Radio Mesh Networks

Date: 26 August 2024 - 13 Jan 2025

Name	KUid
Phillip Lundin	kxg220

Contents

1 Abstract

Radio communication is a widely used technology. In our current society we usually rely on centralized solutions such as telephone towers for our phones. This creates a vulnerability for us as individuals or society as adversaries or natural disasters can ravage key infrastructures. These measures can be mitigated with Dynamic during runtime Mesh network, consisting of low power radio modules. In this thesis I have investigated the Mesh network's *Network Discovery* scales optimally given there is no packet loss, though with 500 Nodes or more, the time efficiency declines. Hop limits scales well if the hop limit is equal to the amount of Nodes with a 50% route requests retrieval rate. Existing solutions to radio Mesh network conclude their routing algorithm is sub-optimal. This is either because of organizational hesitance to change established claims, especially as an open source community or the innate feature of the *Flooding* algorithm spawning exponentially more requests depending on network size.

2 Introduction

2.1 Problem Statements

With the rise of global warming and more frequent occurrences of natural disasters, effective communication at disaster sites becomes increasingly demanding. This project focuses on creating a Dynamic Mesh network during runtime, consisting of low power radio modules that function as Nodes. This investigation will have focus on the effectiveness of Mesh networks based on different algorithms and a hop limit thereof. This will be based on three principles listed below:

- *Reliability Network Discovery.* I will investigate how effective the Network Discovery phase is getting status of the whole network of every Node. This will be implemented in section 4.2.2 and 4.4, with results in 6.1.
- *Hop Limit and Packet Loss* how many hops should a network be configured to and how many packets will successfully get delivered will be discussed in section 6.2. Additionally I will discuss solutions to this problem and its effectiveness in section 7.2.
- *Network performance* with the aforementioned factors I will compare the network's *scalability* in terms of amount of Nodes and algorithm in section 6.3.

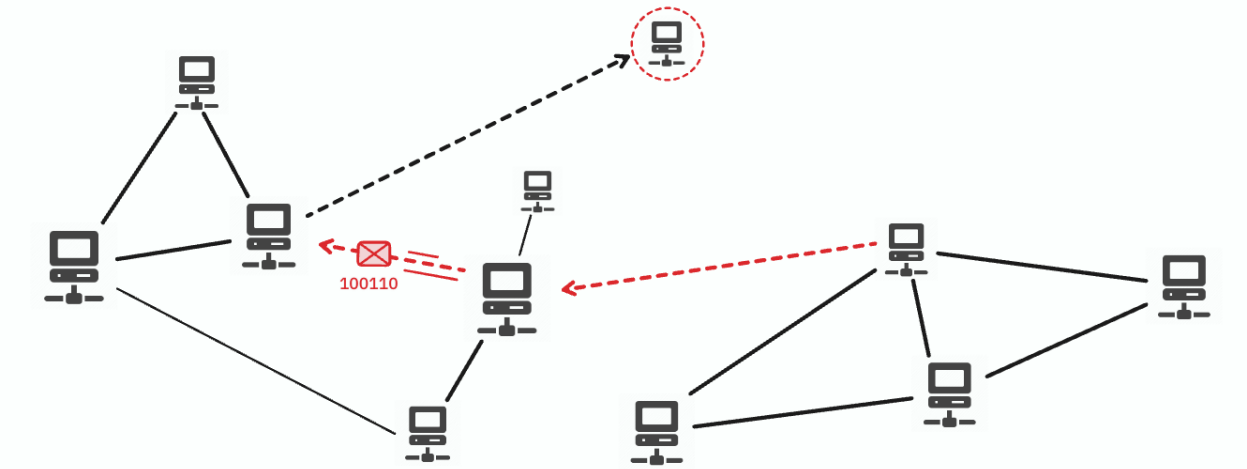

Figure 1: Diagram of a working solution of the Mesh network.

Disaster recovery is the main area in which this could be put to use. We already see fire outbreaks or medical emergencies, using mitigation procedures to establish quick communication to minimize damage. Existing technologies do exists to increase the effectiveness of communication, like walkie-talkies and this drone that can calculate urban patterns to dispatch staff effectively [5]. Though the emphasis of this project is on a dynamically evolving network. This is a mitigation measure towards pre-existing communication that have been rendered unusable and thus providing civilians or staff with a small communication device that creates a distributed network together. These devices can relay messages to each other to send messages across the network. This Mesh network will prevent situations like:

1. The internet could be administrated by the government and interfering by blocking specific IPs or entire networks.
2. Adversaries could intrude into the local technological systems and disable communication systems.
3. Telephone masts or Internet service providers (ISP)s can be destroyed by adversaries or disabled by floods or tsunamis.

The goal of the project can be seen on figure 1. It showcases a Mesh network with a message being sent from one Node to another Node. The red crossed lines depicts the path the message has already taken. Black crossed lines is the path a message is determined to take. All Nodes have their own connection, which is depicted as solid lines to other Nodes. This solid like gets determined through network discovery which will be discussed in section 4.2.1. This will enable the Nodes to send messages across the network without necessarily knowing where your receiver is physically, nor in the Node's routing table. This is made possible because of the various routing protocols Nodes can use, which will be further discussed in section 3.5. The size of the Nodes in the figure depicts transmission power (or signal range) which has many factors to consider and will be further discussed in section 7.1

3 Background

3.1 Motivation

This project is inspired by the vast amount of technologies that are built on radio. Everything from submarines' radar, Wi-Fi on computers and satellites millions of kilometers away from earth all use radio. I have personally experimented with how radio waves can transmit information and wanted to investigate further into finer detail on how to achieve this.

I want to make some clarifications. Mesh networks is not to be confused with a Peer-to-Peer Network. Mesh networks' focus is on routing and relaying messages and the physical location of the Nodes. where peer-to-peer network focuses on the algorithm implementation and how to broadcast to every member of the network [6]. Since we are focusing on the range and frequencies of radio as well as the dynamically routing protocols to relay message, we will create a Mesh network.

3.2 Mesh Principles

During the project, I want to follow some main principles for a Mesh network. This will maintain focus on what is important and to define the scope of this report. We have the following two principles:

> **Network discovery**. Each Node will scan a local version of the network a.k.a every neighbor of the network graph. This is an essential step for both every Nodes' *Local* network as well as the consolidation of a mutual *Global* network shared by each other. I will implement this in section 4.2.2
>
> **Dynamically routing**. This will make sure that message are effectively relayed through the network and delivered to the appropriate Node. This will be done through various Routing algorithms which is covered in section 3.5.

The two focus points are what I consider the most important aspects when it comes to a Mesh network: `instability` and `unreliability` networks. As we do not have a central access point (or Node), we have to compromise reliability as all the book keeping of routing tables is decentralized. That makes the Transmission control protocol (TCP) hard to work efficiently, as Nodes can come and go frequently and quickly. I will get more into why this is in section 4.2. As disaster often induce emergencies we can assume that emergency measures have to be taken into account quickly as we talked about in the problem statement 2.1. Therefore the communication has to be adaptable. We will do this with the dynamic routing principle introduced in 3.5. This adaptability is key as a system should be setup and just work. The thing you want to prevent is, making manual changes to the static routing table for a new network deployment amidst a disaster, and continuously making adjustments to the routing table because Nodes join and leave frequently.

3.3 Meshtastic

An open source community called Meshtastic combines this idea of radio with low-power modules, which uses a technology called LoRa (long range). This is the base idea of this thesis. The low-power modules will increase reliability of a network as Nodes can stay on the network for longer periods of time. The range will also increase the reliability as Nodes can reach each other from further away and therefore create a network that spans much further.

Meshtastic's existence can be summarized into one quote by themselves: "An open source, off-grid, decentralized, Mesh network built to run on affordable, low-power devices" [7]. An open source community gives the development process in the hands of the users instead of corporations that use hidden algorithms and implementations with unclear road maps. This will remove the power of corporations who have power on certain organizational outcomes. Off-grid, decentralized Mesh network which is the core of the projects specification from the Mesh Principles in 3.2.
Open source communities strive to make every implementation of a project (software or hardware) transparent. This will generate a healthy community around the Mesh networks. This enables an abundance of

advantages, mainly: The whole community can lead and develop on projects together instead of it being private and closed-source, like I talked about before. The more enthusiasm, the more people join, the more Nodes on the Mesh network, the more reliable the network will be. Meshtastic uses the term off-grid. The "grid" is the main supply of power usually from a central place distributing power to users. This usually comes in the form of power lines in urban areas, which households depend on. The decentralization part speaks into that a system should be non-monolith in the sense that there should be no central part controlling a system (or network in this case). Essentially Meshtastic has three interesting properties and one claim:

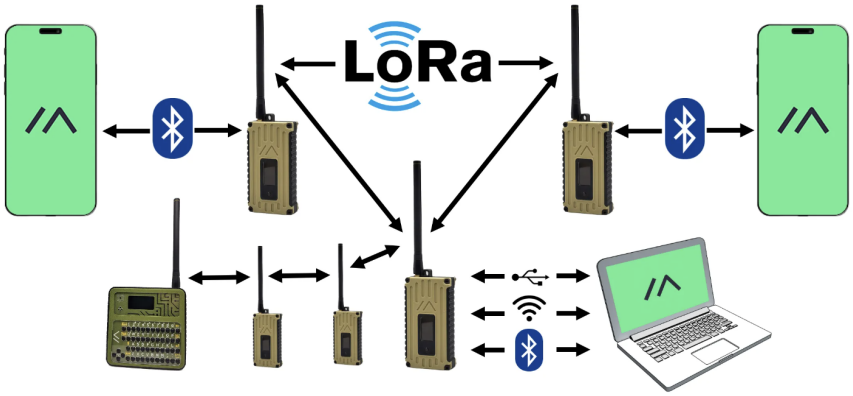

Figure 2: Schematic of how the Meshtastic technology works. Taken from Meshtastic's website [8]

Tech stack: On figure 2 we see computers in the form of laptops, or phones. These devices can connect to your radio via Bluetooth. When you want to send a message from your Meshtastic companion app, which is an app on either phone or computer, the message is relayed to the radio device's memory using Bluetooth, Wi-Fi/Ethernet or serial connection. That message is then broadcasted by the radio This provides benefits from a user standpoint: Many people carry a phone or a simple computer of some sort. Additionally the messaging is done through a User Interface (UI), which makes it more accessible to more people, further strengthening the overall network, as the whole point is to have as many Nodes as possible to relay information from one another. One downside is that a radio with a micro controller which can flash Meshtastic's firmware is required and will require some technological competences. Ideally it should also be done before you want to use Meshtastic's software as flashing is not ideal under an emergency.

