbol, but it is the closest to FSK using NS3 LR-WPAN PHY layer. Also, since FLRC is a spread spectrum algorithm packet collision effects will be different in real life.

7.3 Future Work

This section will cover ideas for future work that uses this research as a basis.

The analysis framework presented in this research can be used for future analysis. This research very specifically focused on the network topology on one site, namely Schiphol Airport. However, analysis for any site can be done using this framework. This would for example yield insights in coverage of a network, which might serve as a proof for switching to a mesh topology. Also, this framework can be used to find optimal spots for fixed infrastructure, such as gateways.

The developed framework is able to test different routing algorithms. Therefore, different kinds of routing algorithms can be implemented fairly easy since OSI stack abstraction is preserved. Propagated node scoring might be one of the interesting additions to the current ETX routing protocol. Propagated node scoring might reveal overall network improvements since neighbours are not selected on direct links, but on overall link quality to a gateway. Propagated node scoring might result in a smarter network since it finds a global optimum and not only a local optimum. Another change that might have a significant influence is to start using txCount per month instead of total txCount. If total txCount is used routing metric, older nodes will always score low on the txCount metric and newer nodes are therefore favoured over older ones. This does not have to be fair, since old nodes have to die eventually.

Appendices

Appendix A Calculations

This section contains several calculations that where needed to support claims and decisions throughout this report.

A.1 Drive By Time

To calculate the time that a node is able to communicate with another node if one of them is moving, one simply applies pythgoras to the following situation.

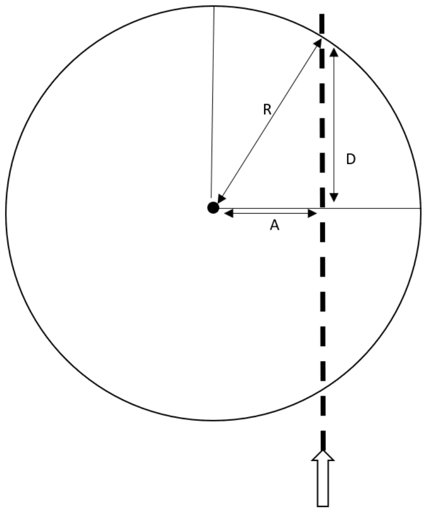


Figure 24: Drive By Time distances

So

$$D = \sqrt{R^2 - A^2} \tag{23}$$

and time in range is

$$T = 2 * D/v \tag{24}$$

If a node has a range of 200 meters and another node drives by at a 100 meters and assuming the maximum speed of a cart (v) is moving is 10 m/s , or 36 kmh, the time in range is

$$\begin{aligned} T &= 2 * \sqrt{R^2 - A^2}/v \\ &= 2 * \sqrt{200^2 - 100^2}/10 \\ &= 34.64[s] \end{aligned} \tag{25}$$

Appendix B RSSI Measurement

This measurement series was performed by Undagrid on a gras field with line of sight. The measurements were done with the BLEtrack 2.16 device (Undagrid). The RSSI measurment was done in an open area near the University of Twente.

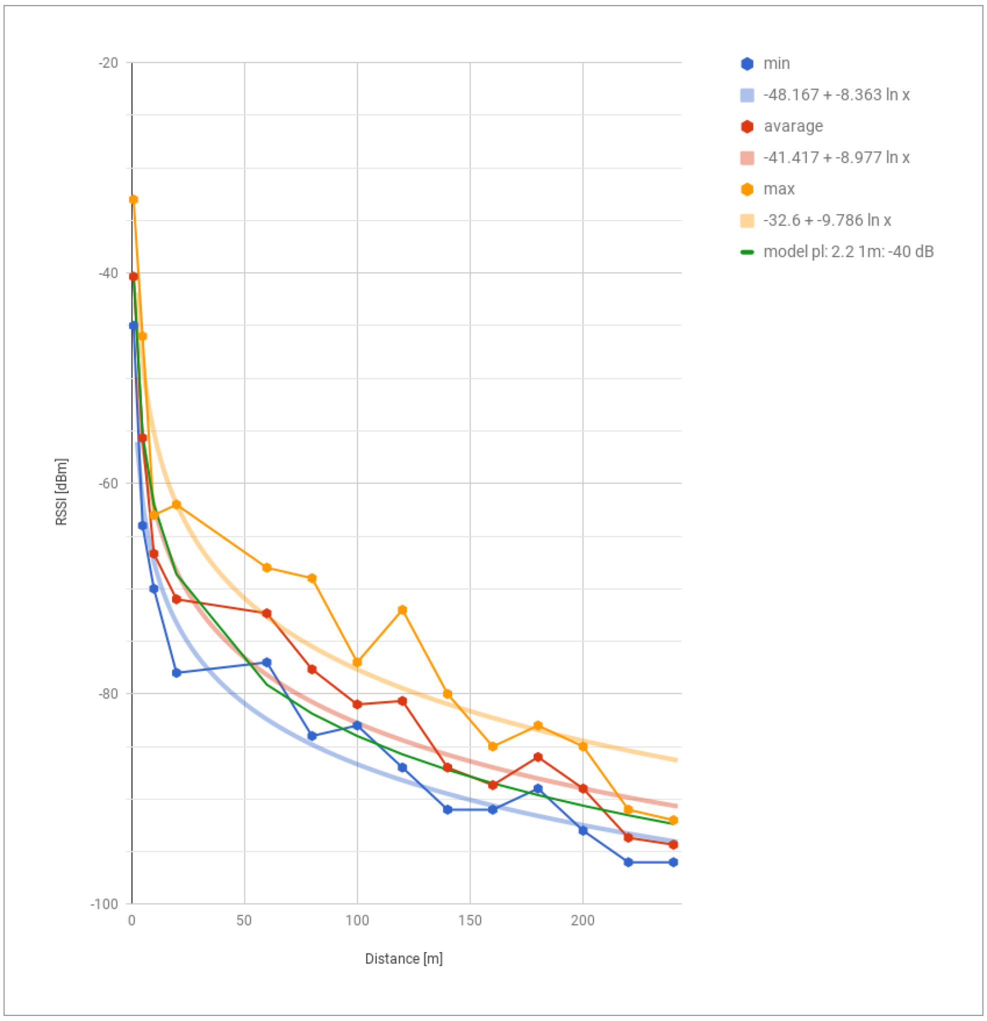


Figure 25: Long distance RSSI measurements using BLEtrack in Line of Sight environment

As can be seen in Figure 25, the received RSSI at 100 meters is around -80dBm if the average RSSI is taken. The receiver sensitivity is -95dBm. Distances of around 150-200 meters are feasible.

Appendix C Data Analysis

This section will try to provide answers to the first research questions. This will be done by analysing data that is in the Undagrid Backend and contains GPS locations of all the nodes at certain airports. As stated before, the scope of this research will take Schiphol Airport as an example airport. However, the analysis algorithm will be written in such a way that with little adjustments, the same analysis can be done for different airports.

C.1 Set-up

The backend of Undagrid is accessible via a REST API. Upon a request, the API will respond with a JSON list with the result of the query. This data will be saved to hard drive and processed via a Python script. Python is opensource and is able to work with classes, structs, lists and other data types that makes data analysis easy. Python packages like NumPi (numerical analysis), Matplotlib (plotting) and NetworkX (graph theory) will be used to analyse the data. The analysis will be on a Windows 10 machine with an Intel Core i7 and 8GB of RAM.

C.2 Methods

The experiment will be split up into two parts: a part focussed on node characteristics and a part focussed on cluster characteristics. The dataset that will be used will be the data set for all nodes on Schiphol Airport in the period between June 1, 2017 00:00 and June 8, 2017 00:00. The API channel that is being used is called TheMainGSETrackChannel. For both parts, the data will be filtered on:

- Is the message send via a gateway in Amsterdam (location filtering)
- Check if a node has sent invalid datestamps. If so, remove all data points from this node.

C.2.1 Data

The API backend responds with a JSON string. An example can be found below.

```
{"channelUuid":"1f98b0a0-36ad-11e4-b5b6-02c2e988c447",
"channelName":"TheMainGSETrackChannel",
"gatewayUuid":"2291e7af-c9f7-4c16-b407-059774830645",
"gatewayName":"UGGWL17030003","nodeUuid":"e73a56b0-c2b8-11e6-acd4-02d573ab71f9",
"nodeName":"UG0133B579",
"sensorUuid":"38f46ed3-e2ea-4625-9371-1e133627f1d3",
"sensorName":"sensor_UG0133B579-GPS",
"ts":1497944592059,
"type":"GPS",
"value":{
        "gatewayRssi":-96,
        "connected":true,
        "queued":false,
        "ts":1497944584,
        "hdop":1.2,
        "fix":9,
        "view":13,
        "snr":33,
        "stationary":true,
        "nodeRssi":-99.5,
        "radioChannel":4,
        "radioMilliWatts":25,
        "lat":52.2940537,
        "lon":4.7585172,
        "location":"S72"}
26 }
```

The first filtering on all data will be done on the type field. This should always be GPS. Then, to detect the change of state, the field "stationary" in the "value" field should be monitored. In Figure 26, the states of a GPS messages based on the "stationary" field can be seen. The field is True, False or absent, which results in three states. On every state change a message will be broadcast with the according information.

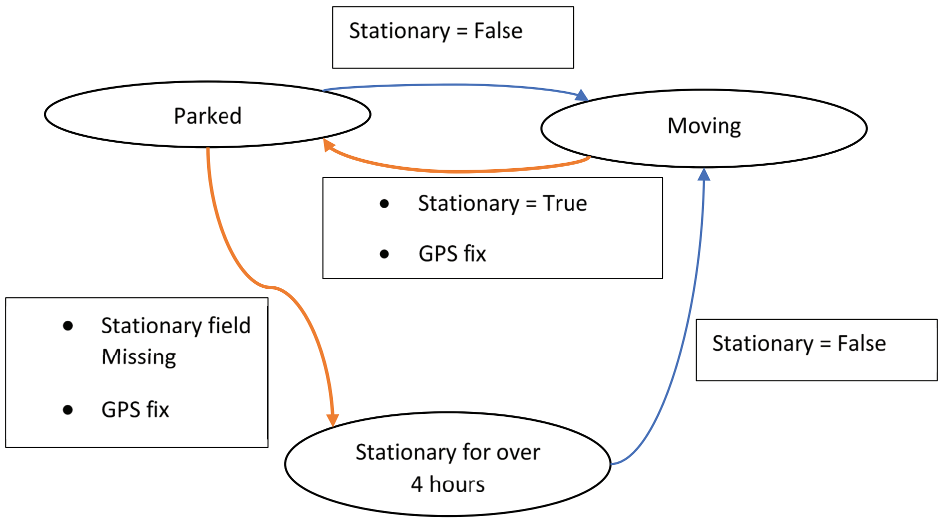


Figure 26: States in the back-end based on the "stationary" field. On all orange edges, a GPS location is published

C.2.2 Time of Travel

For the time of travel of travel analysis, the node data is grouped by node and sorted in chronological. If the messages are ordered correctly, de states as denoted in Figure 26 are begin used to calculate the time of travel. This is done by detecting a message with "stationary" = False and comparing that to the message that comes after that.

