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
Alex Riddell-Webster

Date
April 26, 2023

Acknowledgements

This dissertation owes a huge amount to Matthew Ireland, for supervising me. My UTO, Jon Crowcroft was invaluable. Thanks to Cambridge University Boat Club, in particular Patrick Ryan, Mike Taylor and Rosa Millard, for allowing me to put strange boxes on boats, advising me on communication over water and helping me evaluate the system. 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:	TODO
Code Line Count:	TODO
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.

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. First, the mobile ad hoc network (MANET) allows nodes to communicate a dynamic collection of known obstacles, then the application layer warns users when they are approaching an obstacle and allows the user to add obstacles. The free and open source system will be available online, fully documented, after my graduation. This will include 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 system achieved its goal as it 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]

[1] Simon Ramskill, Twitter post, March 2023, https://twitter.com/lliksmar/status/1637147870682906624?cxt=HHwWgIC8zY_UqLgtAAAA, Accessed March 2023

1.1.1 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]. The 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. Rowing boats frequently move at speeds greater than 15 km/h, making the topology of any network of boats unpredictable and dynamic. Every node in a MANET must forward traffic, making it a router. 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 traffic 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. Media access control protocols control access to the transmission media to prevent collisions between packets. This project implemented media access control to prevent packets colliding and subsequently being corrupted.

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 associated 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 of the boat 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 within 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 implemented in CircuitPython, so it would have been more beneficial to have learnt about this.

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 to 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. This means 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 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 have to be implemented in order to prevent collisions between messages. This was added as 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.

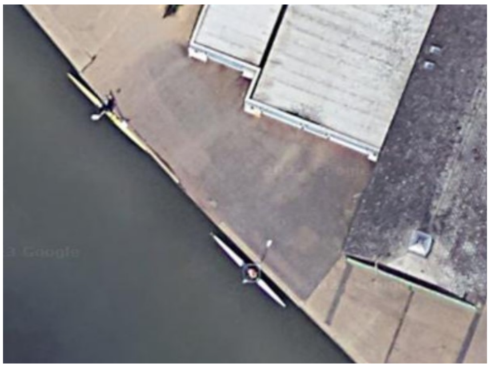


Figure 2.1: Two rowing boats seen on Google Earth [11]

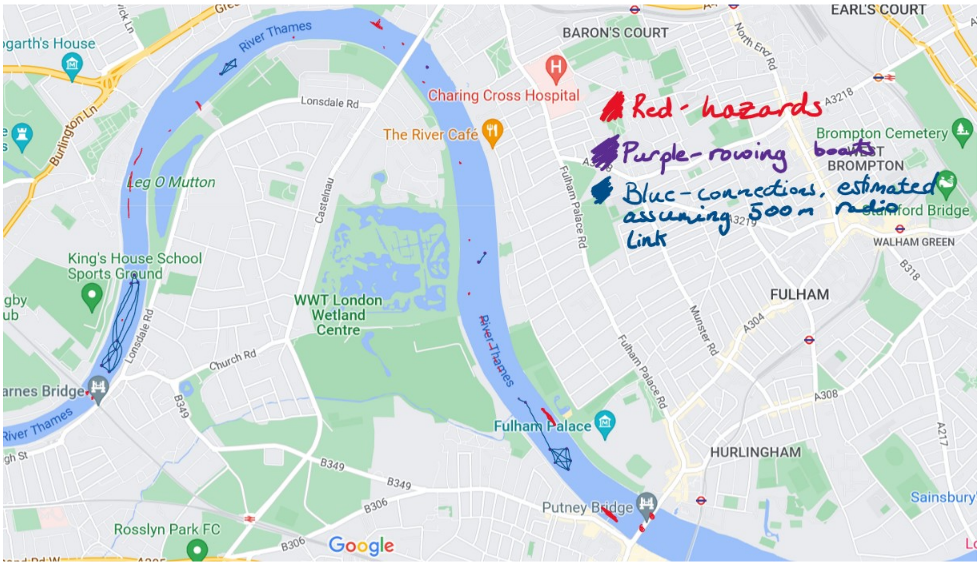


Figure 2.2: Rowing boats, potential obstacles and assumed connections marked on a Google Maps map of the river Thames [12]

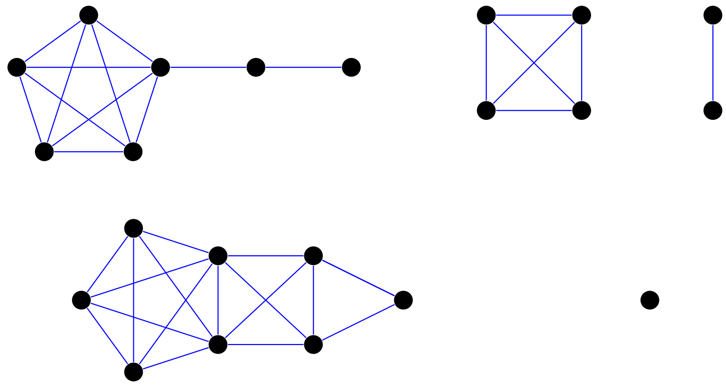


Figure 2.3: The network of rowing boats in an abstracted form, with nodes shown in black and assumed connections in blue

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).

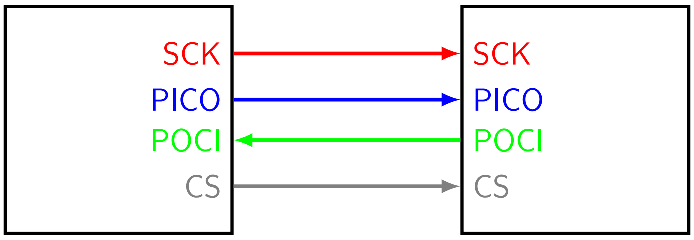


Figure 2.4: SPI logic signals

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.