Retransmission messages: There are two main layers to the retransmission. The Meshtastic algorithm will identify if a message has been received through an acknowledgment packet (ACK). If the sender does not receive an ACK within a timeout, it will retransmit the packet. This is due to lost packets in the network. Nodes might have their packets filtered away from congestion control by other Nodes. Nodes might also have an overloaded buffer and are forced to deny packets. The second phase enables eliminating packet redundancy by refraining from relaying information a Node has already seen. We will define what this includes and solutions thereof in the redundant relaying section 4.3.1.

Hop limits: Another fundamental behavior of Meshtastic's algorithm is the hop limit. [8]. For each message a radio broadcasts, it marks the "hop limit" down by one. When a radio receives a packet with a hop limit of zero, it will not rebroadcast the message. This measure is to prevent overload on the network and prevent packets going too much astray. This is implemented in section 4.2.2. The hop limit varies based on the networks. This will be further discussed in 7.2

On the Meshtastic website. They provide a claim "The routing protocol for Meshtastic is really quite simple (and sub-optimal)." [10]. I do not know what they mean by sub-optimal nor do they elaborate on the topic. One can assume that it is because of the nature of a Mesh network and presumably the usage of the *Flooding* algorithms in networks 3.5.1. What i mean by that is when dynamically constructing a network one would have to use pragmatic solutions that might not be as efficient as conventional static networks. Dynamic

Mesh networks can usually become extremely congested if not configured correctly, if many Nodes try to communicate with high frequency of messages.[1]
This sparked the idea of figuring out the what *is* efficient and what scenarios would the Mesh network principles in section 3.2 be applicable? I will discuss the results of various scenario greater depth in section 6.2 and 6.3.

3.4 Radio

Radio is the fundamental principle for sending information across mediums, such as the atmosphere or water. A lot of technology is built on top of this fundamental physics principle. We have to understand how radio work and the key components of radio.
Radio is just waves of electric impulses. These electric impulses are propagated through mediums via an antenna, which helps direct the waves in different directions (or omnidirectional) i.e. a broad transmission of signals through antennas. Signals are propagated entirely wireless. The antenna needs electricity to be able to propagate radio waves. The waves are encoded with information through modulation 3.4.1 which functions as channels to segregate communication and also lower interference. Radio do this through wave frequencies to specify channels of communication. Certain frequency bands 3.4.2 have different properties which we will also discuss in section 7.1

3.4.1 Modulation

Radio modulation is a technique to superimpose an input signal onto carrier signal to create a modulated output. Radio waves have different formats in terms of encoding and thus also transmitted differently. Therefore it is important that sender and receiver use the same technique together, as the two methods are unable cross communicate. The sender uses the carrier wave as a basis for modulating the signal before transmission. The receiver uses an oscillator to generate a reference carrier wave that matches the frequency of transmitted modulated signal. This allows the receiver to extract the original information signal from the modulated carrier wave. The two main modulation techniques (but more exist) that can be used for transmitting: Amplitude Modulation (AM) and Frequency Modulation (FM).

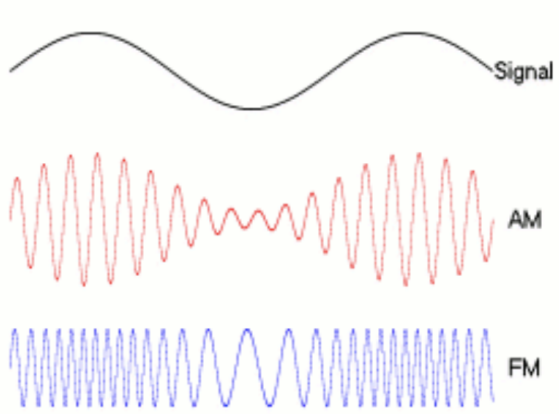

Figure 3: A depiction of how information of a signal is encoded into an Amplitude Modulation and Frequency Modulation

On figure 3 we see an analog signal, which carries binary information in the form of low (0) and high (1). For AM radio, when a signal's amplitude is low the encoded information is low (0). Otherwise the encoded information is high (1). The frequency is constant in this modulation technique. For FM radio, when a signal's frequency is low (or infrequent) the encoded information is low (0) otherwise information is high (1). Like the former, amplitude is constant in this modulation technique. The carrier wave is not visible in the

[1]This is usually to do with rebroadcasting a message which is already seen, and/or sending ACK messages back to senders which will basically double the baud rate of a network

figure but already encoded in the modulated signal of the AM and FM signals.
The CC1101 chip that I will be using in section 3.6.2 uses *Frequency-shift keying* (FSK) which is a *digital modulation method*. FSK will modulate a digital discrete signal into a modulated continuous analog signal. This conversion is called *digital-to-analog conversion* (DAC). In order to reverse, or in other words, demodulate the analog signal, we use *analog-to-digital conversion* (ADC). [2] FM is therefore a continuous analog signals superimposed onto an analog carrier signal to get a analog signal, whereas FSK is a digital signal super imposed onto a continuous analog carrier signal to get an analog signal that shifts between two frequencies to encode binary data (which is the representation of 0 and 1).

The whole point of modulating signals makes sender and receiver able to communicate on different frequencies and therefore segregate the communication channels. (e.g. 440 mHz and another on 880 mHz) If we just had a pure sinus wave, substantial amounts of interference and noise would occur for both channels and would require a greater transmission power. Modulation increases the speed of information transmission as well [13].

3.4.2 Frequency bands

We briefly touched upon how certain frequency bands have different properties in the modulation section 3.4.1. The signal from figure 3 is usually in the form of analog sinus waves which carries continuous information (for analog signals). In order to determine the frequency of which the communication channel should transmit on, depends on the frequency deviation or shift amount, determined of how much the carrier frequency changes during modulation. This means that the variation of the frequency of an FSK can only be the frequency of the carrier signal when the information bit is low, up to the carrier signal when the information bit is high. These variations are inconsequential and we usually just refer to a certain frequency for consistency.
These frequencies are the "channels" of radio. Other devices that are listening on the same frequency can pick up signal of this frequency. These frequencies are measured in *Hertz* (Hz) or cycles per second. Since we talk about bands we often categorized the frequency ranges into different spectrum. A few are classified as low, medium, high and very high etc. frequency bands. The different bands serves different purposes. On table 1 the different spectrum have different ranges. These frequencies should be chosen carefully as it

Low frequency (LF)	30–300 kHz	1-10 km
Medium frequency (MF)	300–3,000 kHz	100-1.0000 m
High frequency (HF)	3–30 MHz	10-100 m

Table 1: Different frequency bands, their frequencies in *Hertz* and their respective ranges [3].

balances between the range and the transmission speed. In general terms the lower the frequency you operate on, the slower your transmission speed is and the bigger antenna you need (compared to high frequencies with the same transmission power). This also means that higher frequencies support higher transmission speeds but decrease the transmission range. As the frequency increases, it becomes possible to transmit more signal elements per second.
Various applications use different frequencies. Examples radio frequency identification (RFID) from your phone that uses low frequency (LF). for medium frequency (MF) we can make use of magnetic resonance imaging (MRI) for patients in a hospital. For High frequency (HF) we have ultrasound for diagnosing conditions without an operational procedure. Keep in mind these technologies might overlap in frequencies or use different frequencies for different purposes (e.g. RFID uses HF and LF).

3.5 Routing Algorithms

Routing is an important aspect of networks and will be widely used in this project. We need to understand how requests are propagated through the network. Here we have *Non-adaptive algorithms* and *Adaptive Algorithms* which manipulate the routing table in different ways. We have these different types of algorithms to balance between the reliability of the network in terms of packets being received and the computation power of either an individual, central Node or every Node.

3.5.1 Non-Adaptive Algorithms

The non-adaptive algorithms are algorithms that do not adapt to changes of the *Global* network. They follow a greedy[2] approach to route and send messages to each other. This non-adaptive solution is simple to implement and inexpensive to run, but theoretically inflexible. To send messages throughout the network, with a non adaptive approach, a Node will send a message to a local known connection (a Node that is one hop away). The receiving Node determines if this request is for itself. if it is not for itself it will repeat the process of sending the message to their local connections. If the packet is for itself, it will parse the packet and evaluate the result which we will talk more about in section 4.2.2. The message might or might not find its way to the desired recipient which is heavily dependent on the network load and hop limit of the packet. The most common algorithms are as follows:

- **Flooding** has a simple definition "Every incoming packet is sent through every outgoing link except the one it arrived on" [4]. This means if a Node A has 3 ingoing connections, and Node A gets a message from connection 1, then Node A will send the message from connection 1 (assuming it was not for itself) to connection 2 and connection 3. This comes at an immense cost as the amount of messages grows exponentially and congest the network.
- **Random Walk** as the name suggests, will take a *random walk* among each known peer in the connections and keep going until it finds the rightful receiver. [4]

3.5.2 Adaptive Algorithms

Adaptive Algorithms on the other hand is flexible in a changing network. These adaptive algorithms are used to relay information through a changing network with a predetermined direction and receiver. This is done in various ways, but essentially the algorithms will change the routing table to depict the current *Global* network and/or determine the best route for a packet, based on the state of the network. Here a "direct" message can be made because all Nodes are in one shared state together, so everyone know where everyone is at all times. The Nodes can calculate these routing tables by consolidating all the information from all the Nodes in the network in various ways. We will talk more about this in section 4.4. The most common adaptive algorithms are as follows:

- **Centralized Algorithm** In this setup a single Node functions as a centralized server / router that keeps track of the state of the network. This Nodes usually does the heavy lifting of calculating every request route of every Node. [4]
- **Isolation Algorithm** This algorithm takes a greedy approach based on its own local information. They operate independently from each Node. [4]
- **Distributed Algorithms** Here all Nodes communicate together to form a *Global* network by updating each other on the network status and updating their routing tables. [4]

Every approach has its potential uses. In the context of disaster recovery a *centralized approach* is not ideal, as the infancy of the disaster still provides chaos, and therefore having a central and powerful Node established already is challenging as ongoing disasters could render it invalid or configuration period would take longer than a small Node which functions on its own and takes its own decisions like *Isolation algorithms*. The implementation of these algorithms are nuanced and does not strictly apply to what I have implemented in the implementation in section 4. For example a network that makes use of the *Reoccurring Discovery* in section 4.3.2 will dynamically determine the network with a greedy approach by pinging neighbors in certain intervals. With this dynamic network we can make use of non-adaptive algorithms like *Flooding* and we have basically created the *Isolation algorithm*. As this algorithm only makes decisions locally. How I like to see it, is that *Flooding*, will without any consideration, send out information to all neighbors in the hopes of the message to reach the recipient. The *Isolation algorithm* will assess which way, based on the *Local* network and its connection information, to whom the best decision is to send the packets to. These are in principle the same, but adaptive algorithms have more nuance to the decisions than what non-adaptive algorithms have.