C.2.3 Distance of Travel

Comparable to the Time of Travel method, this method detects a "moving" state and upon detecting takes a previous stationary or parked point and compares that to the next stationary or parked state. This data is based on GPS locations and distance is measured as the crow flies. This means that the actual distance travelled can be bigger.

C.2.4 Clusters

To analyse if clustering is possible, the GPS locations of the nodes will be used to form virtual clusters. This will be done using the DBSCAN function from the Python library sklearn. This algorithm will form clusters based on a predefined condition. In this case, this condition is that if two nodes are within 85m from each other, they can form a wireless link. This distance is chosen based on the measurements done from Undagrid. This data is included in Appendix B.

Multiple linked nodes can form a cluster. The minimum size of a cluster is 3 nodes. The result of the DBSCAN is fed into a Python graph theory library called NetworkX. This converts the cluster into a graph such that graph analysis can be performed on the clusters. With this, parts of research question 2 can be answered.

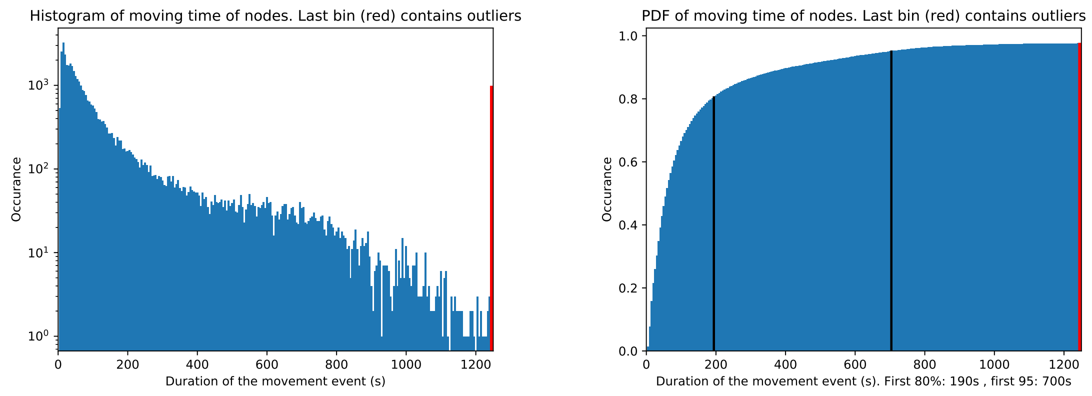
One of the metrics that will be investigated is the node load. Load, or betweenness centrality, of a vertex is the accumulated total number of data packets passing through that vertex when every pair of vertices sends and receives a data packet along the shortest path connecting the pair[73]. This shows which nodes are to heavily loaded and therefore will drain their battery quicker. Also node size, total number of clusters and average cluster size will be investigated.

C.3 Results

The results of the data analysis will be shown below.

C.3.1 Time

First, let's focus on the time of travel analysis. In Figure 27, the time of travel histogram is shown in both histogram and cumulative probability density function. For this analysis, all movements after 1800 seconds are considered as outliers since the occurrence is below 2%.



(a) Histogram of occurrences of durations of a movement event with a bin-width of 5 seconds

(b) Cumulative PDF of event durations.

Figure 27: Analysis of the trip times for nodes at Schiphol Airport for 7 days

If all occurrences from Figure 27a are summed, the total number of movements can be calculated. The analysis also counts the number of nodes that are found. With those numbers, the average number of movements can be found. Another way of calculating the average number of events is by looking at how often each node moves and averaging over that. The results can be found in Table 16.

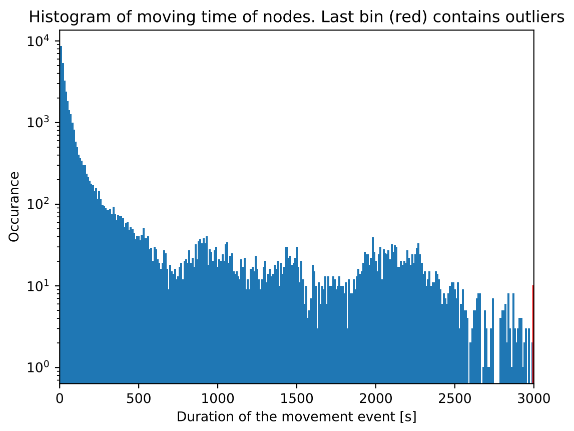
C.3.2 Euclidean Distance

For analysing distance a more strict filtering is applied. Since distance is measured with GPS, using datapoints with false GPS fixes can influence the data. The quality of the fix can mainly be derived from the horizontal dilution of precision (HDOP) metric. [74] The maximum HDOP allowed in this analysis is 15m. A movement is defined as the distance travelled by a node between the data point before and after a "stationary" = True packet. This distance is measured as the crow flies, hence euclidean distance.

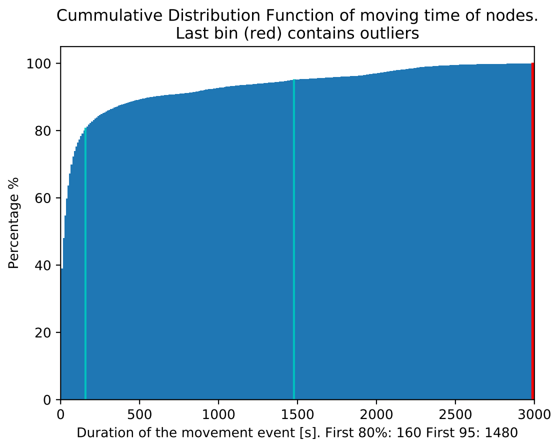
Table 16: Time of Travel analysis results

Total number of movements:	40047
Total number of nodes before filtering:	1306
Total number of nodes after filtering:	846
Average number of movements of a node per day (global) :	6.76
Average number of movements of a node per day (grouped) :	5.78
Maximum travelled time (outlier) :	535718 [s]
Average travel time :	132.269 [s]

First, the characteristics of movements are analysed. The first analysis is on the distance that a node travels normally. The result of this analysis is shown in Figure 28. On the left, in Figure 28a, the distribution of the nodes can be found in absolute figures. On the right, in Figure 28b, the relative distribution can be found. From the latter it can be derived that the 80% of the movements are in the range of 0m to 160m.



(a) Histogram of occurrences of distance of a movement event with a bin-width of 10 meter



(b) Cumulative PDF of event distances.

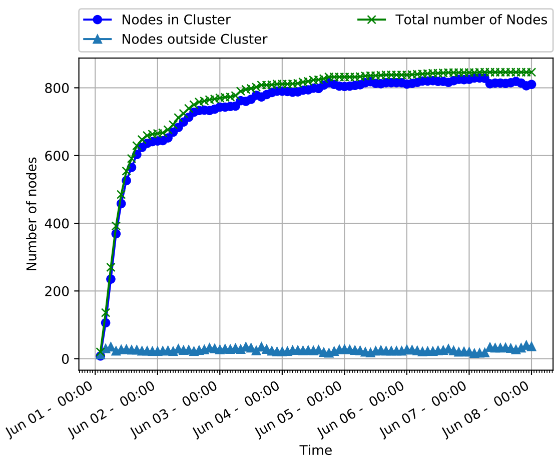
Figure 28: Analysis of the trip distances for nodes at Schiphol Airport for 7 days. The last red bin contains all outliers.

Table 17: Distance of Travel analysis results

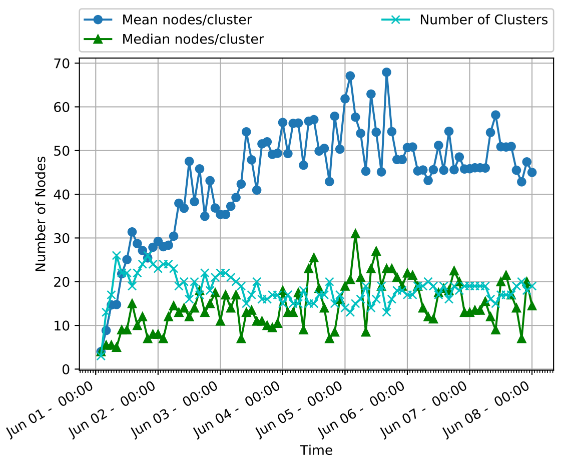
Total number of movements before filtering:	35756
Total number of movements after filtering:	35747
Average distance travelled of a node per day :	30.4m
Maximum travelled distance (outlier) :	4930 [m]
Average trip distance :	213m

C.3.3 Cluster

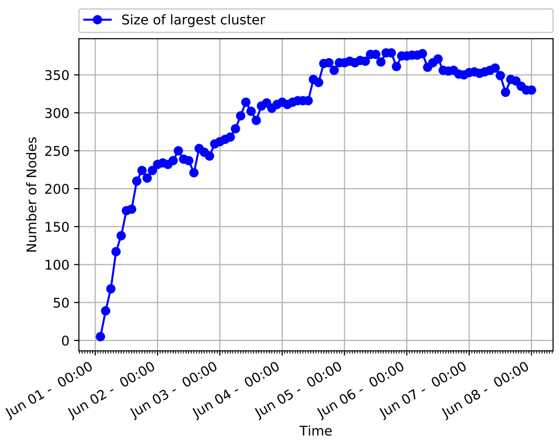
At first, the time dependence of the clusters is investigated. The same data set as before is analysed with snapshots of one hour. In that snapshot, the last known GPS location of a node is taken. These nodes are clustered based on the 85m distance constraint. Then, total number of nodes, the number of nodes inside a cluster, number outside a cluster and average cluster size are calculated. This result is shown in Figure 29.