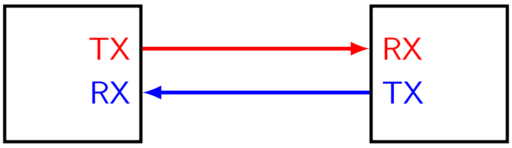


Figure 2.5: UART logic signals

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

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.

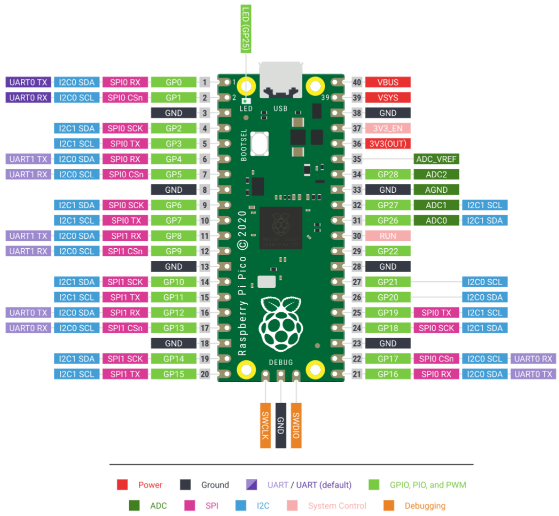


Figure 2.6: The pinout for the Raspberry Pi Pico. Unchanged from Raspberry Pi, licenced under the Creative Commons Attribution-ShareAlike 4.0 licence [21]

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.

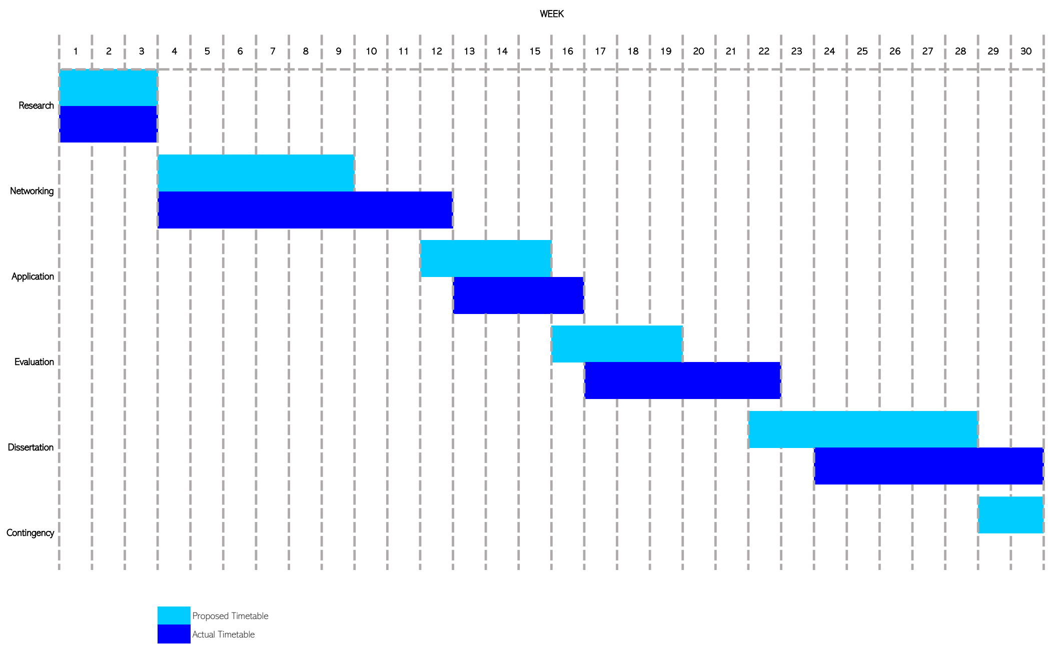


Figure 2.7: A Gantt chart with proposed and actual timelines

Implementation

3.1 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 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 Final System

The final system

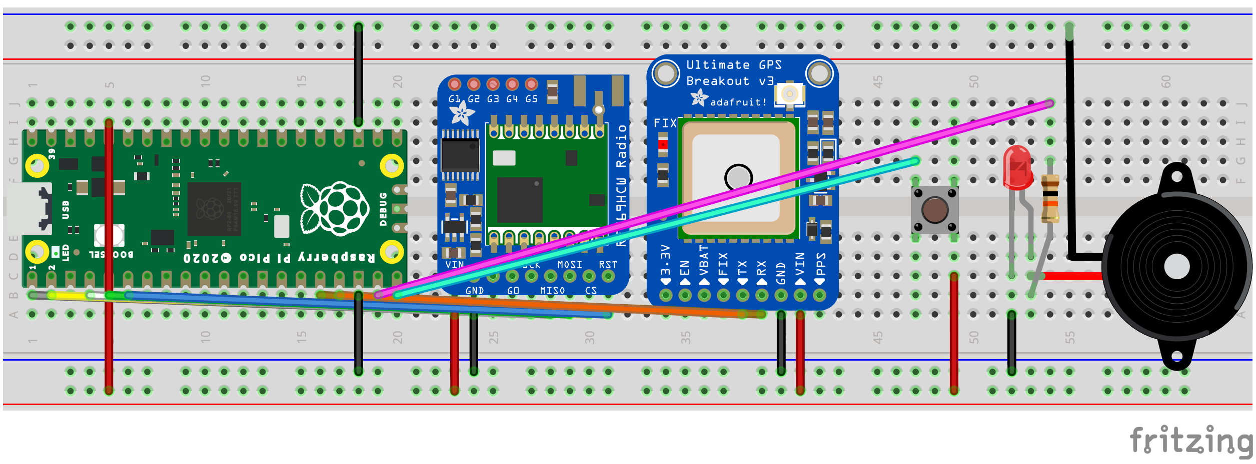


Figure 3.1: An overview of the system

3.3 System Design

Before design, 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 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 designed to pass messages between the two layers and allow both layers access to the GPS.

