VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Improving Real Time Schedule Information Displays on New Zealand Public Transport Networks

Callum Stewart

Supervisors: Craig Anslow, Stephen Winch

Submitted in partial fulfilment of the requirements for
Bachelor of Engineering with Honours.

Abstract

Most public transport networks in New Zealand do not have real-time internal display systems installed to provide passengers with relevant information about the status of their trip; most notably the upcoming stop schedule. This limitation limits the overall usability of New Zealand public transport systems, especially for passengers new to the area (such as tourists). A local company, Radiola Limited, aimed to resolve this issue with a bespoke software application called the **Next Stop Display**, which produces a user interface centred around a list of upcoming stops on a vehicle. However, this system had a number of limitations that prevented its deployment. The purpose of this project was to develop an updated version that would be in a deployable state. Once the new application was developed, a user study was conducted to evaluate its effectiveness and investigate which information display configurations were preferred by customers. The results of this study indicated a high degree of system usability, and that display configurations providing only directly relevant information were preferred to those showing general announcements or advertising.

Acknowledgments

Firstly, I would like to thank Craig Anslow for his seemingly endless knowledge, guidance and enthusiasm throughout the project; particularly with the evaluation and writing of this report. Secondly I would like to thank Radiola Limited for giving me the opportunity to work on this project. I would particularly like to acknowledge Stephen Winch for his technical advice and administration; as well as James West and Jacob Woods for their assistance and development contributions to the frontend redesign. Finally, I would like to thank those who volunteered in my user study, who took time out of their schedules to help me with this project. Without the substantial contributions of all those acknowledged, this project would not have been possible.

Contents

1 Introduction **1**

2 Background **3**

2.1 Visualizations on Public Transport 3
2.2 RNSD2 System 4
2.2.1 Backend 5
2.2.2 Frontend 6
2.2.3 Replay Script 8
2.2.4 Limitations 8

3 Design **9**

3.1 Initialization 9
3.2 Demonstration Programs 9
3.3 Frontend Modes 10
3.3.1 List Mode 11
3.3.2 Map Mode 12
3.3.3 Advert Mode 13
3.3.4 Video Mode 14
3.3.5 Stretched Mode 14

4 Implementation **15**

4.1 Demonstration Programs 15
4.1.1 Wellington East-by-West Ferry 15
4.1.2 GWRC MetLink Buses 17
4.1.3 Algorithm Flexibility 18
4.2 Frontend changes 18
4.3 Recovery and Reliability 19
4.4 Device Management Server 20
4.5 UDP Package Reduction 22
4.5.1 Initial Implementation 22
4.5.2 Name shortening and data simplification 23
4.5.3 Amalgamated Data 23
4.5.4 Encoding 24
4.5.5 Selective Sending 24
4.5.6 Deployment 24
4.6 Summary 24

5 Evaluation **25**
5.1 User Study 25
5.1.1 Research Questions 25
5.1.2 Participants 25
5.1.3 Procedure 26
5.1.4 Tasks 26
5.2 Results and Analysis 27
5.2.1 Visualization Effectiveness (RQ1) 27
5.2.2 System Usability (RQ3) 29
5.2.3 Qualitative Feedback (RQ3) 30

6 Conclusions **31**
6.1 Contributions 31
6.2 Future Work 32

Chapter 1

Introduction

Public transport is a vitally important component to many people's daily routines. Maximising access to real-time data in an easily readable format is a particular priority, allowing commuters to co-ordinate their trips quickly and effectively. To this end, transport networks frequently make use of physical departure screens deployed at bus stops, and real-time information displayed on websites or mobile applications.

In many countries, this information is complimented by displays mounted inside the vehicles themselves, providing passengers with easily accessible real-time information. However, this technology is not widely available in New Zealand, and the few extant solutions tend to be very simplistic. This project, sponsored by local firm Radiola Limited, aims to solve this issue by developing and evaluating software to control a set of physical screens mounted inside vehicles (such as buses, ferries, and trains). External software to allow for easier management of these devices once deployed will also be built. At a minimum, each screen's user interface will provide a real-time list of upcoming stops on the vehicle's route and the current time. However, the intention is to also display supplementary contextual information - such as the vehicle's current location and time predictions for upcoming stop arrivals.

To this end, the goal of the Next Stop Display Redevelopment project was to develop a software system, deployed on portable computers placed inside the vehicles, that provides this information in an easily readable format. The starting point for this project was an existing demo program provided by Radiola Limited (referred to as **RNSD2**) which contained some of the required functionality in a rough, rudimentary form. A new version of this, called **RNSD3**, would be based off a refinement of this original codebase with a number of features missing from the original (specifically, alternative layouts for the user interface that would present various types of auxiliary information).

Once complete, RNSD3 was required to meet the heightened security and reliability requirements of a public-facing device, necessitating substantial testing and re-evaluation. Finally, external software infrastructure required to maintain a large number of RNSD3 devices was be built and tested - notably, a UDP server program which was configured to receive regular status updates from the devices, thus allowing system administrators to quickly identify any issues.

This contextual information has the potential to simplify the act of commuting and related activities. It will allow people to quickly and accurately determine when to perform actions such as get on, leave or change services and whether they are currently running early or late to their desired destination. The remainder of this document covers the conception, development and evaluation of my work over the past few months on this project in three sections. Firstly, I will examine the state of RNSD2, and from there cover what requirements RNSD3 would need and the design I chose to implement them. Chapter 4 then presents and

discusses the substantive work done, which is followed in Chapter 5 by a user study I completed to assess the effectiveness of the new system. Aside from providing direct feedback to me regarding the quality of my work, this study was intended to provide information to other developers and organizations (such as Radiola Limited) regarding best practice for public facing display systems such as this.

The study indicated that the user interface as a whole was highly usable to passengers, but there was a strong preference for auxiliary information being directly related to the stop sequence. Interface configurations displaying unrelated information (such as unrelated announcements or advertising) were less well received. These results indicated that while this system would be a popular addition to New Zealand's present public transport infrastructure, potential operators of RNSD3 systems should aim to keep the data presented on the devices focussed on their primary task.

Chapter 2

Background

2.1 Visualizations on Public Transport

In a public transportation context, a 'visualization' can be understood as a graphical user interface used to display information relevant to a customer's use of the transport network. The goal of such a system is to make the data being displayed as easy to interpret as possible, so that the customer is able to make more appropriate travel decisions based on their intentions.

A popular example of a visualization is a live display system, which provides real-time information regarding an aspect of the via a public-facing physical display. The most common example of this (especially in New Zealand) is a system placed in a static location, such as a bus stop. Generally, these provide a list of upcoming trip arrivals along with time predictions. In many places, systems like this are replacing traditional static timetables as the preferred schedule information tool.

General enthusiasm for public displays such as this is backed up by academic research. A 2007 study by Katrin Dziekan and Karl Kottenhoff found that they reduced passengers' perceived waiting time, improved the predictability of the network and general stress and anxiety among commuters [1].

Some transport networks (particularly in Europe and North America) have expanded this concept by including real-time displays inside the vehicles themselves. An example of this is shown in Figure 2.1 [2].

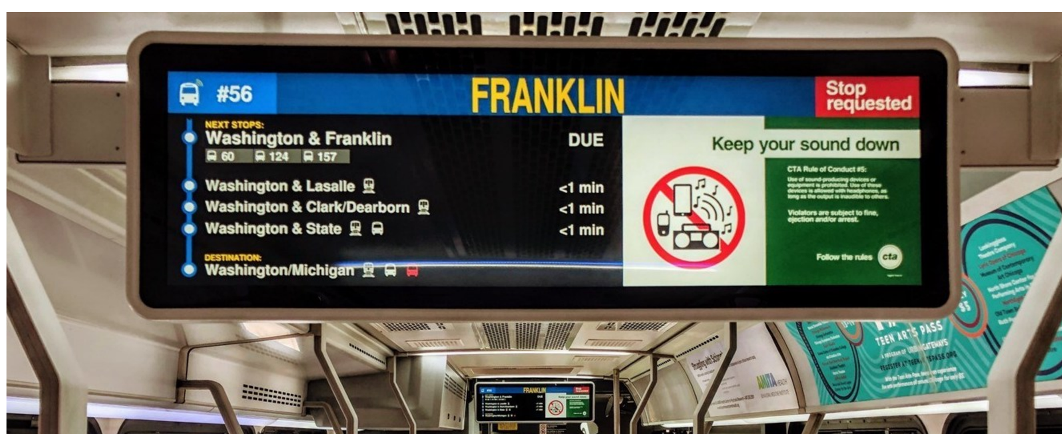


Figure 2.1: Real-time displays on a bus in Chicago. Note the list of upcoming stops on the left side of the screen, and space for image announcements on the right

Specific academic research for devices such as this is unfortunately somewhat limited. However, one recent study worth noting from an adjacent field (conducted by Brewster and Medeiros et al.) investigated the graphical positioning of an augmented reality system designed for passenger use on commercial aircraft. Passengers were found to prefer display layouts that could be easily viewed from within their personal space, while some layouts (such as those where content was placed to the left or right of the passenger at eye level) were found to encourage behaviour that was perceived as antisocial, such as inadvertently staring at another passenger [3].