[2]Make choices that looks best in the moment in hopes of it leading to a global optimal solution

3.6 Hardware

3.6.1 Raspberry pi

A Raspberry Pi is a general purpose computer produced on a single board. This makes the Raspberry Pi perfect for prototyping or having fair share of computing power in small modules.
Raspberry Pi also includes a model called Raspberry Pi Pico, which is a low power solution to minimize the power consumption of the computational device. This is ideal for a couple of reasons: Using low power, increases the battery lifetime of the device, which in turn increases reliability. And the smaller form factor of the Pico model, makes it more portable and can be moved to more strategic places. For this thesis we keep the default Raspberry model B for prototyping, which is configurable with their General Purpose Input output (GPIO) pins and also faster as their processing power is larger than the Pico model. For more determined and polished implementation can the project move onto the Pico model to reap the benefits of less power and smaller form factor.

3.6.2 CC1101 radio chip

The chip used is the CC1101 chip is useful because of it's low power consumption for its radio transmission and retrieval capabilities. It uses the 315/433/868/915 MHZ 3.4.2. I will stick to the 433 MHz range which is the ISM band and is designated for industrial use and does not require a license to broadcast on. This module uses the *Frequency Key Shifting* which was previously discussed in section 3.4.1. I will be using the datasheet from symlink `https://www.ti.com/lit/ds/symlink/cc1101.pdf`. The device's specification can be seen and bought at Amazon: CC1101 transceiver Amazon [3].

3.7 Software and packages

- radioHead library for CC110* chips `https://www.airspayce.com/mikem/arduino/radioHead/classRH__CC110.html`
- radioHead github `https://github.com/epsilonrt/radioHead` from the commit `e78d1e2`.
- Raspeberry PI OS Full (64-bit) released 19-11-2024.

[3]Link in the appendix 8

4 Implementation

4.1 Simulation of a Mesh network

As previously stated in the introduction in section 2, I want to build a Mesh network, which satisfies the Mesh network principles 3.2. As this project is heavily motivated by low level programming, though the feasibility of getting interesting results is far easier with Python thus continuing with Python. Working with integrating software to hardware implementations can be cumbersome and lengthening with low level programming languages which is especially true if you have not tried it before. I will in this implementation section on building a simulation of a Mesh network. I want to see part, why Meshtastic states that their own implementation is "sub-optimal", part to assess the effectiveness of the adaptability and reliability of the routing algorithms that I discussed in section 3.5. Therefore providing this project with a simulation of what the hardware would be running, proves the logical and programmatic standpoint of the implementation.

This simulation is an attempt to depict the real world as close as possible. This means that shared memory is not used, but rather sent through sockets instead (except network measurements). Because this is a simulation and the requests are sent through localhost, the memory used on the machine is shared and read easily, which is not the case over networks.

The project will be build using Python 3.13.0 with an object oriented approach (*OOP*). The initial idea and first attempts were to use C, to implement ideas onto the Raspberry Pi, but quickly chose to program a Python simulation instead, as it was quicker to create concrete results than C. Python creates effective prototyping and validation of attempts quickly.

The code for the simulation project can be found on GitHub under the following repository:
`https://github.com/Nidocq/RadioMeshNetwork`

4.2 Defining a network

It is important to define what to expect from a Mesh network simulation. Compromises have been made trying to simulate a real network on a local machine. I will list parameters that diverge from a real network below and explain the idea behind the design choice.

- The transport protocol for the information being transmitted, can be simulated as User Datagram Protocol (UDP). UDP will send requests to a preconceived recipient without establishing any reliability in terms of connection. This is in accordance to radio, as radio is just transmitted through an antenna in the hopes of another antenna would pick up the electrical waves on another end. The point of Mesh network is to establish communication through an unreliable network, therefore a protocol like Transmission Control Protocol (TCP) is not fit for this case as it tries to establish a reliable connection to a peer. This would both be slow and unstable, both with TCPs establishment requires many ping pongs to establish a handshake, which will likely get lost and fail repetitively. But also if a connection is established, the flow control and congestion control of the TCP protocol would not be as efficient as a reliable connection as Nodes can join and abandon the network at any time.

- `portStrengh` has been added to indicate the radius of a signal. This is equivalent to radio's transmission strength which is measured in decibel-milliwatts for actual radio. This will have two functions ideally: the `portStrength` in The simulation is depicted by how many ports a Node is away from another port. Though this is not the case with real radio. Transmission power is how far a signal can travel. the higher the power, the further the signal can travel and also the easier it is to be conveyed by other Nodes further away. It is important to emphasize that the frequency of the signal stays the same no matter the transmission power (assuming you want to communicate on the same frequency).

- One local machine is running the whole simulation. I have only experimented with localhost (i.e. 127.0.0.1). This serves to loop the requests back to my own machine, instead of sending it over a network. In essence, this is the IP connected to your own machine. This in turn makes it hard to distinguish Nodes from each other because the `serverPort` will be the only unique identifier of the

Nodes. This means that Nodes are unable to (and also restricted by OS) have the same port. This stems mostly from early design choices and having unique identifiers in the actual message would add even more complexity into the processing of header which I will further implemented in section 4.2.2.

4.2.1 Network Discovery

Network Discovery is one of fundamental aspects of the thesis as it involves every Node identifying any neighboring Nodes. I will give the semantics of the discovery phase. How a real world example would behave. Lastly - what the result of this process provides. The implementation of the network discovery can be described a few lines of code in Python: In the code in figure 4 the port scanning is wrapped in threads

```
for portScan in range(self.serverPort - self.portStrength,
                      self.serverPort + self.portStrength + 1):
    self.connections = []
    # Send ping to portScan
    if (recData == b'pong'):
        with mutex:
            if server not in self.connections:
                self.connections.append(server)
```

Figure 4: The `portStrength` implementation, on how Nodes discover the network in the function `reconNetwork` and `tryPing`

to send out a ping to each port at once, then when the Node receives a response, I will mutex lock the `self.connections` to not get any race conditions. This prevents several threads to access the same memory at once leading to unexpected behavior. Every time we will do a scan, we will reset the `self.connections`. This is because if a Node was present, and still is, on certain port we will not do anything. If they had not responded, we would remove them. This will be the same result. I will discuss this more in the Reoccurring network discovery Section 4.3.2.

Ideally as a real world example of this radio communication from section 3.4 depends on the implementation of the underlying communication protocol usually specific protocols run on certain frequencies (or frequency bands) like we talked about in section 3.4.2. If you want to communicate with other devices, you join in on this frequency and demodulate the signal from section 3.4.1. When you want to transmit, you modulate the signal and transmit it over your antenna. An example would be Wi-Fi that runs on a 2.4 GHz frequency (for this example) will identify your computer by specific packets that was created in the *Transport layer* to which the Network layer would know where to send it to, but specifically over this 2.4 GHz frequency.

The result after the Network Discovery process, we have a list of all nearby Nodes within the sender Node's `portStrength` located in the `self.connections`.

4.2.2 Local Network

As mentioned in the introduction, an important aspect of the network is the ability to both send and relay messages from- and to other Nodes. This is necessary in order to send messages to Nodes outside its own local radius. In this section I will introduce how to create requests based on a table of commands. I will also introduce how to create complex packaging of requests and the routing behavior of sending requests outside your *Local* network (`self.connections`)

The distinction between *Local* network and *Global* network from a Node's perspective is: a *Local* network is the network only this Node can see through the Network Discovery 4.2.1. The *Global network* is the consolidation of every Nodes' *Local* network which means every Nodes can see each other. This will be discussed in section 4.4.

Routing is a special case of the messages that are being sent. Therefore I will introduce a message processing as a form of header that will keep track of.

1. If the header has reached its destination
2. If the header has reached its maximum hop counts
3. Miscellaneous, such as errors and undefined behavior.

These headers will have different behavior on the underlying request, based on what their char sequence is. These sequences will always be in the front of the request. If the request is malformed it will default to adding an `ERR` header and notify the receiver. The processing will only process capitalized character sequences. Requests can also have recursive packaging which we will get into later in the section. The exhaustive table of requests options are listed below: This will solve the problem of sending messages across

`MSG` The simplest request is the MSG header. This is processed as a normal message and displayed to the user. No more processing is done.
Usage: `MSG <message>` where the message is a char sequence.

`PING` serves as a simple ping-pong functionality. This is used in the `tryPing` function which is extensively used in the network discovery phase in section 4.2.1.
Usage: `PING`

`ACK` will send an acknowledgment request back to the sender and process the underlying request further i.e. `ACK MSG Hello` will send an acknowledgment back to the sender and then process the `MSG` header afterwards.
Usage: `ACK <request>`

`ERR` is added to a request whenever the Node is unable to parse a request. The two types of errors are (currently) `Malformed request` and `Hop count reached`. A malformed request is whenever the processing is unable to find any of the headers in this table. This could be due to a typo from the initial sender request or a request which hop limit was reached.
Usage: `ERR <message>` where message is a char sequence.