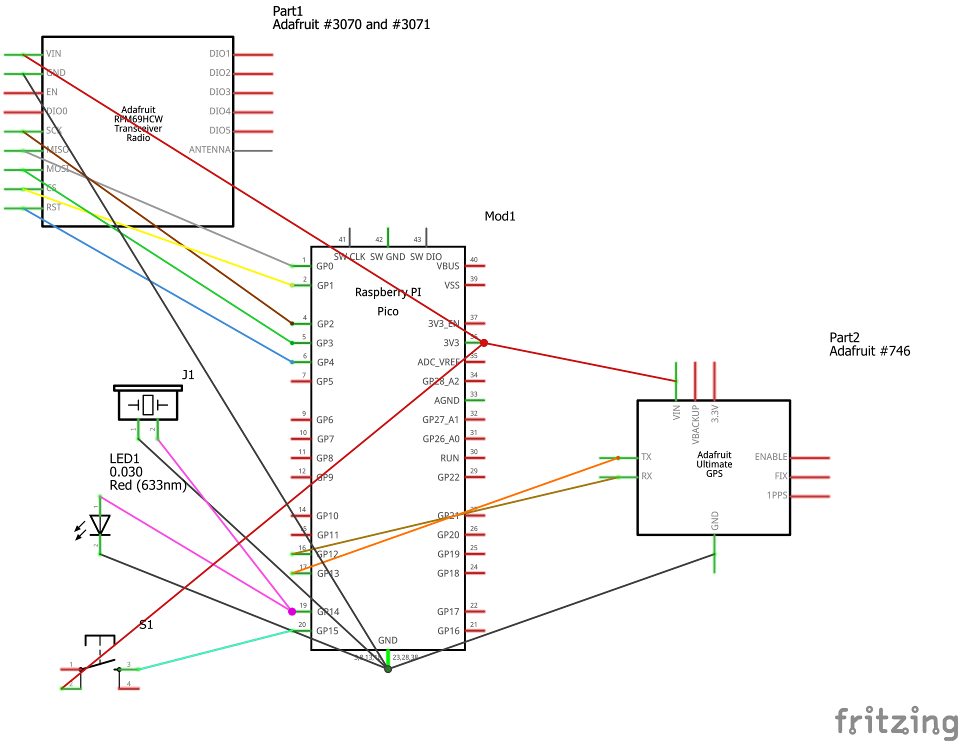


Figure 3.2: The wiring of the system

3.3.1 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 2.4.

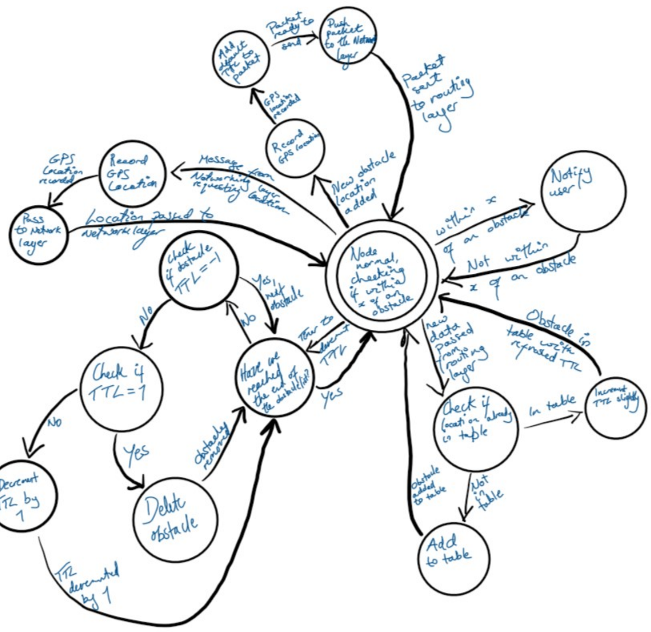


Figure 3.3: (TODO - typeset) The initial state machine for the application thread

3.3.2 Networking

The networking thread contains 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 for my project to prevent work being repeated and use the limited computing resources available most effectively. This means that the implementation of Epidemic will also change. Not only will the node refuse to send any messages after hearing a CTS for a set period of time, or until it hears an ACK, I also integrated the message vector exchange at the start of an anti-entropy session into the CTS and RTS messages. This minimised the number of packets sent over the network, reducing the overall probability of collisions or errors in transmission. This state machine has changed slightly since it was first designed.

