LoRa mesh using LoPy4 microcontroller

Vito Vekić

School of Computing
Final Year Engineering Project

May 6, 2022

Abstract

This paper will outline the planning, creation, and implementation of a mesh network using LoRa technology. The LoPy4 device is utilized, which is a micropython-based microcontroller that have LoRa and Bluetooth functionality. This is an immensely powerful technology if implemented in rural areas or areas with no connection. In worst-case scenarios, one node can last 2 weeks with one battery (li-po 4400mAh); in best-case scenarios, the battery can power a node for up to a month and a half (calculated using (Pycom, n.d.),(Neutert, n.d.)). If one node runs out of power, it will be replaced with another close to it. Such mesh technologies are utilized by the UK military.

Table of Contents

List of Figures

Acknowledgements

I would like to thank Steven Ossont for guiding me through this software project and supplying the devices necessary for it. I'm also thanking my good friend Marko for pointing this idea out to me, and for all the supporting people around me during these shaky times.

Chapter 1

Introduction

1.1 The plan

Nowadays, mobile networks are in everyone's pockets making communication a thoughtless process. The plan is not to reinvent the wheel, but to build upon such paid services with a cheaper alternative to intercommunication. A mesh network would be a perfect framework that could help us to achieve such a goal. The goal is to create a system that would allow users to access the network by purchasing a single node and having access to every other node within a certain range. The more people start using this technology the better and more usable it will become. With the node, the user will need to have a smartphone device with a companion app that would allow them to communicate with the node. One node in the network could be set up to act as a border node and communicate over the internet with other mesh networks allowing interaction with even more users.

1.2 Background

1.2.1 Micropython and Pycom

In 2013 MicroPython was created after a successful Kickstarter campaign by Australian programmer Damien George(George, 2016). When developing the software, he made STM32F4-powered python compatible devices called "Pyboard"s. In 2015 a Dutch company saw potential in the technology so they Kickstarted "WiPy" which is a modified Pyboard with Bluetooth and Wi-Fi functionality("The WIPY: The internet of things taken to the next level", 2016). Afterwards, they created more modules including the LoPy4 which is focused on in this report.

1.2.2 LoPy4 and Pygate

Pycom describes the module as "The LoPy4 is Micropython-programmable quadruple bearer board. It works with LoRa, Sigfox, Wi-Fi, and Bluetooth, making it an exceptional enterprise-grade IoT platform." (Pycom, 2022) Because of its LoRa functionality, it is a suitable candidate for creating a mesh network as it only needs the program that will implement some rules of interaction and roles in the mesh (Child, Leader, etc.), and has Bluetooth functionality so it can interact with a phone/web app. To interact with the device an expansion board or a USB serial converter is needed. In this case, Pygate was used. It's an extension board that comes with a battery input, 8-channel LoRaWAN Gateway, and much more. We will only be using the programming over USB features of it.

1.2.3 Mesh

The mesh network displayed in Figure 1.2 is a network in which devices or nodes are linked together, branching off other devices or nodes. These networks are set up to efficiently route data between devices and clients. They help organizations supply a consistent connection throughout a physical

Figure 1.1: LoPy4 device on a Pygate extension board

space. (Chawla et al., 2015) Pycom created Pymesh, a LoRa mesh technology currently in alpha production.It offers ad-hoc communication over raw-LoRa radio and security on multiple levels. It encrypts the data using AES 128bit key and can use RSA at application level offering private communication channels over Pymesh. The Protocols for the mesh that were implemented were based on Thread by ThreadGroup. With more devices in the network the mesh will stop being a complete mesh and start being a partial one where each node is still connected to every node, but for some nodes to communicate between each other they will need other nodes in the network to forward the data packets.

1.2.4 Flutter and Dart

Flutter is Google's portable UI toolkit for crafting beautiful, natively compiled applications for mobile, web, and desktop from a single codebase.("FAQ", n.d.) Apps are written in the Dart language which is also created by Google. Dart is an object-oriented language with c style syntax used for creating

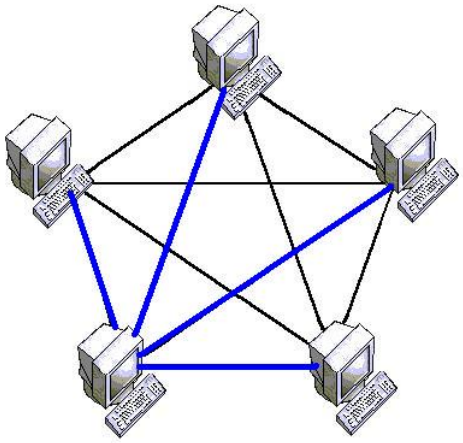

Figure 1.2: A full mesh network
Image from: http://webpage.pace.edu/ms16182p/networking/mesh.html

a server and desktop applications(Google, 2022). To the flutter framework packages such as flutter blue were added. Packages such as that will help us communicate with the microcontroller's Bluetooth functions. Flutter will allow us to create screens easily and give functionality to the user interaction with the application. The main screen of the application is the screen that asks the node to return data that holds a list of its neighbours and allows us to communicate with a specific one.

1.2.5 Drawbacks

Most implementations of technology have some drawbacks that need to be considered while developing a product; Implementing LoRa comes with a lot of things to consider. One of the main problems stems from bandwidth which in LoRa depends on the spread factor. Spread factor (SF) controls the chirp rate, and thus controls the speed of data transmission. A lower spreading factor means faster chirps and therefore a higher data transmission rate("Spreading factors", 2021). Another issue with LoRa is its range, as it is very conditional. The maximum data rate is 5.47 kbps, and the maximum communication range is 2 km in non-line of sight (NLOS) conditions and 20

km in line-of-sight (LOS) conditions(Kim et al., 2020). Another drawback to the entire system is the amount of memory on the LoPy4 device, so different methods of saving data will need to be implemented. If there are a lot of people always using the system, it would generate a lot of traffic. This is only a problem if the system is implemented on a macro scale(Zhu et al., 2019).

Chapter 2

Aims and Objectives

2.1 Aims

This paper aims to produce an application that will interact with the hardware and contact every other device connected to the node. The app will need to be able to recognize who is sending the message, who is receiving the message and if the message has been received, and will be implemented through the following steps:

- The technology will need to be accessible to everyone who owns a node and a smartphone.
- The app will be accessible to everyone who has a google play store or App store (For apple users)
- The Lora will be used to make nodes communicate with each other, for a start we will assess Raw LoRa communication from one node to another.
- When a node receives data, it will blink and that will give the user a visual that a message is waiting to be received, and they should check their device.

- The project will implement Pycom's Pymesh and with that give the user the ability to contact other nodes within the network.
- Implement Protocol buffer to make the size of the packets smaller.
- The nodes can be connected to the internet using border nodes, but in the regular case, it will not be necessary.

2.1.1 Assumptions

- The end user will have a smartphone
- The end user knows how to connect the devices using Bluetooth
- The resulting artifact is not finished enough for practical use, and is a proof of concept.

2.2 Constraints

There are a lot of constraints on this project. A major one is time as the project needs to be done by 5/5/2022 and so the scope of what will be done will be appropriate. To achieve such a goal the quality of the product will be sacrificed to produce a artifact. Another major constraint is the lack of resources necessary to fully implement such network (Batteries, Extension boards, and more nodes to fully test all the features).More devices have been requested, but with no avail. This then constraints the author to only use the provided equipment and make do with what is available. With the sacrifice of the products quality risk management will become more relevant as potentially more severe problems might show up.