`ROUTE` This is the foundation of the Dynamic routing in this section 4.4. The routing consists of `hopcount` which is the maximum 'hops' a request can take between Nodes. A `destAdr` and `destPort` which are the destination address and destination port to specify who or where the packet should ultimately go. And the `packet` is the rest of the packet that needs to be processed by the ultimate receiver.
Semantics: `ROUTE <hopcount> (<destAdr>, <destPort>) <request>`
Usage: `ROUTE <request>`.

Figure 5: Request table constructing and analyzing the messages Nodes send and receive

entire networks. Hop count management and informing the user of potential erroneous header creation as well as implementing new functionality in the future.
The `<hopcount>` is defined in the `ROUTE` header with as an `int` type. A hop count is the amount of times a request can be relayed to another Node. For every receiver of a request, 1 is subtracted from the total hop count.
The second parameter `(<destAdr>, <destPort>)` is the final destination of your request. It is constructed from a tuple format of `('xxx.xxx.xxx.xxx', 1-65550)` where every 'xxx' is a number between 1 and 255. This serves as the destination which uniquely identifies a Node, like we discussed from the definition of the Network 4.2. The `<request>` is the request processing you want to happen after the `ROUTE` request has reached the final destination. This could be in the form of a simple `MSG hello world` or another route `ROUTE 8 ('192.168.1.11', 65006) MSG Do not panic!` that will send out another `ROUTE` request when it receives the aforementioned `ROUTE` request.

Let us take an example: Assume that a Node has been created and initialized with the *Flooding* algorithm mentioned in section 3.5.1. Also keep note that this is the semantics of the routing implementation as the usage is simply `ROUTE <request>` because the function `sendData` handles the networking part.

```
ROUTE 4 ('192.168.1.66', 65006) MSG Hello world
```

This will send a request to the IP 192.168.1.66 through the initialized routing protocol of the Node, which is in this case the *Flooding* algorithm. Because it is a routing request it will send message(s) based on the routing protocol. It will send out a request to every peer it knows (its `self.connections`). in this case each request is independent from each other. When a request is received from the sender's connection, the request will be analyzed according to the request table in this section 5.
The *Local* network can be combined with *Reoccurring Network Discovery*, which will make the routing table dynamic for the *Local* network of the Node. Though a shortcoming will be not providing a global identification of the network and intelligent network path traversal like the *Dynamic Routing* is able to do. This will be introduced in section 4.3.2 and section 4.4.

4.3 Local Network Problem solving

A crude implementation of the Dynamic routing and network discovery protocols of course carries some pitfalls. Therefore I will showcase some of the considerations and fixes necessary to make, to get a functional and efficient Mesh network. When I say efficient, this means that redundancy should be eliminated and maximize the potential of the overall network in the sense that it can reach the farthest and reliably send information across the network.

4.3.1 Prevention of Relaying Redundant Messages

One of the pitfalls of a Mesh network is messages can heavily congest the network if they are not properly managed. This is a big focus point by Meshtastic, as their self proclaimed routing algorithm is sub-optimal. One of Meshtastic's 3.2 principles focuses on ignoring messages it has already seen. It does this by keeping a list of message identifiers. If the message that it is about to relay has already have been seen, the message is discarded.

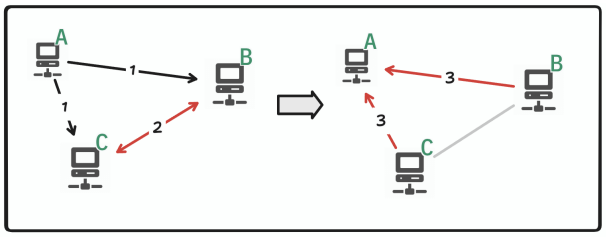

Figure 6: Node A sending message with the *Flooding* algorithm resulting in its own message returning to itself twice

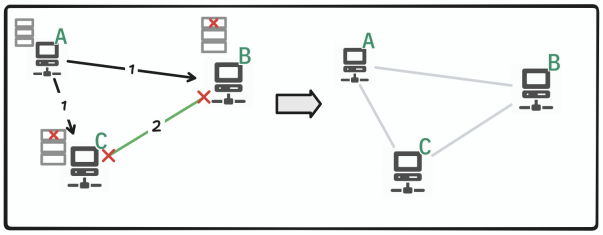

Figure 7: Node A sending message with the *Flooding* algorithm resulting Node B and C determining the messages redundant and dropping them based on their hashed messages list

This functionality is important as this removes substantial overhead on the system. This can be showcased in figure 6 and 7 which showcases a small network. The numbers on the arrows indicate the sequence of events of the requests. All three Nodes A, B, C have a connection to each other $(A, B), (A, C), (B, C)$. A message is sent on path (A, B) and (A, C) indicated by the number 1 as the first message. Path (B, C) with a number 2 indicated it is the second packet sent. Figure 6 showcase the problem of redundant messages. The red colored path (B, C) means it is the beginning of erroneous behavior. Node A sends a packet to both B and C. Both B and C will send a packet to each other hence the double arrowed path (B, C) and (C, B). This is redundant because they essentially share the same message. This redundancy is indicated by the next state of the problem on the right indicated by the hollow black arrow in the middle. The packets which

was sent to B and C are going to be re-broadcasted as the 3rd request back to the original sender A. This is a problem because e.g. the *Flooding* algorithm, which sends packets to everyone expect the receiver, nullifies this behavior which means this will also keep going forever as A now has a new message to process that it will send out to B and C again. On figure 7 we can see that the second request that is sent, is parsed but discarded by Node B and Node C based on their internal storage of hashed messages ultimately preventing the two requests going back to Node A (and more if Node A had more connections)
By having a list of hashed messages, Nodes can prevent *Redundant Messaging* which enables the network to be less congested with the Nodes eliminating at least 30% in the example we looked at in figure 6 and 7.

4.3.2 Reoccurring network Discovery for Dynamic network

In order for our network be be fully functioning for both local- and Dynamic networks 4.2.2, 4.4 we have to perform *Reoccurring Network Discovery*. This means that every Node at a certain interval (e.g. every 3 minutes) will `ping` its `self.connections` again to update the `self.connections`, which was introduced in section 4.2.1. The implementation is based on these 4 rules:

1. Every ping with no response, will continue to have no connection to this port scan.
2. Every ping with a response, will add the responders port to the senders `self.connections`.
3. Every ping with an initially connection, but received no response (i.e. timeout), will remove the connection from the senders `self.connections`.
4. Every ping with an initially connection, and receive a ping will stay in the `self.connections`. (i.e. nothing happens)

Reoccurring network discovery is essential, as it will keep up with the change of the network. As of now without reoccurring network discovery, the network will stay static indefinitely from when the Node(s) were created. This is not useful for networks that are meant to adapt to change. This speaks into the mesh principles in section 3.2 as natural disasters create instability and unreliability and is therefore important that the network adapts to these changes.

After each interval we have an updated list of all nearby Nodes within the sender Node's `portStrength` range. These changes are written to `self.connections` for every Node. This means every Node that did not respond to `ping` are deleted and Nodes responding to `ping` are added (if they are not already in the list).

4.4 Approach to Dynamic Routing

We have lightly gone over *Dynamic Routing* which was introduced in the Mesh Principles in section 3.2. In this section we will go over what it is and how it differs from the *local network* in section 4.2.2. We have already established a way to send messages by relaying messages through a *Local* network, which might reach a Node in the *Global* network. *Dynamic routing* compared to local routing, will provide more information to the individual Node about the *Global network* and what it consists of. This provides more transparency and enables a Node to send messages to a specific Node which it knows exist on the network after the two phases of Dynamic routing which we will get to later. Additionally, it provides direction or more specifically a calculated path from one Node to another. This Dynamic routing solves the static routing in the sense that it provides a direction, but requires more computation power. In this section I will cover an example of how the *Dynamic routing* works. Afterwards I will present the routing table that the Nodes make use of when sending directed messages across the network. Lastly we will talk about what the two phases include when a *Global* network is being established.

The proposed solution is a routing table object that all Nodes have. This table consist of

- `NODE` - The Node that possesses the routing table.
- `ADDRESS` - Unique identifier for a Node.
- `WEIGHT` - The weight measured in either time of message getting delivered or reliability based on SNR[4].
- `TIME` - The UTC time code for the entry update.
- `HOP` - How many hops the represented Node is away in hops.
- `PATH` - Path to take to take for sending messages to the Node. This is represented as a `sequence`.

The routing table is updated by everyone in the *Global* network based on the time code in the routing table. The time code is updated every time there has been a change to a Node's routing table. In this way all the Nodes can determine which of the information is the newest. One problem that would arise right away is how Nodes can determine which Nodes have entered and left from the *Global* network. Consider the network

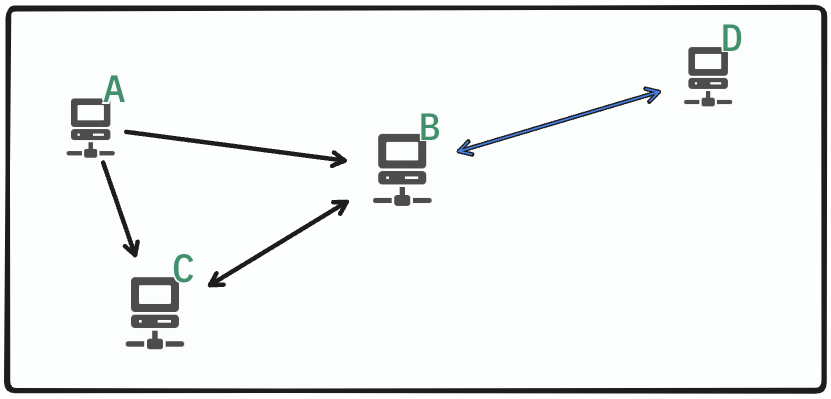

Figure 8: Example cluster for the routing table in figure 9

in figure 8. We have 4 Nodes. $network = \{A, B, C, D\}$. Where A has the connections $(A, B), (A, C)$. B has $(B, A), (B, C), (B, D)$. C has the connections $(C, A), (C, B)$ and lastly Node D has the connection (D, B). from the *Local* network in section 4.2.2, Node A would not know that D exists in the network. In this section we want a definite path from A to D. This is done with a link state algorithm. We can achieve this with Dijkstra's algorithm to find the shortest path from source Node to any other Node.