These findings have implications for physical screens as well, as it indicates that their mounting position on the vehicle can affect passengers' perceptions of the system.

2.2 RNSD2 System

As indicated in the proposal document and introduction, this project is primarily a redevelopment based on the existing RNSD2 system, which was built in mid-2020 by developers at Radiola Limited. Thus, a working knowledge of this system is fundamental to understanding the changes and improvements that have been made so far, and what work remains to be done.

Broadly speaking, RNSD2 can be divided into three interconnected sub-programs – a NodeJS backend used to fetch and send general data on the transport network (which is stored in a local file), a ReasonML frontend used to display the data in a web browser, and a Python-based 'replay' script that can be used to simulate real-time GPS data during development and testing. The entire system is designed to be run on a standalone Linux device with an internet and GPS connection. The system's architecture is outlined in Figure 2.2.

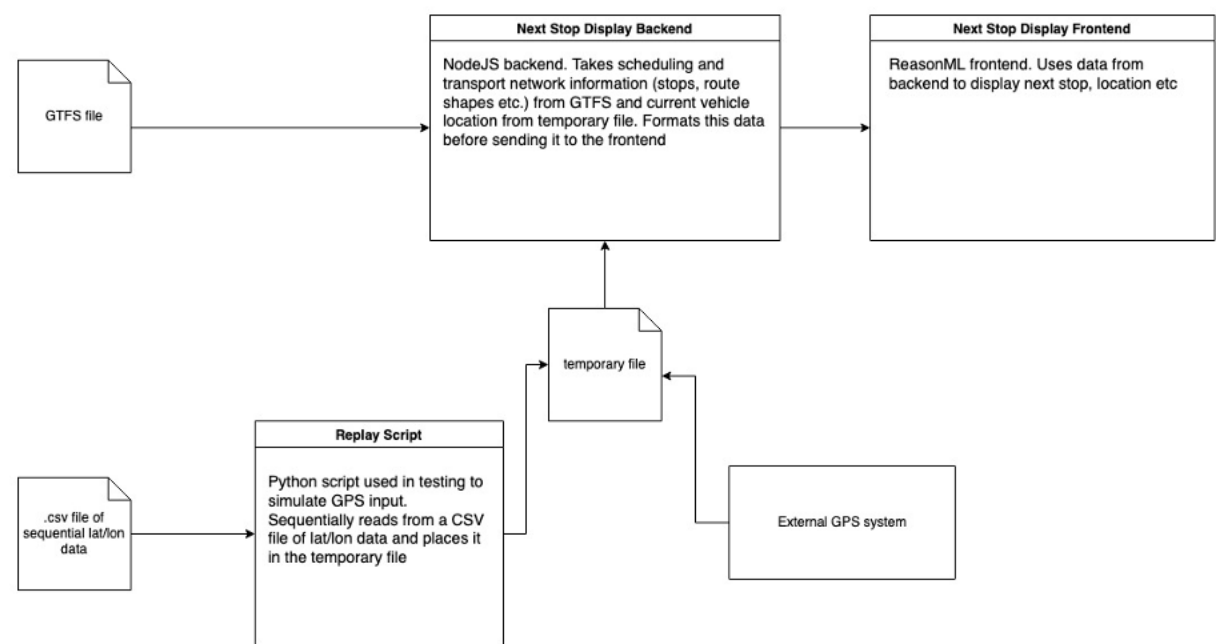


Figure 2.2: RNSD2 Architecture diagram

I do not intend to make any major changes to this structure, and will instead focus on improving and extending what has already been built. Therefore, in the remainder of this section, each of the major features illustrated here will be discussed in detail.

2.2.1 Backend

The most substantial part of the system is the backend, which is written in NodeJS and uses the Koa framework[1] to send relevant data to the frontend. The data in question comes from two primary sources. The first of these is a temporary file (stored in the device's */tmp/* directory) that contains the vehicle's current GPS coordinates. It is updated externally (either by a separate GPS system or the replay script discussed below), and therefore must be accessed at regular intervals to keep up to date.

This is combined with selected data from a populated **GTFS** file. GTFS is a comprehensive specification that contains a list of scheduled 'trip' events (representing each ferry trip on the network), as well as a list of stops on the network and the times when trips are expected to arrive at said stops. Finally, GTFS also contains representations of the physical 'path' of each trip, in the form of a list of short 'segments' between two points. Assuming the current trip has been determined already (this is a contentious issue which is discussed further in section 3.1.1) the correct path as well as the stops and stop times for the trip are then fetched.

Once all the necessary data has been fetched, a two-step process is then used to determine what the next stop is. The first of these is to associate each segment in the path with the appropriate next stop, which is done via the algorithm outlined in Figure 2.2.

```
parameters:
    gpsMargin: meters
    pathSegments: list of path segments for current route, in chronological order
    stops: List of stops on current route

completeSegments = empty array

for each stop in stops:
    nearSegments = all segments in pathSegments within gpsMargin metres of stop
    firstSegment = segment in nearSegments that comes first on the route
    stopSegments = list of all segments in pathSegments before firstSegment
    for each segment in stopSegments
        associate each segment with current stop
    remove values in stopSegments from pathSegments
    add stopSegments to completeSegments
return completeSegments
```

Figure 2.3: The RNSD2 Next Stop Algorithm, which is run at the start of a trip. It iterates over a list of every stop on the trip, and while doing so associates each 'segment' along the route with an appropriate 'Next Stop' value.

Once this algorithm is complete, each segment will have a corresponding 'next stop' field. It is then a fairly simple process to use the vehicle's GPS location to determine which segment it is currently on, and fetch the correct next stop value. Figure 2.4 illustrates how this works in practice.

[1] https://koajs.com/

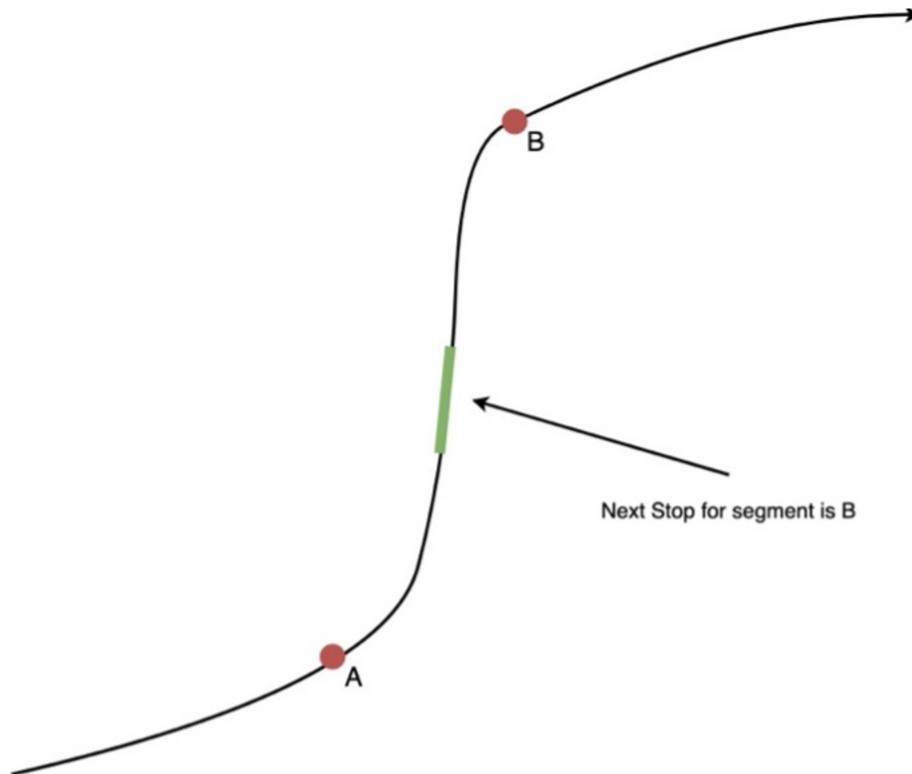


Figure 2.4: If the vehicle (travelling in the direction of the arrow) is on the green highlighted segment, the next stop is B

The path data is also used to calculate time estimations until the arrival at each stop. Once these steps are completed, this data can then sent to the frontend in a JSON format.

