Alexandra Riddell-Webster

A System to Help Prevent Crashes in Rowing Boats

Computer Science Tripos – Part II

Murray Edwards College

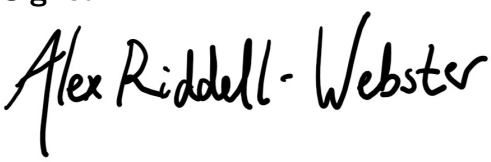
2023

Declaration of Originality

I, Alexandra Riddell-Webster of Murray Edwards College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose. In preparation of this dissertation I did not use text from AI-assisted platforms generating natural language answers to user queries, including but not limited to ChatGPT.

I am content for my dissertation to be made available to the students and staff of the University.

Signed



Date
May 5, 2023

Acknowledgements

This dissertation owes a huge amount to Matthew Ireland, for supervising me. My UTO, Jon Crowcroft was invaluable. Thanks to the members of Cambridge University Boat Club for allowing me to put strange boxes on thier boats, advising me on communication over water and helping me evaluate the system. Special thanks to CUBC members Patrick Ryan, Mike Taylor, Rosa Millard and Imogen Grant. I also thank Duncan Barnes for discussing GPS and electronics on rowing boats with me.

Proforma

Candidate Number:	23487C
Project Title:	A System to Help Prevent Crashes in Rowing Boats
Examination:	Computer Science Tripos – Part II, May 2023
Word Count:	10332 [1]
Code Line Count:	6429 [2]
Project Originator:	The Candidate
Supervisor:	Mr Matthew Ireland
University Teaching Officer:	Prof. Jon Crowcroft

Original Aims of the Project

The objective of this project was to design a system to help prevent crashes in rowing boats. The goal of this system was to warn rowers and coxes when they approach obstacles, be they other boats or items in the river. This project also aimed to evaluate such a system.

Work Completed

All core success criteria of were met and some extension criteria implemented.
This project created such a system to help prevent crashes in rowing boats. Knowledge about obstacles were propagated through a mobile ad hoc network (MANET) of rowing boats using the Epidemic routing protocol. Users were warned when approaching a known obstacle by means of a buzzer and LED light. The MANET and system were thoroughly evaluated.
The whole system was documented and will be available, open source, after my graduation.

Special Difficulties

None.

[1] Word count calculated with '`detex main.tex | tr -cd '0-9A-Za-z \n' | wc -w`'

[2] Line count calculated with '`cloc project --exclude-list-file=.clocignore`' where `.clocignore` contains the libraries I did not write

Contents

Introduction

I built a system to help prevent crashes in rowing boats. The system is implemented in hardware, culminating in a series of boxes that can be attached to boats. The project is split into two parts: the mobile ad hoc network (MANET) that allows nodes to communicate a dynamic collection of known obstacles, and the application layer to warn users when they are approaching an obstacle and allows the user to add obstacles. The free and open source software will be available online, fully documented, after my graduation. It includes a 'how to' guide [Appendix A] for the construction of a node so any rowing club can use my project as a collision avoidance tool.
The final system achieved its goal. It propagated obstacles through the network and warned users of upcoming obstacles.

1.1 Motivation

Crashes between rowing boats and obstacles or other boats injure rowers and cause equipment damage. Unfortunately, crashes are common. Figure 1.1 shows a boat on the Thames, having collided with a bridge at a regatta in March 2023. Some boats have coxswains, responsible for steering a boat, while others are coxless, where rowers who face away from the direction of travel are responsible for steering boats. This project is designed for use in both coxed and coxless boats. My project has a very personal motivation, as a friend was hit by a larger rowing boat while in a single three years ago, causing a severe concussion that resulted in two years of intermission from their studies.

Figure 1.1: An eight colliding with Barnes Bridge[1]
[3]Simon Ramskill, Twitter post, March 2023,
https://twitter.com/lliksmar/status/1637147870682906624?cxt=HHwWgIC8zY_UqLgtAAAA, Accessed March 2023

1.1.1 Feedback from Users

Olympian and Cambridge Student, Imogen Grant said *"Rowing is meant to be a non-contact sport, but it has a surprising incidence of concussions as a result of head on collisions, even at national and international level. We sit facing backwards, so steering involves looking over a shoulder in between strokes. The boats that I row in - singles and doubles - don't even have rudders and our steering is entirely reliant on pressure steering alone. Having warning of an obstacle would improve my confidence while sculling, and would mean that I could focus more on improving my technique rather than worrying that I might injure myself by crashing into something."*

1.2 Background

1.2.1 Networking

The Defence Advanced Research Projects Agency (DARPA) Packet Radio Network (PRNET) [1] was the first wireless data network, using store-and-forward routing over packet radios. Jubin and Tornow lay out the state of PRNET in 1987, nearly 15 years after research began on it, in their paper *The DARPA Packet Radio Network Protocols*. The work on PRNET fed into DARPA's Survivable Radio Network (SURAN) [2]. This connection between the military and ad hoc networking still exists today, with MANETs used in conflicts [3], as well as autonomous vehicles [4] and disaster relief scenarios where previously existing infrastructure is destroyed [5].

Each node in a MANET is free to move in any direction, so the topology of the network changes in an unpredictable way. Every must also forward traffic, making it a router.
The nodes in my MANET would be rowing boats. Rowing boats frequently move at speeds greater than 15 km/h, making the topology of any network of boats very unpredictable and dynamic. In addition to this, the network may move between rivers, meaning no existing infrastructure would be available. These form the primary challenge for my network; routing messages through the network without pre-existing infrastructure, giving each node enough information to pass messages to other nodes.
This is made more difficult by the small CPU and memory on the Raspberry Pi Pico [6], limiting the processing power and information each node has.

1.2.2 Medium Access Control

While routing corresponds to many hops across a network, medium access control considers only one. Medium access control protocols control access to the transmission media to prevent collisions between packets. Collisions could result in packets being corrupted and lower the overall utility of the network. This project implemented medium access control.

1.2.3 OSI Model

The Open Systems Interconnection (OSI) model for computer networking provides a standardisation for communication over a network. The OSI model consists of seven layers: Application, Presentation, Session, Transport, Network, Data Link, and Physical [7]. Based on this model, my project uses the libraries provided with hardware for the physical and data link layers. It focuses mainly on the networking layer, sending messages between nodes.

1.3 Related Work

1.3.1 Networking

There is precedent for using MANETs on rowing boats. I spoke to the team behind the broadcasting and telemetry for the Oxford Cambridge Boat Race. They use a 15-node MANET to get the video and telemetry from the rowing boats. They also take GPS readings for the location of the boat, recording the location up to 20 times a second. Figure 1.2 shows the equipment attached to the boats for the 2023 Boat Race.

Figure 1.2: Equipment placed on the stern of the Cambridge boat [8]

1.3.2 Delay Tolerant Routing

Delay tolerant routing assumes that networks will lack connectivity, with partitions between nodes. Vahdat and Becker's paper *Epidemic Routing for Partially-Connected Ad Hoc Networks* [9] introduces a replication-based, delay-tolerant routing protocol where messages are passed onto nodes that do not have a copy of the message. Vahdat and Becker's paper is the key paper used to implement routing in this project.

1.3.3 Safety in Rowing

ROWCUS [10], a company based in Switzerland, has attempted to solve the problem of crashes in rowing boats. While ROWCUS has similar goals to my project, the technical methodologies are different, using radar rather than GPS location to detect proximity to obstacles. Additionally, ROWCUS does not network nodes together, instead using individual nodes. ROWCUS has *"decided not to pursue the commercial deployment of ROWCUS"*, in a statement on their website [10].

Preparation

2.1 Starting Point

I had no previous experience with microcontrollers, although I had worked with single-board computers. Therefore, over summer I dedicated a small amount of time to learning about microcontrollers and MicroPython, completing basic tasks such as flashing an onboard LED. While this was useful, my project was eventually implemented in CircuitPython, so it would have been more beneficial to have researched CircuitPython.

I had some experience with networking and routing protocols prior to starting this project. This was composed of the Part IB Networking module and two weeks' work during an internship on a MANET with a different routing protocol, much further abstracted than this project. During my project, I took the *Part II Principles of Communications* course, further expanding my knowledge.

2.2 Requirements

The three requirements I identified as my success criteria were:

The Epidemic routing protocol is implemented within the network	Achieved
An evaluation of the network has been carried out	Achieved
An application layer has been implemented to allow utility of the network	Achieved

All of these requirements have been implemented during this project.

I also laid out several extension tasks. The key extension was medium access control. Other extension tasks were drawn from my research into networking protocols other than Epidemic. For instance, using a metric of transmission quality – received strength signal indicator (RSSI) in radio communication – to influence whether an anti-entropy session was initiated. Additionally, the GPS location could be used, either by only contacting nearby nodes to increase the probability that an anti-entropy session is successful, or prioritising sending messages about new obstacles to nodes that are near these obstacles. While time constraints have not allowed me to implement all extensions, I have allowed messages to have two priorities, normal and urgent.

2.2.1 Analysis

The requirements for this project came from analysis of the problem; they were building blocks for a system to help prevent crashes in rowing boats. Nodes would be attached to rowing boats, which the have high mobility and frequently move at speeds around 15 km/h. The networks generated by rowing boats have a high chance of partition and are unpredictable. This topology required a high degree of flexibility within the network, pointing towards a delay tolerant routing protocol. The limited compute power available was also a consideration. I chose to implement Epidemic as it is a delay-tolerant routing protocol and best suited to the network topology likely to be generated by rowing boats [9].

As the nodes in the network were all going to communicate on the same medium, some medium access control would be needed to prevent collisions between messages. This was an extension criteria.

2.2.2 Network Topology

Many of my requirements were based around the topologies of networks generated by rowing boats. The potential topology of networks was therefore analysed. This was done by looking at example distributions of rowing boats on lakes and rivers. I analysed Google Maps and Earth's satellite view of the Thames along a 5.5km stretch of the Championship Course, pinpointing rowing boats and coaching launches, adding them to a map with potential connections between nodes, assuming the radios have a range of 500m, alongside obstacles. Figure 2.1 shows a satellite image of two rowing boats, potential nodes in this network. Figure 2.2 shows the annotated map of the Thames, and Figure 2.3 presents the abstracted network topology of these rowing boats.

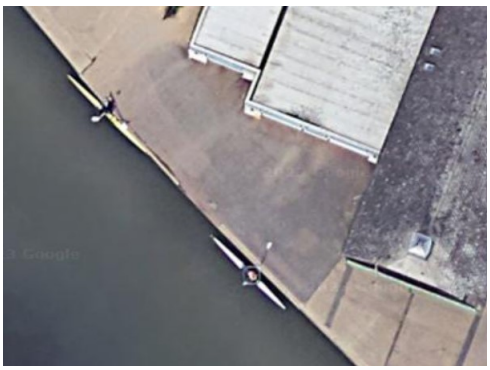


Figure 2.1: Two rowing boats seen on Google Earth [11]

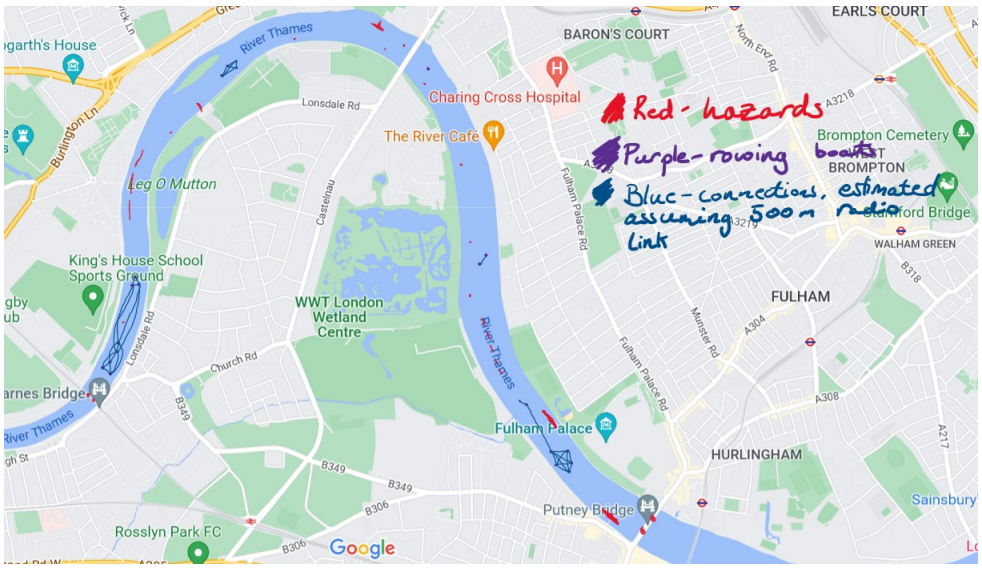


Figure 2.2: Rowing boats, potential obstacles and assumed connections marked on a Google Maps map of the river Thames [12]

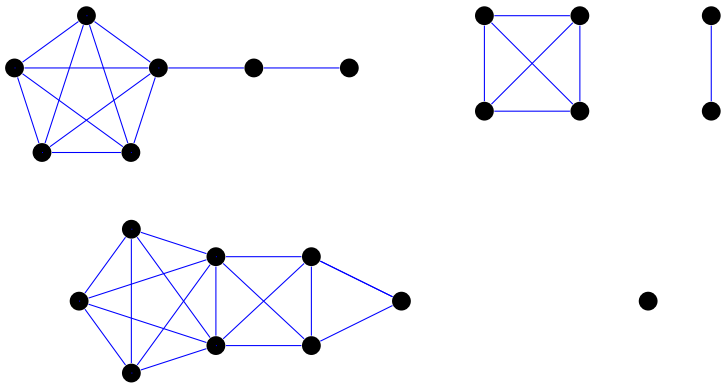


Figure 2.3: The network of rowing boats in an abstracted form, with nodes shown in black and assumed connections in blue

2.3 Summary of Research

2.3.1 MANETs

A mobile ad hoc network (MANET) is characterised by wireless nodes, a frequently changing network topology and no reliance on pre-existing infrastructure. They are decentralised and therefore have no single point of failure [13]. This project constructs a MANET between rowing boats to communicate obstacles to each other. A MANET was implemented due to the lack of pre-existing infrastructure for such a system and the difficulties associated with setting up and maintaining a base station or similar. Requiring an infrastructure like this would also raise the barrier to entry for many clubs with limited funds and technical skills. Additionally, rowing boats can move between stretches of water, further supporting the use of a MANET for this project.

2.3.2 Routing

Routing protocols find a path from a source to one or more destinations destination within the network. Different routing protocols optimise different parameters and are better suited for different network topologies and applications [14]. Within MANETs, a routing protocol must allow the network topology to change over time. They tend to contain node discovery techniques to allow for this.

Before deciding on Epidemic routing for the project, several other protocols were considered. The Better Approach to Ad Hoc Mobile Networking (BATMAN) routing protocol [15] is designed to route messages through MANETs, broadcasting originator messages (OGM) for node discovery. BATMAN has the interesting addition of a transmit quality (TQ) metric in the OGM packets, allowing the quality of connections between nodes to be factored into the route packets take through the network. While BATMAN does allow messages to be broadcast to all nodes, its primary focus is routing messages from one node to another. Additionally, while it allows for message mobility, it is not delay-tolerant.

Greedy Perimeter Stateless Routing (GPSR) is a location-based routing protocol [16]. GPSR exploits the relation between geographic position and connectivity in a wireless network, where each node tells its immediate neighbours its current location. Greedy forwarding is predominantly used to send packets to nodes that are progressively closer to the destination until the destination coordinates can be reached. Where greedy forwarding fails, GPSR uses perimeter forwarding (forwarding the packets around the perimeter of the region) until greedy forwarding can be used

again. This protocol was ultimately deemed to be unsuitable for my project as, similar to BATMAN, its primary focus is on sending messages between two nodes. Additionally, the high mobility of the nodes in the use case means that forwarding packets to a set of coordinates does not mean the message will reach the intended destination, as the node may have moved.

2.3.3 Epidemic

Epidemic routing was implemented as it is a delay-tolerant routing protocol and best suited to the network topology likely to be generated by rowing boats [9]. The networks generated by rowing boats have a high chance of partition, and the nodes are highly mobile. Epidemic routing allows all nodes to carry information through the network. This makes it useful for the potential topology of the use case. Additionally, Epidemic routing supports a broadcast functionality, sending messages to every node in the network, which is necessary for the use case as every boat should hear about potential obstacles.
Epidemic routing gains its name from its similarity to the spreading of infections. Each node holds a buffer of messages that it has seen. When a node comes into contact with another node it has not recently contacted, it replicates and transmits messages this neighbour. This message exchange is known as anti-entropy.