The establishment of the *Global* network has two phases:

1. First phase requires the Nodes to share and consolidate the network with other Nodes until a mutual network has been established.
2. Second phase requires every Node to calculate the shortest path to every other Node with Dijkstra's algorithm.

[4]Signal to Noise ratio - The ratio between how much of a message is mixed with noise. Anything lower than 1:1, means more noise than signal

Node A				
ADDRESS	**WEIGHT**	**HOP**	**PATH**	**TIME**
A	-	0	$\{A\}$	1736432404
B	2	1	$\{A, B\}$	1736432389
C	8	1	$\{A, C\}$	1736432420
D	11	2	$\{A, B, D\}$	1736432433

Node B				
ADDRESS	**WEIGHT**	**HOP**	**PATH**	**TIME**
A	2	1	$\{B, A\}$	1736432405
B	-	0	$\{B\}$	1736432404
C	8	1	$\{B, C\}$	1736432407
D	9	1	$\{B, D\}$	1736432408

Node C				
ADDRESS	**WEIGHT**	**HOP**	**PATH**	**TIME**
A	8	1	$\{C, A\}$	1736432406
B	7	1	$\{C, B\}$	1736432407
C	-	0	$\{C\}$	1736432404
D	17	2	$\{C, B, D\}$	1736432414

Node D				
ADDRESS	**WEIGHT**	**TIME**	**PATH**	**PATH**
A	11	2	$\{D, B, A\}$	1736432409
B	8	1	$\{D, B\}$	1736432406
C	17	2	$\{D, B, C\}$	1736432412
D	-	0	$\{D\}$	1736432404

Figure 9: Routing tables for all Nodes in figure 8

The sharing of the network state requires every Node to share their own *Local* network to their neighbors of their state of the partial *global* network. They can in turn append new entries to the routing table if the Nodes has new information about the network. This appendage will be sent back to the sender. This also needs to be propagated through the network like the *Flooding* algorithm. The reason being that Nodes further out in the network can also get the information that two Nodes have established and connect its own version to it. Here the `TIME` entry in the table is important. It is represented as an integer as the UTC time code. This can be achieved with `date +%s` in the terminal on a UNIX system. Nodes can continually update each other with a new version of their network. Therefore there are two parameters that need to be fulfilled to unanimously agree on a mutual version of the *Global* network. The newest entry (from `TIME`) of the same version of the *Global* network (the whole routing table) and the same routing table entries in their routing table. Every Node in the *Global* network. Figure 9 showcases every Node's routing table after the *Global* network.

This requires more resources than the *Local network* in 4.2.2 because of the calculation of the routing table with Dijkstra's algorithm. Every vertices (connection) the network has the computation time grows exponentially[5].

To summarize - Nodes undergo the *Network Discovery* to establish a *local network* and phase 1 starts. The consolidation of all *Local* networks is When the *Global* network is established, directed message can be sent to whomever is on the network. The messages now have a determined path to take as the Node will consult the routing table and look for the path it has to take based on the address you want to send to. In principle requests will not get lost like the greedy approach in local network are prone to.

5 Tests

5.1 General testing

I use two functions for me to effectively debug the system. `NodeStatus` and `diagnositcPrepend`. The `NodeStatus` will display all the information about the Node calling the function in a pretty and structured manner. The `diagnosticPrepend`, makes it easier to debug the system as the programmer or debugger can use it do function tracing. The function takes two strings. A `caller` and `customMessage`. These two strings have ANSI Color escape sequences [15] and will be rendered as different colors. A good format to use these functions is having the function name as the `caller` parameter. This will make it easy at a glance when debugging of which functions interact and invokes which other functions. This is especially true when working with network and threaded systems. From figure 10 we can see that the Nodes are arranged in a string of 3 Nodes. The Nodes have an identification: `Avocado`, `Fig` and `Quince`. In this example `Avocado`

[5]Based on the asymptotic running time of Dijkstra's algorithm running in $\Theta(|V^2|)$.

```
NodeStatus() Identification : Avocado_DGTZFXKY
address : '':65000
connections : [('127.0.0.1', 65003)]
portStrength : 3 2024-12-30 15:58:55.193365 -c:Avocado_DGTZFXKY '':65000

NodeStatus() Identification : Fig_CWZNIEDA
address : '':65003
connections : [('127.0.0.1', 65000), ('127.0.0.1', 65006)]
portStrength : 3 2024-12-30 15:58:55.193445 -c:Fig_CWZNIEDA '':65003

NodeStatus() Identification : Quince_GMVUOBUX
address : '':65006
connections : [('127.0.0.1', 65003)]
portStrength : 3 2024-12-30 15:58:55.193461 -c:Quince_GMVUOBUX '':65006
```

Figure 10: A capture of the individual Nodes showing their status of found Nodes (which works like a radio scan)

has a relation to `Fig` and vice versa. `Quince` also has a relation to `Fig` and vice versa. We can also see that all Nodes have a `portStrength` of 3 and all of their `addresses` are in the range of each other in a string.

6 Results

Based on the problem statement in the introduction in section 2, I will explore the performance, packet loss and reliability of the different algorithms which I have discussed in 3.5. I will discover the efficiency drop off of the current implementation introduced in 4 by testing the limits of establishing a network of Nodes together in a string. I will also investigate how the hop count from the *Local* network in section 4.2.2 will affect the performance of the routing algorithm. Finally I will discuss the overall reliability of the network from my results based on the scalability of different variables.

6.1 Reliability of Network

I briefly mentioned the problem statement of the reliability of a Mesh network in section 2.1. We want to test the reliability of the network; especially after how it scales based on amount of Nodes in a network. When I say reliability I want to test correctness of the network discovery phase and the speed taken to setting up a correct network. For the network reliability experiment, we have to define an optimal amount of request sent by the Nodes and measure the actual time taken to establish a network of the scenario. This will test the boundaries and limits of the Mesh network in terms of time and correctness.
In this scenario, no messaging takes place, this is purely for establishing a network from the network discovery phase in section 4.2.1. I will define a request as any form of attempt of connecting with another Node. This means that failed attempts of `ping`ing another Node as well as `ping`ing a non-existing Node both counts as 1 request each. If a Node `pong`s the sender, this will count as another request. The various tests' data do not carry over to the next test. They are all distinct from each other. This means a test with 2 Nodes which carries e.g. 24 total requests to form a network, will not include the 24 requests when making the next test with 4 Nodes.
In fig 11 we see a scenario which showcases 4 Nodes forming connections together in a string. The green arrows represent the successful network discovery phase which we implemented in 4.2.1. This results in the sender having the Node in their connections list for any future communication. Every Node in this string have two connections (assuming correct discovery) except the first and the last Node. To create a string you have to have a previous member and a next member with an exception for the first and the last Node as the string has to end at some point. The continuation of this process is represented by the "..." at each end of the Nodes in the figure as we will test various lengths of the string of Nodes. The grayed out "X" of

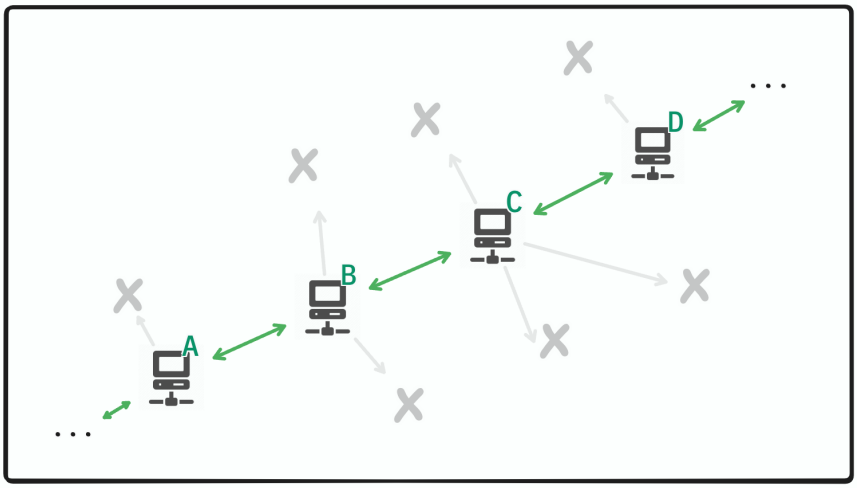

Figure 11: Scenario of `ping`ing Nodes in a string of Nodes as a part of the Network Discovery phase 4.2.1.

the Nodes are the ping requests trying to establish a connection with non-existing Nodes. The amount of requests each Node send out is based on the range (`portStrength`) of each Node. The default range (and also applied in this scenario) is 3.

I want to compare my solution in terms of a theoretical optimal solution. I have set up an optimal case of amount of requests a network with Nodes in a string follow. The formula is presented in equation 1.

$$(range \cdot 2 - 1) \cdot \#Nodes + \#Nodes \tag{1}$$

The *range* which we discussed in the section Network Discovery 4.2.1 is the `portStrength`. The *range* is multiplied by 2 because we go the negative `portStrength` and the positive `portStrength`, which in total makes it double to requests of `portStrength`. We also have -1 because we exclude the Node's own port. We multiply this tally of the *range* of requests with the amount of Nodes. Lastly we add the amount of Nodes because in order to identify a neighboring Node, all the requests that get received, a `pong` request has to be returned from our `ping` request from the sender to properly determine a neighbor. This is the minimum amount of request required to determine a *Local* network and will function as a guideline for how optimal the Mesh network is.

On figure 12[6] we have a graph of the performance of the network discovery phase 4.2.1 from the depicted figure 11. On the left y-axis we have the total amount of request, the right y-axis represents the time taken (in seconds) for the discovery phase to establish. The bottom x-axis represents the amount of Nodes this scenario was tested with. The gray solid line (#Requests sent) represents the total amount of requests that have been sent. The solid green line (#Requests received) represents the total amount of requests received by all Nodes. This is an interesting feature to track as it will let me get an idea of how many packets were lost if we compare sent packets to received packets. The equation 1 is represented by the light pink thick solid line (optimal #Requests). The blue dotted line (Time Taken (s)) represents the time in seconds for the Nodes to be established from the network discovery based on the amount of Nodes. This line and only this line corresponds to the right y-axis which is neatly named Time Taken(s) as well.