2.3 Objectives

2.3.1 PyMesh

One of the aims of the project is to create a network of LoPy4 devices that can communicate with each other using mesh functionality that Pycom created called Pymesh. With this part done with two nodes, we could have a complete mesh network as each node will speak with each other. With more nodes, the network grows and gains the ability to multi-hop which allows one node to have the range of every node connected to the network. The network protocol implemented within Pymesh Is OpenThread which is googles implementation of Thread by Thread group.

2.3.2 Protocol Buffer

To make the data smaller we will use protocol buffer (Protobuf) which is a mechanism for serializing structured data in a forward compatible and backward compatible way. It is similar to JSON, except it is smaller and faster, and it generates native language bindings ("Overview protocol buffers google developers", n.d.). Because we need to use micropython with large memory constraints we will use a specific library called micropython-uprotobuf(jazzycamel, 2018) that will allow us to upload the full package in the lib folder without taking too much space on the microcontroller. The problem with this protocol is that the data is not encrypted so on top of that the data will be encrypted with public-key encryption. There exists another more upkept version of protobuf that wasn't implemented due to time constraints(Dogtopus, n.d.)

2.3.3 Mesh connect app

An application made in flutter will be the last thing to be created and will interact with the LoPy4 microcontroller using flutter blue. It will send data about the user, node name, message, and receiving node data. Using that

the receiving node will be able to gather enough data to figure out where it is coming from. And if someone is using the Pytrack extension board they will be able to send their exact GPS data. The advantage of this application is that it will be able to communicate with people even without Wi-Fi, and it will be able to connect to the people everywhere using border nodes that are connected to a web server. If the user chooses to it even could interact with the node and update its data, such as name and comment on what kind of node it is.

2.3.4 Border Node

One of the nodes in the network will have the ability to connect to the internet to send and receive node data to a server. With this implemented every mesh with a border node will be able to communicate with every other mesh that has a border node. This will make the messaging part of the application more useful as there will be more users to interact with.

Chapter 3

Literature Review

3.1 LoRa

Lora is a powerful communication technique for realizing IoT applications in a large-scale area. As the name says it is a Long-Range physical property radio modulation technique. It is based on Chirp Spread Spectrum (CSS) Technology(Dunlop et al., 2019). It encodes information on radio waves using chirp pulses. A full LoRa frame consists of a Preamble which is used to keep the receiver synchronized with the transmitter, two types of physical layer headers (PHDR): Explicit headers and implicit headers(Jiang et al., 2019). When speaking with the explicit header, PHDR holds the length of data information, error correction code, and the sign of whether the end of the frame carries the cyclic redundancy check (CRC) of the data load. The PHDR is followed by the payload and to finish a message with a CRC if it has one. In Europe, LoRa uses the license-free sub-gigabit radio frequency band of EU868 (863-870/873 MHz). The transfer rate of the device is based on the spreading factor applied. Spreading factor (SF) affects the communication performance of LoRa devices which use the SF of between 7 and 12. A large SF increases the time on the air which increases battery usage, but it improves the range of the devices. Based on all these factors LoRa has a low data rate

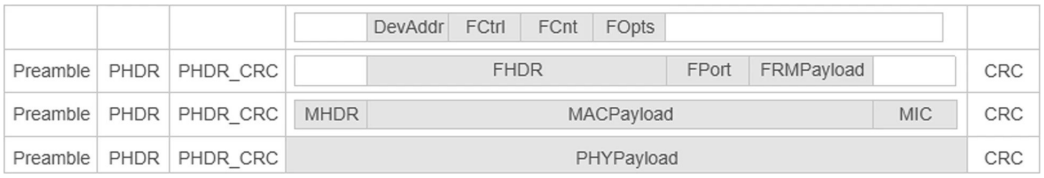

Figure 3.1: LoRa Frame structure
figure from: (Jiang et al., 2019)

(27kbps with spreading factor 7) and a long communication range (2-5km in urban areas and 15km in suburban)("Spreading factors", 2021). On top of LoRa, the Lopy4 devices have Bluetooth, Sigfox, and Wi-Fi functionality.

3.2 Bluetooth

Like LoRa, Bluetooth uses radio signals to get data across devices, but Bluetooth is a short-range wireless technology standard. It was developed in the 89s by Nils Rydbeck with the purpose to develop a fully wireless headset. It runs on frequencies between 2.402 and 2.480 GHz, or 2.400 and 2.4835 GHz (Muller, 2002). It has been synonymous with headphones, but nowadays most wireless devices use it for wireless communication. Bluetooth devices communicate directly which separates them from Wi-Fi devices that need to send their traffic through an in-between device such as the router.

3.2.1 ATT

Attribute Protocol(ATT) defines the communication between two devices playing the roles of Server and client. The server maintains a set of attributes. An attribute is a data structure that stores information managed by the GATT. The client can access the server's attributes by sending requests, which trigger response messages for the server. A client may even send commands to the server to write attribute values.(Gomez et al., 2012)

3.2.2 GATT

GATT stands for generic attribute profile. It defines a framework that uses It comes into use once a connection is set up between devices, meaning that you have already gone through the advertising process governed by GAP. GATT manages information for ATT and is a protocol that works on top of ATT. The client or server roles are determined by GATT and are independent of the slave or master role. (Gomez et al., 2012)

3.3 Ad-hoc Mesh

The wireless mesh is a network made up of nodes that use radio to communicate. They are organized in a mesh topology. A mesh refers to the intercommunication between devices or nodes. This technology was first developed for military applications so that every node could dynamically serve as a router for every other node. This is best as if one node gets destroyed the nearby one will replace the damaged one so that the network never fails (If it has enough nodes). The mesh client that this project will use is the LoPy4, but it can be anything that has wireless connectivity, it all depends on what kind of wireless mesh you are trying to achieve. There is another type of wireless mesh network called wireless ad hoc network. An ad hoc network is fully decentralized and does not rely on pre-existing infrastructure. In such networks, each node takes part in routing by sending data from one node to another. There are multiple types of mesh topologies such as the full mesh and a partial mesh. In a full mesh, every node is connected to every node.(Mohammadi Zanjireh et al., 2013) In a partial mesh network, every node is connected to another node, but every node is not connected to every other node.

3.4 Thread

Thread is an IPv6 networking protocol designed for low-power IoT devices in an IEEE 802.15.4-2006 wireless mesh network, commonly called a Wireless Personal Area Network (WPAN)("What is thread?", n.d.). In a thread, network nodes are split into two forwarding roles which are: Router and End devices. A Router serves the role of forwarding packets for the network device, supplies provisioning services for devices trying to join the network, and keeps its transceiver running all the time. The end device serves as a device that primarily communicates with a single router. The end device does not send packets to different nodes in the network and can disable its transceiver to reduce the usage of power.

3.5 How the mesh works

This section is compiling information on how Thread works. The methodologies on how it works are documented on OpenThreads website, Threads website, and thread's White papers.(Google, n.d.),("What is thread?", n.d.),(Group, 2020)

3.5.1 Device Types

Nodes make up several types; the most used ones are Full thread devices and minimal thread devices.(Group, 2020) A full-thread device Is always on and is subscribed to all the router's multicast addresses and maintains IPv6 address mapping. There are three types of full-thread devices which are: The router, A router eligible end device(REED) that can be promoted to a router, and a Full end device (FED) that cannot be promoted to a router. All the FTD devices can serve as either a Parent (Router) or child (End device). Minimal thread devices do not subscribe to all the router's multicast addresses and send all the messages to their parent. The two device types are: Minimal end device is a device with the transcriber always on and does not need to poll for

messages from its parent, and the Sleepy end device is normally disabled but wakes up occasionally to receive messages from its parent. Minimal thread devices can only work as end devices. When a REED device is the only node in reach of a new End Device wishing to join the thread mesh network, it can upgrade itself into the router. If a router does not have children, it can downgrade itself to an end device.

