Introduction

During this project I built a system to prevent crashes in rowing boats. The system is implemented in hardware, cumulating in a series of boxes that can be attached to boats. The project is split into two parts – the MANET that allows nodes to communicate a dynamic collection of known obstacles, and the application layer that warns users when they are approaching an obstacle and allows the user to add obstacles. The system will be available online, fully documented, after my gradation. This will include a 'how to' guide [Appendix A] for construction of a node so any rowing club can use my project as a collision avoidance tool.

1.1 Motivation

Crashes between rowing boats and obstacles or other boats injure rowers and cause equipment damage. Crashes are common – Figure 1.1 shows an eight on the Thames, having collided with Barnes 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 an eight 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]

1.2 Background

There is precedent for using MANETs on rowing boats. I spoke to Duncan Barnes, part of 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 they attached to the crews for the 2023 Boat Race.

Figure 1.2: Equipment placed on the stern of the Cambridge boat [2]

A technical challenge faced by my project was effectively utilising the small CPU on the Raspberry Pi Pico [3] to propagate messages through an unpredictable dynamic topology, with nodes that frequently move at speeds of more than 15 km/h.

1.3 Related Work

1.3.1 Networking

The Defence Advanced Research Projects Agency (DARPA) Packet Radio Network (PRNET) [4] 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) [5]. The connection between the military and ad hoc networking still exists today, with MANETs used in conflicts [6], as well as autonomous vehicles [7] and disaster relief scenarios where previously existing infrastructure is destroyed [8].

The Open Systems Interconnection (OSI) model for computer networking provides an abstraction of the communications between computers. The OSI model consists of seven layers: Application, Presentation, Session, Transport, Network, Data Link, and Physical [9]. This project uses the libraries provided with hardware for the Physical and Data Link layers. It focusses mainly on the Networking layer, sending messages between nodes.

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 messages.

1.3.2 Delay Tolerant Routing

Delay tolerant routing assumes that networks will have lack of connectivity, with partitions between nodes. Vahdat and Becker's paper 'Epidemic Routing for Partially-Connected Ad Hoc Networks' [10] 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. The term 'anti-entropy', as used in my dissertation, comes from this paper. An anti-entropy session occurs when two nodes come into communication range and exchange messages.

1.3.3 Safety in Rowing

ROWCUS [11], a company based in Switzerland, has attempted to solve this problem with a different technical solution – using radar rather than GPS to detect proximity to obstacles, and without networking. While ROWCUS has similar goals to my project, the technical methodologies are different. The main differences are use of GPS and networking – my project connects nodes together while ROWCUS has individual nodes. Additionally, ROWCUS has "decided not to pursue the commercial deployment of ROWCUS", from a statement on their website [11].

Preparation

2.1 Starting Point

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

My project implemented a modified version of the Epidemic routing protocol. 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 different routing protocol, much further abstracted than this project. Throughout the course of my project I took the Part II principles of communications course, further expanding my knowledge.

2.2 Choice of Hardware

Hardware was ordered before the start of term, in order to satisfy the requirements of the project proposal. The main factors in my hardware choices were size, weight, 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 control the system due to the reduction in size and weight it would offer over a single board computer. The Raspberry Pi Pico was chosen due to large peripheral set, with support for both UART, SPI, and I2C protocols. This allowed for greater flexibility throughout the project.

The 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 a large community surrounding it. The chip 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 additional bureaucracy.
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 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.

2.3 Requirements

The three requirements I identified as my success criteria were:

The Epidemic routing protocol is implemented within the network	**Achieved**
A user interface has been implemented to allow utility of the network	**Achieved**
An evaluation of the network has been carried out	**Achieved**

All of these requirements have been implemented during this project.

I also laid out several extension tasks. Some of them were drawn from my research into networking protocols other than Epidemic. For instance, I wanted to use a metric for 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 these extensions, I have implemented an extension allowing messages to have two priorities – normal and urgent.

These requirements were based around the use case for the project – as devices attached to rowing boats, where the nodes have high mobility. This mobility requires a high degree of flexibility within the network. I analysed the potential topology of networks. This was done by looking at example distributions of rowing boats on lakes and rivers. One potential use case for this network that I am familiar with is the river Thames. 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, then adding them to a map with potential connections between nodes, assuming the radios have a range of 500m, alongside obstacles. Figure 2.3 shows a satellite image of a single sculler, a potential node in this network. Figure 2.4 shows the marked up map of the Thames, and figure 2.5 presents the abstracted network topology of these rowing boats.