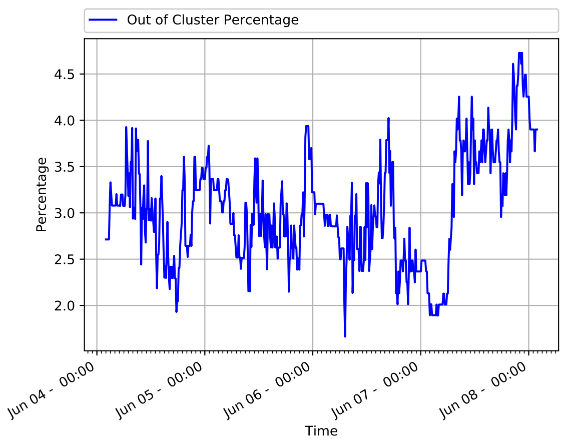
(a) Total number of nodes, nodes inside and nodes outside a cluster



(b) Maximum, mean, and median of the cluster sizes



(c) Largest cluster size



(d) Out of Cluster Percentage

Figure 29: Statistics for the clusters. Stepsize is one hour, and analysis duration is 7 days.

From Figure 29, it can be seen that the clusters have a ramp-up time of a couple of hours. This has to do with the fact that at the first snapshots, only a few data points are available, since nodes only sent every now and then their location. The total number of nodes is stable after 3 days. Therefore, for further analysis a ramp-up time of 3 days will be used.

To get a better feeling of how such a cluster looks like, a visualisation of the clusters is depicted in Figure 30.

Now, let's zoom in on a couple of clusters and investigate what kind of clusters there can be found. Three examples are picked and displayed in Figure 31. The left figure shows a network with a central node. This node is critical to the system, since removing it would break the cluster. The middle image shows a well-connected cluster, and the most right an image of one of the smallest clusters.

How the number of clusters and the characteristics of clusters change over time, was discussed in Figure 29. To investigate the clusters on a node level, two metrics are monitored. The first is the maximum consecutive time that a node is out of a cluster. The second is the how many critical there are at a particular time. The results of the first metric can be found in Figure 32. This figure shows in bins of 10 minutes how many nodes have a certain maximum out of cluster time. The analysis is done for 4 days with snapshots of 10 minutes. As can be seen, in total 284 nodes were not connected to a cluster. If a critical out of cluster time of 3 hours

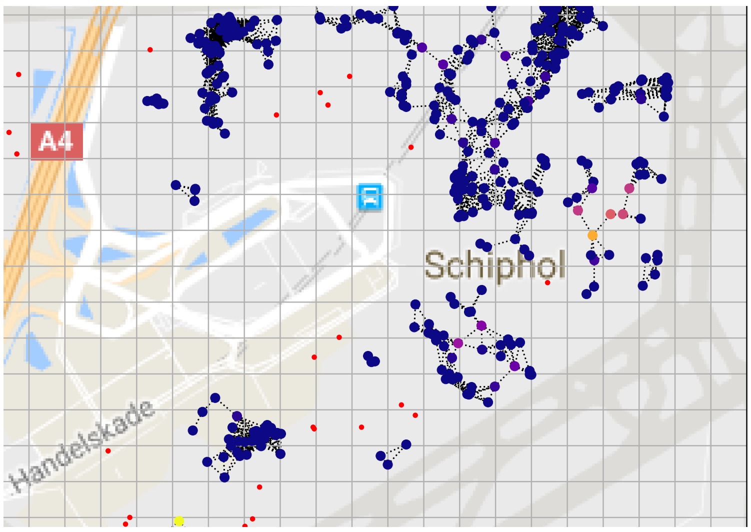
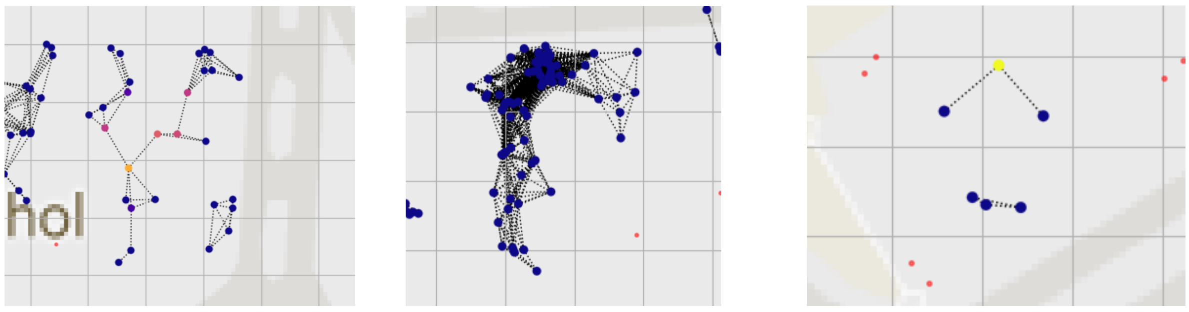


Figure 30: Clusters plotted on Schiphol



(a) a cluster with one critical node (yellow)

(b) A well-connected cluster

(c) Small cluster with outliers

Figure 31: Subset of clusters with different charcteristics

is set, then 112 devices have a maximum out of cluster time that exceeds this 3 hours. Upon investigation, these nodes are mainly situated in remote areas. These nodes belong to ground support equipment that have stationary behaviour rather than nomadic. An example of this equipment is a moveable staircase used in only rare occasions on a remote loading platform.

The result of the analysis on the second metric, the amount of critical nodes, is depicted in Figure 33a. It is best to interpret these results in combination with Figure 34. As can be seen from Figure 34, some of the critical nodes are nodes that belong to a small cluster. Despite that these nodes are critical to that cluster, the absolute load will be relatively low since only a few nodes rely on it.

C.3.4 Overview and Comparison

To summarize the above data, averages over the above mentioned metrics are shown in Table 16. Data for different maximum node communication distances are provided as well.

From Table 18 it can be seen that the connectivity of a cluster goes up if the range increases. This is trivial. However, please note the mean number of critical nodes at 150 meters is 0.51. This means that the clusters are highly interconnected. This conclusion can also be drawn from the average median connectivity.

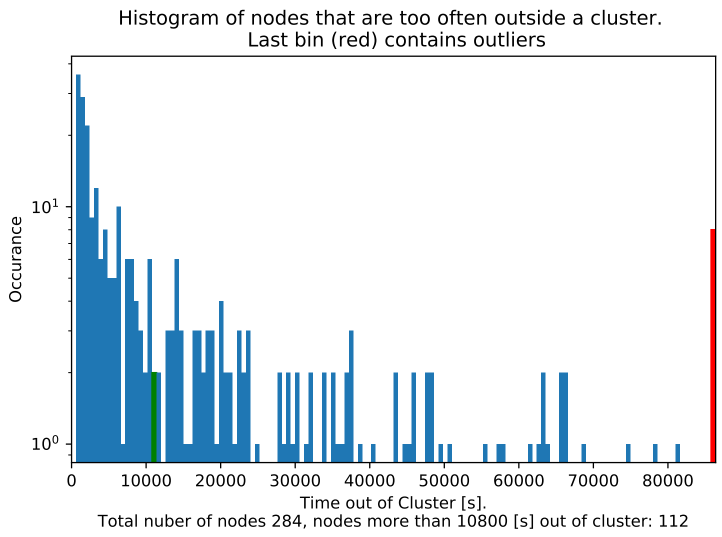
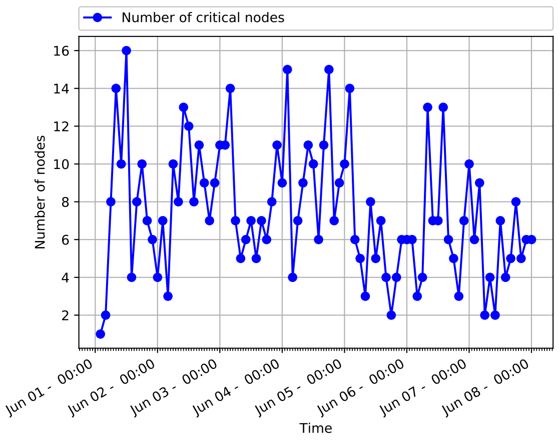
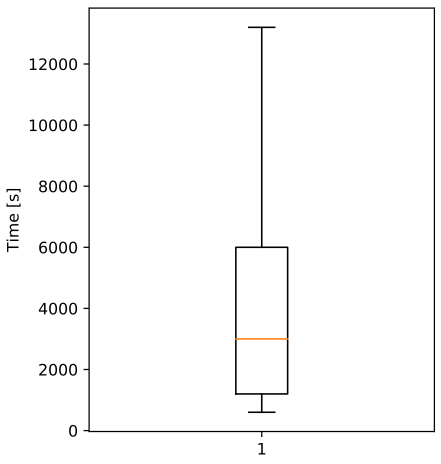


Figure 32: Count of nodes that are out of cluster and its duration



(a) Total number of critical nodes.



(b) Time a critical node is under load. Number of nodes: 465, Median: 3000s, Mean: 5382s. Fliers not shown, maximum load time: 62400s

Figure 33: Critical node statistics

C.4 Discussion

Although the data analysis might be right, the results should be verified and discussed such that the results are interpreted correctly. Lets start with the results shown in Figure 27 and Figure 28. Even after filtering and using valid input data, it might happen that simply because of vibrations a node detects a movement where it is actually just still stationary. This idea can be supported by the distribution of the events. An event of only 10 or 20 seconds (highest peak) would probably not give a big change in cluster formation. However, since this theorem is hard to proof, the results are used as it is.