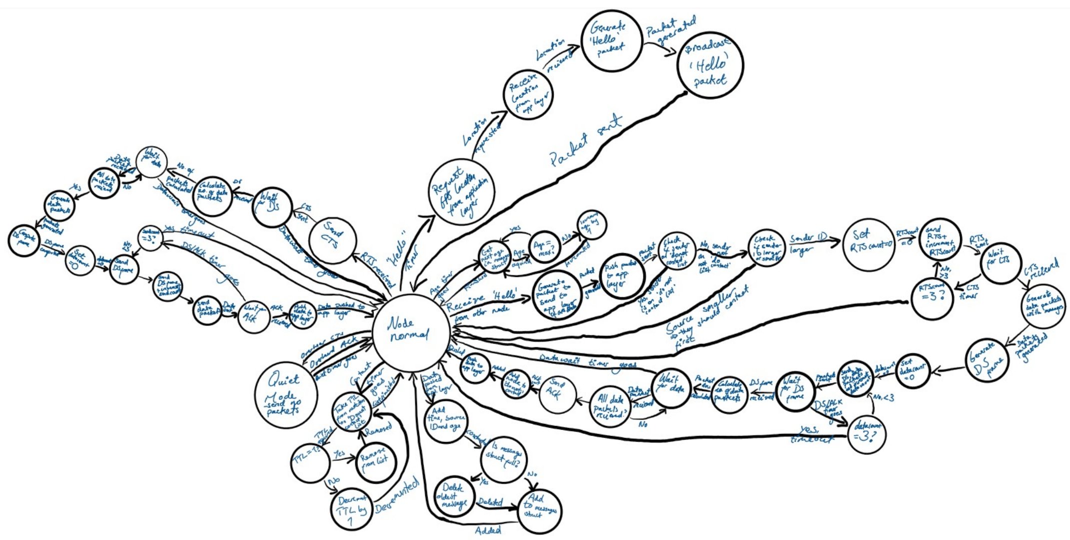


Figure 3.4: (TODO - typeset) The initial state machine for the networking thread

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.

```
|  length  | packetType |  sender  | destination |
|                  payload                       |
```

Figure 3.5: The preamble for all packet types (TODO - typeset, add lengths and different payloads)

3.3.3 Packet Design

After designing the system, I considered the structure of the packets sent by each node. These are influenced by the structures of packets in previous routing protocols I examined. I attempted to keep the structure of the start of the packets Figure 3.3 shows the overall design for these packets. Figure 3.4 shows the breakdown of the packet data by packet type, given that the preamble is the same for all packets. They all start with the length of the packet so the system knows how much data to expect.

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

3.4 Debugging and Logging

Throughout the project, I needed to debug different pieces of hardware and unusual issues. 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 I was debugging 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.

Logging was done through 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
```

3.5 Point to Point

Before starting to implement Epidemic, I ensured that the two nodes could communicate in a point to point way. This was successful; each node could send a packet to another node. I found several interesting variables 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.

I also found that when the rate and length of packets sent increased, more errors crept into 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, 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 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' messages. I used the packet structure previously defined to do this. To test this part of the system, I ensured that 'hello' packets were correctly sent and received. The payload of the 'hello' packets was designed to be the current GPS location of the node, so it could be used by the other nodes as an obstacle to warn users about. When first initiated, this called 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

In Vahdat and Becker's Epidemic paper [9], an anti-entropy session occurs when two nodes come into communication range and exchange messages. Anti-entropy is initiated by the node with the lower ID to prevent collisions. Figure 3.4 shows the first half of anti-entropy. Node A, with a lower ID, initiates anti-entropy by sending a summary vector ($\mathrm{SV_A}$) 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 ($\mathrm{SV_B}$) to produce the messages it has not seen, but the other node holds. It sends this request ($\mathrm{SV_A} + \neg\, \mathrm{SV_B}$) 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 ($\mathrm{SV_B}$) to A. A requests messages it has not seen from B ($\mathrm{SV_B} + \neg\, \mathrm{SV_A}$) 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.

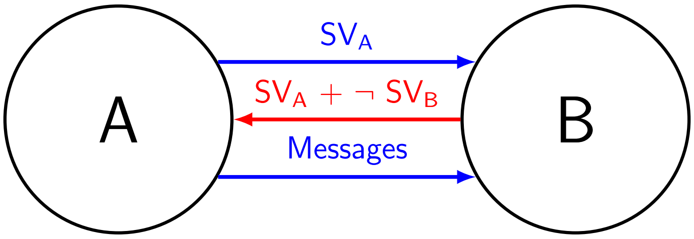


Figure 3.6: The first stage of anti-entropy [9]
If a node receives a 'hello' message from a node that has an address that is larger than it, and the node is not on the list of recently contacted nodes, it attempts to initiate anti-entropy. It does this by sending a request to send (RTS) packet, constructed from the message keys that are held on that node.
When a clear to send (CTS) packet is received from the contacted node, it contains the message

keys that the node that sent it. These two lists of message keys are compared, and any messages the other node contains are added to the list of messages to be sent. These messages are then combined into data frame (DF) packets and sent to the other node.
As an acknowledgement, the other node sends their own DF packets, containing any messages the other node holds but the first does not.
To acknowledge receipt of these messages, the first node sends an acknowledgement (ACK) message. (TODO - timing diagram and node message exchange diagram and code snippet)

3.7 Medium Access Control

Medium access control was implemented using Epidemic. If a node overheard the exchange of an RTS / CTS message exchange between two other nodes, it does not broadcast any hello or CTS messages nor respond to RTS messages. This silence continues 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. (TODO - diagram with state changes)

3.8 Sensitivity Analysis

TODO - the graph with changing of different configurable variables to show how I experimentally determined the values

3.9 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) when started. 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.

3.9.1 Precision

To test the tracking on the GPS, I wrote a short program to generate GPX files. A short 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.

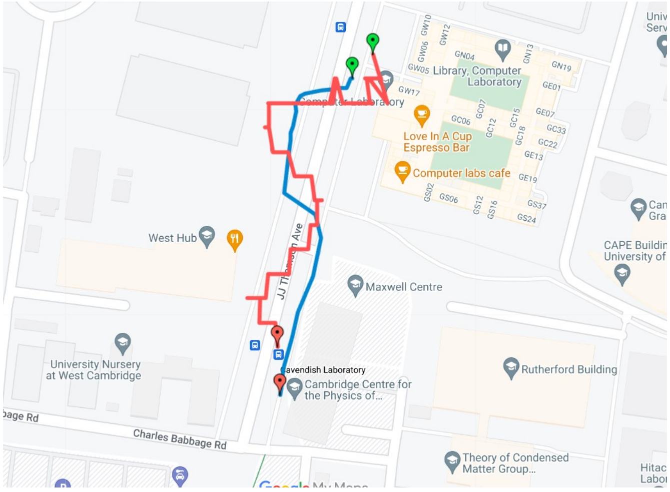


Figure 3.7: The first attempt at tracking a walk with GPS. The red line is the GPX track generated by the board, the blue line is the GPX track generated by a smartwatch [12]

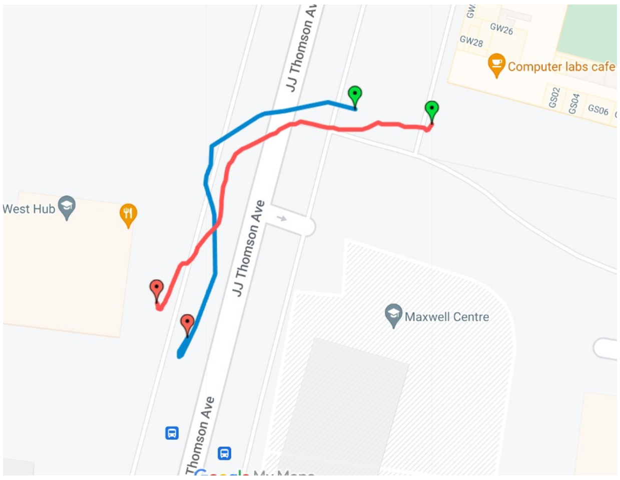


Figure 3.8: The final attempt at tracking a walk with GPS. The red line is the GPX track generated by the board, the blue line is the GPX track generated by a smartwatch [12]

3.9.2 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. 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% [22]. As the distances being measured are relatively small, this was not a problem. Should the system be moved to other planets with a GPS satellite system, this radius will have to be changed. Luckily, this is unlikely due to the absence of interstellar rowing.
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.

3.9.3 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 the noise at the receiver was due to the clock phase noise and interpolation of ephemeris data [23]. Implementing a Gaussian or Kalman filter to remove some of the noise would be possible and is further work.

3.10 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 [24], 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.

3.11 User Interface

3.11.1 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 10k 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.

3.11.2 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. 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. (TODO - writing diagram)

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. (TODO - code snippet?)

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

3.11.3 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.

3.12 Repository Overview

(TODO) Typeset more nicely