Keep in mind that the #Requests Sent number is a super-set of #Request Received. I want to showcase the difference between how many requests were used for the network discovery phase. This also means that the sum of them are not the total requests sent; the #Requests Sent is.
We can see that the total amount of request is the same as the optimal amount of requests. This tells us that the optimal requests that we defined in equation 1 fits the implementation well. The scenario is taken

[6]This plot is also showcased in appendix with the left y-axis limited to 2500 requests to further investigate the behavior and results of fewer Nodes more precisely. 16

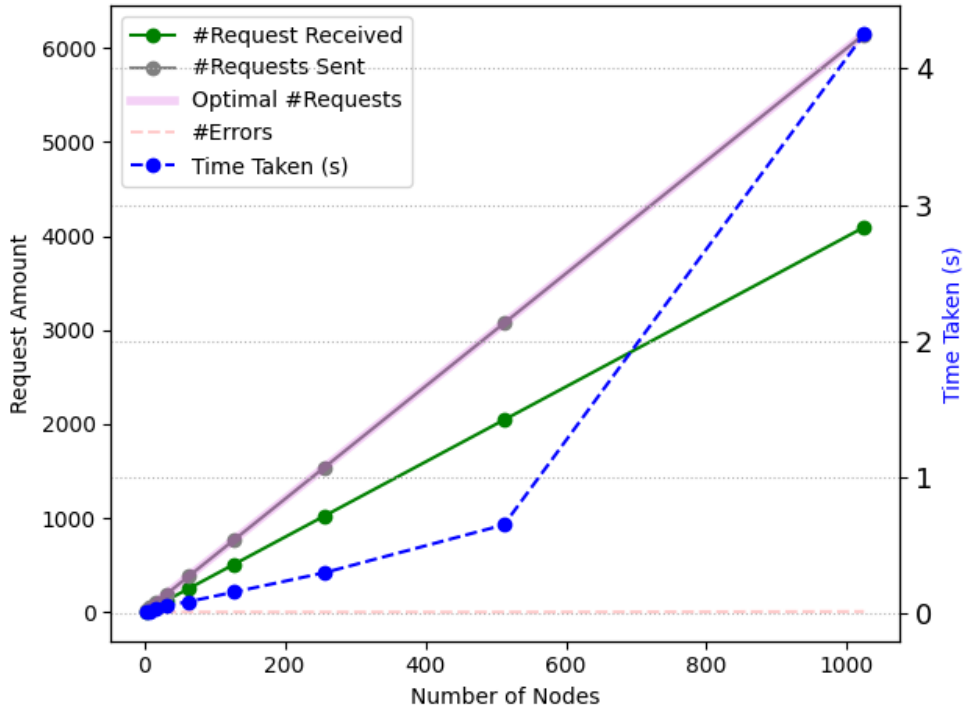

Figure 12: String of Nodes scenarios performance based on optimal defined performance (pink). Blue line represents time taken to establish a correct network from the right y-axis.

with average values over 2 experiments. Therefore the chance of noise or unexpected behavior is decreased and we have something that reflect the solution over time, instead of an isolated attempt (experiment run).

The time taken for the establishment of the Mesh network performs excellent with Nodes below around 500-600 Nodes. Assuming this simulation function as the same as the real world which we will also discuss in section 7.1. Then a *Reoccurring Network Discovery* introduced in section 4.3.2 will be able to maintain +500 devices on a network with sub 1 second update time every interval. In the case where you send a message when the *Reoccurring Discovery* interval happens then your message will be at most 1 second late. On the other hand, as you can see on the graph, the establishment time hastily grows at an exponential rate. The time to establish a network at 1000 Nodes takes a little over 4 seconds which is significantly worse than 600 Nodes.

The defined optimal request in equation 1 reflects the total requests (#Requests Sent) which deems the solution optimal according to how many requests need to be sent to establish a correct network. The reliability of the network strongly depends on how many Nodes are in the network as above 500 Nodes severely harms the time to establish a correct network because of exponentiation. In section 7.1 we will talk about how the effectiveness dwindle based on many conditions. This will affect the reliability of requests, both in the form of requests received, but also partially received requests. This will affect the parsing of the headers or provide fragmented messages to receivers. This reliability experiment is based on the best possible scenario for radio, which means packet loss is non-existent.

6.2 Hop Limit and Packet Loss

In the introduction we introduced a problem 2.1 "how many hops should a network be configured to and how many packets will successfully get delivered" which I will investigate in this section. I will present data which showcases the correlation between the error rate and the successful retrieval of a routed message based on a variable hop limit.

The experiment is configured with the following parameters:

- Amount of messages are twice the amount of Nodes and no persistent *Network Discovery* enabled.
- The size of the packets are all the same
- Errors are requests hop limit reached or other miscellaneous exception that might have occurred. They indicate that the packet was not successfully delivered.
- Clusters of Nodes with random formations. All experiments have the setting random port from 60000 to 60600, and random `portStrength` range between 5 to 100. Data created from averages of 5 experiments.

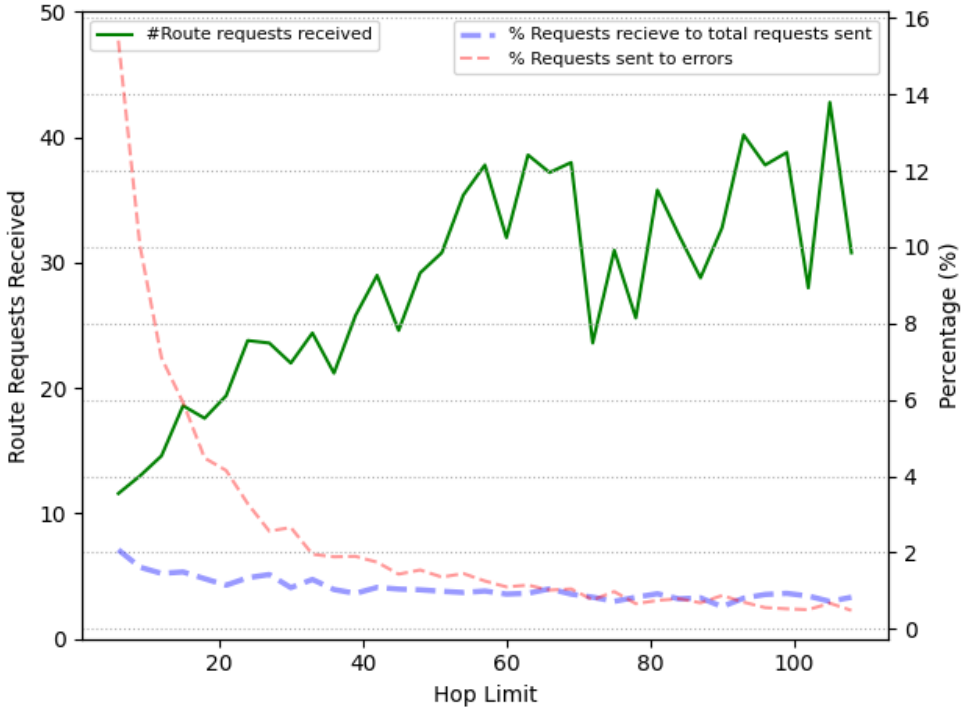

Figure 13: Correlation between successful route requests and errors with 50 Nodes and 100 messages using the random walk algorithm 3.5.

On figure 13 we have the left y-axis which represents the total amount of route requests received (represented as solid green line). On the right y-axis we have the ratio between the errors and requests sent (represented as opaque dashed red line). Lastly we have the ratio between successful route requests and total amount of requests (represented as opaque dashed blue line). On the x-axis we have the the hop limit.
As seen on the graph the percentage error rate has a steady decline, which is also to be expected, since increasingly more requests will be successfully received because of the increased hop limit. One would expect when the error rate decreases drastically, it means that more successful route requests get delivered. You would be right, but the Mesh network carries an edge case, which does not result in an error which there is a reason for. Some of the requests can land in an inert state depicted in figure 14. It showcases the a situation of a request getting stuck because it gets sent a route request which it is unable to propagate further. Node A has connections (A, B), Node B has connections $(B, A), (B, C), (B, D)$, Node C has no connections and Node D has (D, B). In this scenario Node A sends a route request designated to Node D. This initial request message sent is to Node B because that is the only connection Node A has to B. This request will further propagate (at random) the route request to Node C. Node C's range is colored in red because it is unable to send the request to anyone ultimately resulting in a loss of request which is not an erroneous behavior.

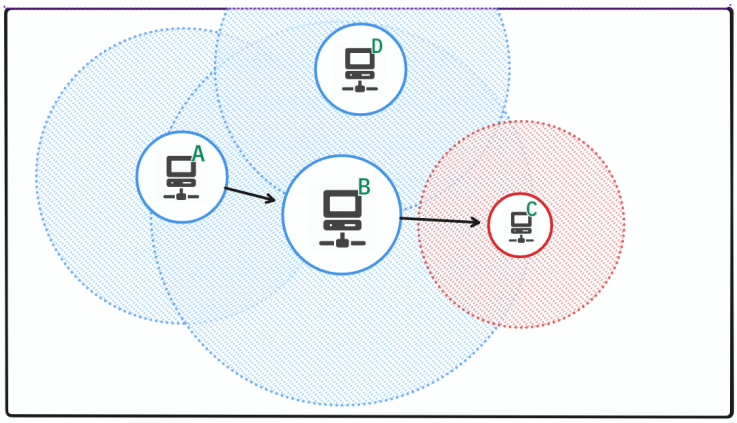

Figure 14: *Random walk* can lead to edge Nodes that have nowhere to go resulting in loss of the request.