2.3.3.1 Anti-Entropy

The term 'anti-entropy', as used in my dissertation, comes from Vahdat and Becker's paper. Anti-entropy is the process of exchanging messages to pass them through the network. It is a key part of Epidemic routing. A more detailed explanation is in the Implementation section, although a brief overview is provided here.
When two nodes come into communication range, they transfer messages the other node has not seen to each other. Each node then stores these messages to exchange later.

2.3.4 Medium Access Control

While these routing protocols often have mechanisms to minimise the probability of collisions, such as adding a jitter in the sending of discovery messages, none of them contain medium access control. Due to the probability of collisions between messages causing disruption to the sending of messages, particularly as all nodes are broadcasting on the same radio frequency (433 MHz). It was therefore prudent to include media access control in the system.
Multiple Access with Collision Avoidance for Wireless (MACAW) is often used by ad hoc networks [17]. MACAW uses request to send (RTS) and clear to send (CTS) messages to minimise the probability of collision. As discussed in the System Design section, MACAW inspired the medium access control in my project.

2.3.5 Serial Communication

Serial communication protocols were used to transmit data between the Raspberry Pi Pico and peripherals. All data was transferred over relatively short distances. The two main protocols used were the serial peripheral interface (SPI) and universal asynchronous transmitter / receiver (UART) protocols.

2.3.5.1 SPI

SPI is a full duplex synchronous serial communication interface. The communication is synchronised by a clock signal, output from the controller on Serial Clock (SCLK). The four logic signals specified

are SCLK, Peripheral Out Controller In (POCI), Peripheral In Controller Out (PICO), and Chip Select (CS).

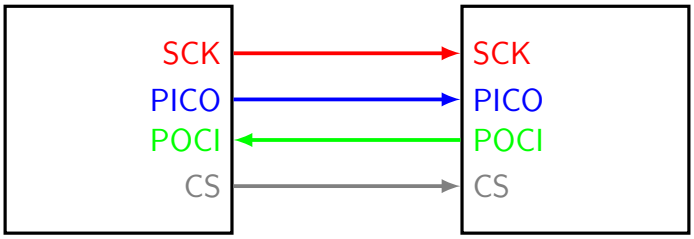


Figure 2.4: SPI logic signals

2.3.5.2 UART

The UART protocol allows for full duplex, asynchronous communication. The controller and peripheral do not share a clock signal, but transmit at the same agreed speed. Data is transmitted using the UART frame format.

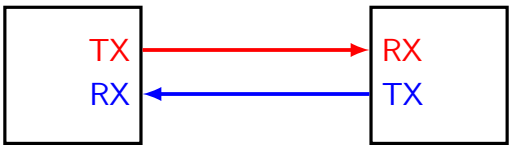


Figure 2.5: UART logic signals

2.4 Software Development

2.4.1 Methodology

Most software development methodologies are designed for use within a team of programmers working over a long period and are not suited for a Part II project. I therefore integrated key components from several software development methodologies, particularly the waterfall and agile methodologies.

The waterfall model of software development is based around the five stages:

Requirements → System Design → Implementation → Testing → Maintenance

From the waterfall model of software development, I took the focus on planning and documentation, particularly as I wanted others to be able to reproduce my results.

Agile software development is based on 12 principles that highlight responding to change and generating working software [18].
I integrated the tenets *"Simplicity – the art of maximizing the amount of work not done – is essential"* [19] and *"Welcome changing requirements, even late in development."* [19] into the project.

The work to be done was broken down into blocks, with concrete deliverables as criteria for completion of each block. This helped prevent scope creep and keep me on track for each part. I also tested each component of the project as it was built, ensuring that it worked and making debugging the final product easier.

2.4.2 Programming

This project used CircuitPython, a branch of MicroPython, an implementation of Python 3 for microcontrollers. CircuitPython was used as it can easily be run on the Raspberry Pi Pico. Additionally, it allowed me to use Pylint to statically analyse code. This was particularly useful as the code was run on an external Raspberry Pi Pico, so running the code to find small errors would have been time consuming. Additionally, the existing AdaFruit libraries I used in the project were written in CircuitPython, so keeping the whole project in the same language reduced complexity.

GitHub was used to perform version control on the code and dissertation for this project. This also allowed me to fork others' repositories and submit a pull request to AdaFruit.

I used Visual Studio Code as my main integrated development environment (IDE) as I have used it in previous projects, and it offered support for CircuitPython via an extension.

2.4.3 Licencing

The project is licenced under the MIT licence. All libraries used in the project are licenced under the same or more permissive licences. The MIT licence was chosen as it allows the project to use these libraries, and as it allows others to use and expand the system.

2.5 Choice of Hardware

The decision to build the system in hardware was not taken lightly. I was aware that it would bring new challenges, such as unusual bugs and more setup issues. However, it would give a better evaluation from a human point and allow parameters to be more carefully tuned for a use case environment.
Hardware was ordered before the start of the Michaelmas term, in order to guarantee its availability for the project proposal. The main factors in my hardware choices were size, weight, power consumption and cost. Items needed to be relatively small to allow them to fit on a rowing boat. If the components were overly costly, it may prohibit other rowing clubs from producing their own networks.

I chose to use a microcontroller to run the system due to the reduction in power consumption and costs it would offer over a single-board computer. The Raspberry Pi Pico is an microcontroller with a dual-core Arm processor [6]. One unit costs £3.90 at the time of writing [20]. The price factored into my choice of microcontroller so it would be affordable for most rowing clubs. It was also chosen due to its large peripheral set, with support for both UART, SPI, and I2C protocols. This allowed for greater flexibility throughout the project. Figure 2.6 shows the pinout for the Raspberry Pi Pico. For GPS and radio, AdaFruit boards were chosen due to the strong community surrounding the hardware, with the AdaFruit boards being supported by open source libraries that allow rudimentary operations to be performed.
The AdaFruit RFM69 radio has an SPI interface and 500m range, appropriate for the use case as rowing boats will take around 2 minutes to cover 500m. Additionally, I chose to use the 433 MHz industrial, scientific, and medical (ISM) band as it is free to use without licencing, allowing me and other rowing clubs to use it without incurring additional costs.
The CD-PA1616S GPS has benefits similar to the RFM69, with community support and libraries for basic communication. It was additionally chosen as it includes a patch antenna and a relatively short cold start time – these were important for the use case, as a delay in the collision avoidance device starting up reduces the overall utility of the system.

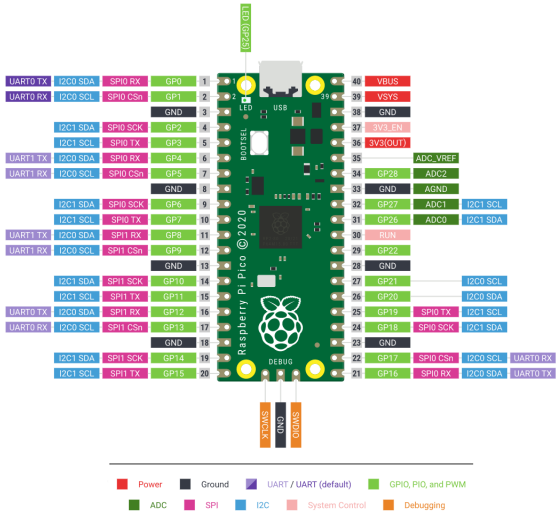


Figure 2.6: The pinout for the Raspberry Pi Pico. Unchanged from Raspberry Pi, licenced under the Creative Commons Attribution-ShareAlike 4.0 licence [21]

2.6 Timeline

The Project Proposal laid out a timeline for the creation of the system. For the most part, this timeline was followed, with the exception of implementing the MANET, which took longer than the time allowed. Figure 2.7 shows a Gantt chart with the proposed and actual timelines.

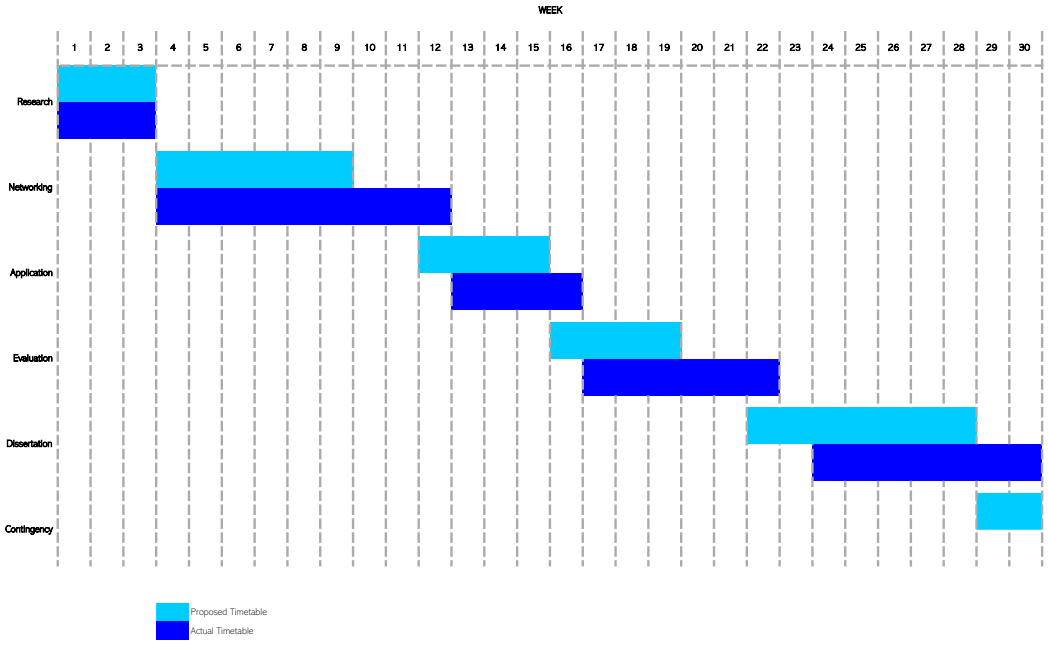


Figure 2.7: A Gantt chart with proposed and actual timelines

Implementation

3.1 Final System

3.1.1 Hardware

Figure 3.1 shows a wiring diagram of the system, produced on Fritzing software, so others can replicate the system. Figure 3.2 shows a virtual breadboard with the system included, again so others can replicate it. Figure 3.3 is an image of the final system, before it is contained in a box. I have soldered the Raspberry Pi Pico 'upside down' so I can more easily view the pin numbers, so while the wiring in Figures 3.2 and 3.3 seems different at first glance, it is all the same.

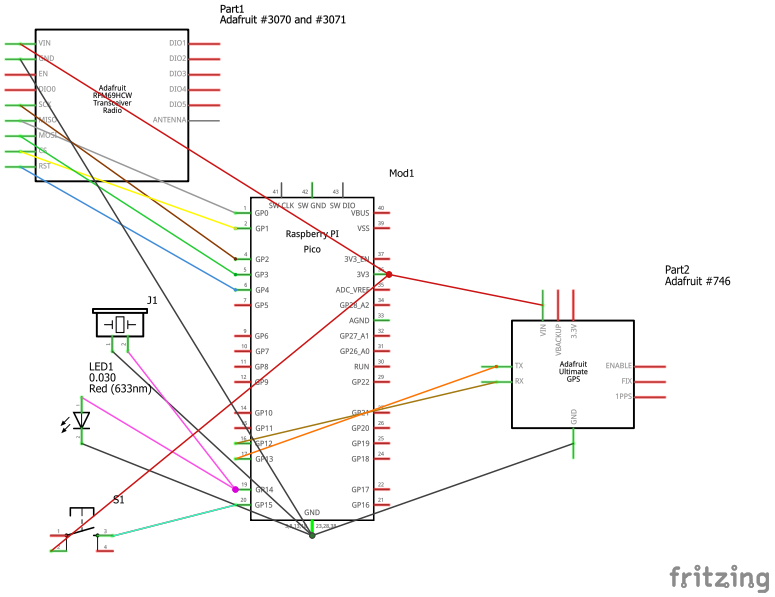


Figure 3.1: A wiring digram for the system

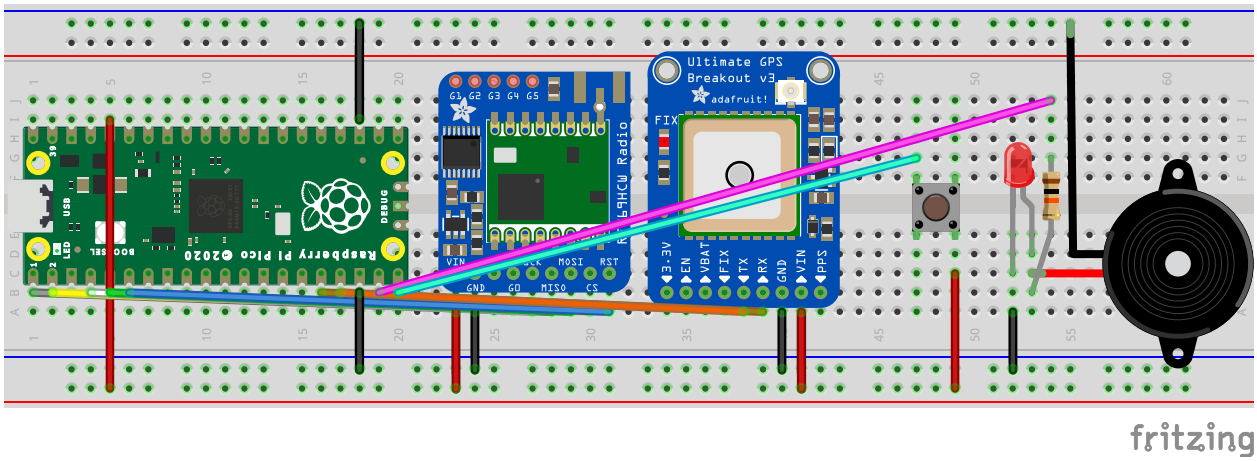


Figure 3.2: A virtual breadboard with the system

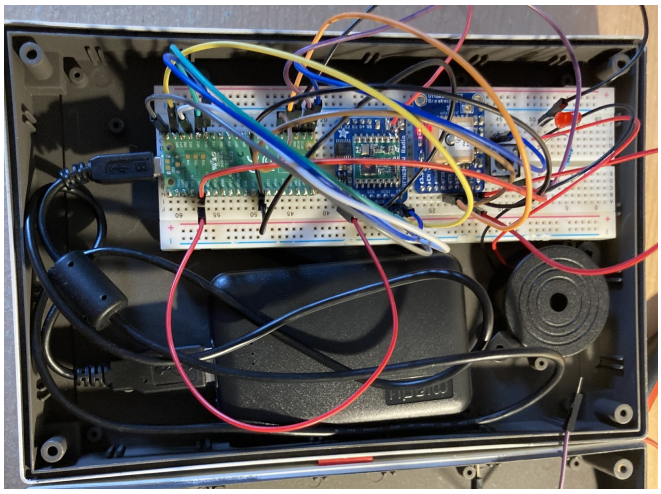


Figure 3.3: The wiring of the final system

3.1.2 Software

Figure 3.4 shows the final state machine representing the software running on the Raspberry Pi Pico. The software differs slightly from the software designed, although both designs have the same effects. The two key differences are in the Epidemic routing protocol and multithreading.
Within Epidemic, routing packets were merged with the medium access control packets to minimise the number of packets used in each anti-entropy session. This is discussed in more detail under the Epidemic section.
Unfortunately, some of the hardware did not have libraries that worked in MicroPython. Therefore, the system was not multithreaded, but ran all on one thread. This is further detailed in the MicroPython section.