2.2.2 Frontend

The frontend is built in ReasonML with ReasonReact. This is a functional, statically-typed language with a syntax designed to be familiar to JavaScript and React developers. The main advantages of this implementation over a more typical React-based one are a more robust type system and less verbose syntax [4]. ReasonReact is an additional plugin that allows React components to be used in the program.

Functionally, the frontend operates by receiving the JSON data from the backend and dynamically rendering it in the browser at the address */localhost:3000/index.html*.This is illustrated in Figure 2.5.

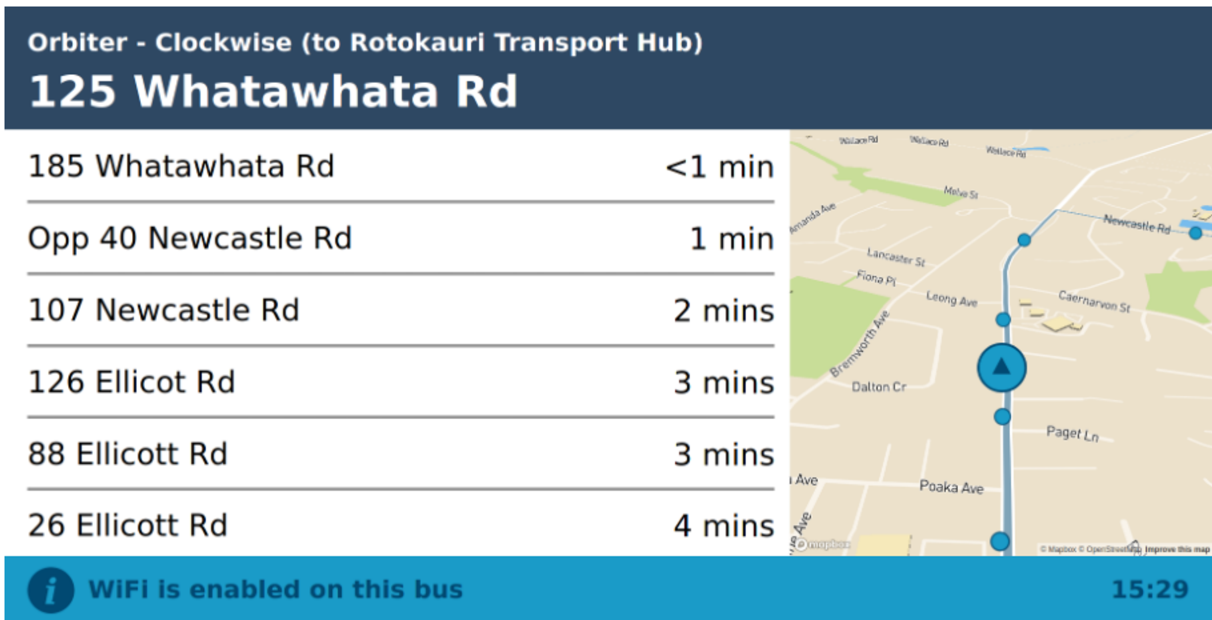


Figure 2.5: RNSD2 user interface running on a simulated bus route in Hamilton, New Zealand. The next stop (125 Whatawhata Rd) is displayed in the header along with the name of the route. The list on the left side of the screen displays the following stops (in order) with estimated durations until arrival.

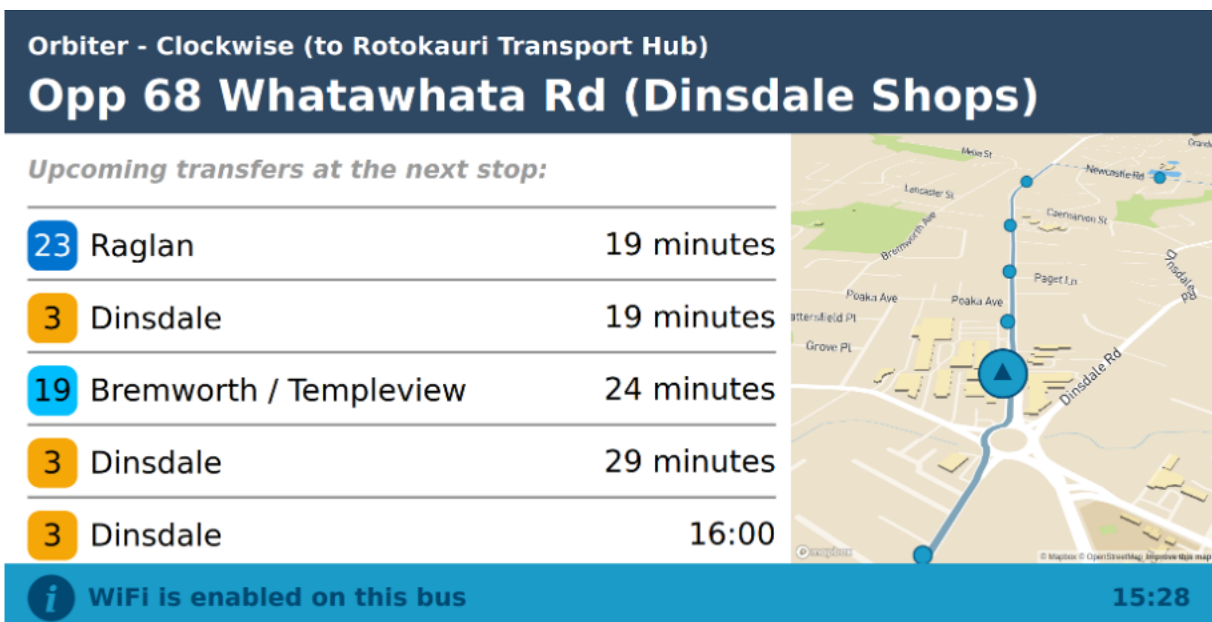


Figure 2.6: In some cases (depending on the GTFS dataset), scheduled data is available for trips departing *from* the next stop. This is displayed (when available) with the intention of assisting commuters who need to catch more than one service on their journey.

On the right side of both Figure 2.5 and 2.6 there is a three-dimensional map displaying the vehicle's current location (using Mapbox GL[2]). The map also includes a line showing the trip's path, with blue markers used to represent the positions of stops. These elements are

[2]https://docs.mapbox.com/mapbox-gl-js/api/

drawn using the GTFS stop and shape data from the backend, which are sent to the frontend along with the upcoming stops.

2.2.3 Replay Script

Developing and testing the system would be impractical while it is physically mounted on a vehicle (or, at the very least, using a feed of GPS data from a real vehicle). To solve this issue, RNSD2 includes a Python script that allows the user to simulate GPS input natively in a controlled manner.

This script works by reading lines periodically from a CSV file (until the end of the file is reached). Each line contains a timestamp, latitude and longitude value delimited by "," characters. This data is then passed into a temporary file (see the general program structure document above) which can then be accessed by the backend. Changing the 'rest' period between file queries can be used to slow down or speed up the vehicle as required.

This system has the advantage that, from the backends' perspective, its output is indistinguishable from an actual GPS feed (as both systems output their data to the same intermediary source). The simple CSV file format also makes it very easy to create mock-up trips to test the system with.