3.5.2 IPv6 Addressing

IPv6 is the next-generation internet protocol that is meant to replace IPv4 when the pool of available IPv4 addresses dries out("FAQs", n.d.). It includes internet protocol security IPSEC within the protocol so depending on the network infrastructure may offer more security over IPv4. The mesh uses IPv6 addressing to decide the id of each device in the mesh.(Group, 2020) There are three scopes in a thread network that use unicast addressing: Link-local which are all the interfaces reachable by single radio transmission, Mesh-local which are all the interfaces within the same network, and Global which are all the interfaces accessible from outside the Thread network. These interaction are displayed in Figure 3.2. Multiple IPv6 unicast

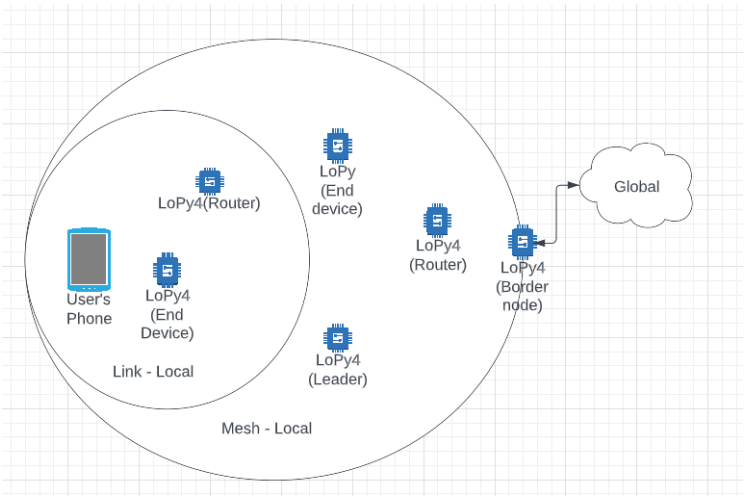

Figure 3.2: A full mesh network
Made using: (http://www.lucidchart.com)

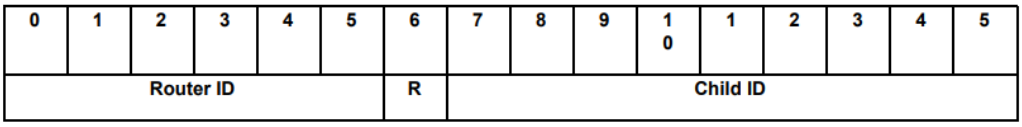

Figure 3.3: Thread ROLC16

addresses find a mesh device. Each has a different function based on the scope and the use case.(Group, 2020) One of the most used ones is Router Locator (ROLC). The ROLC finds an interface based on its location within the network topology. All the devices within the mesh network have been assigned a router id and a child id. Each router holds a table of all its children which uniquely finds a device within the topology. (Google, n.d.) Each child's id corresponds to its parent (Router). Each device joining the Thread network is assigned a 16-bit short address. For routers, this address is assigned using the high bits in the address field with the lower bits set to 0, including a router's address. Children are then allocated a 16-bit short address using their Parent Router's high bits and are appropriate low bits for their address. (Group, 2020) The ROLC displayed in Figure 3.3 is part of the Interface Identifier (IDD), which corresponds to the last 64 bits of the IPv6 address. IIDs can be used to identify some types of Thread devices. The IDD, combined with the Mesh-local Prefix, results in the ROLC.

3.5.3 Network discovery

Thread networks are identifiable by three unique identifiers:
2-byte Personal Area Network ID (PAN ID)
8-byte Extended Personal Area Network ID (XPAN ID) and a human readable network name (Group, 2020). When creating a new network, or searching for an existing one to join, a Thread device performs an active scan for 802.15.4 networks in range. First a device broadcasts an 802.15.4 Beacon Request on a specific Channel, and in return any Routers or REED devices in range broadcast a Beacon that contains their thread network PAN ID, XPAN ID, and the name of the network. Once a thread device has discov-

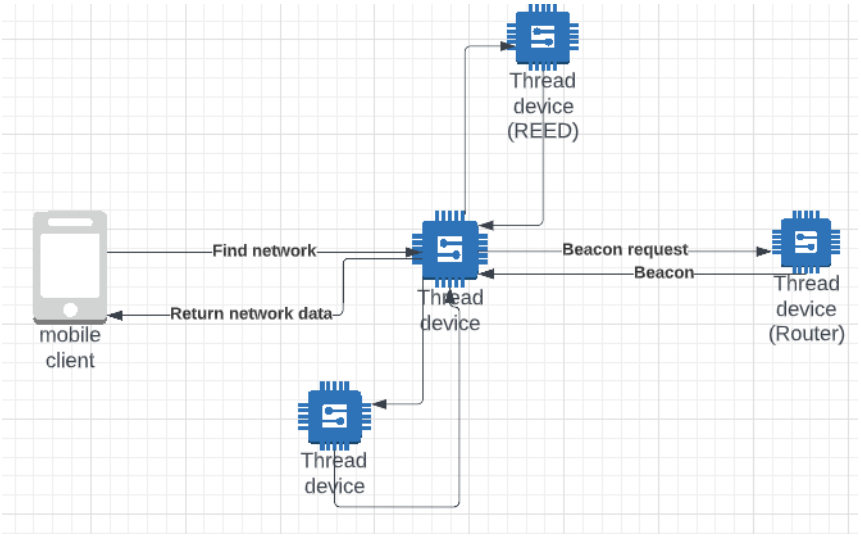

Figure 3.4: Searching for nearby devices
Made using: (http://www.lucidchart.com)

ered all networks nearby, it can either attach to an existing network or create a new one if no networks were discovered (Group, 2020). This is displayed in Figure 3.4.

Chapter 4

Methodology

4.1 Introduction to methodology

A system development methodology refers to the framework that is used to structure, plan, and control the process of developing an information system. This section will focus on different development methodologies and selecting specific ones for the project. This will be based on the comparison of the advantages and disadvantages of each method.

4.1.1 Waterfall model

Displayed in figure 4.1 is the waterfall model where the project is divided into sequential phases, with some overlap and splashback acceptable between the phases. The emphasis is on planning, time schedules, target dates, budget, and implementation of an entire system at one time. This system is ideal for supporting teams whose composition fluctuates or supporting less experienced project teams. The progress in such a system is measurable and it conserves resources. The drawbacks of such a system are that it is not flexible and is more expensive and cumbersome due to significant structure and tight controls. The project only flows forward with it and there is only slight movement backwards, and the problems are usually only found when

evaluating the system("Selecting a development approach", 2008).