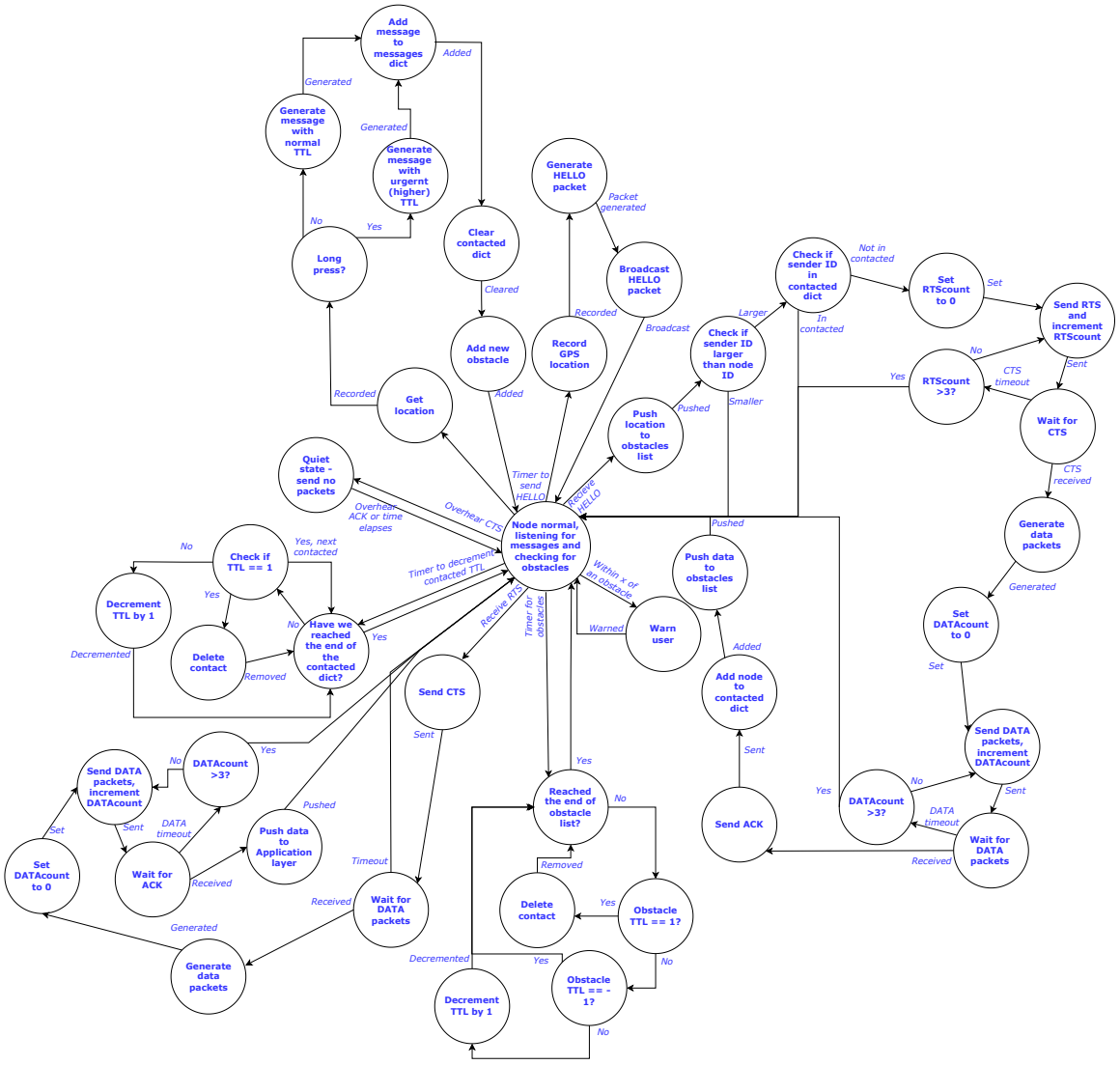


Figure 3.4: The final state machine representation of the software

3.1.3 Packet Design

The structure of the packets sent within the network is standardised. These structures are influenced by the structures of packets in previous routing protocols I examined,particularly BATMAN [15]. I attempted to keep the structure of the start of the packets Figures 3.5 and 3.6 show the overall design for packets sent on the network. The preamble for all packet types start with the length of the packet so the system knows how much data to expect. Figures 3.7 to 3.9 show the breakdown of the payload by packet type, given that the preamble is the same for all packets.

The address `0x00` is designated for broadcast, then each node numbered sequentially and uniquely. This limits the number of nodes a network can contain to be 255.

Bytes / Offset	0	1	2	3
0	Length	Packet Type	Sender	Destination
1	Payload			
2	Payload			
-	.....			
14	Payload			
15	Payload			

Figure 3.5: The preamble for all packet types

Packet Type	Hexadecimal Representation
HELLO	0x00
RTS	0x01
CTS	0x02
DATA	0x03
ACK	0x04

Figure 3.6: The types of packet

Bytes / Offset	0	1	2	3
1	GPS latitude			
2	GPS longitude			
3	Padding			
-	.....			
15	Padding			

Figure 3.7: The payload of HELLO packet type

Bytes / Offset	0	1	2	3
1	Message ID 0		Message ID 1	
2	Message ID 2		Message ID 3	
-	.....			
14	Message ID 26		Message ID 27	
15	Message ID 28		Message ID 29	

Figure 3.8: The payload of CTS and RTS packet types

Bytes / Offset	0	1	2	3
1	Total number of DATA packets expected	Number of this DATA packet	Requested Message ID 0	
2	Requested Message Data			
-	.....			
14	Requested Message Data		Requested Message ID 7	
15	Requested Message Data			

Figure 3.9: The payload of DATA packet types

3.1.4 Open Source

The system was designed and documented so it could be reproduced by other rowing clubs and used to help prevent crashes. Due to plagiarism rules the repositories and documentation, including README, have not yet been made available. They will be made public immediately after my graduation.
The repository is licenced under the MIT licence, allowing others to use and expand on the system.

3.2 System Design

Before designing the system, it was decided the system would identify obstacles using GPS coordinates. The Epidemic routing protocol would be used to propagate obstacles through the network. Users need to be notified of upcoming obstacles and be able to add new obstacles.
From these criteria, the structure of the software to run on each node was designed. The Raspberry Pi Pico uses the RP2040 chip, containing two cores [6]. To make best use of the hardware, application and networking threads were designed to run on each core. Global data structures and concurrency control were chosen to pass messages between the two layers and allow both layers access to the GPS.

While these state machines were broadly implemented, the overall structure of the software changed, with only one thread running due to the limitations of CircuitPython and difficulties transitioning into MicroPython.

3.2.1 Application

The application thread was responsible for notifying the user when they are too close to an obstacle and allowing them to add obstacles. The initial state machine is shown in Figure 3.10.

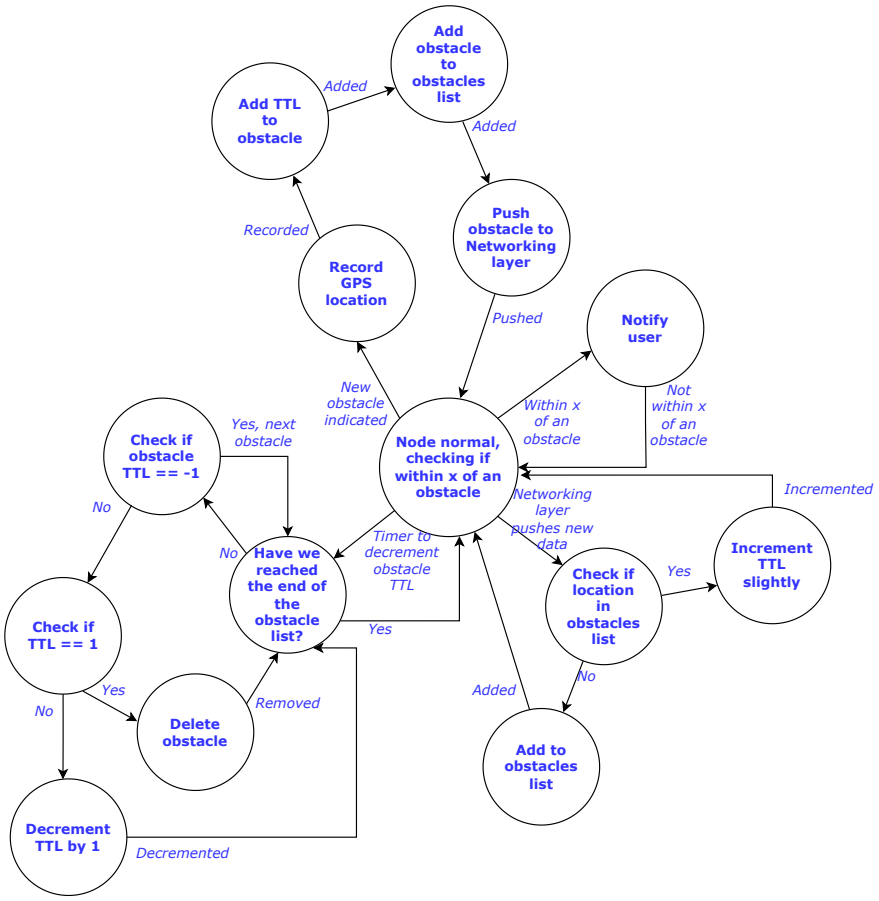


Figure 3.10: The initial state machine for the application thread

3.2.2 Networking

The networking thread was designed to propagate messages through the network. It contained the implementation of Epidemic with medium access control. While routing and medium access control are typically handled separately – in the OSI model they are in the second and third layers respectively – they were considered together in this system to prevent work being repeated and use limited computing resources effectively. These traditionally separate layers were combined as some data structures, such as the contacted node dictionary, would need to be shared between layers, increasing complexity and compute time. Figure 3.11 shows the Networking machine.
This meant Vahdat and Becker's implementation of Epidemic was adjusted. A quiet state was defined, where a node did not send any messages after hearing a CTS for a set period of time, or until it hears an ACK. The message vector exchange was combined into the CTS and RTS messages. This minimised the number of packets sent over the network, reducing the probability of collisions or errors in transmission.

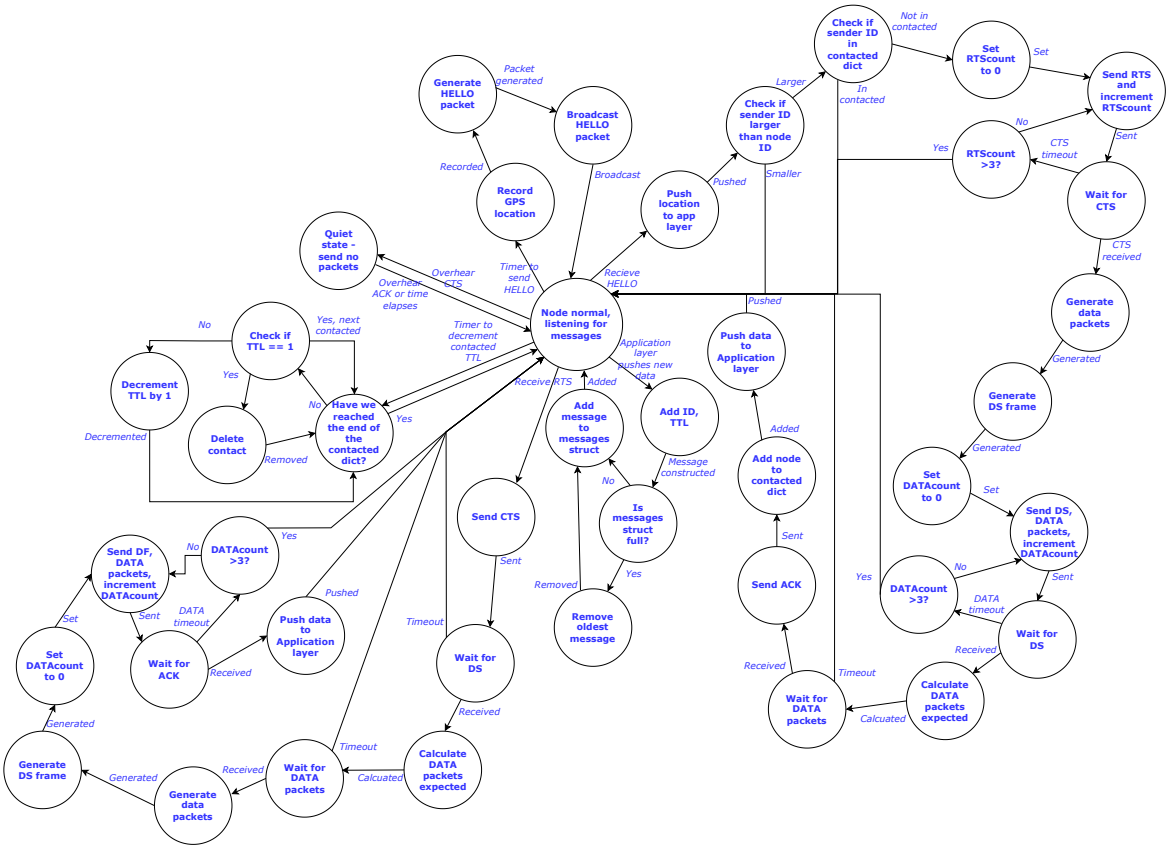


Figure 3.11: The initial state machine for the networking thread

3.2.3 Data Structures

3.2.3.1 Global

The global data structures would be accessed by both threads, with associated locks to ensure that both threads do not access the structure at the same time.

Structure	Construction	Notes
Location	Tuple of (longitude, latitude)	Calls a function that communicates with the GPS board
ApplicationToNetwork	Queue of obstacles	Application machine pushes user defined obstacles onto queue so they can be propagated through the network
NetworkToApplication	Queue of obstacles	Network machine pushes messages propagated through the network to the application layer

3.2.3.2 Application

Structure	Construction	Notes
Obstacles	List of obstacles and their TTL	A list of obstacles and their TTL. Any obstacles with a TTL of -1 are considered permanent

3.2.3.3 Networking

Structure	Construction	Notes
Messages	Dictionary of {Message ID : [Latitude, Longitude, TTL]}	The messages' ID is constructed from the generating node ID and the number of messages that node has generated

3.3 Debugging

Throughout the project, different pieces of hardware and unusual issues needed to be debugged. To debug hardware, a multimeter was used to detect if data was being sent along various connections. On occasions when the multimeter could not be accessed, the anode of an LED was attached to the connection being examined and the cathode grounded. The LED would light up when data was passing through this connection.
When debugging more persistent issues, such as the MicroPython-CircuitPython communication, my supervisor kindly lent me a logic analyser. Figure 3.12 shows a node being examined with a logic analyser.

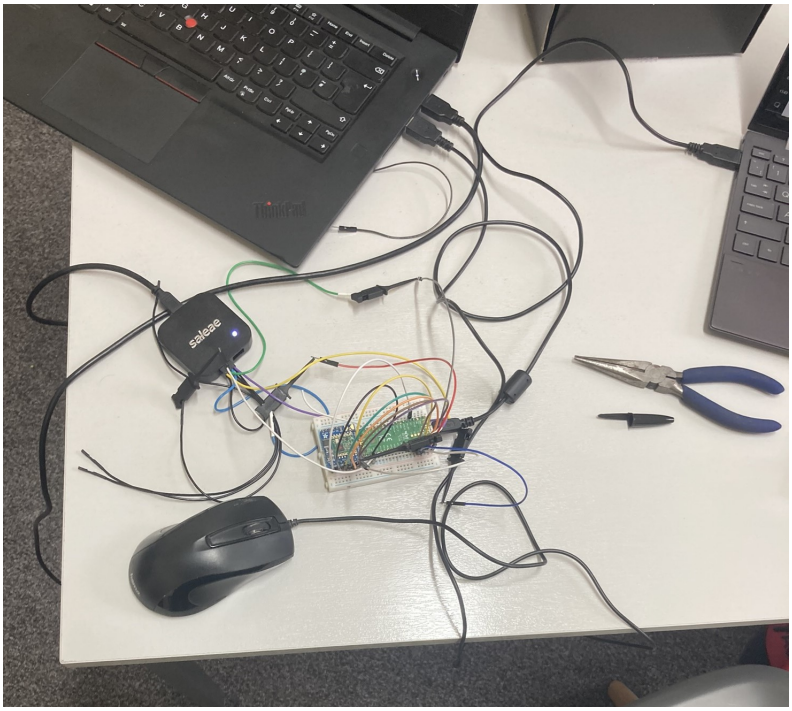


Figure 3.12: A logic analyser being used to examine the signals sent from the Raspberry Pi Pico to the RMF69

3.4 Logging

Logging was done via a bespoke class. The class had functions to handle different types of log. It output series of statements, printed and saved to the flash memory on the Pico in a 'logging.txt' file. This allowed easy modelling of the relationships between nodes. The docstring from the logging class reads:

```
Class with multiple functions to handle logging

Logs:
    Node Address, Time, Time Since Node Startup, Event Type, Event Information

Event Types:
    0   -   General logging string
    1   -   Logging function
    2   -   Logging packet
```

```
3   -   Logging messages
4   -   Logging error
5   -   Logging GPS location
6   -   Logging an alert
```

3.5 Point to Point

Before starting to implement Epidemic, I ensured that the two nodes in line of sight could communicate. This was successful; each node could send a packet to another node. Several interesting variables were found in the communication between point to point nodes.