```
(doc) LICENCE
(doc) README.md

(file) Development
    (file) Application
    (file) MACEpidemic
    (file) 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.

(file) 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.

(file) OnDevice
    (file) lib
    (doc) boot.py
    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 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.
```

Evaluation

The majority of the evaluation was conducted according to the plan in Appendix B [Appendix B]. It was split into a performance evaluation of the MANET and then a more qualitative evaluation of the whole system.

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.

4.1.1 Latency

The latency of the system was as the time between a node being generated at node A and received at node B. Node B was then moved further away from node A, eventually moving to 500m away with an interim node, C. The distances between nodes 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.

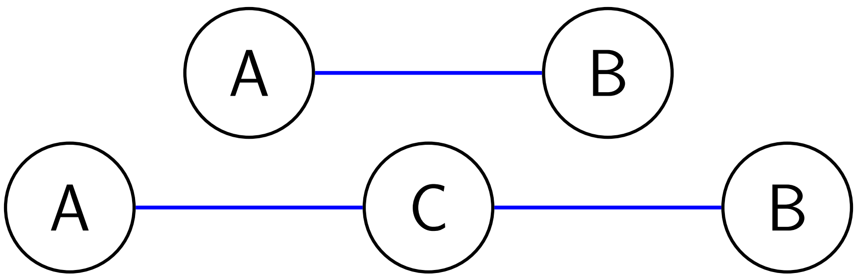


Figure 4.1: The network used for latency testing

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. It is unsurprising to find that the charts 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 then a significant drop-off when the distance is increased to 250 and 500 metres, likely because more than one hop needed to be used 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.

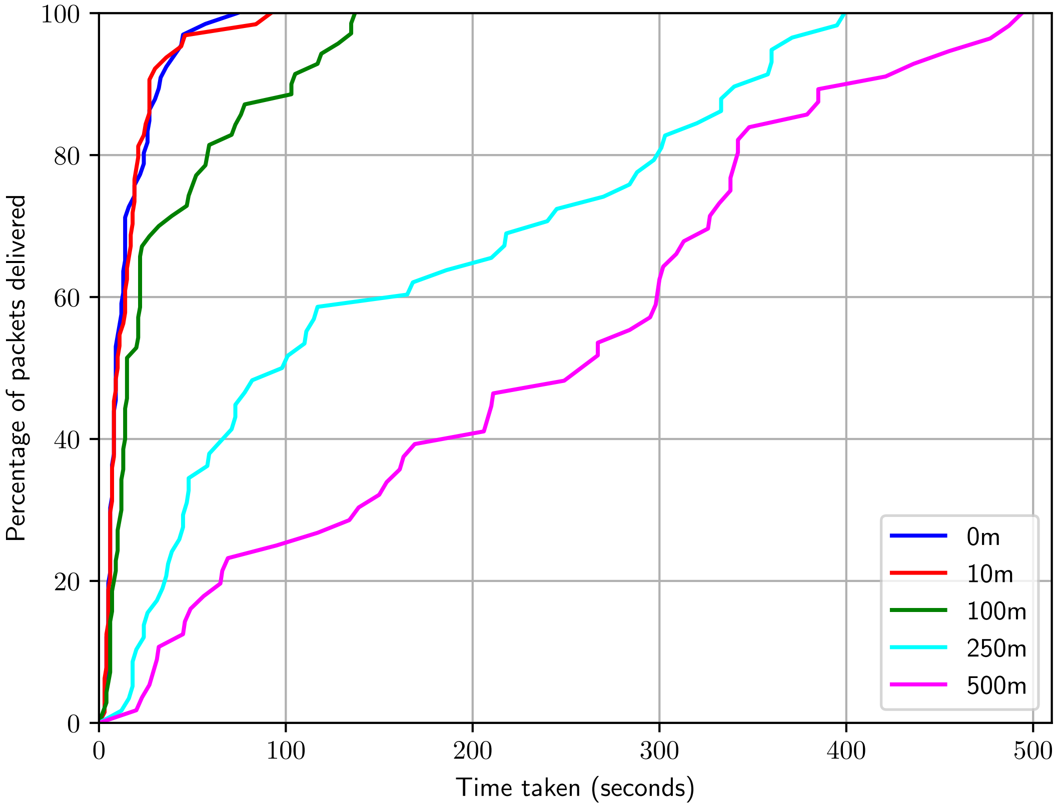


Figure 4.2: The percentage of packets delivered with time

Distance	Average Latency /Seconds	Standard Deviation /Seconds
0	15.045	13.796
10	15.703	16.199
100	34.914	37.966
250	151.741	126.721