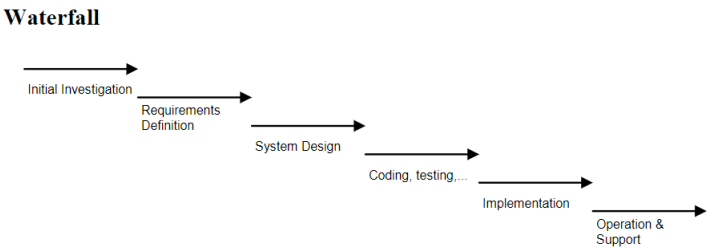

Figure 4.1: Waterfall model methodology source: "Selecting a development approach", 2008

4.1.2 Rapid Application Development (RAD)

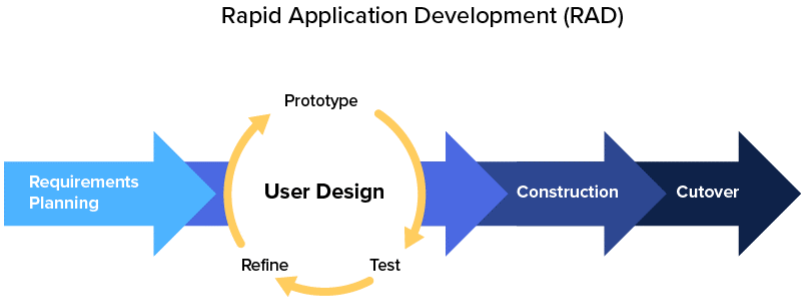

Figure 4.2: Rapid application development methodology source: "Selecting a development approach", 2008

Displayed in Figure 4.2 is an iterative framework with the objective of fast development and delivery of a high-quality system at a low investment cost. It attempts to reduce the inherent project risk by breaking a project into smaller segments and providing more ease of change during the development process. Key emphasis is on fulfilling the business need, while technological or engineering excellence is of lesser importance. The operational version of

an application is available much earlier than with Waterfall, Incremental and similar models. It concentrates on essential system elements from the user's viewpoint. Because the speed and the cost are lower the quality of the system will be lower. Such projects also end up having more requirements than necessary and their design could be inconsistent.("Selecting a development approach", 2008)

4.1.3 Prototype model

Displayed in figure 4.3 is an iterative framework that is not a standalone, complete development methodology, but an approach to handling selected portions of a larger more traditional development methodology. It attempts to reduce inherent project risks by breaking the project into smaller segments and provides more ease of change during the development process. In such a system the user participates in the process, which increases the likelihood of the user's acceptance of the final implementation. In such a process the approval process and control are not strict, and requirements may frequently change significantly. Designers may neglect documentation, resulting in insufficient justification for the final product, and may produce inadequate documentation.("Selecting a development approach", 2008)

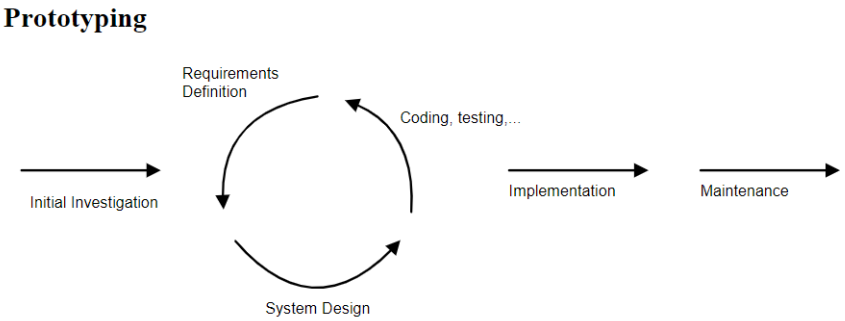

Figure 4.3: Prototype model methodology source: "Selecting a development approach", 2008

4.2 Selected methodology

The methodology that has been adopted for this project is the RAD model. The rationale behind selecting this methodology is the lack of time to fully complete such a massive scale project as using this methodology will produce a prototype. The lack of splash-back in waterfall would limit this project to only incorporate the most basic features and this project will need to be well documented with every part written down so the prototype model would also be ineffective. RAD should provide the the most efficient planing methods and at the end hopefully a working product.

Chapter 5

Software Engineering

5.1 Requirements

The requirements were extracted from the idea of the project and all the functions that it needs to provide to the user. There are multiple levels of complexity in this project so each one needed to be accounted for. Hardware was a crucial deciding factor with the requirements as some aspects and ideas were limited by the hardware's availability, memory, speed, and processing power.

5.1.1 User requirements

This project was inspired by a video from a channel called N-o-d-e where the creator proposes creating a pocket radio terminal. Which is a small computer with a screen, keyboard and the ability to send data using radio instead of Wi-Fi. This device was meant for long-distance off-grid communications. The main idea for that project is to create a small inexpensive device that could be carried easily and hook up to the smartphone using Bluetooth. It would have a companion app that is specifically designed to interact with it. The app would work like a regular messaging app that sends data to the device and then broadcast it over LoRa. The other device would listen for

a specific radio signature and receive the new message. This would create simple communication over a long distance. The device could also work with a mesh network where every device acts as a router. This would increase the coverage by the number of users within a network. If this is combined with a device that is connected to the internet where the phone acts as an exit point. This could allow for users to set up a twitter-like social media where everyone could post data anonymously within the network(N-O-D-E, 2018). The requirements are slightly different from the video as it points to using different techniques to achieve the same goal. The device that was chosen for this project was the LoPy4 because of its availability. To create the application Dart(Flutter) was chosen. This was because the Author has knowledge in the language so to incorporate it would be less problematic than learning a new language.

5.1.2 System Requirements

The following requirements are specified in the order of priority, with the highest priority first. This was chosen based on what will produce a usable system. The lower items on the list are also important to the project, but because of time constraints will be a bit harder to implement.

Functional requirements

Fr1. The system needs to supply a way for the user to send a message on one node and receive it on another. The needs to record the last message just in case there is an error in the sending/receiving process. The following data will need to be stored:

(a) The time/date when the message was sent (Required)

(b) The message that was sent in form of a String (Required)

(c) Who sent the message "Username" (Optional)

(d) Node id of the recipient (Required)

(e) The location of the user. This is only if the user is using Pytrack (Optional)

All this data will be inside a message object that will be saved in the cache. with the ability to be deleted at any point

Fr2. The nodes will need to communicate with each other and if one message needs to be sent to an out-of-range node (From 1 -¿ 3) it is going to use a second node that is connected to both nodes it will "Hop" the message across. And the system will need to display every node that it has access to (Within the mesh).

Fr3. The companion application will be able to communicate with one node to send/receive data using Bluetooth. The app will be made in Flutter, and it will prompt you to first open Bluetooth afterwards it will ask you to connect to a node and ask you if you want to set a username(optional). It will prompt you to select what device in the network you want to communicate with. It will have a Messenger-type interface for communication with a selected node.

Fr4. Protocol buffer will be used to encrypt the message to make communication more efficient as it will make the packet smaller. Every node will have a file with the decryption method. This is useful for lowering the bandwidth of the network by lowering packed size.

Fr5. The ability to change the settings of the node the user is connected to. So, each node will need to have:

(a) The Node id (Required)

(b) The name given to the device by the user "Node name" (Optional)

(c) Node status. Needs to show if a node is a router, endpoint, etc. (Required)

(d) Spreading factor (Optional). If not specified it will default to 10.

This information will be used when deciding where to send the message because it would be easier if the user knows that the specifically targeted node

is an end device with a user on the other end and not just a router.

Fr6. The data from the device will need to be saved somewhere on the device so that after you reopen the app the conversation does not disappear. This can be turned off so that the communication between users is more secure. This will be useful if a user wants to read through the previous messages. This is a limitation on the nodes' part as they cannot act as a database.

Fr7. The end node will need to have the ability to receive the node "health" data so that a user knows of something that happened to the node. The necessary data that the node will need to supply is:

(a) Signal strength (Required)

(b) Battery life (Required)

(c) Uptime (Required)

(d) Send a reset signal to node (Optional)

This would be useful to the user as they would be able to know what devices "health".

Fr8. The system will have the ability to create one of the nodes that will access the internet and communicate with other mesh networks. This node will be the border node and will need to be placed somewhere with a constant internet supply. The implementation of this would allow every user who is connected to the node network to have access to more users to interact with.