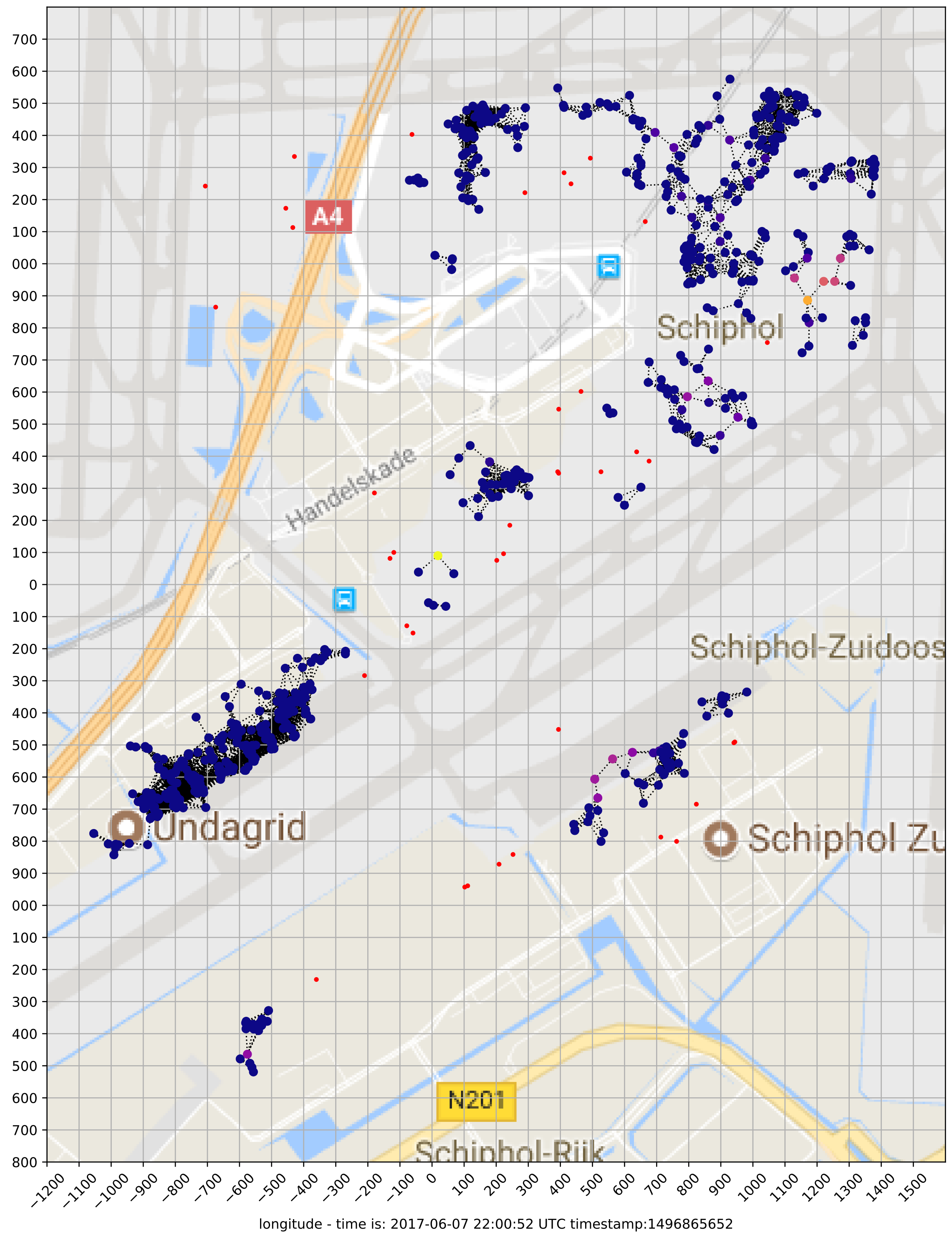


Figure 34: Clusters plotted on Schiphol. Red dots are nodes outside a cluster. Yellow Are critical nodes.

C.5 Conclusion

The results of the data analysis give a first positive impression of implementing a mesh network in the Schiphol area. Assuming an average maximum transmission distance of 85 meters, an average coverage of

Table 18: Overview of different maximum inter node distances.

	Maximum distance between nodes	**30m**	**85m**	**100m**	**150m**
Nodes					
	How often does a node move per day	~5	~5	~5	~5
	Distance travelled per node per day	30,4m	30,4m	30,4m	30,4m
	Average travel time	132,269s	132,269s	132,269s	132,269s
Cluster					
	Mean number of critical Nodes	24,11	7,24	5,27	0,51
	Average Mean cluster size	13,44	50,88	67,88	132,55
	Average Median cluster size	5,42	17,02	18,41	53,29
	Out of cluster percentage	20,69%	3,04%	1,84%	0,84%
	Mean coverage of mesh network	79,31%	96,96%	98,16%	99,16%
	Total number of nodes in cluster	837	837	837	837
	Number of nodes that loses cluster connection once every 7 days	649	284	185	121
	or	77,54%	33,93%	22,10%	14,46%
	Number of nodes exceeding 3 hour out of cluster limit	480	112	67	24
	or	57,35%	13,38%	8,00%	2,87%
	Average median connectivity	3,19	7,25	8,11	23,09

96.96% can be achieved. On the other hand, in a timespan of 7 days, 112 nodes weren't connected to a mesh for more than 3 hours. Furthermore, the analysis also shows that on average only 7.24 nodes are critical nodes. In total 465 nodes serve as critical node with a median load time of 3000s.

In short, the data shows a that a mesh network could be implemented and that the nature of the clusters is suitable for a mesh network. Further investigation in a network simulator should provide further insights into e.g. energy consumption, packetloss, and latency.

Appendix D Preliminary Design Validation with Python Emulator

As mentioned before, prior to implementing the routing design of Section 4, an ETX routing will be implemented in a simplified environment. This will be done using the framework used during the feasibility study in Section 3.2. This will give first insights in behaviour of ETX routing without the amount of effort to that a full network simulator implementation will take. The simplified environment will only emulate routing behaviour. This means that the MAC and physical layer are assumed to be ideal. Implementing a routing layer in a simplified environment will gain insights in ETX routing behaviour and will help to identify exceptional cases. These insights can also be gained from a more advanced simulator. However, implementing this in an advanced simulator will take more time. Moreover, if ETX routing does not work a simple emulator, it will not work in a more advanced simulator or in real life.

D.1 Assumptions and Simulation Environment

The data analysis framework, that was build in Python using the NetworkX graph library, can be altered to send messages across a graph. Using NetworkX, nodes are clustered and placed in a graph. Nodes can use this graph to get information from neighbours and to send data to these neighbours. This is in contrast to a real network that uses beacon packets to spread routing information. In the Python emulator, routing information is therefore present at every node instantaneously. Or in other words, neighbour discovery is done by a lookup in a graph object.

Data transmission is done using send queues and receive queues. Every node in the NetworkX graph is an Python object that resembles a sensor, including properties and methods that a sensor has. If a node wants to send a message to a neighbour it fetches the node object of a neighbour and moves a message from its send queue into the receive queue of a neighbour.

Since timing properties are not simulated, packet collision or channel interference can't be simulated. However, the Python emulator has the ability to calculate packet error rates (PER). This is done by calculating the signal to noise ratio (SNR) using a log distance path loss model with a path loss exponent of 3.5. For a free space environment, this exponent is 2. A higher value is chosen to approximate a worst case bound on path loss. Using a random draw with PER as probability, packets are indicated as faulty or correct. If a packet is faulty, the whole packet will be dropped, i.e. there is no error correction. If a packet is dropped, the sender will try to send a packet again. This results in a higher and more accurate txCount per node. Packets will eventually arrive the selected neighbour.

The emulator flow is as following. The emulator uses a dataset from the Undagrid database. Data of 10 days is used to scan for nodes and gather their location information. This information is used to build up an initial graph. Nodes have a connection if they less are a certain distance apart. Clusters with only 2 nodes ore less are kept as isolated nodes and not placed in a cluster. Also, gateways are defined such that clusters can offload their messages.

After the initialisation phase, messages from the database are fed trough the network as if there were a true multi hop network implementation. First, the sending node finds its target neighbours and sends its message. Packet exchange between nodes is done by means of moving a packet from a send queue to a receive queue at the neighbour side.

The next step is that for all nodes the receive queue is processed. This means that at random all nodes are checked if their receive queue contains a message. If so, it is pulled from this queue and processed. A packet will only be forwarded if it hasn't been forwarded before, as described in Section 4.2.3. Processing of packets is done until all receive queues and send queues are empty.

If all queues are empty, the next message from the database is initiated and send trough the network. This process is continued until all messages in the data set are processed. At that point, the simulation stops.

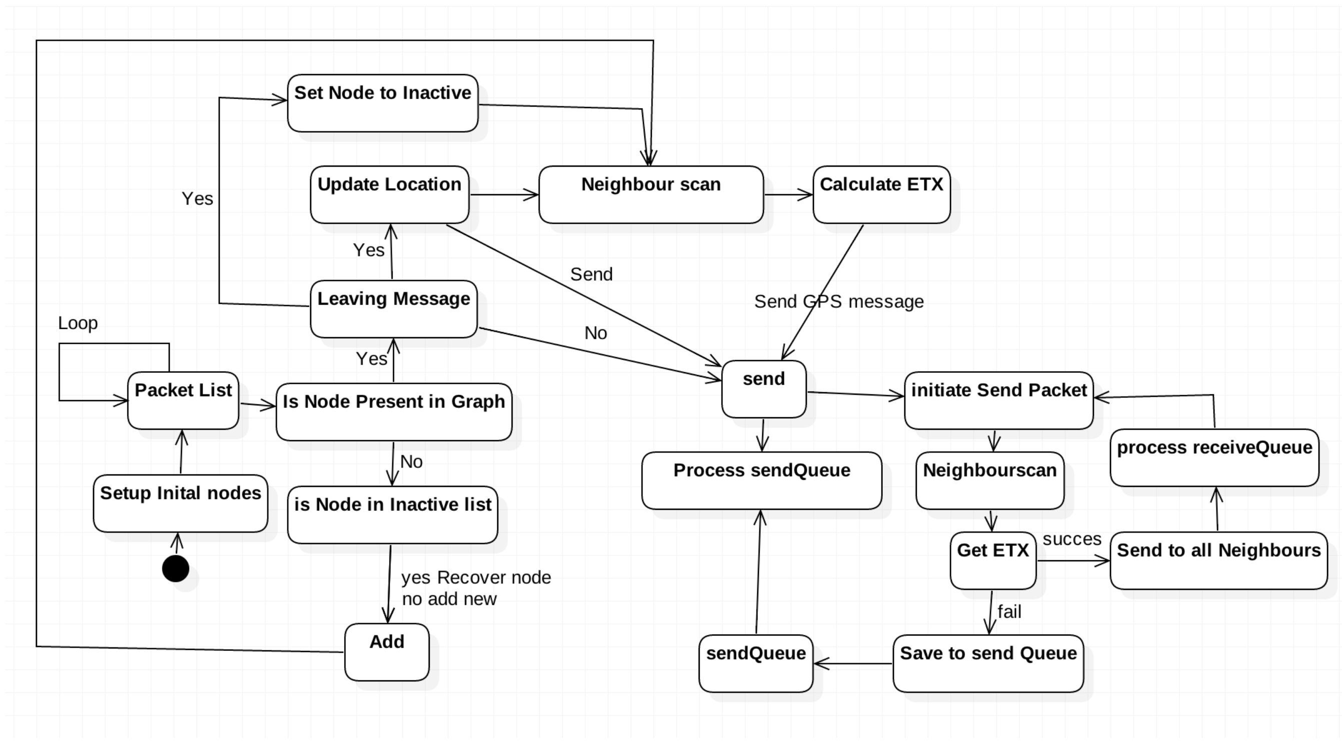


Figure 35: Information flow through the Undagrid Emulator

D.2 Routing Design

Using the routing metrics explained in section 4, each neighbour of a node can be assigned a score. This score is a combination of PathFactor, txCount and RSSI. Since these parameter have different units, they have to be scaled. Each parameter is scaled to a value between 0 and 1, where 0 is a bad score and 1 is a good score.