2.2.4 Limitations

At first glance, RNSD2 may seem like quite a feature-complete system. However, in practice it has a number of notable flaws that make it unsuitable as a functioning, public-facing system. The immediate of these issues are a multitude of bugs, mostly caused by a lack of clear structure in the backend codebase. These bugs mean that the system cannot reliably run for long periods of time, even in the controlled environment of a replay script. This issue is compounded by the fact that it is unable to recognize and recover from bugs and/or abnormal inputs, meaning that even small issues will cause it to stop functioning. On a system such as this which receives data from a number of external sources, this lack of error handling and recovery behaviour is a serious oversight. Functionally, there are also a number of unimplemented backend features that have been worked around with sections hardcoded to fit the provided dataset and replay script - the most notable example of this is the currently assigned trip, which is always set to the same value. For the system to be deployed properly, these code sections will need to be re-implemented in a manner that their behaviour can be adjusted (either through a local configuration file or an external input) to fit the network requirements.

Conversely, the frontend is a far more stable affair. However, it has a number of significant aesthetic flaws which likely reduce its viability as a public facing display; ranging from poor text scaling to perspective issues with the map overlays. It is also impossible to customize the user interface in a meaningful way without editing the code itself, which prevents operators from using RNSD2 for any auxiliary purposes (such as service alert changes) or adjusting the core interface layout to better fit their specific needs.

Chapter 3 discusses the design of RNSD3, which was specifically designed to address these limitations.

Chapter 3

Design

The central objective of this project was to develop a new Next Stop Display system, called **RNSD3**, built on top of the system outlined in Chapter 2. RNSD3 is (unlike its predecessor) in a deployable state and interfaces with additional software infrastructure allowing multiple deployed devices to be operated and maintained easily. The following sections outline the design features necessary to fulfil the requirements as determined by Radiola Limited, in broadly chronological order.

3.1 Initialization

As mentioned in the previous chapter, RNSD2 has a substantial number of bugs, as well as a questionably laid out codebase. The first task of the RNSD3 development process was to perform a substantial code refactor, concentrating on the backend where this issue was the most prominent. The aforementioned bugs were fixed to the point where the system ran reliably on the original (Hamilton) replay script and GTFS file - the most prominent of these were related to incorrect installations on new devices and a severe software crash when the replay script finished or restarted. Fixing these issues allowed for substantial feature additions to the system, which are covered in the remaining sections in this chapter.

Nonetheless, it is worth mentioning again that the overall program structure of RNSD2, as shown in Figure 2.2, will be retained in RNSD3 with new features incorporated into its workflow as required. This will preserve the integrity of a number of resources that will not be the focus of this project, such as text-to-speech functionality for audio announcements and the external GPS system to be used when deployed on a vehicle.

3.2 Demonstration Programs

One of Radiola Limited's main requirements for this project was to get Next Stop Display systems running on self-contained devices that could be shown to potential customers, such as councils and other public transport operators. To this end, functioning instances of RNSD3 that ran on customized replay scripts for two possible network deployments were built and tested. Both of these were for networks in Wellington, New Zealand - one for the city's bus network (referred to henceforth as the Greater Wellington Regional Council, or **GWRC**, network) and the other is for the East-by-West Ferry network.

These programs are likely to be connected to screens with different dimensions - while the GWRC network will use a typical 16:9 display ratio, the ferry network is intended to use

an unusual 16:3.2 'stretched' screen[1], as shown below:

Figure 3.1: A Litemax 1916-I digital display similar to the one to be used by the East-by-West demonstration program. It would be mounted horizontally on the vehicle, in a similar manner to the bus display shown in Figure 2.1.

As a result of this and the nature of the East-by-West network this variant was designed for; a temporary frontend user interface was also built to take advantage of it (this was eventually replaced by the 'Stretched Mode' design shown in section 3.3, so I have not included an example of it here. However, it is discussed in greater detail in section 4.1.1).

Finally, due to physical differences between buses and ferries, new algorithms had to be implemented to allow RNSD3 to function properly on either type of network.

3.3 Frontend Modes

While the RNSD2 frontend was easily its most stable part, it had a number of aesthetic deficiencies. Thus, a substantial refactor was performed to get it to a standard where Radiola Limited would be comfortable advertising, selling and shipping units running the system. Due to the relative success of the RNSD2 frontend, this process was done in the same ReasonML language.

This section saw the implementation of features which most obviously distinguish RNSD3 from its predecessor. The new frontend has five different 'display modes' which can be switched between on-the-fly using a configuration value. Four of these modes, in addition to the system's basic purpose of listing the upcoming stops, include different types of additional contextual information. The fifth mode is a permanent version of the alternative frontend mentioned in section 3.2.

The five modes are as follows: List, Map, Advert, Video and Stretched. They are each discussed in the following subsections.

[1]https://www.litemax.com/product-detail/1916-I-spanpixel-industrial-display/

3.3.1 List Mode

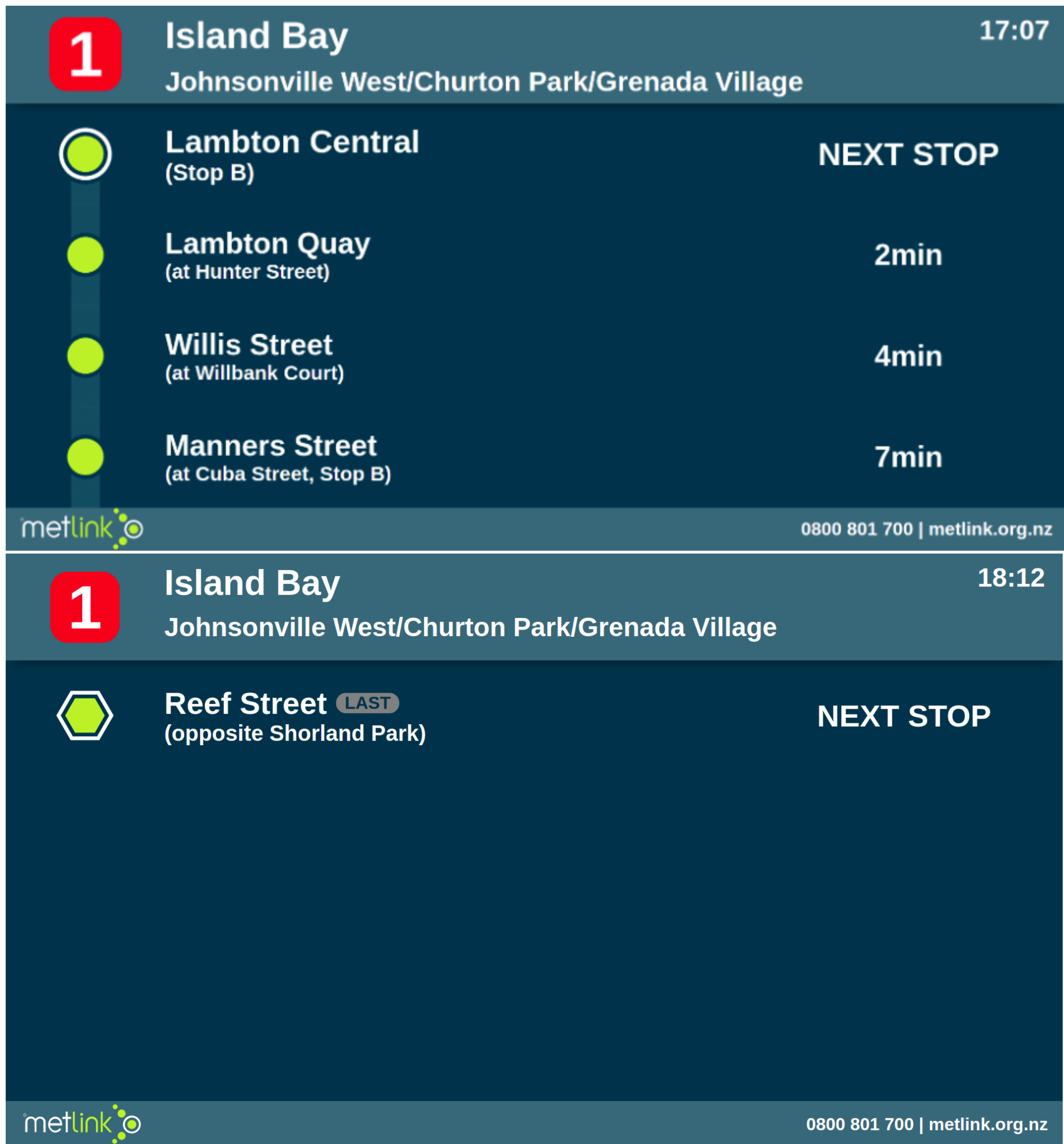


Figure 3.2: List mode configuration. The last stop is denoted via a text indicator and a unique hexagonal icon.