Figure 2.3: A rowing boat seen on Google Earth [12]

Figure 2.4: Rowing boats, potential obstacles and assumed connections marked on a Google Maps map of the river Thames

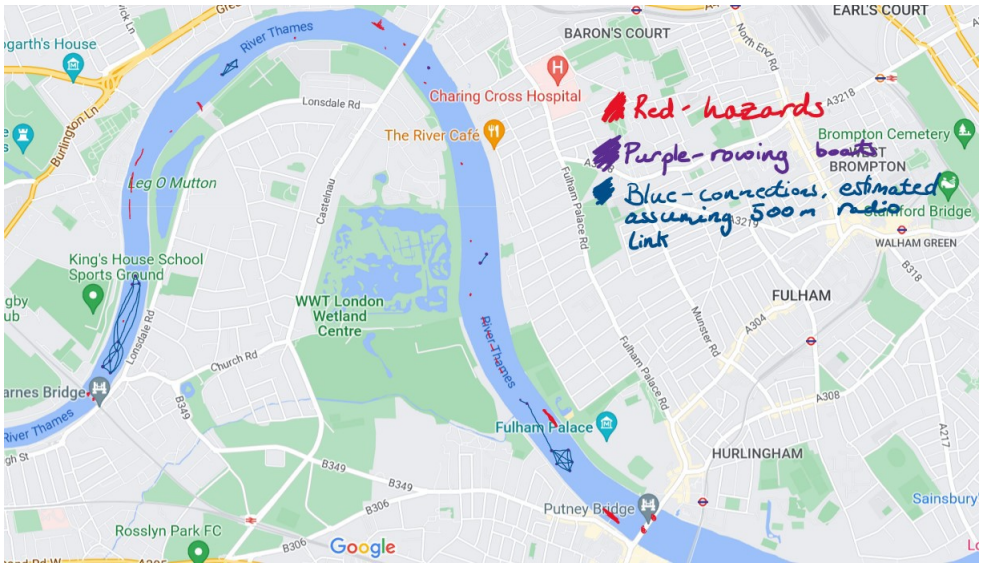
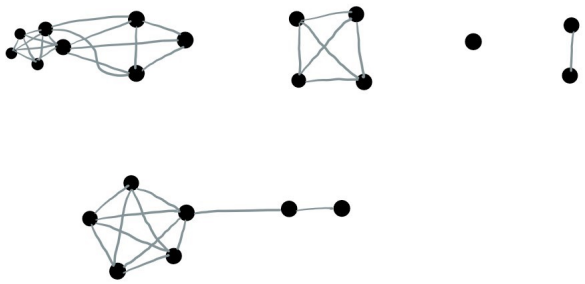


Figure 2.5: The network of rowing boats in an abstracted form



2.4 Choice of Programming Language

This project used CircuitPython, a branch of MicroPython, a version of Python for microcontrollers. CircuitPython was used as it can easily be run on the Raspberry Pi Pico, with no need to configure new development environments. 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 an inconvenience. Additionally, the existing AdaFruit libraries I used in the project were written in CircuitPython, so keeping the whole project to 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 repositories and submit a pull request to AdaFruit.

I used Visual Studio Code as my main Integrated Development Environment as I have used it on previous project, and it offered support for CircuitPython.

2.5 Summary of Research

2.5.1 MANET

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 due to the lack of pre-existing infrastructure and the difficulties associated with setting up and maintaining a base station or similar. Requiring an infrastructure like this would also raise the barrier for entry for many clubs with limited funds and technical skills. Additionally, rowing boats can move between stretches of water, further supporting use of a MANET for this project.

2.5.2 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 as the routing protocol to implement for my project, I considered several other protocols. The the Better Approach to Ad Hoc Mobile Networking (BATMAN) 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) [16] is a location based routing protocol. 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 in 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.

2.5.3 Epidemic

I chose to implement Epidemic in my project [10]. Epidemic routing gains its names from its similarity to the spreading of infections. Each node replicates and transmits messages to neighbours that have not recently been contacted. These neighbours are discovered when each node broadcasts its existence to its neighbours. Epidemic was implemented in this project because it is a delay-tolerant routing protocol and best fits the likely network topology generated by rowing boats, as detailed in the Requirements section below. The networks generated by rowing boats have a high chance of partition, but the nodes are highly mobile. As Epidemic allows any node to carry network information it is best suited to the network. Additionally, Epidemic 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.
Anti-entropy – message exchange – within Epidemic is initiated by the node with the lower ID. These lower ID node initiates anti-entropy by sending a summary vector a, representative of the messages

they have to another node. This node calculates logical AND between this summary vector and the negation of its own summary vector to produce the messages it has not seen that the other node has. It requests these messages. The node with the higher ID then sends its updated summary vector to the other node, and the node with the lower ID requests any messages it has not seen before. After a successful message exchange, the node is added to a list of recently contacted nodes, preventing anti-entropy from being conducted many times between two sets of nodes. This list is periodically cleared.