A quarter wave whip antenna was used – a piece of wire cut to 17.4 cm (1/4 of the wavelength) and soldered to the antenna port and ground plane. I found that the thickness of the wire had an impact on the signal, with thicker wire having a longer range. This is likely due to the lower resistance of ticket wire. Figure 3.13 shows the antenna of varying thickness on three of the radio. The antenna were one of the main practical difficulties I had with the choice to implement the system in hardware. They had a tendency to break or fall off and a small error in application results in a significant reduction in communication range.

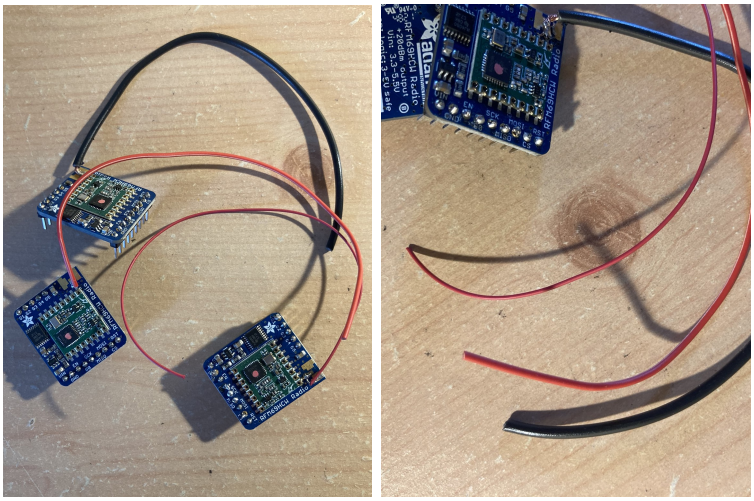


Figure 3.13: A selection of antenna on the radios

It was also found that when the rate and length of packets sent increased, more errors were found in transmission and decoding. To fix this, I wanted to move to fixed length packets, padded with 0s. However, the `adafruit_rfm69` library did not allow for fixed length packets. I therefore added this functionality. I consulted the RFM69HCW datasheet [22], set the packet format register to 0 (indicating fixed length) and the packet length to 64 (the size of the FIFO buffer). I integrated configurable length fixed length packets into a fork of the `adafruit_rfm69` library [23] and have submitted a pull request so others can use this functionality.

3.6 Epidemic

3.6.1 Discovery Messages

The point to point messages were then moved to sending and receiving discovery messages, of type `0x00`, also known as HELLO packets. These messages used the broadcast address, `0x00`. These

packets used structure defined in the Final System section. To test discovery packets, it was ensured that HELLO packets were correctly sent and received. The payload of HELLO packets contained the node's current GPS location, so it could be recognised as an obstacle by other nodes. When first initiated, this location was generated by a stand in function that gave a random set of numbers, later replaced by a call to the GPS board.

3.6.2 Anti-Entropy

When a new node is discovered, anti-entropy will occur. An anti-entropy session allows nodes to exchange messages and occurs when two nodes come into communication range. In Vahdat and Becker's Epidemic paper [9], anti-entropy is initiated by the node with the lower ID to prevent collisions. Figure 3.14 shows the first half of anti-entropy. Node A, with a lower ID, initiates anti-entropy by sending a summary vector (SV_A) to node B. This summary vector is represents the messages node A holds. Node B calculates the logical AND between this summary vector and the negation of its own summary vector (SV_B) to produce the messages it has not seen, but the other node holds. It sends this request ($SV_A + \neg SV_B$) to node A. Node A responds by sending the requested messages to node B.
The roles are then reversed, with node B sending its summary vector (SV_B) to A. A requests messages it has not seen from B ($SV_B + \neg SV_A$) and receives them.
After a successful message exchange, node B is added to node A's list of recently contacted nodes, preventing anti-entropy from being repeatedly conducted between two nodes. This list is periodically cleared.

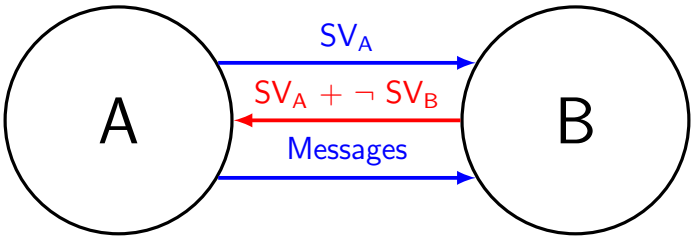


Figure 3.14: The first stage of anti-entropy [9]

The implementation of Epidemic routing in this project is broadly the same as Vahdat and Becker's, initiated by the lower ID node. However, the implementation in this project combines medium access control with the exchange of summary vectors, with SV_A being combined with the RTS and SV_B combined with CTS. These contained the message keys that are held on that node.
The other key difference is in requests. As all nodes use the same mechanism to construct a request to send to the other nodes, this can be moved off node. Once a node has received the summary vector from the other node, it will calculate the messages the node wants by comparing the message keys each node holds ($SV_A + \neg SV_B$). This removes two messages from the anti-entropy process, reducing the opportunity for error. Figure 3.15 shows updated anti-entropy.

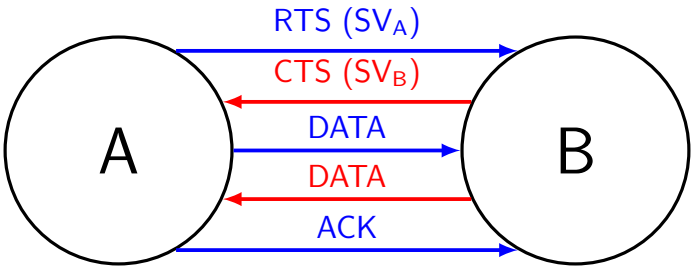


Figure 3.15: The version of anti-entropy I implemented

The message exchange in the system design was also tweaked. Again, these changes were made to reduce the number of packets sent, and therefore the bandwidth taken up by the anti-entropy

process and possibility for errors. The design has DATA FRAME (DF) and DATA packets, where the DF packet sets up the number of expected DATA packets. Instead of sending a separate packet, the number of DATA packets expected was designated the first byte of the payload. Each DATA packet given an ID, its number from the total. When the initiating node (A) has sent all its packets, it waits for the DATA packets sent from the other node as an acknowledgement all DATA packets have been received. If DATA packets are not seen within the timeout, set in the `config.py` file, the node resends the DATA packets. If this is repeated as many times as defined in the `config.py` file, the initiating node (A) ceases anti-entropy and enters the listening state again. If node A successfully receives all DATA packets from node B, it sends an acknowledgement (ACK) to indicate no resends are necessary and ends the anti-entropy process.

When the non-initiating node (B) sees it has received all the DATA packets from node A, and has sent all DATA packets, it waits for the ACK from A. If unseen after a certain time, specified in the `config.py` file, it resends the DATA messages. Should the resends still not be acknowledged, the anti-entropy times out. Figure 3.16 shows a timing diagram for the implemented Epidemic routing.

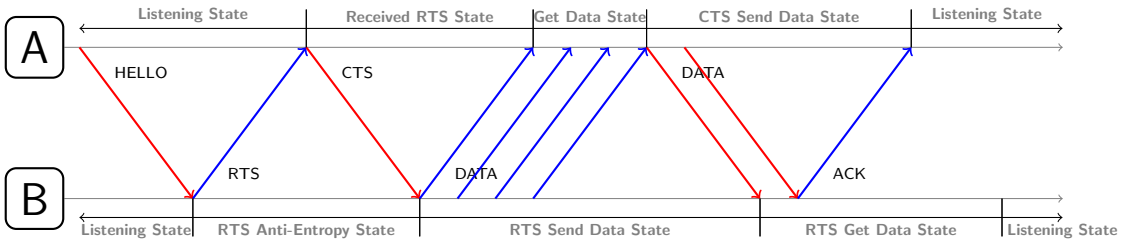


Figure 3.16: Messages and states of two nodes completing anti-entropy with no missed messages

These variables remain configurable to allow each user to choose values that work best for their use. The sensitivity analysis was the basis for the values I used when Evaluating and using the system.

3.6.3 Pseudocode

Figure 3.17 shows an abstracted and commented version of the Epidemic routing. Note that logging, error messages have been cut out. The full code is available in the code repository, under `/OnDevice/DeviceCode.py`.

```
if state == config.LISTEN:
    args = rfm69.receive()
    # Update the state based on any packets received
    state = handleReceive(args)

elif state == config.RECEIVED_HELLO:
    # Add the location of the node that sent the HELLO to the obstacles list
    obstacles.append([float(newObstacle[0]), float(newObstacle[1]), 1])
    if sender not in contacted and sender>config.ADDRESS:
        # Anti-entropy initiated by this node
        success, messages = RTSAntiEntropy(dest = sender, messages = messages)
        if success:
            # Add the other node to the dictionary of contacted node
            # on successful anti-entropy
            contacted.update({sender : config.CONTACTED_LIVES})
    state = config.LISTEN


elif state == config.RECEIVED_RTS:
    # Initiate anti-entropy from the side of sending a CTS
    success, messages = CTSAntiEntropy(sender, messages, RTSpacket)
    state = config.LISTEN

if state == config.LISTEN and timers.hello():
    # Broadcast HELLO packet
```

Figure 3.17: Abstracted Epidemic routing code

3.6.4 Sensitivity Analysis

The value of each configurable value in the network was determined experientially. The most important of these was how often a HELLO packet would be sent. This has a direct impact on how often anti-entropy would be initiated, and therefore the latency of messages.
Figure 3.18 shows the latency of 30 messages being sent between two nodes with interference from another node. From this experimentation, I decided to send HELLO packets every 5 seconds (with a random jitter). This was a trade-off between collisions of HELLO packets, the power consumption involved in sending packets, and the latency of packets sent through the network. The average latency of a message in when HELLO packets are sent every 5 seconds was 17 seconds.

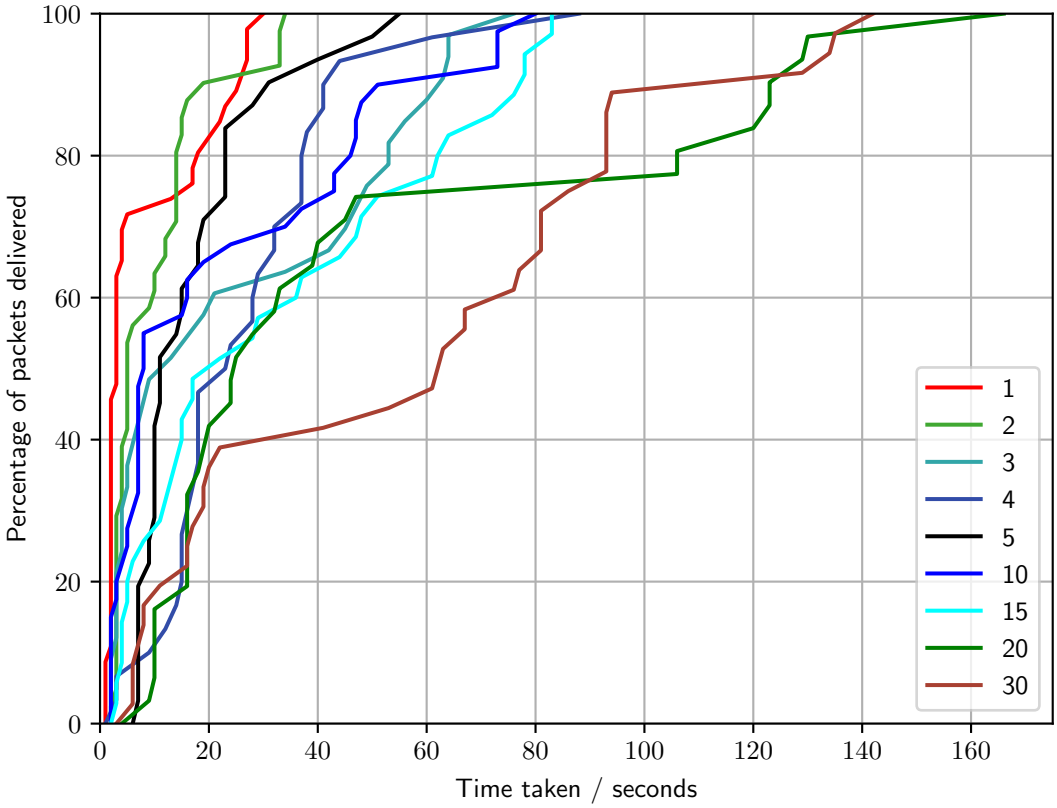


Figure 3.18: The time taken to deliver messages over distance intervals

3.6.5 Testing

Testing the Epidemic implementation ensured that it worked, particularly with edge cases. The edge cases tested involved:
Both nodes containing full messages dictionary, with 30 messages
One node holding some messages and one node with no messages
Both nodes having identical messages
Both nodes holding no messages

This testing highlighted an issue with anti-entropy, where no DATA packets were sent if there were no messages the other node needed. This was rectified by sending one empty DATA packet to indicate that the node had no new messages.

Once the edge testing was finished, I demonstrated the utility of the network by allowing the user to type in a message on the terminal of a laptop on one node. This message was passed to another node and displayed on the attached screen.

3.7 Medium Access Control

Medium access control was implemented alongside the Epidemic routing protocol, detailed in the previous section. If a node overheard the exchange of an RTS / CTS message exchange between two other nodes, it enters a silent state. The node does not broadcast any HELLO or CTS messages, nor respond to RTS messages. The node remains in the silent state until an ACK is seen, sent by the appropriate node matching the exchange of RTS / CTS messages, or until the timer has

elapsed, based on a time set in the `config.py` file, currently set to 15 seconds. This timeout was chosen as a worst case scenario for a very long exchange of messages between two nodes, so even an unusually long exchange will not be interrupted.

3.7.1 Pesudocode

```
if packet == CTS:
    state = config.QUIET
    # Record the node that sent the CTS
    # Allowing us to check if any ACKs overheard are correct
    sendingCTSNode = sender

if state == config.QUIET:
    args = rfm69.receive()
    # Listen for the ACK connected with the node pair
    # Wait for timeout the timer until
```

Figure 3.19: Abstracted version of the code for medium access control

3.8 GPS

Before integrating the GPS into the system, I ensured that it worked. Several different serial communication protocols were used, staring with SPI. In the end, UART was used as the asynchronicity worked best with the GPS. To ensure that the GPS generated the correct coordinates, I checked that the coordinates given were my location.

I found that the GPS had a long time to first fix (TTFF) on a cold start. This was because there was no long term memory when power was lost to the device, so any real time clock (RTC) and downloaded almanac data would be lost. A holder for and CR1220 coin cell were added to allow the GPS to keep accurate RTC data when the system was powered off. This significantly reduced the TTFF. Additionally, it allowed the GPS chip to use its built in EASY assist system [24], which helps with quick positioning by calculating up to three days of ephemeris data and saving these predictions.

The messages generated by the GPS chip are National Marine Electronics Association (NMEA) 0183 messages, which are then parsed by the system using the `adafruit_gps.py` library.

3.8.1 Precision

To test the tracking on the GPS, a short program was written to generate GPX files. A brief walk was tracked with the GPS. For comparison, the same activity was tracked with a smart watch. Figures 3.4 and 3.5 show the GPX files overlaid on Google Maps.

As can be seen from Figure 3.4, the track generated by the system's GPS was jagged and inaccurate. On examination of the GPS coordinates generated by the board, they are only precise to five significant figures, so it is unsurprising that the coordinates are not precisely accurate. Therefore, the degrees and minutes were combined to generate more precise coordinates, to eight significant figures. Figure 3.5 shows the GPX track with the more accurate coordinates. Based on the route taken when walking, the board's GPX track is slightly more accurate than the smartwatch's.