Non-Functional Requirements

Nr1. The app will need to have an intuitive UI so that every user knows how to access and communicate with different users.
Nr2. No user data will be tracked and will only be found on the sender's

and receivers' devices. It will not be sent for analysis to a third party.
Nr3. The message will be sufficiently encrypted so that if someone spoofs the packet that individual will not be able to access any of the data inside the packet.
Nr4. The system must be scalable. If more nodes are added to the system it needs to quickly add them to the mesh, and they need to work within the network without any problems.
Nr5. All the parts of the system need to work together flawlessly to supply a good user experience.

5.2 Risk Analysis

This section is dedicated to potential problems that the system may run into. This is a massive project so it could hold a lot of security flaws. The selected risks are ranked based on how severe they would affect the performance, security, and general user satisfaction while using the system.
R1. The nodes are fragile.
If the user removes the antenna connected to the lopy4 device while it is using its LoRa functionality it will fully destroy the device.
R2. A different user can connect to the node and access the data meant for the original owner of the node.
This is a problem with how the messaging works is a design flaw and it could be worked out so that the user contacts a specific user connected to the mesh network
R3. The packet could be intercepted and decrypted.
The current encryption that the device uses is protocol buffer, and it does not encrypt strings. It just makes the data smaller so it can be more efficient to send on low-power devices.
R4. Listening to Bluetooth advertisements
Bluetooth devices send and receive data all the time. If a user were to listen

to what data is being sent from the phone to the node it would be able to see all the messages that the user has received and sent

5.3 System design

The system design is divided into three parts: the mobile application, the node, and the methods of interaction between the devices. Each part was designed with the system requirements in mind. The first part that was thought of is the interaction between the application and the node.

5.3.1 Node and app interaction

The interaction was described using Figure 5.1 which describes the interaction between the node and the application where the device gets the information of the devices in the mesh network and all the possible errors that may happen based on what part of the system malfunctions. This is the first idea for the mesh and app interaction. After more research, the mesh was already developed by Pycom and the only thing necessary was the way the application communicates with the mesh. This was made possible with the use of Bluetooth low energy (BLE) and the services it provides.

5.3.2 Application layer

For the app, the first thing designed was the progression of the screens in the application. This was displayed in Figure 5.2 Where we can see that as soon as the user opens the application the app checks if the user's Bluetooth is open, and if it is not working it prompts the user to turn it on. If at any point the user turns off Bluetooth this will be displayed. After the check passes, the user will need to select the device it wishes to connect to as sometimes one person can be connected to multiple devices. After the user chooses what device, he wants to connect to he can change the data of the node or

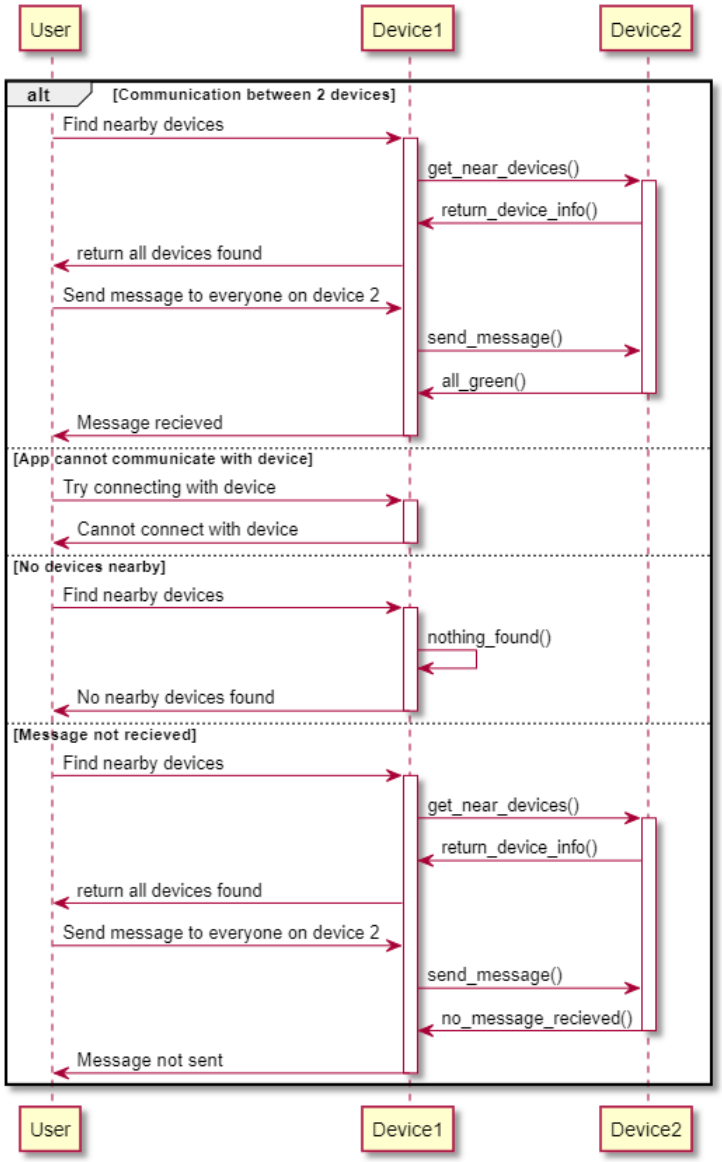

Figure 5.1: UML Diagram of 2 devices in a mesh network

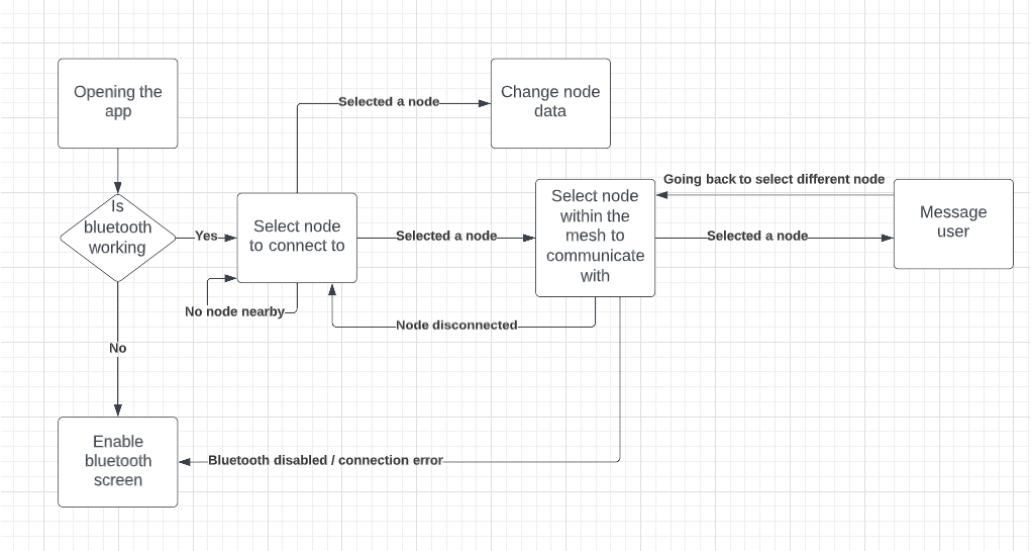

Figure 5.2: Chart displaying the progression through the app screens
Made using: (https://www.lucidchart.com/)

select a node within the network to communicate with. The data that would change is the name and description of the node that the other devices within the mesh can see. If the user chooses a specific node within the mesh to communicate with, they will be given a messenger-type texting screen where every person can send a message to that node. This is all that the app must be able to do. This was achieved using Flutter to generate screens. In the repository for this project, there is a Flutter app folder that contains all the code for the application.