Every parameter is scaled to a value between 0 and 1. These parameters can be scaled in various ways. Since the Python Emulator is only a temporary tool, the scores are scaled by rough estimations. In a later stadium more accurate scaling relations will be used. The goal of the scaling system in the Python Emulator is to be able to select nodes on a easy criteria in stead of random. The focus of the ETX implementation in the Python Emulator is on ETX routing itself and less on load balancing.

The $Score_{txCount}$ of a node is a linear relation running from a score of 1, the the txCount is near 0, and 0.5 for large txCount values. Since the simulator runs a limited set of packets, the txCount of a node will not become too high. The relation is shown in Equation 26.

where

$$Score_{txCount} = \frac{0.5}{(txCount + 1) + 0.5} \tag{26}$$

$Score_{RSSI}$ is calculated via Equation 27. Using a log-distance path loss model with a path loss exponent of 3.5, an artificial radio sensitivity and noise level of $-145dBm$ and a transmission power of $25dBm$, a range of approximately 100 meters is achieved. Very good RSSI links are rewarded with a 1, mediocre and bad RSSI links with 0.5

$$Score_{RSSI} = \begin{cases} 1 & \text{if } RSSI > -100 \\ 0.5 & \text{else} \end{cases} \tag{27}$$

The PathFactor scale is presented in Equation 28. It separates nodes mainly on higher levels, since a PathFacor score of above 20 is generally only possible on ETX levels higher than 3. This is also the region where PathFactor is more important, finding the highest successfully path on a higher ETX level is more important than close to a gateway.

$$Score_{PathFactor} = \begin{cases} 1 & \text{if } PathFactor > 20 \\ 0.8 & \text{else} \end{cases} \tag{28}$$

All these separate scorings have to be combined to on single score metric. By multiplying every scaled parameter with each-other, the combined score is also a value between 0 and 1. By multiplying instead of taking an average, one single parameter can have a big influence. For example $1 \cdot 1 \cdot 1 = 1$, and $1 \cdot 1 \cdot 0.1 = 0.1$, but $(1 + 1 + 0.1)/3 = 0.7$. This is expected to give better results since it filters neighbours with, for example, very bad RSSI or a very high txCount.

$$Score_{neighbour} = Score_{txCount} \cdot Score_{RSSI} \cdot Score_{PathFactor} \tag{29}$$

D.3 Measurement Procedure

The Python Emulator is now ready to run packets through a network and deliver them to gateways. The list of data packets is fetched from the Undagrid database. The simulator will perform 5 different simulation runs, each with different simulator settings. Each run will show a certain characteristic of ETX routing or will help verifying ETX routing design choices. The necessity of PathFactor parameter will be tested, just as the possibility to send packets via a multitude of routes instead of one.

The first run is a bounded flooding test run. In this test, a packet that is received by a node is forwarded to all neighbours with a lower ETX count. Packets will be filtered on packet duplication. In this run, neighbour scoring is disabled, since a packet is sent to all neighbours. Pathloss computation is also disabled in order to get an idea of an upper-bound on total packet transmission count.

In the second run, packets are forwarded to just one neighbour using Equation 29. However, PathFactor will be disabled. This test will be compared with test number 4 to test the influence of PathFactor scoring. The influence of RSSI and txCount on routing performance will not be tested. This is done because PathFactor is a propagated scoring metric and might be unstable. Whether or not the PathFactor helps with routing can be investigated in this test.

Then, in the third run, the same settings as run 2 are used but the number of neighbours a node selects is increased from 1 to 2. This means that not the two best nodes according to Equation 29 are selected. This test will show weather or not redundant paths should be used for packet forwarding.

Test 4 focuses on the influence of PathFactor. As said before, this test will be compared to test 2. It might be that this factor is of no importance to finding an optimum route.

The last test, number 5, will test the influence of the RSSI metric by disabling the packet error rate (PER) calculation and assuming that every packet will always be delivered to the receiving neighbour.

For each run, 49999 packets were processed by the simulator, initiated by 726 nodes. However, some messages can not be processed due to wrong GPS fixes, timing issues or other anomalies. All messages from a node are deleted if an anomaly is detected. These failed messages add up to a total number of 5202. This results in a total number of initiated packets of 44797.

Table 19: ETX routing results from Python Emulator for different runs described in Section D.3

Test Number	**1**	**2**	**3**	**4**	**5**
Description	Flooding	Base test	Dual Path	PF Enabled	PER influence
Number of Paths	many	1	2	1	1
PathFactor Enabled	No	No	No	Yes	No
PER Enabled	No	Yes	Yes	Yes	No
Successfully Initiated Messages	44797	44797	44797	44797	44797
Message In Queue Left At Simulation end	1407	1407	1407	1407	1407
Total Transmitted Initial Packets	43390	43390	43390	43390	43390
Messages Received By Gateway	42815	43303	43237	43290	43312
Messages Lost	575	87	153	100	78
mean txCount	391	107.5	152	107	108
mean rxCount	3037	52	89.5	52	52
mean txSelf	55	55	55	55	55
mean txCount/txSelf	6.17	1.84	2.51	1.82	1.78
max txCount/txSelf	1288	288	707	295	234

D.4 Results

The result of the runs are presented in Table 19 and Figure 36. Figure 36 shows the spread of several metrics from Table 19. In this table, txCount and rxCount are given. txSelf indicates the number of messages a node initiated, i.e. this node is the source node of the message. Then, the txCount/txSelf indicate how many packets a node had to forward compared to the number it initiated. Then, in Figure 36, boxes in the box-whisker plot indicate the 25th percentile, median and 75th percentile. As can be seen, the median of each box is comparable to the mean of that metric in that run in Table 19. Therefore, outliers are equally distributed from median and mean.

From Table 19, It can be seen that run one, where bounded flooding is emulated, performs the worst of all runs. It scores the worst on every metric. On average, the txCount of a node is between 2.6 and 3.7 times as high as other runs. The amount of messages lost, is between 3.8 and 7.4.

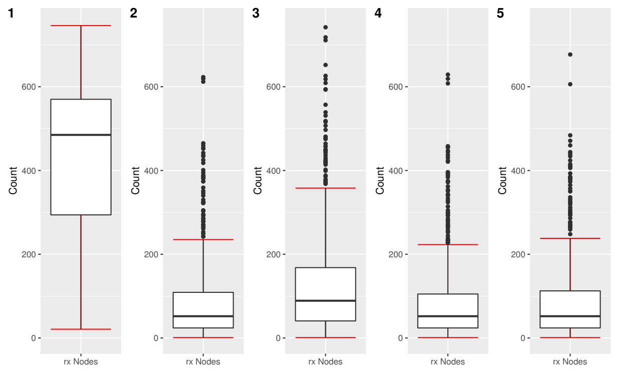
Run two is the baseline for the emulation runs. Comparing it to run three, it can be noted that mean txCount is 1.4 times as high. The number of messages lost is higher as well. It should also be noted that the maximum txCount/tsSelf is 2.5 times as high.

Comparing run two to run four show little difference. Run four performs little worse on messages lost, txCount, and txCount/txSelf.

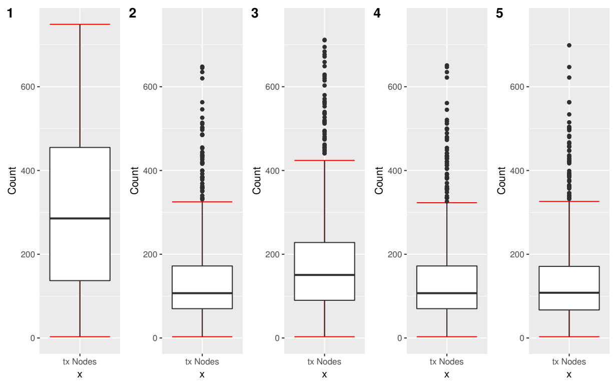
When run two and five are put side by side, it can be seen that the number of lost messages drops, as well as the maximum txCount/txSelf .

D.5 Discussion and Conclusion

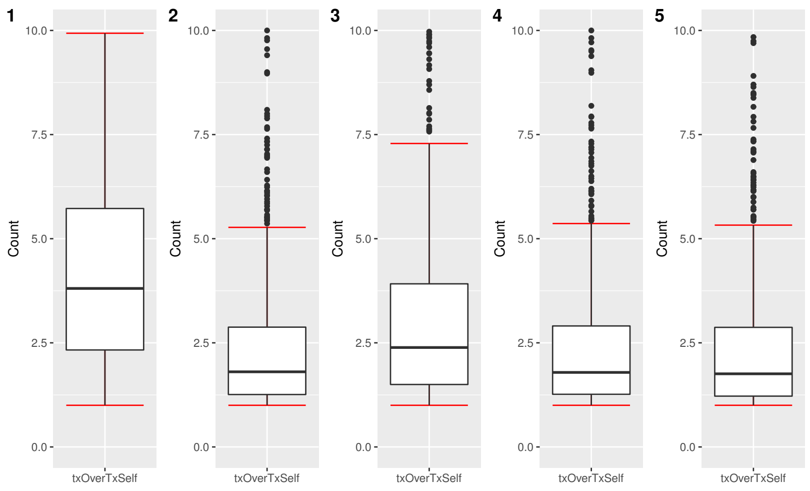
Considering the input of the simulator, it should be noted that the number of messages left in queues of nodes is over all runs equal. This indicates that in all cases cluster properties are the because since packets only end up in a queue if a node is isolated. An interesting result of the emulator is the number of packets lost in the emulator. Since MAC and physical layer properties are not emulated, no messages should get lost. This indicates a flaw in the routing mechanism. The only part of the emulator that drop packets is the packet duplication filter. The PER module only emulates packet retries and wont drop packets and therefore would not be a problem. This is also supported by data from run 1 and run 5 where PER is disabled but



(a) rxCount per node per run



(b) txCount per node per run



(c) txCount/txSelf per node per run

Figure 36: Results of five ETX routing variants. Each number corresponds to the run number presented in Table 19. These boxplots indicate from top to bottom: outliers (black dots), 1.5 times the Inter Quartile Range (red line), 75th percentile, median, 25th percentile. The range on the y axis is limited, therefore, not all outliers are shown

messages still get dropped. Since the number of packets lost is below 0.35% it was decided not to improve the Python Emulator but rather redesign the packet filter and implement this in the final design.