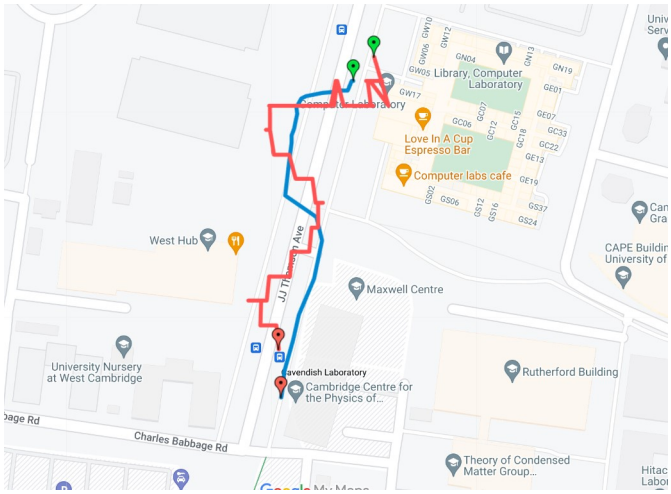


Figure 3.20: The first attempt at tracking a walk with GPS. The red line is the GPX track generated by the board, the blue line is the GPX track generated by a smartwatch [12]

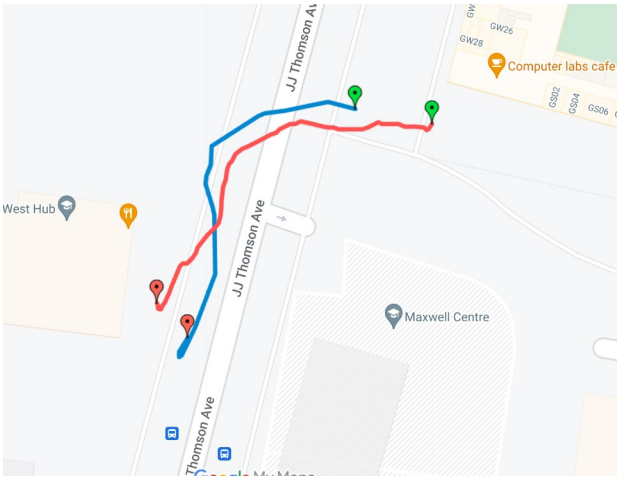


Figure 3.21: The final attempt at tracking a walk with GPS. The red line is the GPX track generated by the board, the blue line is the GPX track generated by a smartwatch [12]

3.8.2 Haversine Formula

The haversine formula was used to calculate the distance between two sets of GPS coordinates. This was chosen as it is relatively simple to compute. The haversine formula is very commonly used to find distance between two sets of GPS coordinates [25]. It is notable that this formula finds two points on a sphere, and the earth is an ellipsoid. The radius of the earth is not consistent across the globe. The average radius of the earth was used instead. As this is an approximation, it will result in errors of up to 0.5% [25]. The distances being measured are relatively small, so this was not a problem.
Vincenty's formulae takes the ellipsoidal nature of the earth into consideration, but is more complicated and due to the small distances involved it was decided that the saving made on compute time would be more beneficial than the additional accuracy in distance measurements.

3.8.2.1 Equations

The haversine is defined:
$haversine(\theta) = sin^2(\theta/2)$

Where ϕ is latitude and λ represents longitude and $haversine(\theta)$ is abbreviated to $hav(\theta)$.

$hav(\theta) = hav(\phi_2 - \phi_1) + cos(\phi_1)cos(\phi_2)hav(\lambda_2 - \lambda_1)$
θ is the angle between the two points on the sphere. d is the distance between the points and r is the radius of the sphere.
$\theta = d/r$

Combining these equations gives us the distance between the two points as
$d = r * 2 * sin^{-1}(\sqrt{sin((\phi_2 - \phi_1)/2))^2 + cos(\lambda_1) * cos(\lambda_2) * sin((\lambda_2 - \lambda_1)/2))^2})$

3.8.3 Satellite Error

It was found that the GPS location moved even when the board was stationary. I believed this to be an error in the library, perhaps related to the precision of floating point numbers. However, research indicated that this was due to noise at the receiver causing the GPS location to move around. This noise was due to the clock phase noise and interpolation of ephemeris data [26]. Implementing a Gaussian or Kalman filter to remove some of the noise would be possible and is further work. Another solution to this problem is demonstrated by the telemetry in the Boat Race, where an external antenna is uses and compared to the values from a base station that has been previously gathering data.

3.9 MicroPython

The libraries to allow rudimentary interface with the RFM69 and GPS boards were both written in CircuitPython. I opted to use these libraries to interface with this hardware rather than write my own libraries.
MicroPython supports both multithreading and interrupts, neither of which CircuitPython supports. It was therefore prefered to use MicroPython to implement the core of the system, particularly so the application and networking threads could run concurrently, maximally utilising the two cores in the Raspberry Pi Pico [6]. I ensured the Raspberry Pi Pico could run two threads by defining two functions that calculate the Fibonacci sequence using two global variables and a lock to ensure concurrency control. This code successfully calucalted the Fibonacci sequence when I allocated a thread to each core.

CircuitPython should have a library, Blinka [27], that emulates the CircuitPython API, and allows hardware operating in CircuitPython to work with MicroPython. However, when attempting to run a simple sending packet and receiving packet setup, I found the code did not work. To debug this problem, I ran two nodes trying to send messages to each other one running CircuitPython and one running MicroPython with the Blinka libraries. A third node read any packets that were sent and logged them. The MicroPython node continued not to receive or send any data. I lowered the baud rate in case the Blinka or SPI interfaces were being overwhelmed. Even with a dramatic reduction, no change was found.
This was when my supervisor kindly lent me his expertise and logic analyser. We found that while the Pico was sending data to the RFM69, it was sending it twice, and the RFM69 clearly did not know how to deal with this so did not respond.

Rather than trawl through the Blinka library looking for the bugs that caused this, we decided instead to use CircuitPython for the whole system. This meant it had to be redesigned to run on only one core. This ensured project time was spent working on the system, rather than searching for a bug.

3.10 User Interface

3.10.1 Hardware Choice

In order for the MANET and GPS to be useful, a rudimentary user interface was built. The plan initially contained only a buzzer and button. An LED was added to allow the device to be more accessible for those who cannot hear the buzzer. Therefore, a button with a built in LED was chosen. This kept the design of a node simple, so easy to construct, and less likely to get moisture inside the box. The button chosen was also waterproof, the system will get wet in rowing boats. It was a momentary, non-locking, relatively large button to allow the user to effortlessly press it when they encounter an obstacle.
A 2kΩ resistor was chosen to be used in series with the buzzer. This resistance modulates the tone of the buzzer. A brief survey of rowers indicated the eventual tone to be most suitable as a warning.

3.10.2 Integration

The buzzer and LED were wired in parallel to the same pin in the Pico, to allow the flashing and buzzing to occur simultaneously. This is shown in the wiring diagram under the Final System section. If the system detected that the current GPS position of the node was within the distance specified by the `GPS_DISTANCE` value in `config.py`, using the haversine formula.

As CircuitPython does not allow interrupts, the value of the button was polled. If the value was true, the button was pressed. The value was polled again after 0.5 seconds. If the value was still true, this was considered to be a 'long' press, meaning the obstacle that had been detected should be marked as urgent and propagated through the network at an accelerated rate.

With the user interface integrated, the third success criteria, *'An application layer had been implemented to allow utility of the network'*, was achieved.

3.10.3 Testing

The testing of the user interface first ensuring each piece of hardware worked. I wrote a short program to flash the LED and sound the buzzer when the button was pressed.
The next stage of testing ensured the system generated new obstacles, adding them to the message buffer, when the button was pressed. This was done by generating messages and ensuring that these messages were received by nearby nodes.

A surprising issue occurred when a node was powered from a power bank, where the node would shut down in a nondeterministic fashion. After experimenting with other power packs and powering the node from mains power, I decided that the power bank likely had some fail safe if a certain level of current was drawn from the power pack, it shut off power as a safety concern. This was rectified by using a different power bank, although batteries could also be used.

3.11 Repository Overview

- LICENCE
- README.md
- OnDevice
  - boot.py
  - lib
  - This is the final product, the code that should be uploaded to the Raspberry Pi Pico. The boot.py file contains the implementation of the system to help avoid collisions. The lib file contains `adafruit_gps.py`, a library I did not write but use, licenced under the MIT licence. `rfm69.py` is a modified version of `adafruit_rfm69.py`, which I have modified to include fixed length packets and fixed some bugs, so it is a combination of code I wrote and others' code.
- Development
  - Application
  - MACEpidemic
  - Misc
  - Contains the code used in development of the system. `MACEpidemic` and `Application` contain the code generated when building the MANET and interface to it respectively. `Misc` contains a more varied set of files, generated when trying out hardware and other small components of the system.
- Evaluation
  - Split into various folders depending on the evaluation being conducted. It contains the code used to run each subsection of the evaluation, results and python files for analysis of the results.

Evaluation

The goal of the evaluation was to prove correctness and utility. The evaluation also allowed me to compare the MANET to other implementations of Epidemic routing using common benchmarks.
The majority of the evaluation was conducted according to the plan in Appendix B [Appendix B].
Evaluation was split into a performance evaluation of the MANET and an evaluation of the whole system, including a qualitative evaluation.
The key metrics evaluated when considering the MANET were percentage of packets delivered, latency, bandwidth and time taken to propagate message after partition.
The whole system evaluation considered the GPS readings taken by the system and its interaction with the user.

4.1 Evaluating the MANET

The evaluation of the MANET was designed to mirror the evaluation performed by Vahdat and Becker's paper, with the key difference that my project was implemented in hardware, while theirs was simulated. The similarities between the evaluations allowed me to prove correctness through comparison of results. It also offers insight into both implementations and any improvements that could be made.

4.1.1 Latency

The latency of a message is the time between the message being generated at node A and received at node B.
Node B started next to node A and was moved further away, finishing at a maximum distance of 500 metres. There was a third node, C, interacting with the {A, B} pair. This allowed messages to pass over a greater distance. The distances (denoted d) between nodes A and B were 0, 10, 100, 250 and 500 metres. 60 messages were generated by A at each distance. Figure 4.1 shows the setup, with radio connections between nodes marked in blue and the distance measured in grey.

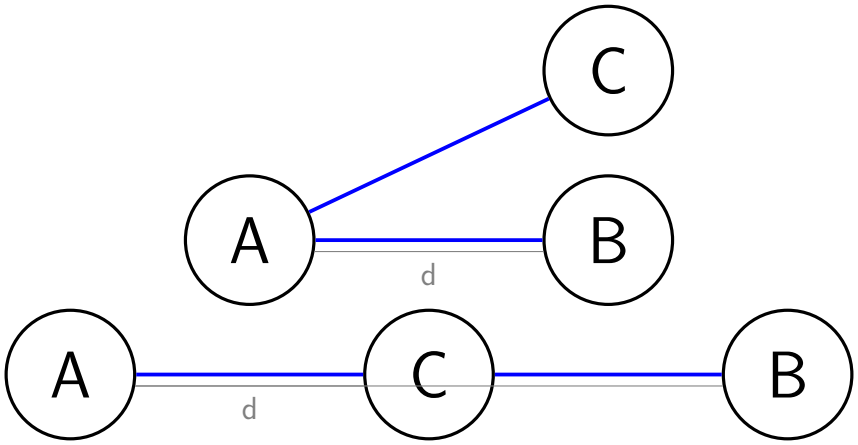


Figure 4.1: The network used for latency testing

The results of the latency evaluation are shown below. Figure 4.2 shows the percentage of packets delivered over time. Figure 4.3 breaks down the data behind this, showing the times that each packet arrived at each distance. Table 4.1 shows the raw data. The metrics for 0 and 10 metres differ little, with an average delivery time of 15.0 and 15.7 seconds respectively. The 100 metre interval follows a similar pattern. There is a significant drop-off when the distance is increased to 250 and 500 metres, likely because more than one hop is needed to move messages from one node to another. There is a significant increase in the standard deviation, due to the increased variability from passing messages over multiple nodes. As distance increases, the latency increases. This is probably due to the spreading of signal causing more errors in transmission.
At first glance, these latencies seem unusually high. When compared to the latencies in Vahdat and Becker's paper [9], the latencies in the paper are also high. This is due to the design of Epidemic routing. Epidemic routing is optimised to transmit messages through networks with a high chance of partition, rather than being optimised for latency. A message is not sent as soon as it is generated. Instead, neighbouring nodes periodically poll each other to find any messages that are not on both nodes. This polling also explains the increased variability in delivery times when more than one hop is taken, as the anti-entropy process needs to occur multiple times to allow a message to reach its destination.

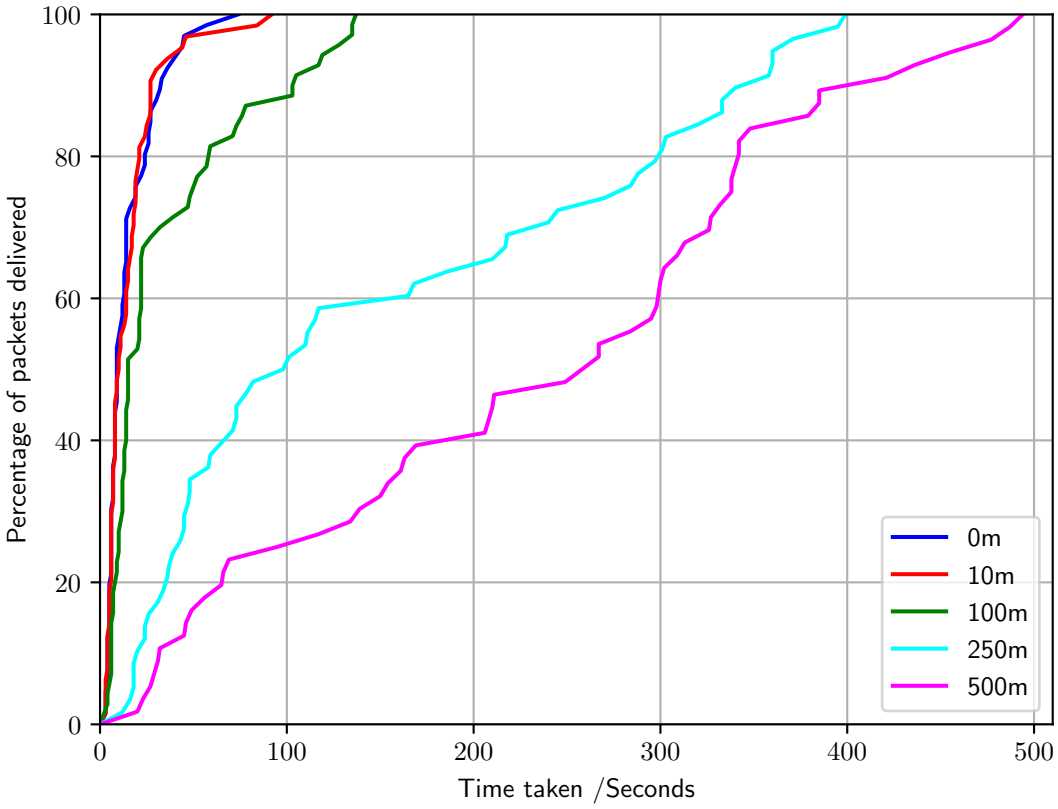


Figure 4.2: The percentage of packets delivered with time

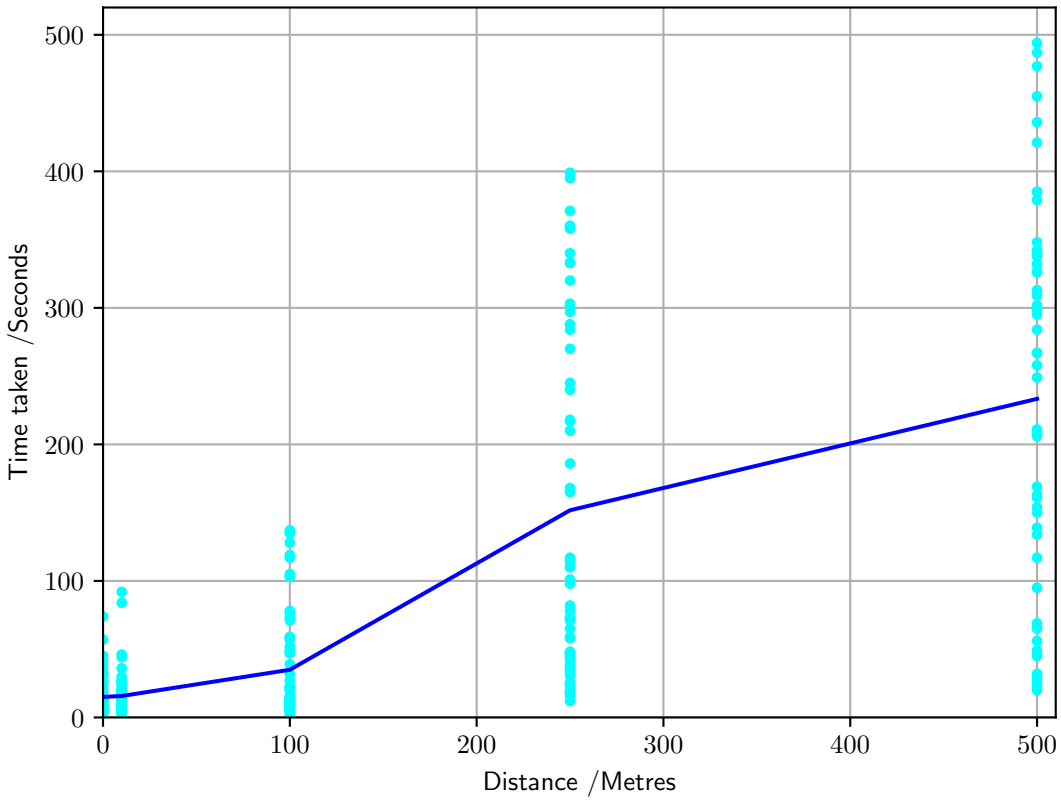


Figure 4.3: The time taken to deliver messages over distance intervals. The dark blue line represents the average, the cyan dots represent individual datapoints

Distance /Metres	Average Latency /Seconds	Standard Deviation /Seconds	Minimum Latency /Seconds	Maximum Latency /Seconds
0	15.045	13.796	1	57
10	15.703	16.199	3	92
100	34.914	37.966	2	137
250	151.741	126.721	12	399
500	233.357	138.679	20	494

Table 4.1: Packet latency

4.1.2 Bandwidth

Evaluating the bandwidth of the system was done to work out how many messages the MANET can transfer. In reality, it is unlikely the MANET will need to process as many messages as sent in this test as obstacles will not be generated this quickly by human users. Bandwidth testing was conducted between two nodes, with one node generating messages at a set rate and broadcasting them to the other node. Messages were generated at an increasing rate until the number of messages received by node B plateaued. Figure 4.4 shows the setup for this test.