5.3.3 Hardware layer

The node is the main part of the system as it communicates with the application using BLE. The device is using Pycom's implementation of Thread which needs to be initialized using an update. After the device gets the data using Wi-Fi it gains mesh functionality. It will display colours based on what kind of relationship it has with nearby nodes. It will gain the ability to search for nearby nodes to add to the network and upgrade itself from

child to parent based on its relationship with other end devices/routers. It also gains the ability to access OpenThread's Command Line Interface which allows the user to interact with the mesh.

5.4 Project Structure

Python part

The python part of the project will comprise of a folder called data with all the files that will be used for memory such as the startup.json file that would allows the program to retrieve user-generated data at startup, and message.txt that will hold the cached data of the message in case it gets corrupted while sending. Outside that folder is another folder called lib which contains all the libraries used for the project. At the root of the project there are three main python files one for startup another for the main loop and a final one for the protobuf configuration.

Flutter part

The Flutter side of the project will contain six screens that interact with the node. It's going to use Flutter_blue to interact with the Bluetooth services provided by the node that is acting as a Bluetooth server. A file will be created when messaging that will contain all the previous messages from a given chat.

Chapter 6

Implementation

6.1 boot.py

6.1.1 Before first boot

Before the first boot, the LoPy4 device would need to go into a standby mode to await relevant data can be inputted into the JSON file. To achieve this during boot a function checks for JSON and if it's not located it would await the user's Bluetooth connection and setup. If the setup was left incomplete it would default to standby. Every other time the LoPy4 would start normally except if the user "Factory resets" the device. This loop is located within the boot.py which is the first file run when the microcontroller boots.

6.1.2 Initialisation station

The first step Micropython does is initialising Lora and Bluetooth. To achieve this a JSON file is read and its value pairs are used to initialise other parts using user input. It needs the JSON file to initialise the lora region as different locations have different available frequencies. After we have the values necessary Bluetooth and LoRa functionality is initialised. This was done with the commands:

```
bt = Bluetooth()
```

Used to initialise Bluetooth

```
lora = LoRa(mode=LoRa.LORA, region=jsn[lora_region])
```

Used to initialise LoRa
After Bluetooth and LoRa are initialised a function is used to erase the previous data of message.txt. This was created by using a built-in function that checks for files, specifically for a file called message.txt removing it and re-initialising it. message.txt is a file that contains data of the last sent message, and before each boot, the previous message is erased. The message gets erased to prevent anyone from tinkering with the device and accessing the data. Another thing that was implemented to help with device reliability is increasing the thread stack. After a lot of debugging, it was determined that the default stack size was too small and the device would occasionally crash. This is done with one command which is:

```
_thread.stack_size(4096 * 3)
```

this function increases the 4Kb default value by three times. The LoPy4 devices if left unchanged will blink blue light every 4s. This can be disabled with

```
pycom.heartbeat(False)
```

6.1.3 Pymesh configuration

To add Pymesh to your LoPy4 device PyBytes would need to be firstly enabled and configured. This is done in a couple of ways. The most reliable way is through the https://pybytes.pycom.io/ website and the Pycom firmware update tool. There are more ways of enabling it other than the update tool like using the App pycom made for it and using The command line tool while connected to the device. These methods still need the user to be using the PyBytes website. While on the website the user can create a project and

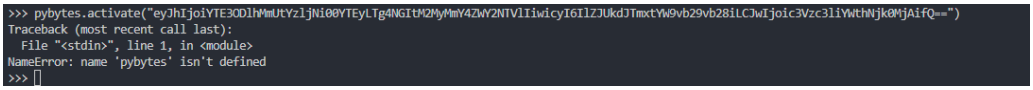

Figure 6.1: Configuring PyBytes failing with device connected

select the devices to add to this project. After that is done the ability to create a mesh is enabled. When the mesh is selected the user will be prompted to a licence agreement and select the region of the node, spread factor and the bandwidth. With that, a join key is required. A join key is a 128-bit string every device within the mesh shares and requires for identification and verification. After selecting the relevant information the user is prompted with a summary of the selected data. After that PyMesh can be deployed to a device that is currently connected to PyBytes.

6.1.4 Pymesh implementation

To test if Pymesh is initiated properly is to try inputting this code into the terminal.

```
>>> pymesh = pybytes.__pymesh.__pymesh
>>> pymesh.cli_start()
```

With this, the user has the ability to interact with the Pymesh command line interface. Here the user can send a message, print configure file content, display the current IPv6 address, etc. Pymesh is a set of frozen scripts in the Pymesh firmware binary. If PyBytes is used mesh can be started and initialised by calling a Pymesh object using:

```
pymesh = pybytes.__pymesh.__pymesh
```

Pymesh parameters are saved in memory so the next time the node boots it will try to maintain its IP address and connect to the neighbour nodes.

6.2 main.py

The first thing that the main file needs to do is to import the boot.py file. Within main a couple of classes are created that hold the data of a message and the data of a node. Message data gets used every time the user wants to send a message to another within the mesh to convert it so protobuf can format it before it gets sent. More methods are implemented that retrieve the device's "Health" information. There is another function that interacts directly with the microcontroller which is reset_device():

6.2.1 Continue Pymesh

After all the necessary objects are retrieved a function runs to check if the device is connected to pymesh. If it's not it will halt and start searching for one. If we are connected to the network we can use functions like:

```
print(get_mesh_pairs())
```

to display every device in the mesh and their MAC address. After specifying a node we could get detailed node info and send it to the application.

6.2.2 Bluetooth Implementation

To interact with the app we need to set up the microcontroller as a GATT Server. For that, we first need to advertise our device so the user can "See" The node. Afterwards establish characteristics which encapsulate a single data point (Device battery percent, run-time seconds, etc.) and group them using services. When a change in characteristic occurs it will create a callback with updates that will send the data to the application. Using this we can specify a power service that receives data from the phone to Reset/PowerOff/HardReset. Afterwards we can specify another service for device "Health" which would retrieve an object containing the temperature/battery voltage/runtime. Another necessary service is gathering mesh

data and retrieving specific characteristics based on the request. The final service is used to change the data of the node and update the JSON file based on the changes after the next boot.

6.3 Flutter app

Before we can start creating screens all the packages need to be installed and setup. The main one that this will be focusing on is Flutter_blue which requires a couple of steps before working. Firstly we need to go into .android/app/build.gradle and add a line to defaultConfig that limits the minimum android SDK version. Afterwards, we need to add permissions to use Bluetooth and access the location. This is done differently for android and IOS. The first screen to be made is the screen that tests if Bluetooth is enabled on the smartphone. If it should lead you to another screen containing a list of devices that you are connected to. For that we first need to obtain an instance of FlutterBlue and using it we can start scanning for devices. And we can display the services using:

```
List services = await device.discoverServices();
print(services)
```

With this we can create a function which will extract characteristics and give us an ability to interact with them. One of these functions communicates with the node's configuration data. This is used to rename the node, change the region or change the spread factor.

6.3.1 Other Screens

The most essential screen is the one that allows the user to select which node within the mesh they wanna communicate with. To achieve this data is awaited from the mesh service and after its retrieved a list of devices is generated. After selecting a node a messaging screen will be displayed. When sending data to a node the user's username, message and time of delivery is

sent. When another user opens the message another message will be sent back to the sender of the message notifying them that the message was opened. This screen will read all the information from a JSON file containing all the previous messages from this chat. This file is updated every time a new message is received or sent. Another screen can be chosen from the start to change the user data and the message settings (username, if messages should be deleted after a certain time).