D.5.1 Effect of RSSI scoring

Comparing run 2 with run 5, the influence of the PER module can be evaluated. The number of packets lost is 9 lower, but the mean txCount is slightly increased. The max txCount/txSelf is significantly changed. This is probably due to a remote node that is almost isolated and the only connection to a cluster is bad. By disabling the PER module performing retries becomes unnecessary and the effect of the bad connection is not noticeable any more.

D.5.2 Effect of PathFactor scoring

By taking PathFactor in account during neighbour selection, only the mean txCount/txSelf is reduced. This is concluded by comparing run two with run 4. The number of packets lost increases and the max txCount/txSelf is also increased. The increase of the latter can be explained by looking at the score combination formula in Equation 29. By enabling PathFactor, the influence of other parameters lowers. This can result in selecting neighbours with high txCount (low score) and a high PathFactor (high score), and therefore counter acting load balancing.

The influence of the PathFactor metric is disputable. It shows both small improvements as well as small a decrease of performance on other metrics. More testing should be done to prove that PathFactor scoring does not contribute to optimum route finding.

D.5.3 Effect of using route redundancy

Considering run 1 and run 3, where in both cases the number of neighbours selected is more than one, performance is on all metrics worse than run 2. Therefore, if correct node-to-node communication can be realised, i.e. using acknowledgement packets, redundant routes are not beneficial.

From this preliminary design verification it can be deduced that flooding or multicast is unnecessary for a proper functioning of ETX routing. Only if node-to-node communication is unreliable, multicast could be reconsidered. For now, multicast and redundant routes will be excluded from the routing design.

D.5.4 Overall Conclusion

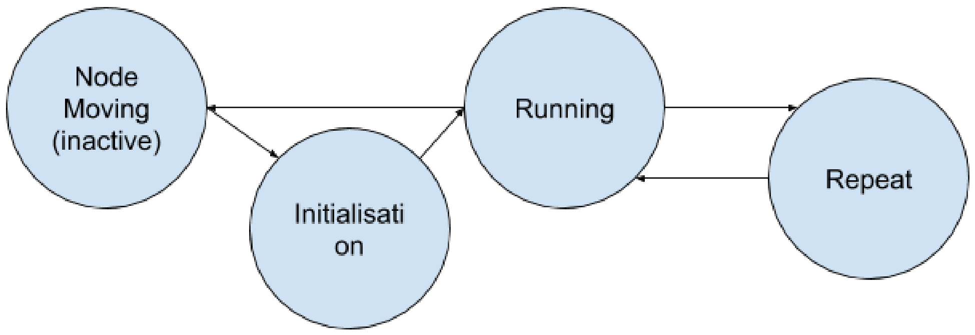
The goals of this preliminary design test was show that ETX routing works for a nomadic sensor network. Based on the results shown in Table 19, it can be concluded that ETX routing is promising enough to develop a more elaborate simulation environment. If multicast is ignored, the amount of transmissions per node is sufficiently low for a multihop network. With a txCount/txSelf figure between 1.78 and 1.84, average node contribution is not much higher than a star topology network, where this figure is equal to 1. The spread in Figure 36 however indicates that certain nodes act as critical nodes. Some nodes have a txCount/txSelf of almost 300. It is expected that this figure will balance out when sensors are repositioned. After repositioning, the probability of being a critical node again is low.

In the Python Emulator, the effect of RSSI and PathFactor scoring is tested by either enabling or disabling these parameters. This also means that the weight of every parameter is equal. This does not have to end up in a optimal neighbour selection. Therefore, in the next design of ETX routing, a weighted parameter function should be used. These weights cannot be determined analytical and should therefore be calculated brute force. How this can be achieved, will be explored later.

Due to the lack of timing properties in the Python emulator, route information retardation cannot be modelled. If a node has outdated route information, it might send packets to the wrong neighbour. Also, delay in updating can result in oscillations in update ETX levels. This should be included in an implementation where timing is modelled.

All things considered, the ETX routing design is has been proven to work correct in an routing emulator. To test routing algorithm stability and agility, a network simulator that is capable of simulating time influences should be used.

Appendix E UndaMAC states



- Node Moving (inactive)
  Node is moving and therefore it is not useful to do medium scanning, since the environment is changing all the time.
  - All hardware disabled
    - beaconing off
    - rx off
- Initialisation
  Node just became stationary
  - Beaconing on
  - RX while idle is on
  - send beaconACK upon beacon rx
  - set timer for TrxInit to move to state Running
- Running
  Node has all information of neighbour nodes. It will now run in minimal energy mode: only wake up for beaconTx and consecutive rx window
  - enable beacon
  - RX while idle is off
  - rx to ALL neighbours
- Repeat
  To resync with nodes that might went missing or due to packet misses, perform a full scan for a short period of time.
  - Beaconing on
  - RX while idle is on
  - send beaconACK upon beacon rx
  - set timer for TrxRepeat to move to state Running

Figure 37: UndaMAC states

Appendix F UndaMAC Software Architecture

The MAC layer should hook onto the PHY layer that is written in C++. Therefore, the MAC layer will be in C++. All callback functions to and from layers are shown in the MAC overview in Figure 38. To hold all parameters and settings for a node efficiently, the native Node class will be expanded and receive the UndaMesh (UM) prefix. Resulting in the name of UMNode class. This class will hold settings for MAC, such as states and helper functions, but also routing information structs and wrappers for higher layers.

All nodes will be hold in a UMNodeContainer, which as an extension on the native NodeContainer class. This class is extended to search nodes on MAC address rather than the native NS3 id.

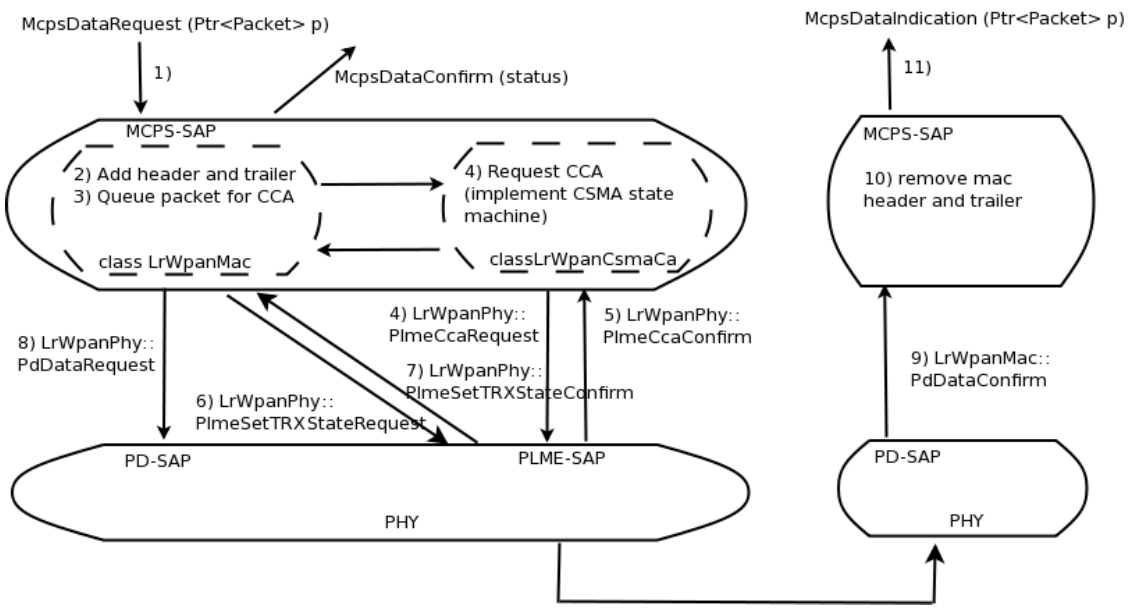

McpsDataRequest (Ptr<Packet> p)
1)
McpsDataConfirm (status)
MCPS-SAP
2) Add header and trailer
3) Queue packet for CCA
class LrWpanMac
4) Request CCA (implement CSMA state machine)
classLrWpanCsmaCa
8) LrWpanPhy:: PdDataRequest
6) LrWpanPhy:: PlmeSetTRXStateRequest
7) LrWpanPhy:: PlmeSetTRXStateConfirm
4) LrWpanPhy:: PlmeCcaRequest
5) LrWpanPhy:: PlmeCcaConfirm
PD-SAP
PLME-SAP
PHY
McpsDataIndication (Ptr<Packet> p)
11)
MCPS-SAP
10) remove mac header and trailer
9) LrWpanMac:: PdDataConfirm
PD-SAP
PHY

Figure 38: LR-WPAN MAC Layer overview with internal and callback functions [61]

Appendix G UndaMAC Delay test results

Table 20: UndaMAC packet delay test in milliseconds. Every node is only connected to the node before and node behind it, i.e. node 2 is only connected to node 1 and node 3

ETX	number of packets	total delay[ms]	delay per hop[ms]
1	50	1997.22	1997.220
2	50	4037.36	2018.680
3	50	5993.08	1997.693
4	50	8303.70	2075.925
5	50	10349.72	2069.944
6	50	12348.18	2058.030
7	50	14705.82	2100.831
8	50	16479.64	2059.955
9	50	19003.70	2111.522
10	50	21213.32	2121.332
11	50	23618.86	2147.169
12	50	25399.48	2116.623
13	50	27435.26	2110.405
14	50	29923.42	2137.387
15	50	32046.44	2136.429
16	50	34412.90	2150.806
17	50	36152.06	2126.592
18	50	38852.72	2158.484
19	50	40695.44	2141.865
20	50	42735.44	2136.772