As shown in Figure 4.5, the MANET has a relatively low bandwidth, with plateauing at around 0.4 messages transferred on average each second, equivalent to taking around 3 seconds to pass one message from node to node. To improve the bandwidth of the system, the time between discovery (HELLO) packets sent by each node could be decreased.

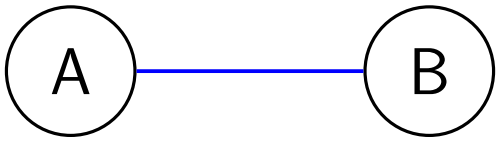


Figure 4.4: The network used for bandwidth testing

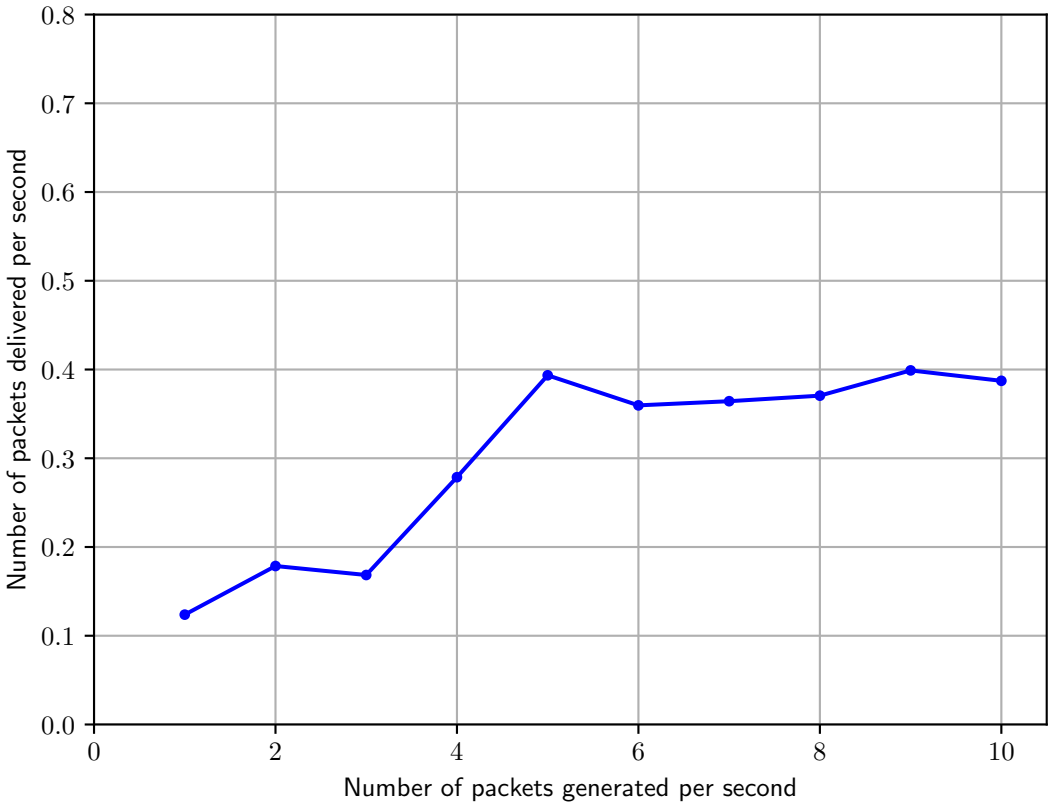


Figure 4.5: The bandwidth of a fully connected two node network with an increasing number of messages generated each second

4.1.3 Percentage of Packets Delivered

With the exception of bandwidth testing, when more packets were generated than could be delivered, 100% of packets were delivered throughout testing. This mirrors the results found by

Vahdat and Becker, where their simulation gave a 100% delivery rate when the number of messages was smaller or equal to the size of the message buffer.

4.1.4 Five Node Field Test

The percentage of packets delivered is best illustrated by the test with five nodes. To run this test, I recruited four volunteers to move around a large field, approximately 500 x 500 metre area. This allowed nodes to move in and out of range of each other. Each node had a 0.1 probability of generating a message each second, with up to 30 messages being generated. The nodes moved around this test area for approximately 15 minutes. This is a reduced version of the evaluation performed in Vahdat and Becker's paper [9], where 50 nodes were simulated within a 1500 x 300 metre area.
This test was run twice, with results being compared to ensure no anomalies occured. In both these tests, 100% of packets were delivered. Unfortunately, Vahdat and Becker do not release the implementation details for their paper, so a direct comparison of the two implementations is not available. However,

4.1.5 Partitions

The network was broken into smaller networks to ensure that messages are propagated through the network after partition and compare the time taken to propagate messages after partition.
Partition evaluation was conducted with four nodes. The first experiments used two pairs. Each pair was set up to communicate with each other, then brought into range of the other pair. Figures 4.6 and 4.7 show the starting and finishing states of the network.

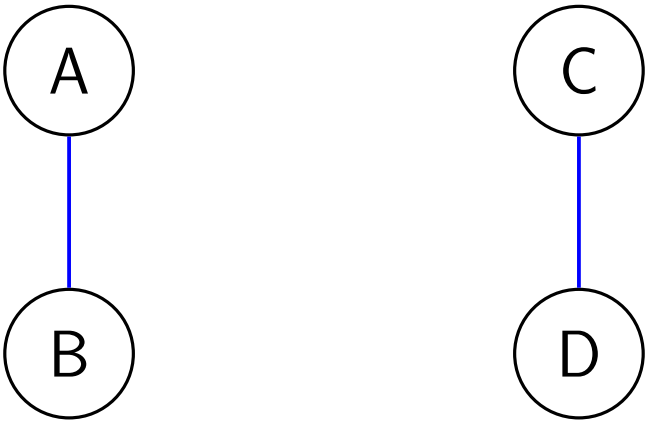


Figure 4.6: The network at the start of partition testing

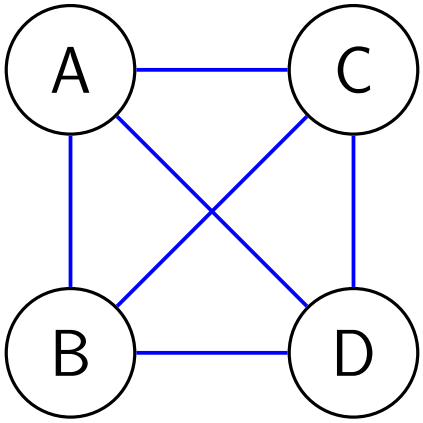


Figure 4.7: The network at the end of partition testing

4.1.5.1 Asymmetric

Asymmetric partition testing consisted of the {A, B} pair holding 30 messages before partition. After the partition was removed, the {C, D} pair held the same messages {A, B} held. Over 2 partitions, the average time taken for all nodes in the network to hold the same messages was 106.5 seconds. Given the approximate speed of a rowing boat is 15km/h, and the radio range is 500 meters, this would allow two boats to be in range for 120 seconds, just enough time for the maximum number of messages to be transferred between two boats.

Run	Time for Node C /Seconds	Time for Node D /Seconds
1	50	135
2	78	44

Table 4.2: Time taken for all messages to be received in asymmetric partitions

4.1.5.2 Symmetric

Symmetric evaluation of the network gave the two pairs of nodes different sets of messages. The average time for nodes to receive all 30 messages was 130 seconds, 23.5 seconds greater than an asymmetric partition.

Node	Time for messages to be received /Seconds
A	87
B	159
C	145
D	129

Table 4.3: Time taken in seconds for all messages to be received in symmetric partitions

4.2 Evaluating the System

4.2.1 On Water

The water evaluation was conducted last. I was concerned that there might be an accident and the project become water damaged or fall into the river, so all experiments before this were conducted on land. The system works on rowing boats, notifying rowers of other boats who are using the system and of registered obstacles.

4.2.1.1 GPS

Rivers and lakes are in many ways ideal for the GPS, being large open spaces with few obstacles to the system's view of the sky. However, I was concerned that bridges or similar obstructions to the sky would interfere with the GPS tracking on each boat. This would be of particular concern as a bridge could be considered an obstacle. I therefore examined the GPS tracking on the water by logging the location at regular intervals in a GPX file.
Figure 4.7 shows the GPS trace underneath a bridge. While the error caused by the bridge is minimal, it could be improved by applying a Gaussian or Kalman filter, as done in other GPS tracking for rowing boats. An issue this experiment inadvertently highlighted is that the trace does not appear to be accurate, showing the boat moving along the ground at some points. When comparing this to the GPS points taken in later experiments, this does not seem to be a systematic error, but rather a random error in the placement of the GPS in that outing. This suggests that an obstacle may be recorded in the wrong place due to an inaccurate GPS reading.

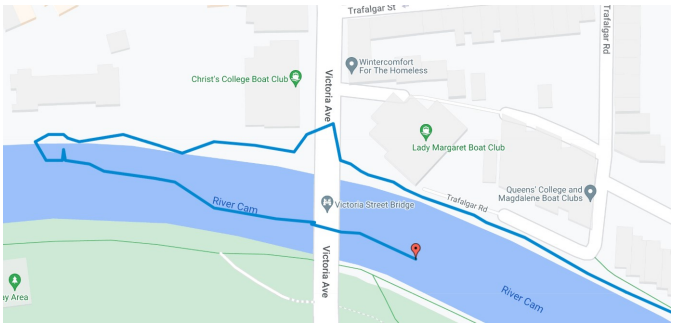


Figure 4.8: The GPS track taken by a node, shown on Google Maps [12]

4.2.1.2 Crash Prevention on Water

To ensure the system worked in the use case environment, I went rowing with a volunteer, each of us with a device on our boat. We ensured that the boats were notified when they were approaching each other. We found this notification to come at around 10 metres between boats. However, when the boats are rowing alongside one another, with no chance of collision, they continually alert the user to the other boat.

The nodes also transfer knowledge of obstacles between them. To test this, one boat went further up the river and registered a 'new obstacle' at a pre-agreed point (shown in Figure 4.8, opposite the Goldie Boat House) by pressing the button on the device. The first boat then returned to the second, which moved towards the potential obstacle. This gave a time delay between the boats approaching the obstacle, showing that messages are propagated through the network to warn other boats of the obstacles. The node warned the other boat of the obstacle.

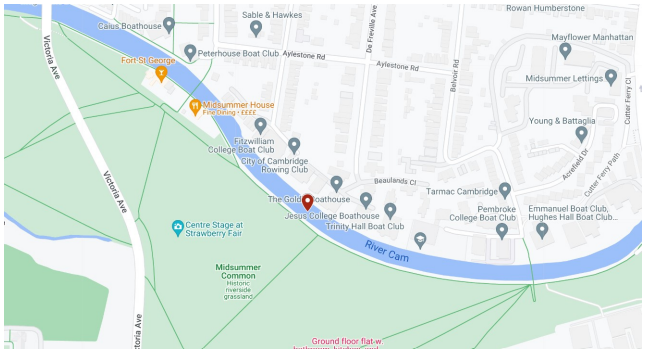


Figure 4.9: The registered obstacle as a red marker on Google Maps [12]

4.2.2 Qualitative Evaluation

4.2.2.1 Survey

For a qualitative evaluation, I surveyed the rower in the other single. While their overall impression was positive, the survey highlighted extra work that needed to be done to the user interface. They pointed out that the buzzer, even if the bug for notifying users about boats they are rowing alongside is fixed, might be a violation of rules about noise during certain times, so it could include another button to turn off the buzzer if needed.

4.2.2.2 Limitations Caused by Implementation

There are some limitations to the system due to its design. The format of the packets has meant that there can be at most 255 nodes in the system, as there are two bytes reserved for the node address and address 0 is reserved for broadcast. The number of obstacles stored in memory is limited by the available memory on the chip, 1.9MB as 1MB is dedicated to the code running the node. Assuming each obstacle takes 10 bytes, 95000 obstacles can be stored on a device. This is sufficient. However, the message buffer is limited to 30. This limit on the number of messages a node can propagate is placed there to allow the unique ID for each message to fit into a single 64 byte packet, but means that the maximum number of obstacles a node can pass to a neighbour is 30.

Conclusion

5.1 Achievements

In the introduction I highlighted the issues around crashes in rowing boats and the need for a system to help prevent such crashes. This dissertation has presented such a system, propagating knowledge about obstacles through a network of rowing boats using the Epidemic routing protocol. Figure 5.1 shows a node from the final product, a box attached to a single scull.

Figure 5.1: The final product in use, attached to the canvas of a rowing boat

Three success criteria were laid out as part of the preparation:
The Epidemic routing protocol is implemented within the network
An evaluation of the network has been carried out
An application layer has been implemented to allow utility of the network

This project has met and exceeded the success criteria. Amongst the extension criteria achieved, medium access control was implemented, high and low priority messaging was defined within Epidemic routing, and a review of the application layer was conducted.

As part of this project, I have contributed to open source libraries and furthered my knowledge of microcontrollers, GPS and networking.

5.2 Reflections

The preparation chapter lays out the choices I made with respect to the software development methodology, tools and programming language. These choices were all found to be effective in designing and building the system. It was particularly helpful to have a plan that split the project into stages with defined deliverables at each stage.

While the research phase is designed to cover broad parts of the area in order to find the best solution to a problem, I spent too much time researching various potential routing protocols. With the benefit of hindsight, it would have been beneficial to settle on the Epidemic routing protocol earlier in the process.

There were both upsides and downsides to implementing the project in hardware. It taught me huge amounts about serial communication protocols and other low level implementation details. It also allowed for a more holistic approach to the evaluation, using the system on rowing boats. However, it is undeniable that the hardware made some aspects of the project more difficult, particularly when it came to debugging a problem. For instance, I soldered an antenna to a radio module in such a way that caused errors in the received packets, a problem I did not diagnose until I swapped the module for another one.
An aspect that went well was the choice of hardware. The availability of widely tested libraries for that hardware that I could modify gave me a level of flexibility in the low level aspects, but allowed me to delegate some of the boilerplate coding.

5.2.1 Other Approaches

There have been other approaches to this problem. ROWCUS used radar to attempt to solve this problem. This technological approach has the benefit that no person has to mark the location of an obstacle to be warned of it but is likely to miss some obstacles, particularly those submerged underneath the water, arguably the most dangerous obstacles as the users cannot see them. ROWCUS has "decided not to pursue" their system any further.

5.3 Future Work

There is future work that could be conducted in relation to both the networking and application components of the project.