The List mode serves as the system default, and provides predicted arrival time data for each upcoming stop. Note the graphic on the far left of the screen, which provides bulleting while referencing a typical public transport map design.

3.3.2 Map Mode

Figure 3.3: Map mode configuration, showing the vehicle's real-time location.

The Map mode shows a three-dimensional map of the current surrounding area. This map has three overlayed elements to provide relevant information – a central forward-pointing icon indicating the location of the vehicle (this is icon does not move, instead the map position and alignment is adjusted accordingly); a pink line indicating the vehicle's scheduled route; and green markers (styled similar to those on the left side of the screen) to shown the locations of scheduled stops along the route.

3.3.3 Advert Mode

Figure 3.4: Advert mode configuration, displaying one or more static images.

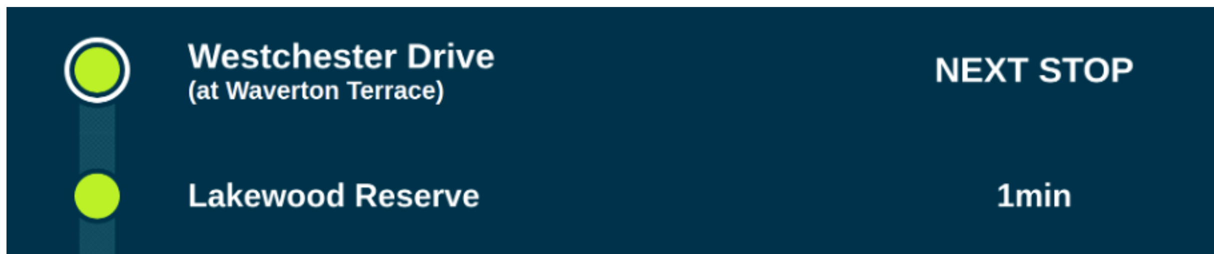
The Advert mode uses the right side of the screen to display one or more static images. These images could be used to show announcements (such as in Figure 3.4, where a request to use the NZ COVID Tracer application is displayed) or advertising. Images for display are stored in a local directory, allowing the operators to easily switch between different adverts. If there are multiple images in this directory, the system will automatically switch between them every 30 seconds. If the directory is empty, the List view shown in Figure 3.2 is used as a stop-gap measure.

3.3.4 Video Mode

Figure 3.5: Video mode configuration. The video is intended to play automatically, and in a window identical in size to the Map and Advert modes.

The Video mode allows the operator to display a video on the right side of the interface. The potential use cases of this are similar to the Advert mode discussed prior - to show announcements, advertising or other promotional materials (the screenshot above shows the trailer for the 2020 Wellington On a Plate festival). The video plays automatically and on a continuous loop. Unlike with the Advert mode, only one video can be shown without a manual configuration change.

3.3.5 Stretched Mode



The Stretched mode has a reduced interface of information compared to those shown prior. It is intended to replace the interface variant mentioned in Section 3.2, and is optimized for the smaller 19.1′ stretched display shown in Figure 3.1. This mode is intended for use in very small networks, where the amount of content displayed on the other modes is not necessary.

Chapter 4

Implementation

4.1 Demonstration Programs

4.1.1 Wellington East-by-West Ferry

Due to deployment requirements at the time, the first demonstration I started work on was a variant of the system designed to be deployed on the aforementioned East-by-West ferry network, shown in Figure 4.1.

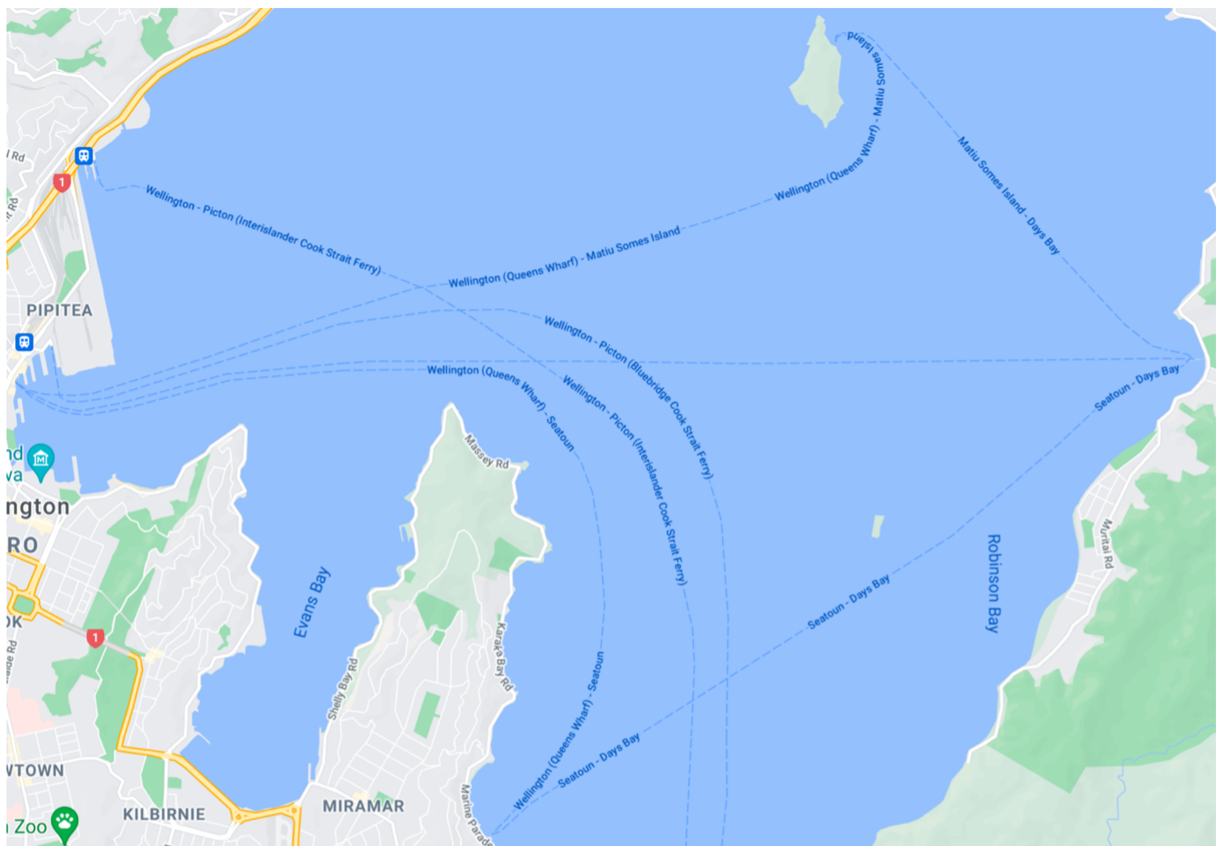


Figure 4.1: an overview of the network, which contains ferry terminals at three locations on the perimeter of Wellington Harbour and an additional terminal on Matiu/Somes Island. Unfortunately the ferry company itself does not provide a graphical map of any kind, so this image is sourced from Google Maps.

RNSD2 was designed specifically for a bus network. Because of this, a number of substantial design and algorithmic changes had to be made for this to be viable. This in effect made for a brand new implementation, which made this section far more time consuming than originally anticipated.

The first change was made due to the fact the East-by-West Ferry only has four stops, and at most a given route has three stops (including the starting point). The stops are also separated from each other by distances of up to 10 kilometres, whereas on bus networks it is common to have stops within a few hundred metres of each other. Because of this, the purpose of a system such as this changes somewhat. With such a low number of stops, the order of upcoming stops becomes less important (customers can be expected to know where to get off). Conversely, however, the time taken between stops is far longer (sometimes in excess of half an hour). Thus, RNSD3's main purpose in this context becomes more to inform customers of how soon the upcoming stops are, rather than the order in which they are coming up. This observation made RNSD2's extant UI unnecessarily complicated, so the frontend underwent a major redesign (shown in Figure 4.2) that only shows a maximum of two upcoming trips.