So coming back, the reduction in errors with no apparent increase of route request retrieval is caused by requests landing in an inert state. This problem does not reflect the reality fully. From the Network discovery phase in section 4.2.1 in order for a Node to establish a connection with another Node, both Nodes have to be in range of each other. In reality Node B is unable to see Node C because Node C is unable to send a message back to Node B. In the simulation, Node B is able to respond to Node A because the socket carries sender information and is established on the same machine. This network discovery problem is also interesting to investigate on its own, but is out of scope in this thesis.
Most importantly the percentage difference between route requests received and total requests has a consistent steady decrease, meaning that the network will send increasingly more requests compared to successful route requests. This does not mean that the highest percentage is the best, but the highest percent with most amount of successful route requests. The downfall is likely due to the fact that requests have to travel further to reach the destination resulting in more total requests overall. At around the 55 hop limit seems to be optimal based on all the averaged results. This is the point where the least amount allowed requests (by hop limit) is sent with the highest success in retrieving route requests. We will see later how an increasing number will do to the this ratio in section 6.3 to see if the amount of Nodes play a big role in how we should set the hop limit.

A hop limit of the same as the Nodes itself on the network provides the best result based on the experiment in figure 13, though this does not take into account of cluster formation, as they could play a huge role of how messages are relayed and chances as paths are chosen at random.

6.3 Network Performance

In this section we will explore the scalability of the network. We talked about the results of the *Network Discovery* based on how many Nodes are present in the network in section 6.1. Therefore I will conduct the same experiment as we did in section 6.2 but with different variables; namely the amount of Nodes. This will shed light on if there are any correlation between amount of Nodes and the hop limit that should be used.

An initial experiment was conducted to showcase the behavior of a network when 120 hop limit was set with 350 Nodes over 2 experiments. The figure 17 can be seen in Appendix B. The graph showcased the route requests received growing logarithmically where the point of diminishing returns is at 120 Nodes, which coincidentally is also the hop limit of the experiment. Therefore I want to conduct another experiment where Nodes are always the same as the hop limit, to see if this would provide a better route requests received that would grow quicker than logarithmically.
On figure 15 we can now see that the route requests grows linearly instead of logarithmically by just changing the hop limit to be equal to the number of Nodes.

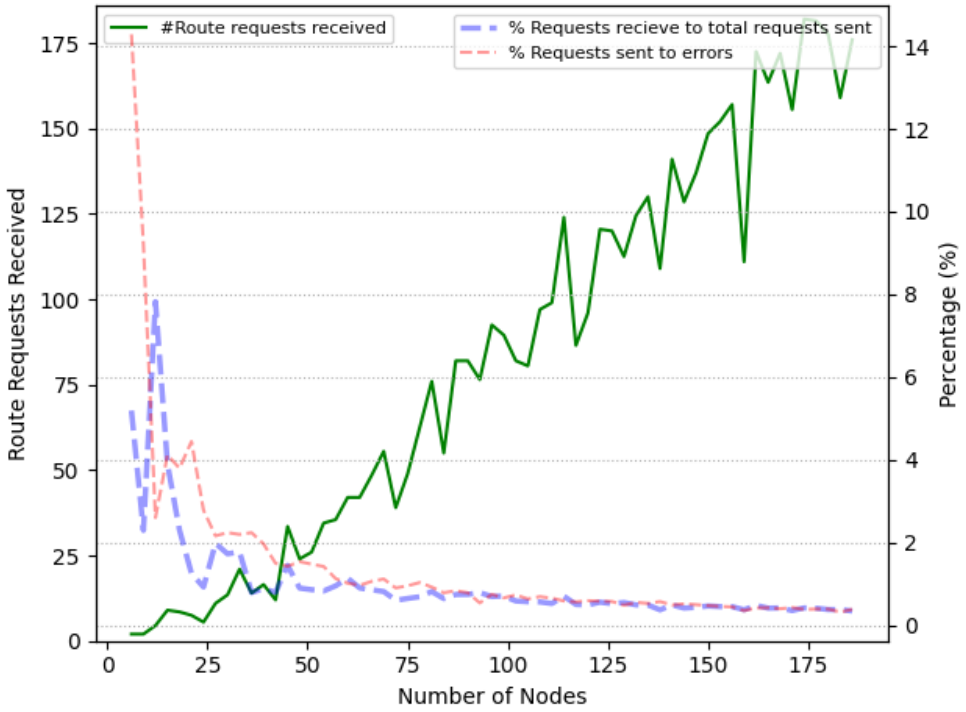

Figure 15: *Random Walk* algorithm with variable Nodes and with twice the messages sent than there being Nodes. Hop limit is equal to Number of Nodes, averaged over 2 experiments

It is the fewest scenarios where 500 Nodes will cover major disasters. if everyone carries a Node device, the number would be much bigger. If we extrapolate the data from figure 17 and figure 15, the hop limit equal to the number of Nodes is clearly preferable. It showcases a steady linear scaling compared to the amount of Nodes instead of a logarithmic. The route requests received, has a 50% chance of getting successfully received. Since twice the messages are being sent only the number of the Nodes are being received which means a 50% fall off. This is also why there are some areas of the graph where more requests are being received than there are of Nodes. This can provides further optimization if the Nodes can resend their request potentially increasing the retrieval chance.
This speaks against the Meshtastic implementation, whose hop limit is by default 3 and strictly maxed at 7. Meshtastic have other measures in place that might make them able to keep the hop limit so low which we will discuss in section 7.3.
A higher hop limit provides better results in terms of successful route requests received. Though the effectiveness begins to dwindle as soon as the number of Nodes surpass the hop limit. Keeping the hop limit dynamic to correlate to the number of Nodes yields good results consistently.

7 Discussion

7.1 Reach and effectiveness

In the real world there are many aspects to consider when you want to construct communication over long distances. In the section Frequency Bands 3.4.2 I talk about the range of different frequencies. These numbers are to be taken with a grain of salt, as aspects not concerning radio itself, but the environment plays a huge role. This source lists different conditions that plays a role in the reliability of the signal getting to its destination. [1]

1. Frequency Bands - Specifically UHF (Ultra High Frequency) and VHF (Very High Frequency) which

we discussed in section 3.4.2.

2. Power Output - High wattage transmission makes signals travel further. Not only the transmission power but how sensitive the receiver antenna is also plays a huge role in catching signals.

3. Terrain and Environment - Buildings, trees, walls and hills play a big role of how reliable the signal gets to its receiver. Using the proper frequency is key here as, according to the source, UHF can penetrate buildings, but sacrifice the range (compared to VHF).

4. Weather Conditions - Rain, snow, and fog can weaken radio signals.

5. Antenna Quality - The length, type, and quality of the antenna play a major role in the radio's range. [1]

6. Use of Repeaters - Some radios allow the use of repeaters to extend their range by re-transmitting the signal over a larger area. This is a similar concept to Mesh networks as every Node essentially function like a repeater, but relaying information to each other. [1]

Be advised that many frequency bands are restricted for military services or licensed by corporations. This is why we only transmit on certain frequency ranges mentioned in the modulation section 3.4.1 like 440 MHz or 880 MHz depending on geographical location.

A deployer of a Mesh network would have to continuously strike a balance between the range (and therefore frequency and antenna) with obstructions and environments to determine the reliability and effectiveness of a system. Not to mention the limits that we talked about in the end of the Network Reliability in section 6.1 as well as the computational limit that we talked about in Network Performance in section 6.3.

7.2 Packet loss compared to size

It is hard to determine the correct amount of hop limit. We did conclude that the hop limit should correlate the number of Nodes. The problem is when a new Node joins a network with *Local* network approach - which only see their nearest peers. It is not possible to determine the size of the network. The joined Node would have to make a guess to how big the Mesh network is. The Node could also rely on *Global* network (if already established when joining the network) to calculate the size of the network
Another huge factor that affects the successful retrieval of routed messages is the shape of the network. Throughout the thesis we have assumed the experiments conducted created partially connected graphs. Scenarios could happen where Nodes form in a straight line like I showcased in 11 or two clusters forming, wanting to send requests to each other with no connection in between them to relay the requests.
Depending on the scenario different configurations need to be installed to provide the best results as a number of Nodes correlation to hop limit will not fit all use cases. And this is especially true and important for real world scenarios where a string of Nodes or other formations might be more common than anticipated. What i mean by this, is with fully connected Mesh networks, a hop limit of the amount of Nodes seem substantial, and could be lowered.

7.3 Why is the Meshtastic algorithm sub-optimal?

A question that has repetitively come up throughout the report and also stated in section 2 has been *Why is the Meshtastic algorithm sub-optimal?*. Meshtastic state that the algorithm is simple which resembles the *Local* networks I talked about in 4.2.2 and based on the discussion of the hop count 7.2 of the results in 6.2. They are in "theory" simple because they use the *Flooding* algorithm, which basically floods the network with requests and you might be asking, "But is there a solution to this hop limit problem?". I have not been able to find a definite answer to the problem. Although Meshtastic themselves have configured the limit to be 7 with a default value of 3. They are very adamant that a hop count of 3 covers most cases [9]. This is questionable since keeping the hop limit in correlation to the number of Nodes provides good results because of the linear growth with no fall-off on successful requests received based on the growing number of Nodes in section 6.3. Meshtastic have implemented features to broaden the network as far as possible. This

is done by sending messages to the Node with the lowest SNR (signal to noise ratio) and will in principle send the message to the furthest Node in a *Global* network. This might be why this number is so low and static compared to my optimal hop limit we talked about in section 6.2.
An interesting aspect of Mesh network comes from the mathematical concept called *Percolation Thresholds* that describes the formation of long-range connectivity in random systems[12]. Thereby you can calculate the probability of requests reaching from a random place in the network to the edges (or other destinations) to get a good estimate. This would be another good way of estimating the hop limit of different cluster formations based on probabilities. The results thereof could be compared it to the simulation to see differences and similarities. Meshtastic have improved this themselves by creating "Next Hop-based routing with fallback to flooding" which delivers the next packet direction based on information from a previous successful delivery [11]. Meshtastic has still not considered removing the sub-optimal comment from their page which was committed on commit *8b5ca44* on the Meshtastic GitHub repository back in April 2021 [14]. This means it could either be that the routing algorithm still is considered sub-optimal (perhaps based on the *Flooding* algorithm) or that they have forgotten and no action has been taken on this sub-optimal claim.

7.4 Shortcomings

Based on the time limit given for a project like this, shortcomings have been made. I will provide my procedure of implementation or fixes, thus others can help along.

1. Redundancy packets in section 4.3.1. I would use a hashing algorithm to hash the messages and have every Node have a list that would check if it has seen the message before. This was a much bigger implementation than I originally thought because the routing information in the request changes every hop. Therefore I would have to introduce another request type in the request table in *Local* network section 4.2.2.