There are many parameters in the `config.py` file that impact routing. These could be tweaked to optimise communication over water. The experiments to determine the values for these parameters were performed indoors rather than on water (the use case) as it was not pragmatic to perform these experiments on the water. The MANET could be further improved by integrating ideas from routing protocols other than Epidemic. For instance, as in GPSR, the GPS location could be used, either by only contacting nearby nodes to increase the probability that an anti-entropy session is successful. The location could also be used to prioritise sending messages about new obstacles to nodes that are near these obstacles. A metric for transmission quality, in radio communication the received strength signal indicator (RSSI), to influence whether an anti-entropy session was initiated.

The application component could be further improved by taking the direction of the rowing boat into account when notifying the user of an obstacle. The GPS could be further improved by applying a Gaussian or Kalman filter to reduce the noise in the location, particularly under bridges.

Additional improvements could be made from the suggestions of users, such as a switch to mute the buzzer.

5.4 Other applications

Such a system could be utilised for other applications. The most obvious is collision avoidance for other water sports, such as canoeing and kayaking. There are more avant-garde applications for this system. Deciding if boats have collided within competitions, removing human bias, is one of these potential applications. The low latency may be a concern in other uses of the system. The latency needs to be comparable to the speeds achieved by the nodes in the MANET to allow messages to propagate through the system.

Bibliography

[1] J. Jubin and J. Tornow, "The DARPA packet radio network protocols," *Proceedings of the IEEE*, vol. 75, no. 1, pp. 21–32, 1987.

[2] D. Beyer, "Accomplishments of the DARPA SURAN program," in *IEEE Conference on Military Communications*. IEEE, 1990, pp. 855–862.

[3] P. Nicholas and K. Hoffman, "Computational challenges of dynamic channel assignment for military MANET," in *MILCOM 2015-2015 IEEE Military Communications Conference*. IEEE, 2015, pp. 1150–1157.

[4] R. Davidescu and N. Eugen, "Ad hoc networks for the autonomous car," in *IOP Conference Series: Materials Science and Engineering*, vol. 252, no. 1. IOP Publishing, 2017, p. 012094.

[5] Y. Jin, X. Tan, W. Feng, J. Lv, A. Tuerxun, and K. Wang, "MANET for disaster relief based on NDN," in *2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN)*. IEEE, 2018, pp. 147–153.

[6] Raspberry Pi, *RP2040 Datasheet A microcontroller by Raspberry Pi*, 2022.

[7] A. Moore, "Computer networking slide set for IB computer science," 2022.

[8] M. Taylor, "Equipment on the Cambridge women's boat," 2013.

[9] A. Vahdat and D. Becker, "Epidemic routing for partially connected ad hoc networks," 2000.

[10] ROWCUS. (2022) ROWCUS | Rearview radar for collision avoidance. [Online]. Available: https://www.rowcus.com/

[11] Google. (2023) Google earth. [Online]. Available: https://earth.google.com

[12] ——. (2023) Google maps. [Online]. Available: https://www.google.co.uk/maps

[13] S. Corson and J. Macker, "Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations," Tech. Rep., 1999.

[14] J. Crowcroft, "Pat II lecture notes of principles of communications," 2022.

[15] K. Kiran, N. Kaushik, S. Sharath, P. D. Shenoy, K. Venugopal, and V. T. Prabhu, "Experimental evaluation of BATMAN and BATMAN-Adv routing protocols in a mobile testbed," in *TENCON 2018-2018 IEEE Region 10 Conference*. IEEE, 2018, pp. 1538–1543.

[16] B. Karp and H.-T. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking*, 2000, pp. 243–254.

[17] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, "Macaw: A media access protocol for wireless lan's," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 4, pp. 212–225, 1994.

[18] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. (2001) Agile Manifesto. [Online]. Available: https://agilemanifesto.org

[19] ——. (2001) Agile Principles. [Online]. Available: https://agilemanifesto.org/principles.html

[20] (2023) Raspberry Pi Pico | the Pi Hut. [Online]. Available: https://thepihut.com/products/raspberry-pi-pico

[21] Raspberry Pi. (2022) Raspberry pi pico pinout. [Online]. Available: https://datasheets.raspberrypi.com/pico/Pico-R3-A4-Pinout.pdf

[22] Hoperf Electronic, *RFM69HCW ISM transceiver module V1.1*.

[23] GitHub. (2022) Adafruit Industries. [Online]. Available: https://github.com/adafruit

[24] CD Technology, "CD-PA1616S GPS patch antenna module data sheet V.04."

[25] C. C. Robusto, "The cosine-haversine formula," *The American Mathematical Monthly*, vol. 64, no. 1, pp. 38–40, 1957.

[26] A. Mohamed and K. Schwarz, "Adaptive kalman filtering for INS/GPS," *Journal of geodesy*, vol. 73, pp. 193–203, 1999.

[27] AdaFruit. (2022) Adafruit blinka library. [Online]. Available: https://docs.circuitpython.org/projects/blinka/en/latest/index.html

Appendices

A – Guide to Building a Node

A System to Help Prevent Crashes in Rowing Boats
How to Construct Your Own System

This is a 'how to' guide to building the nodes and downloading the software needed to build your very own MANET. Each node *should* beep (or flash an LED, depending on what you choose) when the user is approaching a known obstacle, much like parking sensors on a car! A disclaimer - I do not guarantee this will work and take no responsibility for any collisions.

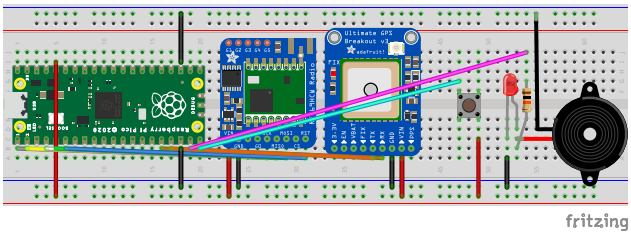
Components Needed
This is a list of what I used in my project, and therefore what the software I wrote has been designed for. Any microcontroller, GPS and radio can be substituted.

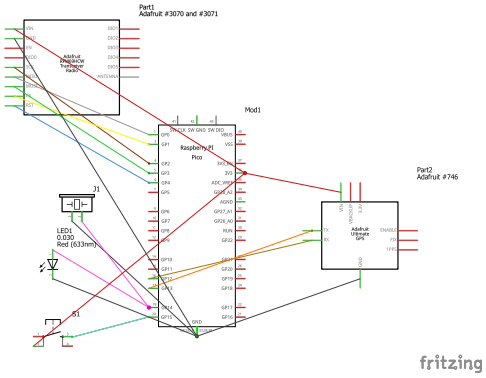
- Raspberry Pi Pico
- Adafruit RFM69HCW Transceiver Radio Breakout
- Adafruit Ultimate GPS Breakout
- Non-latching Button (mine has an LED ring around it used to tell if the node is on)
- Buzzer and resistor (resistor moderates the pitch of the buzzer, I use 10k)
- Waterproof box
- Power pack (to power the device)
- 3M Dual Lock or other method of attaching the final node to the boat
- Breadboard (optional but recommended)
- Wires

Tools Needed

- Soldering iron
- Drill and bit of the same diameter as your button (I use 12mm)

Wiring





Software

After running CircuitPython on your device (AdaFruit has a good little tutorial here: https://learn.adafruit.com/getting-started-with-raspberry-pi-pico-circuitpython/circuitpython) Go to the OnDevice folder in this repository and upload the boot.py file and deviceCode.py to the Pico, then place the contents of the lib folder into the Pico's lib folder. When you're done, the Pico's filesystem should read:

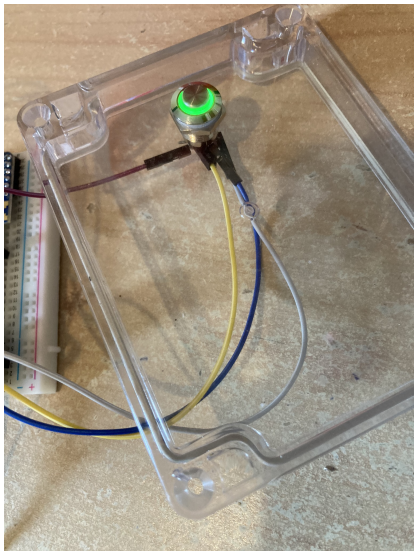
```
boot.py
deviceCode.py

lib
    adafruit_gps.py
    rfm69.py
    config.py
```

You'll need to go into `lib > config.py` and manually set the `ADDRESS` constant to a unique value for each node. Remember each address must be smaller than 0xFE, 254!

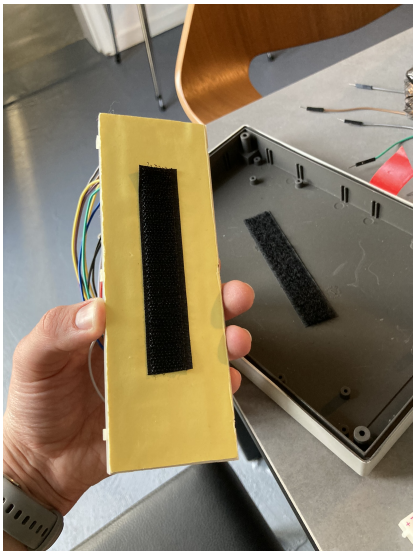
If you want to change any other parameters, they are all held in the config.py file, with comments above. Simply plug the device into your computer and edit.

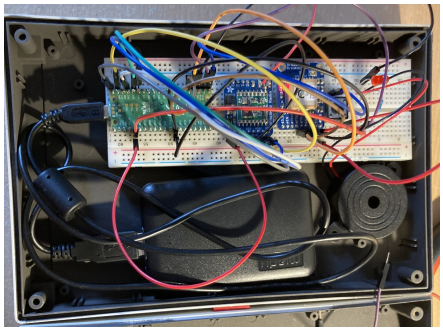
SetUp

As the button and any LEDs need to be on the outside of the box, you'll have to drill a hole into your waterproof box.



You'll now need to place the device into its waterproof box with a powerpack. I held all the components in place with dual lock to make sure they didn't move when inside the box.





Avoid accidents!
The device should work immediately on powerup. Row safely!

B – Evaluation Plan

Part II Project - Plan for Evaluation

March 2023

Overview

The evaluation for the mobile ad-hoc network (MANET) will be split into two parts – first the evaluation for the pure MANET and a whole system evaluation. The evaluation of the MANET will examine the implementation of Epidemic, the delay tolerant networking protocol will be checked for correctness and performance. The system evaluation will look at the MANET in the context of the use case, on the water.

All data will be logged on the node in a CSV file with columns
`Node Address, Time, Time Since Node Startup, Event Type, Event Information`
where the Event Information varies with the event being logged and may contain the GPS location and message keys. This data will then be analysed on my device using Python code.

Evaluating the MANET

These tests be conducted in a field, as this will give an outdoor environment similar to the use case, but I will be able to better control the conditions the node is in. They will use four nodes, as this is the maximum number of nodes we can accurately calculate time for. Two nodes will use GPS to find the current time and two will use the serial connection to laptops to calculate the time. These tests will be conducted first on a small scale, with short tests using a small number of nodes, to ensure the tests can be run. After this has been confirmed, the tests will be run for a longer time with the maximum number of nodes.

The tests can then be compared to the evaluation in the initial epidemic paper [3], which simulated the nodes with 50 mobile nodes in a 1500 x 300 m space using the Monarch extensions to the ns-2 packet-level simulator rather than hardware as I am doing. Evaluation metrics examined included message delivery latency, delivery rate, the average and maximum number of hops a message took to get to a node.

The first test will be the percentage of packets delivered in the four node network. This will be time limited (i.e. if a message is not received in x minutes, it is considered undelivered). Nodes will randomly generate a new message every second, for a total of 25 messages, mirroring the structure of testing used in Vahdat and Becker's paper [3]. The nodes will be in a box of area $20m^2$ with the range of the radios reduced to approximately $5m$ to simulate the environment in which the system will be used. The nodes will move constantly.

Next, the transfer delay will be measured. This will be the average time taken for each message to be delivered, and will use the same setup as percentage of packets delivered testing.

Finally, the time taken to propagate messages after partition will be measured. This will include one-sided, asymmetric partitions, where only one set of nodes has messages the other set has not seen. It will also include symmetric partitions, where both sets of nodes have messages the other set has not seen. The number of unseen messages will be increased and each test will be run five times.

Figure 1: The setup and axes for delivery rate and transfer delay

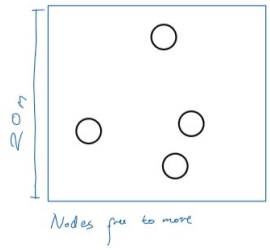
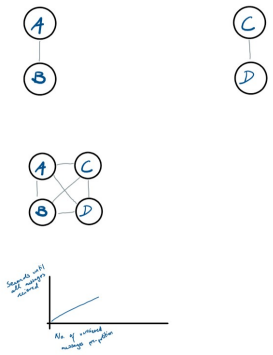


Figure 2: The setup and axes for partition testing



Evaluating the System

This will be performed on the water, the environment the MANET will be used in, after a 'dry run' on land to ensure there are no obvious flaws with the system. To examine the use of the system and how long it takes messages to arrive at other nodes, a well known obstacle has been selected. This is the red buoy at 51.48236626931181, -0.22641424521527762, shown below [1]. Using the time given from the GPS chips, we can then map the locations and time since the obstacle was added that other nodes receive messages about the buoy. This experiment will be run multiple times to gather a sufficient volume of data about the propagation of new obstacles.

Additionally, this will allow me to look at the behaviour of users when adding a new obstacle then tweak the MANET to fit. While there is a ground truth about the location of an obstacle, it is likely that most users will not be directly above the obstacle when they log it.

Figure 3: The location of the 'red buoy' obstacle [1]

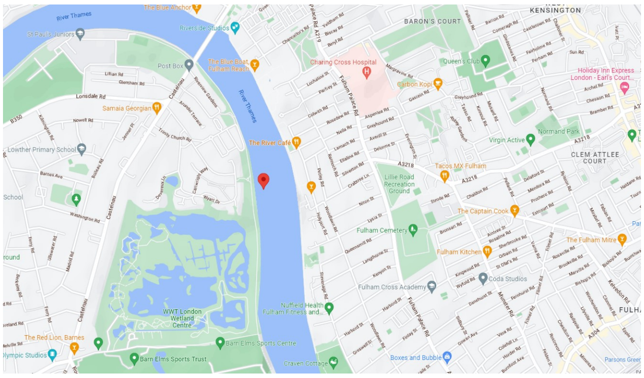


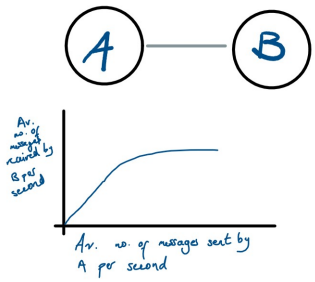
Figure 4: An image of the 'red buoy' obstacle [2]

Extensions

If there is sufficient time, further evaluation can be performed on the MANET. This will be structured in a similar way to the tests in the 'Evaluating the MANET'. Bandwidth will be tested in a fully connected network, where the number of messages per second transferred between two nodes may be found by generating random packets at set intervals at one node (node A shown below), and seeing how many are passed to another node (node B below). This test could then be performed with both nodes generating and receiving messages, to examine bandwidth on a two way connection.

Further extensions to the system evaluation will include a questionnaire for those who use the

Figure 5: The setup and expected graph for testing bandwidth



device, covering a range of users, including both coxes in coxed boats and rowers in coxless boats.

References

[1] Google Maps. 51.4823, -0.2264. [Online] Available at: https://www.google.co.uk/maps/place/51%C2%B028'56.5%22N+0%C2%B013'35.1%22W/@51.4820341,-0.2295313,15.5z/data=!4m4!3m3!8m2!3d51.4823663!4d-0.2264142!5m1!1e4 [Accessed March 2023]

[2] Riddell-Webster, A. Red buoy. Taken March 2023.

[3] Vahdat, A, Becker, D. Epidemic Routing for Partially-Connected Ad Hoc Networks. Published 2000. Duke University.