500	233.357	138.679

Table 4.1: Average delivery and standard deviation in seconds

4.1.2 Bandwidth

Testing the bandwidth of the system was done with the aim of working out how many messages the system can cope with. In reality, it is unlikely the MANET will need to process as many messages as in this test. 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 0.35 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, I could decrease the time between discovery (‘hello’) packets sent by each node.

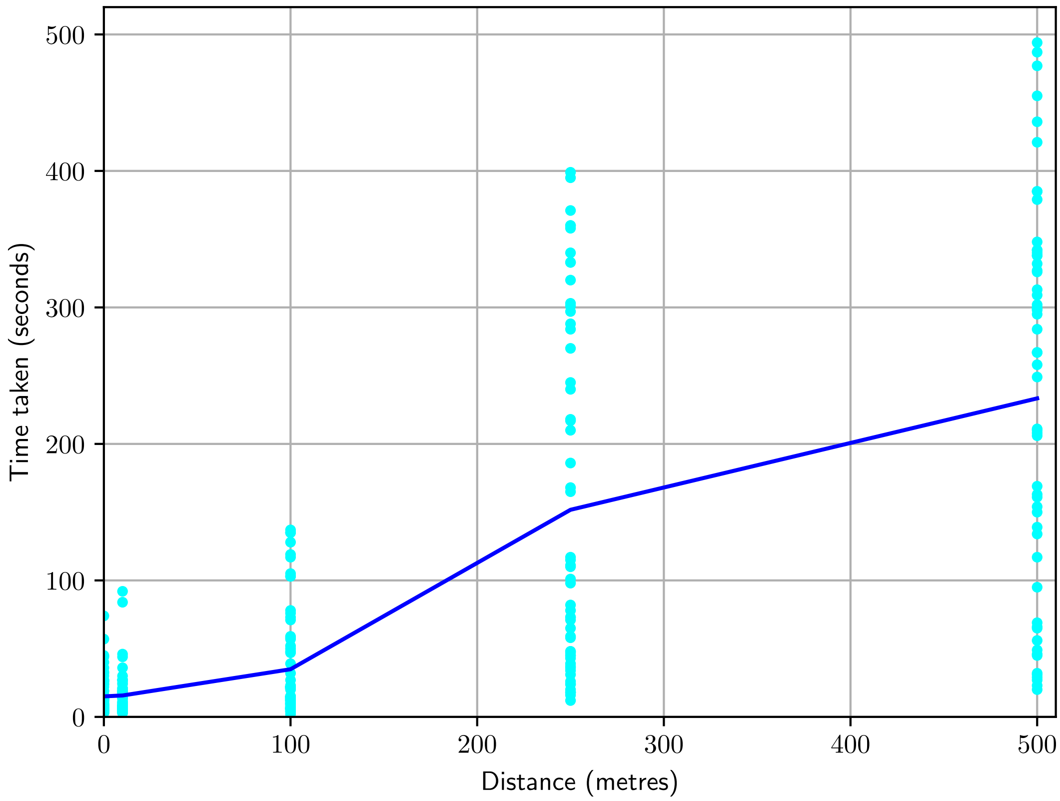


Figure 4.3: The time taken to deliver messages over distance intervals

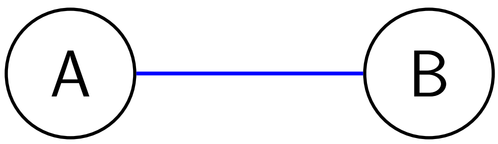


Figure 4.4: The network used for bandwidth testing

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.

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, 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.

While this test was run three times, only two of the results can be used due to a loose connection meaning a node did not work on the second run. In both these tests, 100% of packets were delivered.

4.1.4 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.

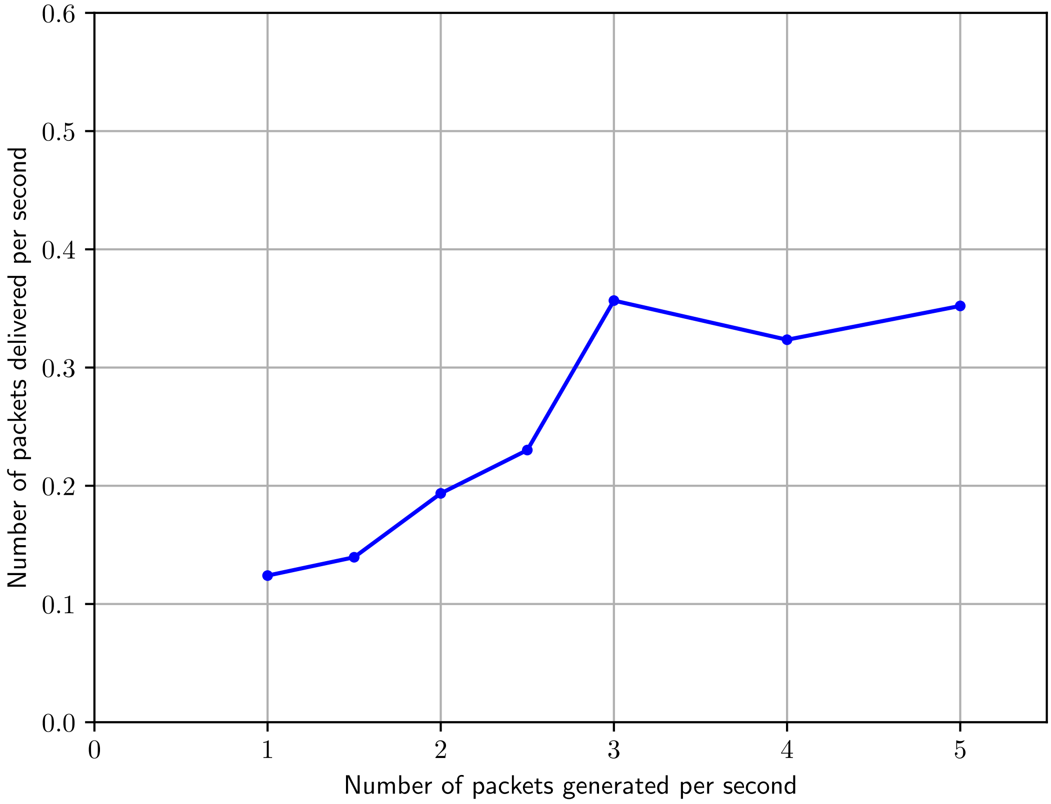


Figure 4.5: The bandwidth of a fully connected two node network with an increasing number of messages generated each second (TODO - extend to 10)

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.

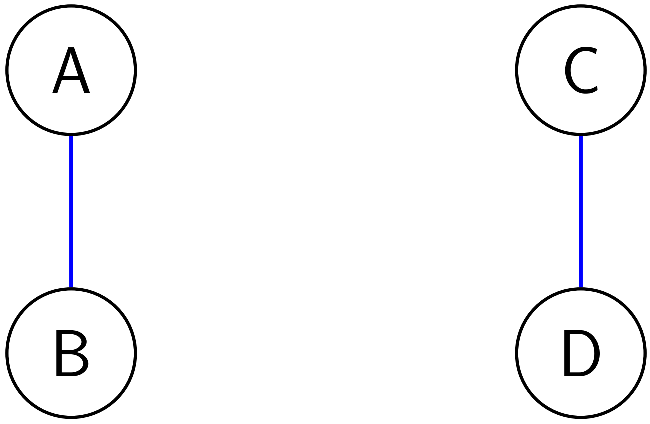


Figure 4.6: The network at the start of partition testing

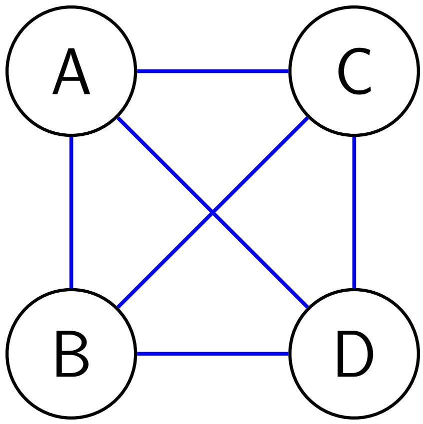


Figure 4.7: The network at the end of partition testing

4.1.4.1 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.