2. Limited computational power. I am unable to test larger formations of Nodes due to insufficient RAM. My device has too much information stored in RAM at one time which by then it uses swap memory, which is far from ideal. I suspect that persistent sockets fail to close from one test to another. Despite closing sockets after every run. Only around 4500 threads can be spawned on my machine. This equates to around 1500 Nodes when performing the Network discovery in section 4.2.1. This is because it is roughly equivalent to the $1500 * 3$ where 3 is the default `portStrength`. This is most likely due to the socket handling of all the Nodes. When performing various tests, it's hard to strike a good balance between waiting for all sockets and getting results fast.

8 Conclusion

The dynamic Mesh network scales optimally and reliably based on the *Network Discover* for *local* networks. When +500 Nodes establish a network, the time for a *Network Discover* increases dramatically. If the hop limit is always equal to the number of Nodes in the Mesh network, it provides a linear growth which is ideal because if the data is extrapolated it will scale well with a consistent 50% of the route requests being received. This enables disaster sites to quickly and reliably send information to each Node. Meshtastic's self-proclaimed sub-optimal algorithm is still unknown, although reasons could be that Meshtastic still consider their algorithm sub-optimal because of natural behavior of the *Flooding* algorithm broadcasting to all peers. Or it is an organizational relic that still has not been updated, despite Meshtastic optimizing their algorithm. Meshtastic also uses a different approach to hop limit, as their default value is 3, but have many similar but also different routing approaches compared to the solutions provided in this thesis.

References

[1] OneSDR - A Technology Blog. "Walkie Talkie Range – How Far Can They Really Go?" In: (2024). URL: https://www.onesdr.com/walkie-talkie-range-how-far-can-they-really-go/.

[2] Loring Chien. "What is the difference between frequency shifting keying and frequency modulation?" In: *Quora* (2020). URL: https://www.quora.com/What-is-the-difference-between-frequency-shifting-keying-and-frequency-modulation.

[3] "Frequency Range". In: *wikipedia* (year). URL: https://en.wikipedia.org/wiki/Radio_spectrum.

[4] Editorial Team of journal. "Routing Algorithm". In: *Network Encyclopedia* (2023). URL: https://networkencyclopedia.com/routing-algorithm/.

[5] Miriam McNabb. "Drones Transform Emergency Response with Advanced Communication Technology [VIDEO]". In: *dronelife* (November 30, 2024). Pulled 2024, Dec. URL: https://dronelife.com/2024/11/30/drones-transform-emergency-response-with-advanced-communication-technology-video/.

[6] Ben Mendis. "What is the difference between peer-to-peer and mesh topology?" In: *Quora* (2015). URL: https://www.quora.com/What-is-the-difference-between-frequency-shifting-keying-and-frequency-modulation.

[7] Pulled 2024, Dec. URL: https://meshtastic.org/.

[8] Pulled 2024, Dec. URL: https://meshtastic.org/docs/overview/.

[9] Meshtastic. "LoRa Configuration". In: *Meshtastic* (2024). URL: https://meshtastic.org/docs/configuration/radio/lora/#max-hops.

[10] Pulled 2024, Dec. URL: https://meshtastic.org/docs/overview/mesh-algo/.

[11] GUVWAF & misc. "Next Hop-based routing with fallback to flooding". In: *GitHub* (Oct 2, 2023). URL: https://github.com/meshtastic/firmware/pull/2856.

[12] "Percolation threshold". In: *Wikipedia* (2024). URL: https://en.wikipedia.org/wiki/Percolation_threshold.

[13] Linkyfi Team. "Understanding the Role of Modulation in WiFi Radio Transmission." In: *avsystem* (18 Jul 2024). URL: https://avsystem.com/blog/role-of-modulation-wifi-radio-transmission.

[14] Sacha "sachaw" Weatherstone. "Add docusaurus site". In: *GitHub* (Apr 2, 2021). URL: https://github.com/meshtastic/meshtastic/commit/8b5ca44531189f2386f4b4246780f6ed41cd4d7b#diff-b803fcb7f17ed9235f1e5cb1fcd2f5d3b2838429d4368ae4c57ce4436577f03fR11.

[15] Wikipedia. "ANSI Escape Code". In: (2024). URL: https://en.wikipedia.org/wiki/ANSI_escape_code.

Appendix A

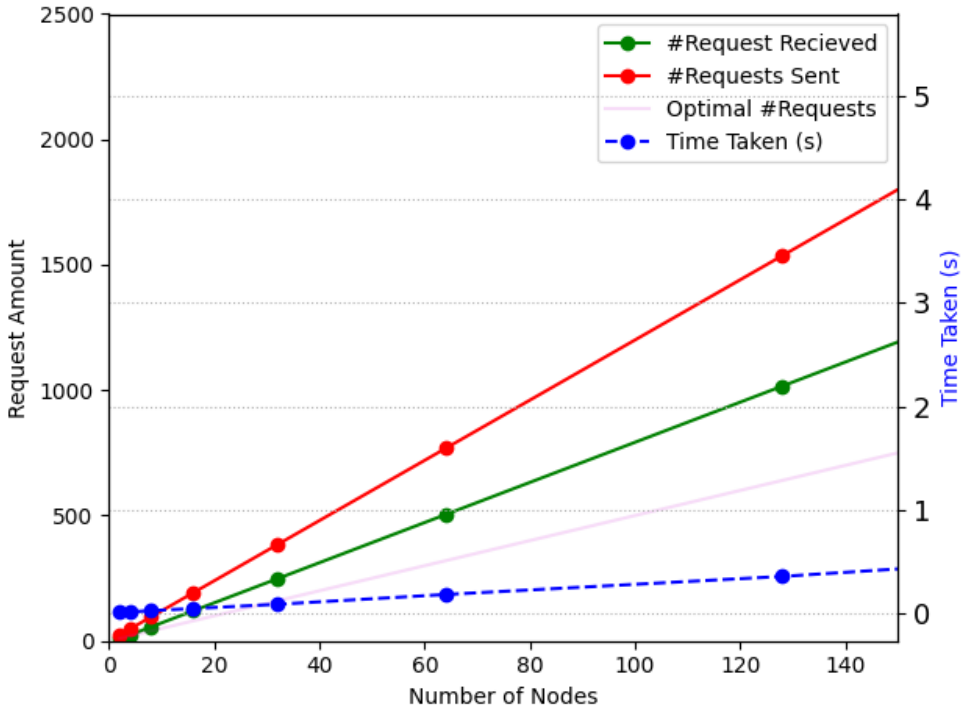

Figure 16: Same plot from Requests vs time and optimal in sec12 with the request amount reduced to 2500 on the left y-axis

Appendix B

Appendix C

https://www.amazon.com/CC1101-Wireless-Module-Antenna-Transceiver/dp/B0CP78Y5TT/ref=sr_1_13?
adgrpid=133370997569745 9&dib=eyJ2IjoiMSJ9.XZSSSyfdV50ShzaxXXO6sr16YuIMMmU9fV5WYAeJgd7hz34pR2Y7pkMGtKl0t
PzASHIG0xevafvr4Rzv9b_lOn47BheRoG_0FrRJ2cG2Nr5jc9_juJBi5kGx4EY1hNPX41K8GtnKQq4wHWPVnqmuuQcyS5TH5hvq9FR5
tMzzT_GfvAkh4bAgrXV3LZWsj22JBHH72nwH9899sh0&dib_tag=se&hvadid=83357128046722&hvbmt=be&hvdev=
c&hvlocphy=1054&hvnetw=o&hvqmt=e&hvtargid=kwd-83357901031700%3Aloc-53&hydadcr=4960_13164893&
keywords=cc1101+wireless+module&msclkid=61a958c9dcd51945dfcb3f91d7ab7248&qid=1733823363&sr=
8-13

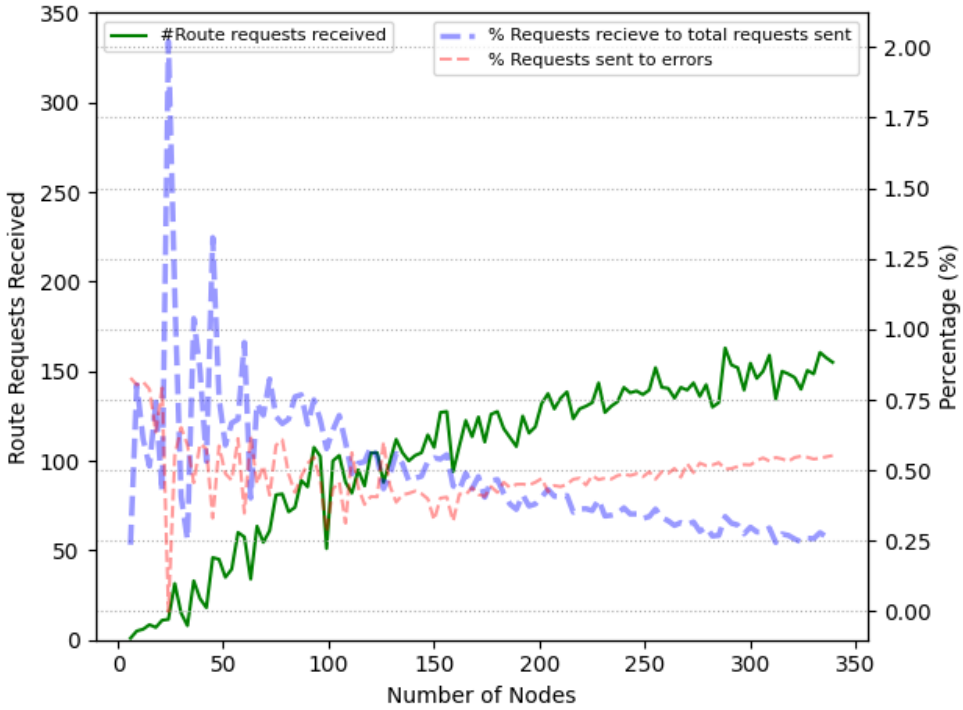

Figure 17: *Random Walk* with variable Nodes, with twice the messages sent than there being Nodes. Hop limit is equal to 120, averaged over 2 experiments