Chapter 7

Problems with development

7.1 Introduction to problem solving

This section is dedicated to every roadblock the author ran into while developing the product. Because of the final state of the artefact, there is a lot to speak about.

7.1.1 Time

One of the biggest problems stems from a lack of time management on the author's part. The scope of this project would suit a team of well organised computer scientists and a long time to complete because of all the moving parts. The project should've only focused on one aspect of this project such as the companion application, the node itself, or the border node that needs a web server implemented so that it has someone to communicate with. Because the project was all of these parts it's mostly unfinished.

7.1.2 Pycom Problems

Because the pycom products are not as widespread as microcontrollers such as the ESP32 or the Arduino uno there aren't that many resources available

for the regular user of such a device to firstly test features before implementing them in bigger projects. Pycom uses its library for the pycom devices that don't yet have IDE support and assistance. Because of that features like auto-fill or error checking are not available. Every pycom specific line of code is flagged as incompatible with python. The only way of seeing if something works are to find the specific command on the Pycom documentation website and try implementing it. This often works, but going back to a different command and scrolling through the website every time something doesn't work is very time-consuming. To connect to the devices there are 2 options: Using Pybites which is a cloud-based device management platform which is still in beta or the IDE extension called Pymakr with version 2.0 still in alpha.

Pymakr

One of the biggest inconveniences caused by Pymakr is that it's enabled globally and every time the user were to open a new Visual Studio Code window it opens a new Pymakr terminal even if one is open from a previous session. This can be disabled, but not limited to a single project. Currently, on the VSC marketplace it has 2 stars with some comments like: "A nice reminder that programming is 90% messing with half broken tool-chains and 10% programming. "command pymaker.connect not found" It's going to be a longggg road to get to "Hello World"."(online, 2022) This is not the only one-star review with many more on the Rating section of the add on. Some of the most common issues to occur regularly are:
The add-on not finding the device even when its plugged in the specific port (Fix: Reset the PC, Update the firmware of the microcontroller),
The upload to the device failing for some reason corrupting the data on the device(Fix: Updating the firmware and re-uploading the data),
Uploading the PyMesh can mess with uploading any other programs onto the device.

Pymakr2

Currently in the preview is Pymakr2 which is a better version of Pymakr that takes to account a lot of issues with the previous version and attempts to fix them. To start a console first you need to select the device that you want to interact with. It adds the ability to interact with the files within the microcontroller which adds better interaction with the device, and the ability to have multiple devices connected to one PC. This add on is still in preview so it does come with a lot of bugs and problems. It promises some things that it doesn't deliver like the auto-fill and documentation for functions within the IDE. Some of the most common problems a user will run into in this version of the software(2.10.0) are:
The add-on crashes when trying to access the flash memory(Fix: Unplug the device and plug it back in and retry the operation),
Failing to build because of the JSON error of the pymakr.conf file that hasn't even been changed. Even with such problems, it shows a lot of promises and the current version is way more advanced and better working than the previous version.

7.1.3 Hardware

The hardware is very powerful for such a small device. The LoPy4 device by itself cannot be programmed without an extension board or a USB serial converter. The only available device at the time was a PyGate that came quite handy for programming the devices, but the problem with that came when every device needs to be programmed. To do that the LoPy4 devices would need to be removed and reattached every few minutes to test if the program works. Another problem with the devices is that if the antenna is disconnected mid usage the 35£ device would fry itself. The only reason this is stated is that it happened during development once.

Chapter 8

Evaluation against requirements

8.1 What has been done

The following section contains Table 7.1. which displays every system requirement and how complete it is. Afterwards, reasoning will be provided why it's left in such a state, and to what extent each part is done.

8.1.1 FR1: Sending data from one node to another

One of the most important parts of this project is sending data from one device to another. This part was done in python by firstly enabling LoRa and a specific socket on the device, selecting whom to broadcast to, and sending a message. Another device needs to be set up to listen for such a packet, receive it and display it in the terminal. This was turned into two functions called send_message() and message_rcv(). The only reason this isn't fully done is that it needs to be inside a loop that will forever listen to/receive messages. And for that to be done it needs the companion app to send data to it to forward it to another device.

System requirement	Description	Percent done
Fr1	Message from one node to another	90%
Fr2	Mesh implementation	70%
Fr3	Companion app	50%
Fr4	Protocol buffer	100%
Fr5	Change node data	30%
Fr6	Save data	40%
Fr7	Node health	70%
Fr8	Border node	0%

Table 8.1: Displaying the current completeness of specific requirement

8.1.2 FR2: Mesh implementation

The mesh fully works with the OpenThreads command-line tool available on the terminal. This was implemented using PyBytes and sending the mesh configuration to the microcontroller to enable the frozen python scripts in the firmware. This process is needed to be done every time the device is updated which is a slight inconvenience as the devices need to be updated regularly. It can display PyMesh parameters, and the devices it's connected to. The reason this isn't fully done is that it requires the companion app to interact with the data and select what node it wants to communicate with within the mesh.

8.1.3 FR3: Companion app

The application is developed using flutter. It has prototype screens that currently don't have much function (Except progression through screens), but if more time was available all the functionality would be implemented. One of the biggest roadblocks with the application was learning how flutter works, and more importantly how Bluetooth devices interact with one another. Because of how vital the application is to the whole project, other parts are nearly done but left unfinished.

8.1.4 FR4: Protocol Buffer

To implement protocol buffer onto a microcontroller running micropython micropython-uprotobuf was used(jazzycamel, 2018). It's a lightweight implementation of Google's Protocol buffer for micropython. One problem with this distribution of protobuf is that it's unsupported by the person who made the repository, but this version works fine and was not too hard to implement. Saying that it's made for Linux which caused some problems with implementation. In the end, it ended up being three files that contain the bare-bones Google API that enables protobuf to work, a file generated using protobuf cli that specifies what kind of data will be sent and full implementation of protobuf. After this was implemented another version that was more up-kept and better was found online, but because of the time constraints was not successfully implemented(Dogtopus, n.d.)

8.1.5 FR5: Change node data

A system was created that would read a JSON file and set the node name and node comment to the device during the boot part of the program. A class was created to hold all that data and retrieve it, but because of the state of the application it"s unfinished.

8.1.6 FR6: Save data

To save data two methods are implemented. The first one is to create a cache on the micropython device that would save the string of data to a file called data.txt. This data is utilized if the device fails to send the data from one device to another. This data is deleted every time the device is booted so that people who want to tinker with the device don't have access to the last message sent. The other method of implementing save features was creating a file that gathers all the data from data.txt and compiles a file on the companion application that would give the user the ability to see all the past messages. The second save feature is not fully implemented.

8.1.7 FR7: Node health

Methods were created to access device data such as the ability to send a reset signal to the device and to get the run time of the device, The temperature of the ESP32 core, and battery voltage. This is split into two functions: one to soft reset the device and another to display the relevant data.

8.1.8 FR8: Border node.

The time ran out before the border node was fully implemented. To implement this feature would require knowledge of interacting with web servers and hosting such a service. This is a great point of continuation for the project as if the mesh had the ability to communicate with the internet it wouldn't only be bound to that one mesh.

Chapter 9

Future work

9.1 Continuation of work

In this section improvements to the design and possible future developments in this technology may improve this way of mesh networking. Because of the artifact's current state, this is a continuation point for the project. The following sections are some future technologies and their implementation within the mesh.

9.1.1 Finishing the artifact

A big part of what can be done on this project is finishing all the parts based on the software requirements. If this project were to be completed all the other sections of this chapter could be started to be implemented.