4.1.4.2 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.

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

Node A	Node B	Node C	Node D
87	159	145	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.

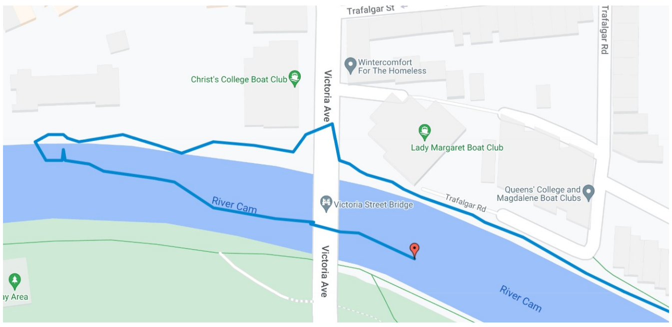


Figure 4.8: The GPS track taken by a node, shown on Google Maps [12]

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.

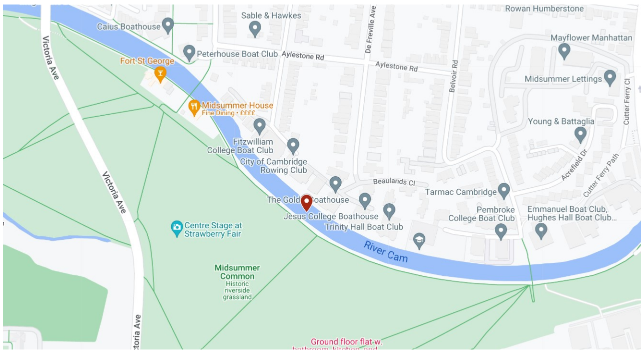


Figure 4.9: The registered obstacle as a red marker on Google Maps [12]

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 low level details 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.

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.

Some of the networking extensions were drawn from my research into networking 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, for instance decide if boats have collided within competitions, removing human bias.

Bibliography

[1] J. Jubin and J. Tornow, "The darpa packet radio network protocols," 1987.

[2] D. Beyer, "Accomplishments of the darpa suran program," 1990.

[3] P. Nucholas and K. Hoffman, "Computational challenges of dynamic channel assignment for military manet," 2015.

[4] R. Davidescu and N. Eugen, "Ad hoc networks for the autonomous car," 2017.

[5] Y. Jin, X. Tan, W. Feng, J. Lv, A. Tuerxun, and K. Wang, "Manet for disaster relief based on ndn," 2018.

[6] R. 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]

[12]

[13]

[14]

[15]

[16]

[17]

[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] T. P. Hut. (2023) Raspberry pi pico | the pi hut. [Online]. Available: https://thepihut.com/products/raspberry-pi-pico

[21]

[22]

[23]

[24]

Appendices