C – Progress Report

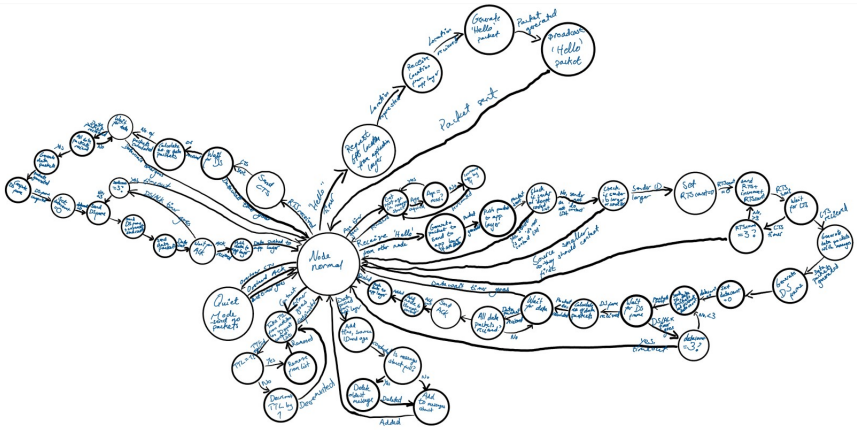
Part II Project - Progress Report

May 5, 2023

Name: Alex Riddell-Webster
College: Murray Edwards
Email Address: ahr38@cam.ac.uk
Director of Studies: Luana Bulat
Supervisor: Matthew Ireland
UTO: Professor Jon Crowcroft
Overseers: Ferenc Huszar and Andreas Vlachos
Title: A MANET to Facilitate Collision Avoidance in Rowing Boats

My project, a mobile ad-hoc network (MANET) to facilitate collision avoidance in rowing boats, attempts to reduce the frequency of rowing boat collisions. The technical core of the project is the Epidemic routing protocol [1], modified to include medium access control based on the Multiple Access with Collision Avoidance for Wireless (MACAW) protocol [2]. Both medium access control and Epidemic are implemented in the by the same state machine, to simplify the implementation and prevent work being repeated – a waste of the limited resources on the Raspberry Pi Pico [3]. The initial state machine is shown in Figure 1, although it has been changed during implementation. Most notably, data send (DS) packets have been removed and the information being put into the data packets to reduce the number of packets sent.

Figure 1: Network state machine

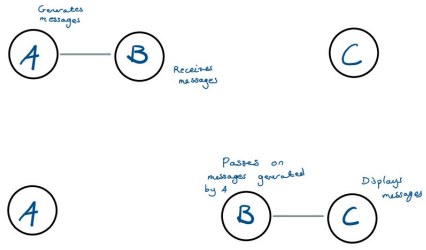


Construction of the MANET is going well. It has been constructed in hardware, working with the Raspberry Pi Pico, an ARM-based microcontroller without an operating system [3] and RFM69 radio. I have finished the networking machine so am currently tweaking and evaluating the network state machine while finishing the application machine. As the network has a physical implementation, I intend to test the network on rowing boats, the environment it would be used.

To ensure the network was delay tolerant, I ran a test with three nodes, A, B and C. At the start, nodes A and B were in range of each other and node C was out of range. I set node A up to generate random messages every 40 seconds, with a time to live (TTL) greater than 2 so the messages would survive for two 'hops' across the network. I then moved node B out of range of

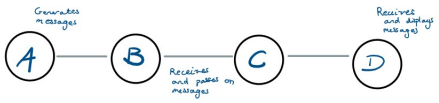
node A and into range of node C, which then displayed any messages it received so I could check that they were the same as those generated by A, with a reduced TTL. Figure 2 shows the setup.

Figure 2: Using three nodes to check the network is delay tolerant



Another test involved four connected nodes, A, B, C, and D, where the transmit power of each node was significantly reduced, so each node had at most two connections. Node A generated messages, and I checked to see if D received them. I will use a similar set up in the future to test the percentage delivery and latency of packets. The setup is shown in Figure 3.

Figure 3: Using four nodes to check the network can transfer packets over several links



The most significant obstacles have been in radio communication. The FIFO buffer on the RFM69 was occasionally being overwritten as the controlling library was not clearing the FIFO. I modified the library to clear the buffer and allow the sending of fixed length packets. Changing to fixed length packets (64 bytes, the maximum length of the FIFO) allowed for more reliable communication. Additionally, CircuitPython (the language in which AdaFruit's libraries are written) does not support interrupts. To work around this, I poll to see if a condition is met when a corresponding timer elapses.

Given the work completed so far, I am two weeks behind the timetable laid out in October. As I am working on the application machine and evaluation of the network in parallel, the project will likely be back on timetable by mid-February.

The remaining work is first to finish the application machine and evaluate the MANET. Evaluation metrics will include the percentage of received packets, transfer delay and variance, and the time taken to propagate messages in a previously segmented network. Finally, I will pull the application and network machines together, running them on the two cores in the Pico, with concurrency control over key data structures and the GPS chip. A concern here is the Adafruit Blinka libraries allowing interoperability between CircuitPython and MicroPython [4], given the errors and incompleteness found in other libraries.

References

[1] Epidemic Routing for Partially-Connected Ad Hoc Networks. Vahdat, A, Becker, D. Duke University. 2000.

[2] MACAW: A Media Access Protocol for Wireless LAN's. Bharghavan, V, Demers, A, Shenker, S, Zhang, L. ACM SIGCOMM Conference. 1994.

[3] RP2040 Datasheet A microcontroller by Raspberry Pi. Raspberry Pi Ltd. 2022.

[4] GitHub - adafuit/Adafruit_Blinka: Add CircuitPython hardware API and libraries to MicroPython & CPython devices. https://github.com/adafruit/Adafruit_Blinka. Adafruit Industries. GitHub. Accessed January 2023.

D – Project Proposal

Part II Project – Project Proposal

October 14, 2022

Preliminary Information

Name: Alex Riddell-Webster
College: Murray Edwards
Email Address: ahr38@cam.ac.uk
Director of Studies: Luana Bulat
Supervisor: Matthew Ireland
UTO: Professor Jon Crowcroft
Overseers: Ferenc Huszar and Andreas Vlachos
Title: A MANET to Facilitate Collision Avoidance in Rowing Boats

Introduction

My project will implement routing in a Mobile Ad-Hoc Network (MANET).

Routing protocols find a path from a source to one or more destinations destination within the network. Different routing protocols optimise different parameters, and are better suited for different network topologies and applications [1].

A MANET is characterised by wireless nodes, a frequently changing network topology and no reliance on pre-existing infrastructure. They are decentralised and therefore have no single point of failure [2]. MANETs have a large range of uses, from the military [3] to facilitate communication, to autonomous vehicles [4] or disaster relief scenarios [5] to gather and move data across locations where previously existing infrastructure has been destroyed.

My project wishes to use a MANET to share a set of locations throughout a set of rowing boats, in order to facilitate collision avoidance. Collisions in rowing can cause damage to both rowers and their equipment. This project's motivation is to avoid rowing boats colliding with each other and with other obstacles. A radar and AI based obstacle detection system exists [6]. However, to the best of my knowledge, collision avoidance has not been attempted by networking boats together. My project will represent each boat as a node in a network. Each node will store a set of locations the user should be warned about; passing location data throughout the network will be the technical core of the project.

My project will be implemented in hardware, in the real world. In general, networking protocols can be implemented in simulation or in hardware. Depending on the nature of the simulation environment, it might not be possible to use exactly the same code in the simulator as in the real hardware. Due to the time constraints on a Part II Project, I intend to implement my project only in hardware.

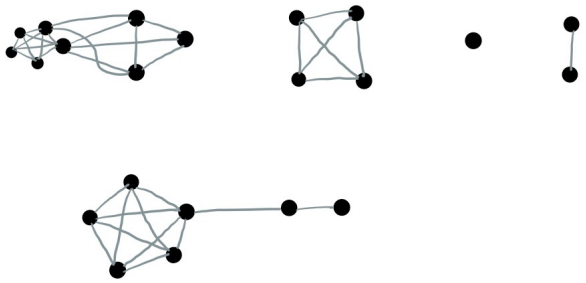
As stated in the cover sheet, Human Participants will be used to help test and evaluate the project. This will comprise a few volunteers to row boats, allowing the network to be run on the water. These volunteers will all be members of Cambridge University Boat Club, able to safely row a boat and navigate the river where the network is being tested.

Structure

The first part of the project will be dedicated to research, looking at the Epidemic routing protocol. Epidemic was chosen as it is a delay tolerant routing protocol, and best fits the likely topology of networks generated by rowing boats. There is a high chance of partitions in these networks. However, the nodes in the networks will be highly mobile, meaning that data can still be transferred through the network through exploiting this mobility. A potential topology for the network, generated from looking at satellite images of a 5.5km stretch of the river Thames [7], is shown in Figures 1 and 2, both on a map and as an abstracted topology.

Figure 1: Rowing boats, potential obstacles and assumed connections marked on a Google Maps map of the river Thames

Figure 2: The network of rowing boats in an abstracted form



The first research phase will also decide on any changes that need to be made to the routing protocol to make it better suit the application. Finally, the first research phase will refine the evaluation metrics needed for the project.

The second research phase will look at microcontrollers, particularly the Raspberry Pi Pico. It will also consider multi-threading on the Pico and in CircuitPython and how this can be exploited to most effectively implement my Part II project.

After the research phases, I will implement point to point communication between two neighbouring (in radio connection range) nodes in hardware.

The next part of the project will implement broadcast and controlled flooding routing between at

least three nodes. A packet format will be defined as part of this, as flooding will form the start of the Epidemic routing protocol.

After a flooding protocol has been implemented, I will build on it to implement the Epidemic routing protocol. The code will be written in a two week block, then tested on hardware in a third week.

The application layer will then be implemented. This will ensure that the information passed through the network can be used. The application layer should warn the user when they are approaching an obstacle, and allow the user to add obstacles, which are then passed to the routing layer to be propagated through the network. My project aims to keep the routing and application layers separate for ease of construction, testing and evaluation.

The hardware will then be tested, tweaked and evaluated. Correctness will be evaluated first on land, likely in a field, where analysis of the network is easier and larger numbers of metrics can be examined than in the use environment. Performance will be evaluated in the use environment - on rowing boats on the water. I intend to evaluate the routing and application layers separately. Evaluation of the network will likely consider connected and partitioned instances of the network separately. Evaluation is likely to look at the time taken for the network state to be flooded through the nodes, and routing tables then updated, as well as a packet loss ratio [8]. Further evaluation would conduct some case studies, tracking a packet through the network and ensuring no unnecessary latency is added. The power consumption of each node could also be measured, giving a proxy measure for the traffic passing through each node. Due to time constraints, I consider these further evaluations to be extensions.

Success criteria

There are three success criteria I will hold for my project:

1. The Epidemic routing protocol is implemented on the network
2. An application layer to demonstrate the utility of the network has been implemented
3. An evaluation of the performance of the network has been carried out

Extensions

The success of my project will be defined by completion of the core criteria listed above. If there is time, I have set further challenges:

1. Case studies on the path and timing of individual packets are performed
2. The network is further evaluated by examining the power consumption of individual nodes as a proxy metric for traffic passing through a node
3. The User Interface of the device is evaluated
4. The application layer is further enhanced, using heuristics and extra data such as angle of attack from GPS and combining sensor data
5. The routing layer is further enhanced by passing additional data, such as location awareness, to the routing protocol

Plan of Work

Start of Block	End of Block	Block Length	Notes	Work to be Done	Milestones
14/10/2022	21/10/2022	7		Research - how to implement Epidemic protocol, evaluation methods used for ad-hoc networks	Develop a greater understanding of the Epidemic protocol and a plan for implementing it, create an evaluation plan
21/10/2022	28/10/2022	7		Learning how to use the microcontroller and boards	Ensure all the necessary hardware is available, develop a greater understanding of the Raspberry Pi Pico and CircuitPython
28/10/2022	04/11/2022	7		Start to work with the hardware - implement point to point communication between two nodes	Two nodes can send point to point messages
04/11/2022	18/11/2022	14	07/11 - Robotics Assignment 1	Implement controlled flood routing	Messages are flooded between at least three nodes
18/11/2022	02/12/2022	14	18/11 - 4s head; 28/11 - Robotics Assignment 2	Implement Epidemic routing protocol	Routing state information is shared between at least two nodes, Epidemic is implemented
02/12/2022	10/12/2022	8		Test, tweak and debug Epidemic implementation on hardware	Epidemic is implemented on hardware
10/12/2022	26/12/2022	16	14/11 - Trial 8s; Christmas	Time off	-
26/12/2022	02/01/2023	7	01/01 -> 11/01 - Camp	Implementing application layer - read location data from GPS and warn user when approaching known obstacle	The device warns the user when they are approaching a known obstacle
02/01/2023	16/01/2023	14	01/01 -> 11/01 - Camp	Implementing application layer - transfer data between the application and routing layers	The application layer is implemented on hardware
16/01/2023	23/01/2023	7		Tweaking the hardware, testing point to point links on land	The hardware runs on land, finish proof of concept
23/01/2023	30/01/2023	7		Water testing and tweaking	The hardware is implemented and run in the application environment (water)
30/01/2023	03/02/2023	4	03/02 - Cybercrime 1	Write progress report and presentation	Progress report and presentation
03/02/2023			**Progress report and presentation**		
03/02/2023	14/02/2023	11		Evaluation and tweaking on land	The hardware is evaluated for correctness on land
14/02/2023	21/02/2023	7	17/02 - Cybercrime 2	Evaluation and tweaking on water	The hardware is evaluated for performance on water
21/02/2023	07/03/2023	14	03/03 - Cybercrime 3	Dissertation - plan and bullet point what will be said	Dissertation bullet point form (first draft)
07/03/2023	21/03/2023	14	17/03 - Cybercrime 4	Dissertation - write out preparation and implementation	Dissertation has implementation and preparation written out
21/03/2023	04/04/2023	14	26/03 - Boat Race	Time off	-
04/04/2023	18/04/2023	14		Dissertation - write introduction, conclusion, evaluation	Dissertation fully written, sent to supervisor to proofread (second draft)
18/04/2023	02/05/2023	14		Dissertation - Take on criticism, add references and appendices	Dissertation - final draft
02/05/2023	12/05/2023	10		Contingency	-
12/05/2023			**Final deadline**		

Starting Point

I have a little experience with networking and routing protocols. My experience is limited to the Part IB networking module, although it is being expanded by the Part II Principles of Communications module and my research. I will need to add to my knowledge of networking and routing protocols.

I have previous experience using Raspberry Pi single-board computers with AdaFruit boards. I have no previous experience with microcontrollers. I will need to improve my knowledge of microcontrollers to complete this project, something I have set aside time for in my Plan of Work.

Resource Declaration

I plan to use my laptop to implement, evaluate and write up the project. It has a comprehensive system of backups through OneDrive and disk images. A backup of the project will exist with Git version control, hosted on GitHub. My own hardware, including Raspberry Pi Picos, breadboards and AdaFruit radio and GPS modules will be used to develop and implement the project.

Libraries to interface with the AdaFruit boards are written by AdaFruit in Circuit Python, and in my experience tend to be robust, although they occasionally contain bugs. If necessary, I can fork the code and implement bug fixes.

My project will partially rely on the correctness of routing protocols, work that others have already published. [9]

As the project has a real-world implementation, I have permission from Cambridge University Boat Club to test devices on their boats.

References

[1] 2022 Part II Lecture Notes on Principles of Communications. Crowcroft, J. 2022.

[2] Mobile Ad Hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations. Corson, S. and Macker, J. Network Working Group. 1999.

[3] MILCOM'09: Proceedings of the 28th IEEE conference on Military Communications. Richardson, K, Jimenex, C, Stephens D. 2009.

[4] AD HOC Networks for the Autonomous Car. Davidescu, R, Negrus, E. IOP Conference Series Materials Science and Engineering. 2017.

[5] MANET for Disaster Relief based on NDN. Jin, Y, et all. IEEE. 2018.

[6] DEVICE | rowcus. https://www.rowcus.com/device. ROWCUS. Retrieved 07/10/2022.

[7] Hammersmith Bridge - Google Maps. https://www.google.com/maps/place/Hammersmith+Bridge,+London/@51.4883478,-0.2302753,17z/data=!3m1!4b1!4m5!3m4!1s0x48760fb3f5c78f85:0x932267a237304c18!8m2!3d51.4883478!4d-0.2302753. Google. Retrieved 10/10/2022.

[8] SERIES Y: GLOBAL INFORMATION INFRASTRUCTURE, INTERNET PROTOCOL ASPECTS AND NEXT-GENERATION NETWORKS Internet protocol aspects – Quality of service and network performance. International Telecommunication Union. 2011.

[9] Epidemic Routing for Partially-Connected Ad Hoc Networks. Vahdat, A, Becker, D. Duke University. 2000.