9.1.2 Algorithm For Determining The Spread Factor

Currently, the user needs to set the spread factor for the node. If the spread factor is lower the data won't travel as far as with a higher spread factor, the perk of this is that the amount of data sent is larger and the speed of transfer is larger. If the user manually states the spread factor of the

device quite often it wouldn't be the most optimal for the range. There are propositions for an adaptive spread factor selection scheme for corresponding spread factors. This scheme enables the maximum throughput and minimum network cost(Kim et al., 2020). Using this technology, the mesh could find the most optimal spread factor for each device within the mesh. By doing so the data transfer of the devices would be quicker and the battery usage lower. This technology has yet to be implemented within multi-hop networks, but it's highly suitable for such systems.

9.1.3 Helium Implementation

Helium is a global, distributed network of Hotspots that creates public, long-range wireless coverage for LoRaWAN-enabled IoT devices. The Helium hotspots produce and are compensated in HNT, the native cryptocurrency of the helium blockchain(Amir Haleem & Nijdam, 2018). This technology could be implemented within the mesh so that each time it has utilized the user who interacts with it gets some crypto as a by-product. This would give a lot of people the incentive to switch to this mesh technology as every message you send will gain you some value. This goes well with the idea of the artifact which is the ability to permanently gain access to a network with a single payment and upkeep. If Helium is implemented the device itself could generate enough of the crypto to pay for itself.

9.1.4 Extra Features

This technology is not limited to messaging users within the network but getting data from sensors out of Link-Local reach. This may include creating a different part of the app for monitoring sensors within the mesh. This will need to be implemented with specific profiles for each sensor device. This would be useful if the user wants to use the mesh for more than messaging as currently, the application is very limited.

9.1.5 Web app

Because futures direction towards wireless technology nowadays more and more PC's/laptops come with Bluetooth built in. A web app could be developed for desktop devices that would interact with the node. This would give more people the ability to access the network and use it to communicate with other users or check their IOT devices.

9.1.6 Application Account

More work needs to be done on the application, and the next thing that needs to be incorporated is sending the data not to everyone who is connected to the node, but to the specific user connected to that node. This would help with the security of devices that are connected to the mesh as the interaction with them could be limited to a single user or multiple people within the network.

9.1.7 Message board

Another thing that can be implemented within the mesh is a message board website that can be accessible to everyone who is connected to the mesh. It would allow the users to post small messages anonymously. The data for the website could be saved onto the LoPy4 device. The device will need to separate storage into two parts: Data that is necessary for the device to work normally and another to hold the Website data. With more devices added to the network, the storage capacity for the message board will increase. If the device is to run out of storage then it would delete the earliest messages.

Chapter 10

Conclusion

10.1 In conclusion

This project aimed to create a way of a secure and widespread network that be accessed at a low entry price. This paper acts as a foundation on which to build upon and create a fully decentralised wireless network with the ability to send data over, and access the data of nearby IoT devices (which if they have a LoRa module could be programmed as a node), get compensated (Using Helium) and communicate privately. The wireless mesh technology shows a lot of promise with a future that is pivoting towards online all the time. This would allow us to stay connected even if we get disconnected from other networks.

Bibliography

Amir Haleem, A. T., Andrew Allen, & Nijdam, M. (2018). *Helium a decentralized wireless network* (tech. rep.). http://whitepaper.helium.com/

Chawla, M., Mundra, A., Rakesh, N., Agrawal, A., & Ghrera, S. P. (2015). Fault tolerance based routing approach for wmn. *2015 International Conference on Computer and Computational Sciences (ICCCS)*, 177–182. https://doi.org/10.1109/ICCACS.2015.7361345

Dogtopus. (n.d.). Dogtopus/minipb: Mini protobuf. Retrieved May 5, 2022, from https://github.com/dogtopus/minipb

Dunlop, B., Nguyen, H. H., Barton, R., & Henry, J. (2019). Interference analysis for lora chirp spread spectrum signals. *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*, 1–5. https://doi.org/10.1109/CCECE.2019.8861956

Faq [Accessed: 2022-05-03]. (n.d.). https://docs.flutter.dev/resources/faq

Faqs [Accessed: 2022-05-01]. (n.d.). https://ipv6.org.nz/index.html%3Fp=743.html

George, D. (2016). Micro python: Python for microcontrollers. https://www.kickstarter.com/projects/214379695/micro-python-python-for-microcontrollers

Gomez, C., Oller, J., & Paradells, J. (2012). Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, *12*(9), 11734–11753.

Google. (n.d.). Node roles and types openthread. https://openthread.io/guides/thread-primer/node-roles-and-types

Google. (2022). *Dart overview*. Retrieved May 3, 2022, from https://dart.dev/overview

Group, T. (2020). *Thread network fundamentals* (tech. rep.). https://www.threadgroup.org/Portals/0/documents/support/Thread%20Network%20Fundamentals_v3.pdf

jazzycamel. (2018). Micropython-uprotobuf. Retrieved May 5, 2022, from https://github.com/jazzycamel/micropython-uprotobuf

Jiang, Y., Peng, L., Hu, A., Wang, S., Huang, Y., & Zhang, L. (2019). Physical layer identification of lora devices using constellation trace figure. *EURASIP Journal on Wireless Communications and Networking*, *2019*(1), 1–11.

Kim, S., Lee, H., & Jeon, S. (2020). An adaptive spreading factor selection scheme for a single channel lora modem. *Sensors*, *20*(4), 1008.

Mohammadi Zanjireh, M., Shahrabi, A., & Larijani, H. (2013). Anch: A new clustering algorithm for wireless sensor networks. *Proceedings - 27th International Conference on Advanced Information Networking and Applications Workshops, WAINA 2013*. https://doi.org/10.1109/WAINA.2013.242

Muller, N. (2002). *Networking a to z*. McGraw-Hill Education. https://books.google.co.uk/books?id=0qv4KbasX7wC

Neutert, S. (n.d.). Simple calculator for estimating a (lipo) battery's life. https://www.of-things.de/battery-life-calculator.php

N-O-D-E. (2018). Help build an off grid communications / mesh network device. Retrieved May 4, 2022, from https://www.youtube.com/watch?v=zXTEWFb_6w4

online, U. (2022). Retrieved May 4, 2022, from https://marketplace.visualstudio.com/items?itemName=pycom.Pymakr&ssr=false#review-details

Overview protocol buffers google developers. (n.d.). https://developers.google.com/protocol-buffers/docs/overview

Pycom. (n.d.). Lopy4 documentation. https://www.farnell.com/datasheets/5913.pdf

Pycom. (2022). The lopy4 is a quadruple bearer micropython enabled development board including lora, sigfox, wifi and bluetooth [Accessed: 2022-05-02]. https://pycom.io/product/lopy4/

Selecting a development approach. (2008). *Center for Medicare & Medicaid Services*, 10. https://pdf4pro.com/amp/view/selecting-a-development-approach-1432b1.html

Spreading factors [Accessed: 2022-03-27]. (2021). https://www.thethingsnetwork.org/docs/lorawan/spreading-factors/

What is thread? (n.d.). https://openthread.io/guides/thread-primer

The wipy: The internet of things taken to the next level. (2016). https://www.kickstarter.com/projects/wipy/the-wipy-the-internet-of-things-taken-to-the-next

Zhu, G., Liao, C.-H., Sakdejayont, T., Lai, I.-W., Narusue, Y., & Morikawa, H. (2019). Improving the capacity of a mesh lora network by spreading-factor-based network clustering. *IEEE Access*, *7*, 21584–21596. https://doi.org/10.1109/ACCESS.2019.2898239

Appendix A

First Appendix