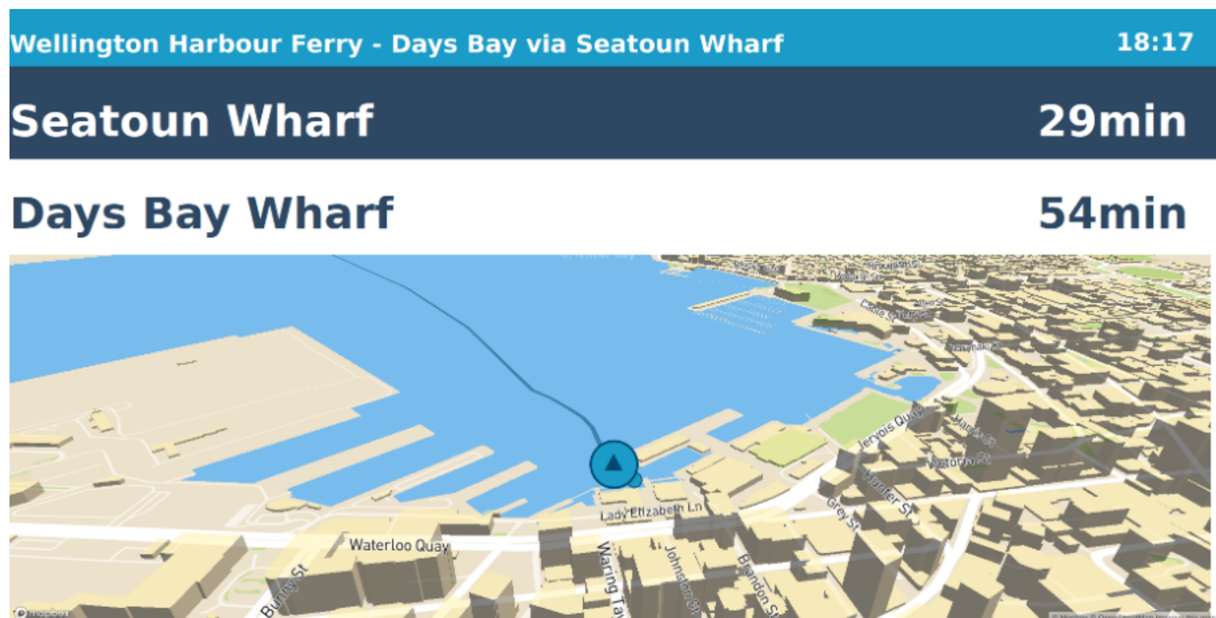


Figure 4.2: The original frontend interface for the Wellington Ferries pilot program. This was a precursor to the Stretched mode shown in Figure 3.6.

The dimensions of this new interface were designed to be compatible with the Litemax display shown in Figure 3.1, which was intended to be used for deployment onto a ferry. The interface's top portion (which shows the time, route and upcoming stops) is designed to fit the display's dimensions perfectly while the map is cut off.

A second major change is the implementation of an automatic trip assignment feature. In RNSD2, the exact trip value is fetched from an external URL, where the assignment itself must be handled by a system manager. This is necessary on bus networks as the different trips take are often very similar, making it hard to distinguish between them. In the case of the East-by-West Ferry, however, the trips are infrequent enough that it is possible to determine what trip it is onboard. This is done via algorithm outlined in Figure 4.3.

```
for (every trip scheduled to run today)
    construct a JSON object containing the trip ID, start/end co-ordinates and scheduled
    start time (this data is fetched from GTFS). Store the object in a list
    for later access.

while(current trip ongoing)
    if (ferry is within 100m of the trip's end co-ordinates)
    break

if (program has just been initialized || the current trip has just ended)
    fetch all trips from the list that start within 100 meters of the ferry's current
    location. Then, find the one whose scheduled start time is closest to the
    current time. Set this is the new trip.
```

Figure 4.3: Trip assignment algorithm for East-by-West Ferry variant of RNSD3, which selects the correct trip event given the time and the ferry's current location.

This variant of RNSD3 can therefore be deployed with minimal outside input from a system manager or the operator of the vehicle. This makes it an ideal pilot program, as it requires minimal external input to function properly.

Finally, the next stop algorithm described in Figure 2.3 proved unsuitable for a ferry network such as this, due to the fact the vehicle is liable to deviate substantially from the GTFS-defined path due to currents, weather conditions and the paths of other ships (unlike a bus, which is confined to a road and will only deviate if the driver makes a wrong turn). This issue is compounded in the East-by-West Ferry network due to the presence of Matiu/Somes Island – vehicles can choose to go round its north or south depending on external factors rather than following a pre-determined route. This means that the ferry can deviate as much as a kilometre from the GTFS path without being functionally 'off-route'. In fact, the only time we can guarantee the ferry will actually be *on-route* is at the stops themselves. This meant that despite this issue the stops *did* update correctly without any changes, but the time predictions were extremely inaccurate. To solve this, a new algorithm was devised to calculate these using the vehicle's average speed and the straight-line distance to the next stop. As most of the routes in the East-by-West Ferry network are fairly direct, this gives a reasonable estimate. However, it is worth noting that if RNSD3 were to be deployed to a ferry network with substantial turns the algorithm is less likely to be accurate.

4.1.2 GWRC MetLink Buses

The second demonstration programme required was for the GWRC bus network. Unlike the ferry system discussed prior, this was relatively easy to set up as the original next stop algorithm and trip assignment behaviour would work as intended. One notable change I *did* make to the program here was to the GTFS initialization process, which is done in the backend via the Node module *node-GTFS*. This module provides an 'import' function which loads the content of the selected GTFS file into a database - meaning the import does not need to be re-run unless the data is changed externally or the database is cleared manually. Despite this, in RNSD2 the initial import was always executed in the program's start-up sequence. This wasn't an issue with the original Hamilton network data but proved problematic when the Wellington network's GTFS file was used - due to its large size, the program would take too long to start up. To solve this, I separated the import action into a new 'sub-program', which is only run automatically from the installation script. This ensures the

system launches quickly.

4.1.3 Algorithm Flexibility

Once the two prior subsections were complete, there now existed two separate 'versions' of RNSD3 implementing the bus and ferry-specific algorithms. To prevent the situation of now having two separate codebases to independently manage, we then worked towards merging them into a single program that could operate using either process, depending on a config value. This proved fairly easy to do in the backend, and allowed for a workflow restructure resulting in a more efficient and easy-to-understand codebase (for instance, large files were divided into smaller ones - restricting each file to one main function where possible).

Conversely, implementing switching behaviour between the two frontend implementations would have been very time consuming and ultimately redundant, as at a later stage it was planned to overhaul this part of the program anyway (see Sections 3.3 and 4.2). As a temporary measure, the frontend configurations used on each pilot device were manually set.

4.2 Frontend changes

The second major development focus was an overhaul of the user interface that would, alongside aesthetic streamlining, add the ability to switch between the five display modes. Due to time constraints and business requirements on behalf of Radiola Limited, this involved substantial design and implementation contributions from Radiola employees.

In total, there are five frontend modes. Two of these (the List mode and Map mode) were implemented by the developers acknowledged at the start of this report, along with the basic structure and components that conformed to the updated design language. However, the three other modes (Advert, Video and Stretched) were implemented by myself.

The List, Map, Advert and Video modes are based around the following three core components, of which only the second changes (the Stretched mode is the exception to this, as it is a reimplementation of the interface shown in Figure 4.2.):

1. A header, displaying the current route (including its name and code) and the time in 24-hour format.

2. A central section, which contains the list of upcoming stops as well as mode-specific data.

3. A footer, which displays the transport company's insignia and contact information.

A new config file (called *app.config.json*) contains an entry that can be used to quickly switch between the different modes without manually re-rendering the frontend in the browser.

Implementation of the modes themselves was straightforward for the most part, with the notable exception of a still outstanding issue with the Video mode. Modern browsers will typically block videos from playing automatically; and to enable this behaviour I was required to run raw JavaScript code upon loading the video – unfortunately, ReasonML does not natively implement the functions required for this behaviour. However, it is possible to write raw JavaScript inline, so I was still able to preserve a single unified frontend codebase. Even with these changes, auto-play still only works when audio is disabled. So far the only workaround for this is to manually change the web browser's settings to allow for auto-playing video at the local address.