2.5.4 Media Access Control

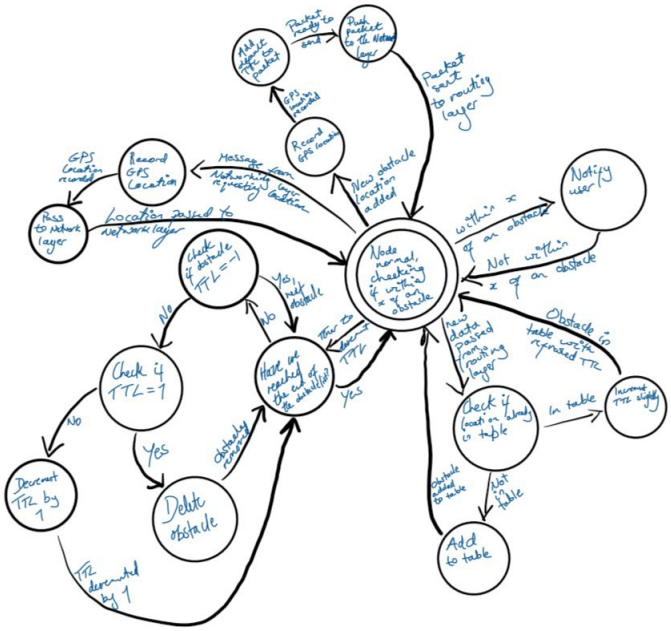
While these routing protocols often have mechanisms to minimise the probability of collisions, such as adding a jitter in the sending of discovery messages, none of them contain medium access control. Due to the probability of collisions between messages causing disruption to the sending of messages, 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. 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.

2.6 System Design

Having decided on the Epidemic routing protocol, I planned the structure of the software that would run on each node. I knew that the Raspberry Pi Pico uses the RP2040 chip, containing two cores. To make best use of the hardware, I decided to design application and networking threads that would run on each core, with global data structures and concurrency control to pass messages between the two layers and allow both of them to access the GPS board.

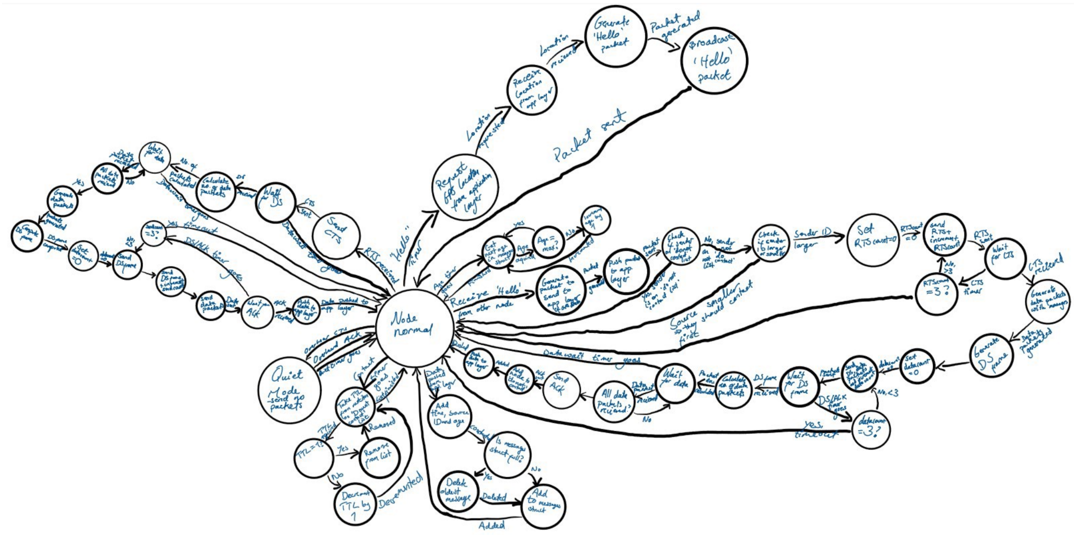
The application thread is responsible for notifying the user when they are too close to an obstacle. It also takes input from the user, generating new obstacles when a button is pressed. The initial state machine is shown in Figure 2.4. The a time to live (TTL) for each obstacle. If the object is permanent, such as a bridge, the TTL is set to -1, indicating that it should never be removed.

Figure 2.6: The initial state machine for the application thread



The networking thread contains the implementation of Epidemic with media 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 2.7: 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.