A – Guide to Building a Node

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

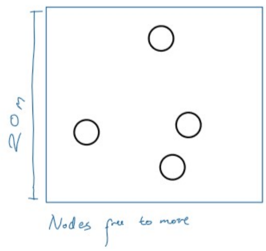
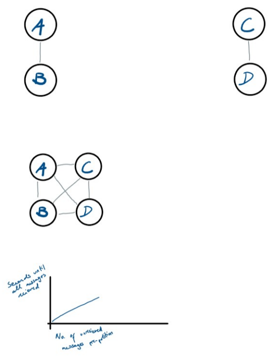


Figure 2: The setup and axes for partition testing



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]

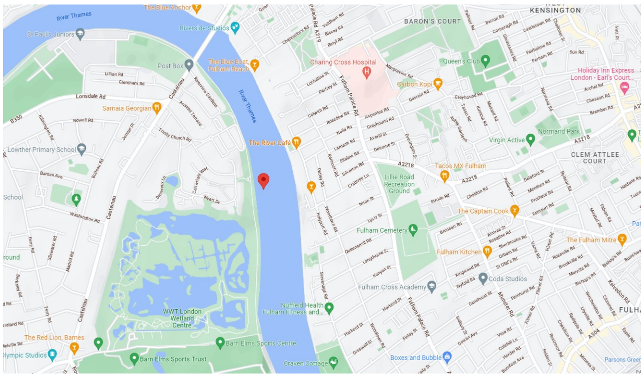


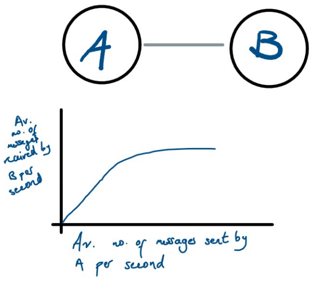
Figure 4: An image of the ‘red buoy’ obstacle [2]

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

Figure 5: The setup and expected graph for testing bandwidth



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

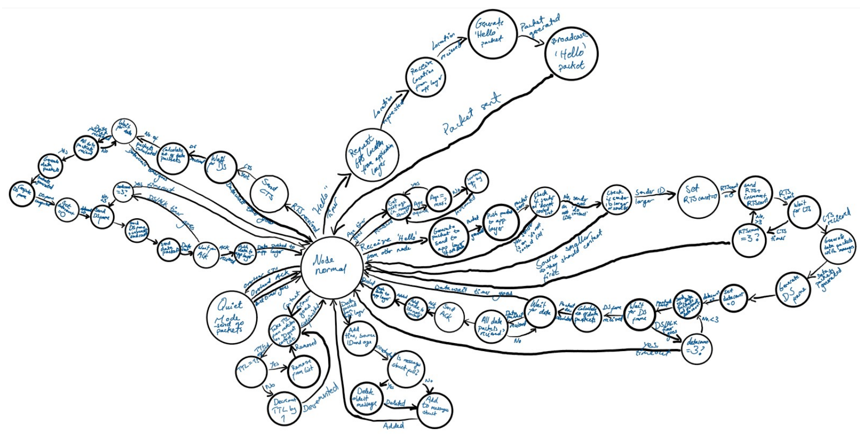
Part II Project - Progress Report

April 26, 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.

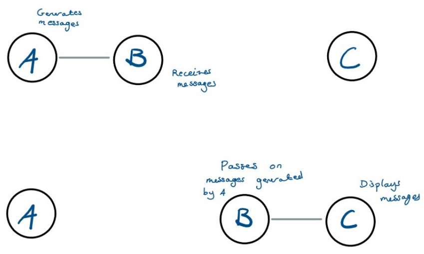
Figure 1: Network state machine



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.

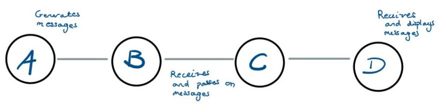
To ensure the network was delay tolerant, I ran a test with three nodes, A, B and C. At the start, nodes A and B were in range of each other and node C was out of range. I set node A up to generate random messages every 40 seconds, with a time to live (TTL) greater than 2 so the messages would survive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

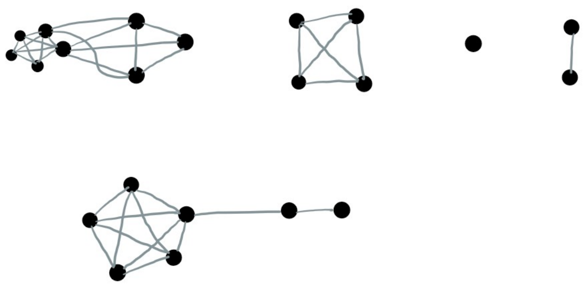
As stated in the cover sheet, Human Participants will be used to help test and evaluate the project. This will comprise a few volunteers to row boats, allowing the network to be run on the water. These volunteers will all be members of Cambridge University Boat Club, able to safely row a boat and navigate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