Appendix H UndaMAC Packet Loss Test Set-up

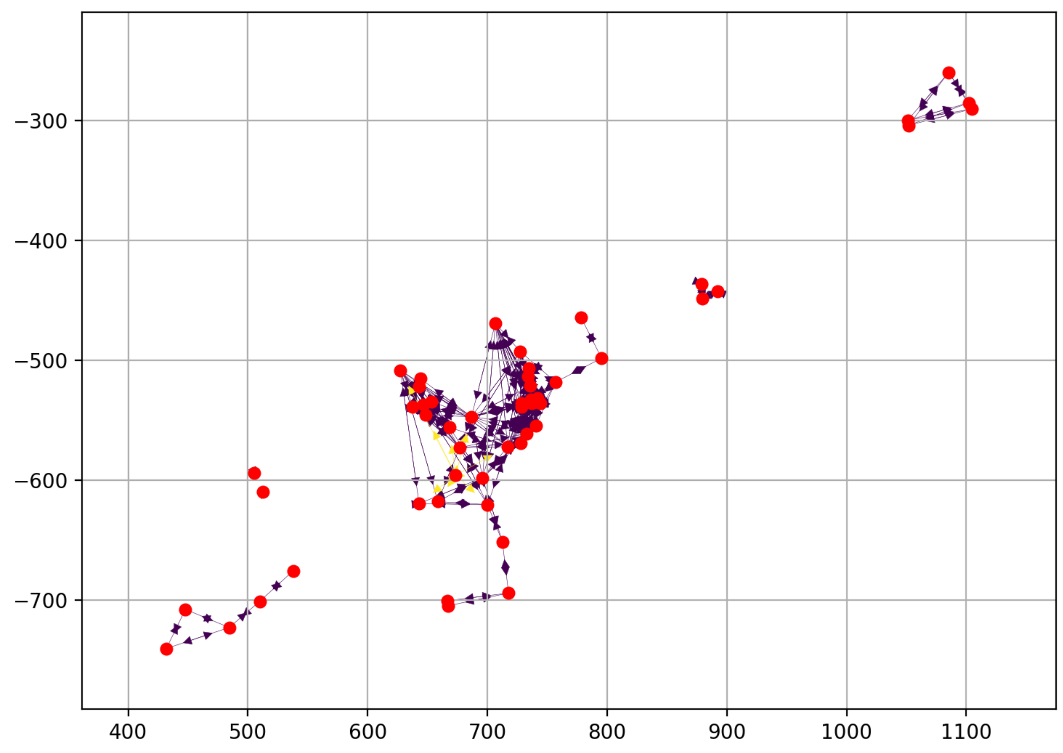


Figure 39: Test setup for packet loss for testing UndaMAC MAC layer in NS3. Environment contains isolated nodes, small isolated clusters and a connected cluster with a gateway and multiple ETX levels

Appendix I Commercial Mesh Network Solutions

Mesh networking is both an active research area for academics but also a trending topic for business solutions. A study from May 2017 gives an elaborate view on the current BLE mesh solutions. It distinguishes the solutions between standard, academic and proprietary. At the end, the overview gives a list of current challenges for mesh networking. Below one can find the results of a short analysis of available mesh solutions are found in the before mentioned studies, completed with own work.

The listed solutions are filtered on relevance to this project. Since Undagrid mesh solution should be energy efficient, certain solutions are left out of this summery.

I.1 Bluetooth SIG Mesh

At the end of 2016, the Bluetooth Special Interest Group (SIG) announced the BLE Mesh specification. It uses their own BLE stack, which uses the 2.4GHz ISM band and follows the IEEE 802.15.1 specification. It uses Gaussian Frequency Shift Keying with a modulation index of 0.5. An index of 0.5 is close to a Gaussian Minimum Shift Keying (GMSK) scheme and lowers the radios power requirements. The Bluetooth Mesh uses flooding as routing mechanism. The two main methods used are the network message cache method and the time to live method. The network message cache is designed to prevent devices from relaying previously received messages by adding all messages to a cached list. When a message is received, it is checked against the list and ignored if already present. If not already received, then it is added to the cache so that it can be ignored in the future. To prevent this list from becoming too long, the number of messages that are cached is limited by implementation. Each message includes a Time to Live (TTL) value that limits the number of times a message can be relayed. Each time a message is received and then relayed (up to a maximum of 126 times) by a device, the TTL value is decremented by 1 [51]. As far as I can find at the moment, there is no load balancing implemented. [49, 50]

I.2 Wirepas

Wirepas is platform independent and can run on a wide variety of communication protocols. It uses a hardware abstraction layer such that it is able to run on different ICs[52]. For BLE, the Wirepas stack depends on a Nordic Nrf52 chip. The maximum packet size is 102 bytes and typical delays for a battery critical solution are around 10 seconds for 10 hops. The Wirepas architecture different types of devices. The first is the gateway that is the connection to the internet. Another type of device are called anchors. These anchors are placed every 20 to 30 meters to make indoor positioning possible. They double as routing devices between nodes and the gateway. An anchor receives packages from neighbouring nodes and transmits these to the gateway. An anchor uses approximately 45uA on average. Then, we've got the nodes that are mounted to assets that have to be tracked. There exist special types of nodes. One of these is a node that does not participate in the mesh. This saves energy for that individual node. A typical node uses between 25uA and 40uA, depending on load. According to Wirepas data this will result in a typical battery life of 5 years.
Routing is done by a Wirepas developed routing algorithm that incorporates load balancing. This algorithm is one of the Wirepas unique selling points and belongs to the intellectual property of Wirepas. The current version of Wirepas, as of August 2017, needs 265kb of internal flash memory. If the Wirepas stack will be loaded onto an Undgrid tracker there would be 20kb of memory space left for the application environment. Financially speaking, a commercial license should be bought to get access to source code, SDK and documentation. On top of that, a default fee per device has to be paid.

I.3 DASH7

DASH7 is an open source wireless communication standard for applications that require only modest bandwidth. Its main focus is to provide active radio frequency identification in harsh RF environments (RFID

[53]. It operates primarily in the sub-1GHz band and focussed on long range and low power. A typical DASH7 network topology is a tree based structure that uses gateways and sub-controllers to get coverage to their nodes. This means less delay and easier networking. However, infrastructure is needed to get coverage. Its latest specification, version 1.1, was released in January 2017. In comparison to version 1.0, that was released in 2015, its main improvement is in the area of security and improvement in ease of use.

DASH7 supports a master/slave structure and is focused on light weight data transfer. Its MAC layer supports filtering, addressing (ID and 64 bit Extended ID), and a maximum frame format of 255 bytes.[54]

I.4 Serval Mesh

The Serval Project enables cell phones to build up a network such that calls can be made from one node to the other without a central cell phone tower. It also supports messaging and file transfer capabilities. It establishes this by using a WiFi mesh network between cell phones. Stand-alone range extenders can be used to give a better coverage to the end users [55].

I.5 Firechat

Just like Serval, Firechat enables cell phones to create a mesh network to send messages from node to node in a distributed manner. It supports WiFi and Bluetooth and can use LTE to connect to the internet. The manufacturer, Open Garden, has opened up their platform for external parties to make use of the Open Garden API. This allows smaller devices to take advantage of the network. TrackR, for example, uses this Firechat network to be able to transmit messages without the need of LTE network. These tags do not participate in forwarding messages, but can use the infrastructure of the Open Garden mesh. The nodes that do participate in forwarding are mainly focussed on cell phone like devices, and therefore do not run for years on a single battery [citation needed].

I.6 Filament

Filament is a company that designed a mesh network with focus on having fully decentralized communication and focuses on privacy and security. A Filament network uses a couple of components to establish a mesh network, named Telehash, TMesh, Blockname and Blocklet. Telehash is used to uniquely generate node addresses such that decentralized network discovery is possible. It uses JSON as data structure and JSON Web Encryption and JSON Web Signing to ensure privacy and security of all data. Messages can be tunnelled over almost any transportation mechanism, including TCP, UDP, BLE and IEEE802.15.4 radio links. Another component of the Filament environment is TMesh. This part takes care of the forming of a mesh network and sharing resources amongst any number of nodes. Blockname is a technique to gain trust from other nodes by reference that is recorded in a blockchain. It also uses blockchain as a currency to pay for the costs of forwarding messages. This part of Filament is called Blocklet [50].

I.7 DigiMesh

DigiMesh is a mesh networking protocol that reduces the complexity of the ZigBee protocol for point-to-multipoint configuration. One of these improvements is the ability to deal with broken link connections and sleeping nodes. DigiMesh does not require separate router or coordinator devices, each device is capable of doing every task[57].

I.8 Thread

The IPv6 mesh network called Thread is a royalty-free, but closed-documentation network focusing on smart home applications. It uses 6LoWPAN on top of Zigbee, which in turn uses IEEE 802.15.4. Thread is able to address up to 250 nodes in one mesh [58].

I.9 FruityMesh

This BLE mesh network solution is an open source mesh network that uses the Nordic SoftDevice framework to generate a mesh network. It supports connection intervals between 7.5ms and 4 seconds. It uses 2 bytes as address space, which means a theoretical limit of around 65,000 uniquely identifiable nodes per mesh. However, this field is used for group definitions as well. This reduces the number down to 20,000 uniquely identifiable nodes. FruityMesh did not put effort in implementing the best routing algorithm, since every situation favours a different routing protocol. However, it supports broadcasting and routing based on network quality metrics.

I.10 CSRmesh

CSRmesh has been developed by Cambridge Silicon Radio (CSR). It operates on top of BLE 4.0 and uses advertisement channels to communicate. There is no hierarchy in a CSRmesh network. It uses flooding to deliver its packages.

References

[1] Nicolas Hunke, Zia Yusuf, Michael Rüßmann, Florian Schmieg, Akash Bhatia, and Ninpun Kalra. Winning in IoT: It's All About the Business Processes, 2017.

[2] Louis Columbus. 2017 Roundup Of Internet Of Things Forecasts, 2017.

[3] Peter Bowen, Asit Goel, Michael Schallehn, and Michael Schertler. Choosing the Right Platform for the Industrial IoT, 2017.

[4] Ahmed Banafa. Three Major Challenges Facing IoT - IEEE Internet of Things, 2017.

[5] A Frost and Sullivan Whitepaper For. The Rise of Autonomous Device Networks What drives IoT business models ? pages 1–19, 2016.