4.3 Recovery and Reliability

By this stage, RNSD3 was capable of running properly in normal operation. However, the system must also be able to handle unforseen errors in the system or an external component while deployed - due to its relative complexity and public facing nature, this is a particular concern. Therefore, this next step was to develop features that allow RNSD3 to automatically resolve functional issues while in operation - examples of this include a loss of internet, display or GPS connectivity. The system is now able to identify when something is wrong and take action to resolve it, such as restarting the offending component or, if need be, the entire system.

Broadly speaking, once an issue is detected the desired protocol is to first restart the offending component, and if this does not resolve the issue then reboot the whole system. This was implemented via the use of two new 'recovery' components. Issues pertaining to or interfacing with the backend were managed by functions in the file *recovery.js*, while the frontend status is checked using the shell script *browser-checker.sh*. Additionally, some components are run automatically using Ubuntu's built-in systemd [9] service, which will relaunch them if a crash occurs. The exact types of error checked for, and a brief description of how the checking works, are outlined in Figure 4.4.

Issue	Checking Function Location	Checking Function Description
GPS disconnected	*recovery.js*	Checks when file in */tmp/* was last updated. If there has been no update for a configured time, action is taken.
Internet disconnected	*recovery.js*	Pings *www.google.com* every time function is called (by default, every second). If there is no successful ping for a configured time, action is taken.
Web Browser Crash	systemd servive	Browser is launched at *localhost:3000/index.html* automatically, and in full screen mode, by a systemd service. A second service ensures it is always at the front of the window stack.
Web Browser / Frontend Hang	*browser-checker.sh*	Principal function of the *browser-checker* shell script. Checks for a frontend hang by reading the main tab's title – if it contains 'localhost' this indicates a crash.
Backend Crash	systemd service	The backend program itself is run in a systemd service – if it crashes for any reason it is restarted automatically.
Backend Hang	*recovery.js*	Checks when the last time a complete data object was sent to the frontend (indicating the backend is working correctly with no errors). If none is sent for a configured time, action is taken.
Display disconnected	*recovery.js*	Uses NodeJS's *exec* function to run an internal xrandr [10] script. The output of this tells us if the display/monitor is connected correctly.

Figure 4.4: Table detailing how a set of potential system issues are automatically identified and resolved (via NodeJS functions, specialized shell scripts and systemd Linux services).

4.4 Device Management Server

Fundamentally, the Next Stop Display is a standalone system capable of running with minimal external input. This makes individual instances more secure and reliable, as they are not dependant on shared resources. However, this feature becomes an impediment when numerous devices are deployed across a vehicle fleet. Without a central access point, each device would have to be monitored and updated individually – which in the case of larger fleets is unfeasible. To solve this, two key pieces of external software infrastructure were required – a server that receives regular status updates from the devices themselves and processes them into a database, and a system that, when the user provides a vehicle ID, establishes an SSH connection to the corresponding device. As of the writing of this report only the former program is in a functional state, and is thus the focus of this chapter.

Its purpose is to receive packets sent by RNSD3 devices and store the data internally for administrator access (ideally, in a database). These requirements make the User Datagram Protocol (UDP) a perfect fit. A server program that fulfilled them was subsequently built in NodeJS, the architecture of which is outlined in Figure 4.5.

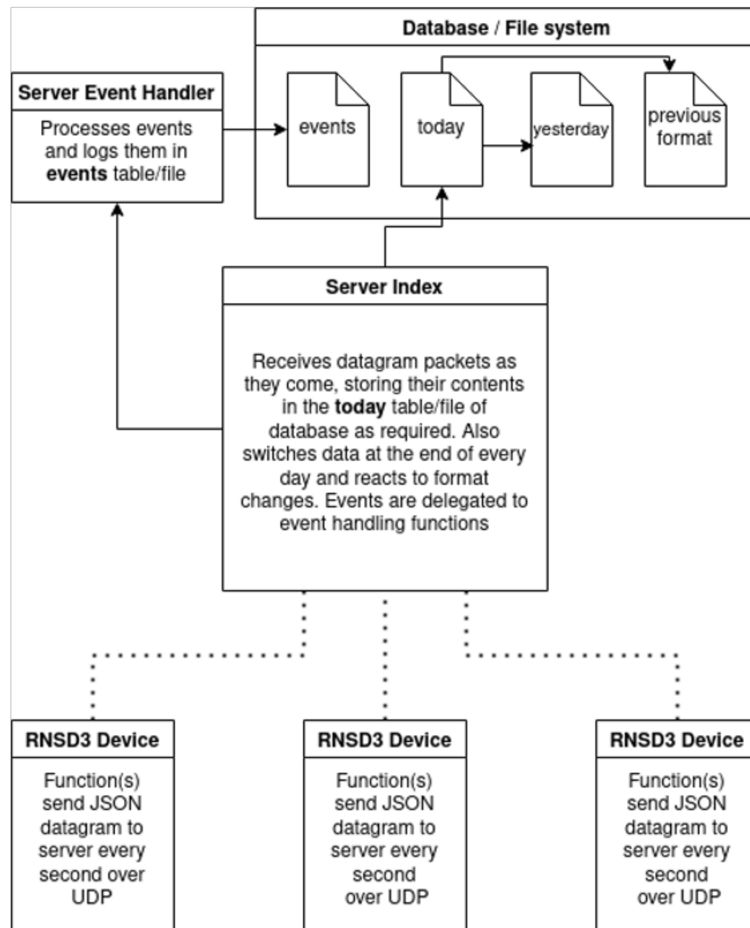


Figure 4.5: Architecture diagram of the UDP server program

Packets are sent from each RNSD3 instance in a JSON format via an NPM package called *udp-json* . This means every piece of data in the packet has an associated key, allowing us to tell what it represents. Currently, the program outputs appropriate formatted data into a total of four CSV files, corresponding to those in Figure 4.5 (However, the long-term intention is to replace these with an SQL database). New logs are initially saved to *today.csv* when they come in, with each row representing a log and each column representing a key. If the keys of the incoming packets change, the contents of *today.csv* are transferred to a new file, *previous-format.csv*. This allows operators to change the data being sent to the server without crashing it or destroying/corrupting the extant data. At the end of every day, the logs in *today.csv* are transferred to *yesterday.csv* where they are cached for another day before being discarded entirely. Finally, *events.csv* records when a Next Stop Display system property (derived from the received packet data) changes – examples of this include an automatic system reboot, loss of GPS connectivity, loss of display connectivity, or if no package is received for more than a minute and the bus was last recorded somewhere along its route (which indicates a loss of internet connectivity). This file is designed to allow operators to quickly identify if one of the deployed devices has a problem without physically checking

it on the vehicle.

4.5 UDP Package Reduction

The server program is dependent on quality log data being sent regularly (ideally every second to ensure accurate time-and-location stamps of issues) to it over the Internet using the UDP protocol. However, this must be balanced with the need to keep system running costs as low as possible. To this end, each RNSD3 device will operate on a fixed 500MB/month cellular data plan, and must therefore consume data at a rate that ensures this limit is not exceeded.

This 500MB would have to cover not only the UDP logs, but potentially large GTFS file updates and display file transfers (i.e., for the Advert and Video modes). Therefore, it was decided that only around 100MB a month could be allocated specifically to the UDP logs, leaving the remainder of the data for these tasks. Each device can be assumed to run for an average of 12 hours a day while constantly sending logs every second - meaning that each log should be under 70 bytes in size, if possible. The subsections below discuss the steps taken to resolve this issue.

4.5.1 Initial Implementation

As discussed in Section 4.4, the NPM package *udp-json* [11] was used both on RNSD3 itself and the server program to send and receive a JSON object containing the relevant data. This allows for a very convenient data format, however the packet does have a 12B overhead which has to be accounted for. Ultimately, it was decided the trade-off was worth it.

The initial set of data sent in each packet is outlined in Figure 4.6:

Name	Type
Date	string
Latitude	float
Longitude	float
Heading	float
No. of satellites	integer
Last GPS connection	Large integer
Last display connection	Large integer
Vehicle ID	integer
uptime	Large integer
CPU usage percentage	float
Total memory	float
Free memory	float

Figure 4.6: Description of UDP data packet contents sent from RNSD3 to the server program.