[6] PwC. Leveraging the upcoming disruptions from AI and IoT. Technical report, 2017.

[7] Statista.com. IoT: number of connected devices worldwide 2012-2025, 2018.

[8] Richard Draves and Brian Zill. Routing in Wireless Mesh Networks. pages 114–128, 2004.

[9] Jing Sun and Xiaofen Zhang. Study of ZigBee Wireless Mesh Networks. In *2009 Ninth International Conference on Hybrid Intelligent Systems*, pages 264–267. IEEE, 2009.

[10] Statista.com. IoT average sensor costs 2004-2020, 2016.

[11] Ron Miller. Cheaper sensors will fuel the age of smart everything, 2015.

[12] Dirk Jan Lemkes. Radio based ground support equipment positioning in a mesh network at the schiphol airport platform. Master's thesis, University of Groningen, The Netherlands, 6 2016.

[13] Undagrid. Website. `https://www.undagrid.com`, 2017. Accessed: 2017-07-17.

[14] GSEtrack. Website. `http://gsetrack.com/`, 2017. Accessed: 2017-07-17.

[15] Richard Jacob Kers. *Low Energy Multi-hop Mesh Networking : Feasibility study and Master Thesis preparation.* PhD thesis, University of Twente, The Netherlands, 2017.

[16] Duncan MacMichael and Ted Hudek. Windows Network Architecture and the OSI Model — Microsoft Docs, 2017.

[17] CISCO. Internetworking Technology handbook. 2012.

[18] Matthew Syme and Philip Goldie. *Optimizing network performance with content switching : server, firewall, and cache load balancing.* Prentice Hall, 2004.

[19] Wikipedia. OSI-model - Wikipedia.

[20] Beard Beard and Willian Stalling. *Wireless Communication Networks and System.* 2016.

[21] A.G Fallis. *Getting Started with Bluetooth Low Energy*, volume 53. O'Reilly, 2013.

[22] Michael Venezia. BLE 4.1 vs. BLE 4.2 - New Features and Advantages, 2016.

[23] Lance Looper. Understanding Bluetooth 4.1, 4.2 and Beyond, 2016.

[24] Semtech. Long Range, Low Power, 2.4 GHz Transceiver with Ranging Capability datasheet, 2017.

[25] Muhyi Bin Yaakop, Izwan Arief Abd Malik, Zubir Bin Suboh, Aizat Faiz Ramli, and Mohd Azlan Abu. Bluetooth 5.0 throughput comparison for internet of thing usability a survey. In *2017 International Conference on Engineering Technology and Technopreneurship, ICE2T 2017*, volume 2017-Janua, pages 1–6. IEEE, sep 2017.

[26] Instruments National. Understanding Spread Spectrum for Communications, 2012.

[27] Ulf Eriksson. Functional vs Non Functional Requirements, 2012.

[28] Dmitri A Moltchanov and D Moltchanov. Routing protocols for ad hoc networks Ad hoc networks Lecture: Routing protocols for ad hoc networks 2 Ad hoc networks, 2009.

[29] Padmini Misra. Routing Protocols for Ad Hoc Mobile Wireless Networks, 1999.

[30] Mehran Abolhasan, Tadeusz Wysocki, and Eryk Dutkiewicz. A review of routing protocols for mobile ad hoc networks. *Ad Hoc Networks*, 2(1):1–22, jan 2004.

[31] Sjoerd Langkemper. Scalability of routing protocols in wireless ad-hoc networks. Technical report, ., 2006.

[32] Charles E. Perkins and Pravin Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. *ACM SIGCOMM Computer Communication Review*, 24(4):234–244, 1994.

[33] Hemanth Narra, Yufei Cheng, Egemen Çetinkaya, Justin Rohrer, and James Sterbenz. Destination-Sequenced Distance Vector (DSDV) Routing Protocol Implementation in ns-3. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, 2011.

[34] Jangsu Lee, Seunghwan Yoo, and Sungchun Kim. Energy aware Routing in Location based ad-hoc networks. In *2010 4th International Symposium on Communications, Control and Signal Processing (ISCCSP)*, pages 1–5. IEEE, mar 2010.

[35] Charles E Perkins, Elizabeth M Royer, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. Technical report, jul 2003.

[36] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). Technical report, oct 2003.

[37] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized link state routing protocol for ad hoc networks. In *Proceedings. IEEE International Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century.*, pages 62–68. IEEE, 2003.

[38] Zygmunt J Haas, Marc R Pearlman, and Prince Samar. The Zone Routing Protocol (ZRP) for Ad Hoc Networks, 2002.

[39] Kilinkaridis Theofanis. Hierarchical Routing Protocols Special Course In Mobility Management.

[40] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, David Moss, and Philip Levis. Collection Tree Protocol. 2009.

[41] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, David Moss, and Philip Levis. Collection Tree Protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys'09)*, November 2009.

[42] C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100. IEEE, 1999.

[43] Hristo Asenov and Vasil Hnatyshin. GPS-enhanced AODV routing. 2009.

[44] Vasil Hnatyshin, Malik Ahmed, Remo Cocco, and Dan Urbano. A comparative study of location aided routing protocols for MANET. *IFIP Wireless Days*, 1(1):11–13, oct 2011.

[45] Stefano Basagni, Imrich Chlamtac, Violet R Syrotiuk, and Barry A Woodward. A distance routing effect algorithm for mobility (DREAM). In *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking - MobiCom '98*, pages 76–84, 1998.

[46] H Lim and C Kim. Flooding in wireless ad hoc networks. *Computer Communications*, 24(3-4):353–363, feb 2001.

[47] J. Postel. RFC791 Internet Protocol. Technical report, Darpa Internet Program, 1981.

[48] The Linux Information Project. Time-To-Live Definition, 2005.

[49] Digi-Key's European Editors. Bluetooth brings its mesh networking to the internet of things, 10 2015. Accessed on: 2017-08-11 `https://www.digikey.com/en/articles/techzone/2015/oct/bluetooth-brings-its-mesh-networking-to-the-internet-of-things`.

[50] Sally Johnson. Using mesh networking to interconnect iot devices. Accessed on: 2017-08-11 `http://internetofthingsagenda.techtarget.com/feature/Using-mesh-networking-to-interconnect-IoT-devices`.

[51] Bluetooth SIG. *Mesh Profile*, July 2017. V1.0.

[52] Northstream. White paper massive iot: different technologies for different needs. Technical report, Northstream, 2017.

[53] jpnorair. *Internet of Things and Data Analytics Handbook*, volume 1. Wiley, isbn13 9781119173649 edition, 2017.

[54] Dash 7 Alliance. Dash7 alliance protocol technical presentation, 6 2014. Accessed on: 2017-08-25 `http://dash7-alliance.org/wp-content/uploads/2014/08/005-Dash7-Alliance-Mode-technical-presentation.pdf`.

[55] Serval. Serval project home page, 02 2017. `http://www.servalproject.org/`.

[56] Filament Project. Fondation for the next Economic Revolution. pages 1–14, 2016.

[57] Digi. Digimesh xbee user guide, 6 2017. Accessed on: 2017-08-11 `https://www.digi.com/resources/documentation/digidocs/pdfs/90000991.pdf`.

[58] Silicon Labs. Thread silicon labs home page, 6 2017. Accessed on: 2017-08-11 `https://www.silabs.com/products/wireless/mesh-networking/thread`.

[59] MWayLabs. FruityMesh Project Website, 2015.

[60] P.; Moreira, W.; Mendes, D Mahmood, N Javaid, U Qasim, and Z. A. Khan. *Routing in Opportunistic Networks.* 2012.

[61] Tom Henderson. Low-Rate Wireless Personal Area Network (LR-WPAN) - Model Library ns-3, 2011.

[62] NS3. Source Code 3.27, 2018.

[63] Rohde-Schwarz. LR-WPAN ZigBee, Thread, 6LoWPAN, Wireless Hart, ISA100 device testing — Solution — Rohde & Schwarz.

[64] Semtech Corporation. LoRa Modulation Basics. (May):1–26, 2015.

[65] Moshaddique Al Ameen, S. M. Riazul Islam, and Kyungsup Kwak. Energy Saving Mechanisms for MAC Protocols in Wireless Sensor Networks. *International Journal of Distributed Sensor Networks*, 6(1):163413, 2010.

[66] G P Halkes, T Van Dam, and K G Langendoen. Comparing Energy-Saving MAC Protocols for Wireless Sensor Networks. *Mobile Networks and Applications*, 10:783–791, 2005.

[67] Philipp Hurni, Torsten Braun, Markus Anwander, P Hurni, T Braun, · M Anwander, and M Anwander. Evaluation of WiseMAC and extensions on wireless sensor nodes. *Telecommun Syst*, 43:49–58, 2010.

[68] Lei Tang, Yanjun Sun, Omer Gurewitz, and David B Johnson. PW-MAC: An energy-efficient predictive-wakeup MAC protocol for wireless sensor networks. In *Proceedings of the 30th IEEE International Conference on Computer Communications, INFOCOM'11*, pages 1305–1313, 2011.

[69] Laraib Hamada and Shagufta Henna. AS-PW-MAC: An adaptive scheduling predictive wake-up MAC protocol for wireless sensor networks. In *2016 Sixth International Conference on Innovative Computing Technology (INTECH)*, pages 525–530. IEEE, aug 2016.

[70] Fernaz Narin Nur, Selina Sharmin, Md Abdur Razzaque, Md Shariful Islam, and Mohammad Mehedi Hassan. A Low Duty Cycle MAC Protocol for Directional Wireless Sensor Networks. *Wireless Personal Communications*, 96(4):5035–5059, 2017.

[71] Kuei-Ping Shih. Course on Wireless Networking, Chapter IEEE 802.15.4 mac layer overview. Technical report.

[72] J M Chambers Cleveland, W. S., Kleiner, B. and Tukey, P. A. Graphical Methods for Data Analysis., 1983.

[73] Byungnam Kahng Kwang-Il Goh and Doochul Kim. Universal behavior of load distribution in scale-free networks. *Physical Review Letters*, 87(27):1–4, 2001.

[74] TTU. Lecture notes on geospatial; principles of gps operation, Fall 2012. `http://www.depts.ttu.edu/geospatial/center/gist4310/documents/lectures/Fall%202012/4310-04%20Principles%20of%20GPS%20Operation.pdf`.