This covered all the information the UDP server would need in a user-readable format, but at the expense of package size. A server-side calculation showed that packets in this format generally had a size of around 330B. Thus, the monthly data usage of the logging system would be:

$$\frac{(330+12)\times 60\times 60\times 12\times 30}{1.0\times 10^{6}} = 440MB$$

Figure 4.7: Calculation giving monthly data usage of UDP packet sending.

This value is clearly far too high – nearly 90% of the device's monthly data allocation would be used for logging, which when it can be expected to need to download GTFS and configuration updates as well is not feasible without incurring additional running costs.

4.5.2 Name shortening and data simplification

The first step taken to reduce the packet size was to remove superfluous data values and reduce the names of the remaining values to a single character. This would reduce human-readability, but as it is intended to be further processed server-side this isn't a particular concern:

```
const sendUDPLog = (data, new_status) => {
    udp_client.send({
        V: config.vehicleID, // vehicle ID
        T: Date.now(), // timestamp
        t: data.trip.tripID, // trip ID. Data only sent on change
        c: cpuPercentage(), // cpu usage percentage
        m: memPercentage(), // memory usage percentage
        g: gpsStatus(), // binary GPS status
        d: displayStatus(), // binary display status
        L: data.lat, // lat
        l: data.lon, // lon
        H: Math.round(data.heading), // heading (degrees)
    }, config.udp_port, config.udp_host)
}
```

Figure 4.8: Code sample from RNSD3, showing the function used to send UDP packets. Relevant data is placed in a new JSON object via fields and/or sub-functions. The title of each data entry is now only one character to save space.

This new format produced packets between 100 and 110 bytes in size. Applying the calculation from earlier to the worst-case scenario of this gave a monthly data consumption of 158MB. This alone is a significant improvement and is *probably* serviceable for deployment, however further compression is still possible.

4.5.3 Amalgamated Data

The packet built in figure 4.8 contains two separate binary status values. To save space, these were condensed into a single value under an 'S' (representing 'Status') heading. This further reduced the packet size to between 100 and 104 bytes, translating to a monthly consumption of 150MB.

4.5.4 Encoding

The final compression method attempted was to send the data in a hexadecimal format. Surprisingly, however, this actually *increased* the package size with the data being sent – from 100-104 bytes to 130-140 bytes. This increase was mostly due to the fact that complex decimal numbers (i.e., GPS co-ordinates) ended up being more character intensive as hex values. As these numbers cannot be rounded or simplified without losing data, it is better these values are left in their original encoding.

Integer values (such as timestamps) *do* have a reduced length (in terms of ASCII characters), but this change is not translated to data savings as the string format they are saved in is less efficient than the original integer format. Additionally, sending selective hex data has semantic problems on the server-side – without either a manually set schema or additional client data there is no way of determining which data types are hex encoded and which are not.

4.5.5 Selective Sending

With the hex encoding proving to be a dead end, the data packet size was instead substantially reduced through the following two-step process. First, all non-essential data (essential data being basic identifiers such as the timestamp, co-ordinates, heading and vehicle ID) was moved to the status amalgamation. After this, a clause was added so that this value (and the trip ID) were only sent to the server when a change occurred (the data section of each element was left blank otherwise). This design proved quite effective, with the average packet coming in at a size of around 80B. Applying the formula, this gives a monthly consumption of around 119MB – not perfect, but more than serviceable.

4.5.6 Deployment

Once complete, the server program was installed on an Amazon EC2 instance. Its execution on this instance is done via the PM2[1] service, which ensures the program is running at all times. Each RNSD3 device must be able to send data packets to this server via a static IP address. This behaviour was implemented using an AWS load balancer; which ensures that UDP packages can always be sent to the server using the same address.

4.6 Summary

The features covered in this chapter represent the bulk of this project's substantive content. These can be summarized as functioning demonstration programs for the East-by-West Ferry and GWRC Bus networks, automatic system recovery features, and a device management server which can be used to monitor the status of a deployed device externally. To make the latter of these features commercially viable, steps were taken to reduce the size of RNSD3's log packages while still sending the data necessary for their intended use. These features, being confined to the sphere of development and operation, could be evaluated with a thorough internal testing regime to identify and fix any bugs. Conversely, the public facing elements of the system discussed in Chapter 3 would require a user study to determine their effectiveness, which is the focus of Chapter 5.

[1]https://pm2.keymetrics.io/docs/usage/quick-start/

Chapter 5

Evaluation

To evaluate the effectiveness of RNSD3's four primary display modes (List, Map, Advert and Video) a user study was conducted. This chapter describes the user study protocol and results.

5.1 User Study

The user study aims to evaluate the perceived effectiveness of the Radiola Next Stop Display 3 (RNSD3) system's user interface to passengers on Wellington's bus network. This is achieved via a within-subjects design [5], where each participant is shown the set of display modes.

5.1.1 Research Questions

The following research questions were addressed:

- **RQ1:** Which of the four display modes of RNSD3 is the most effective?
- **RQ2:** How usable is the overall system to passengers?

5.1.2 Participants

As the RNSD3 is intended to be public-facing, a user study using a number of volunteer participants who use public transport at least semi-regularly appeared to be most appropriate for recruitment. Participants were recruited from Victoria University students and associates of the Human Computer Interaction (HCI) group; and given local supermarket vouchers supplied by the University upon completion of the study as an honorarium for their time.

Participant ID	Age Category	Gender	Main Type
1	25-34	Male	Bus
2	25-34	Female	Bus
3	18-24	Male	Bus
4	18-24	Male	Bus
5	25-34	Male	Train
6	25-34	Male	Bus

Figure 5.1: Table of User Study Participants. A total of six were selected, five males and one female. All participants caught public transport on at least a monthly basis, and all but one participant primarily took the bus.

5.1.3 Procedure

Each participant was first required to sign a consent form, and then provide the demographic data outlined in Figure 5.1. Once this was done, each participant was shown a working instance of the system running on a replay script, giving an experience as close to actually using it on a bus as realistically possible given my temporal, logistical and financial constraints.

The original plan was for the participant to then be shown four of the primary display modes for a minute each (the List, Map, Advert and Video modes were selected for this; while the Stretched mode was excluded as it is intended for use in a slightly different context to the others). However, as the tests were scheduled to take place in late August and through September of 2021, an unfortunately timed nationwide COVID-19 lockdown meant testing had to wait until late September, which in turn meant that the large sample I was planning on taking for the study would not be possible.

To get around this, I restructured the testing schema such that each user would now be shown only three modes - the List, Map and *either* the Video or Advert modes (as their fundamental designs and intended use cases are very similar). By treating these modes as one I would be able to achieve every possible testing order with only 6 tests, instead of 24 in the original configuration.

The order in which each candidate would be shown the different modes would vary, counterbalancing the learning bias (for instance, candidates might show a slight preference for modes they saw later as they got more 'used' to the system the more they looked at it). Thus, If the modes were always shown in the same order, the results would have an unnatural bias towards whichever mode was shown last.

Once they had seen three modes they were assigned, they would be asked to fill out the questionnaire itself, which itself had three parts (these are outlined in Section 5.1.4).

5.1.4 Tasks

Once the participants had been shown each mode, they were asked to complete a short questionnaire do determine the effectiveness of the system, which was centred around the following three questions:

1. **Visualization Effectiveness (RQ1):** A section where the candidate would rate each mode on a 5-point scale, from 'Least Effective' to 'Most Effective'.

2. **Feedback (RQ2):** Space for (optional) written feedback on the design of each mode.

3. **System Usability (RQ3):** A standard set of ten System Usability Scale (SUS) questions, to determine the overall ease-of-use of the system.

5.2 Results and Analysis

Results from the quantitative sections (Visualization Effectiveness and System Usability) are shown below. A number of qualitative feedback responses for each of the modes were also received, and are provided in full in the appendix.

5.2.1 Visualization Effectiveness (RQ1)

Participant ID (PID)	List	Map	Image	Video
1	5	4	-	1
2	5	4	-	3
3	4	5	4	-
4	4	5	-	1
5	4	4	2	-
6	4	5	2	-

Figure 5.2: Table of response scores for each visualization mode. Scores are from 1 to 5, with a higher score indicating greater effectiveness.

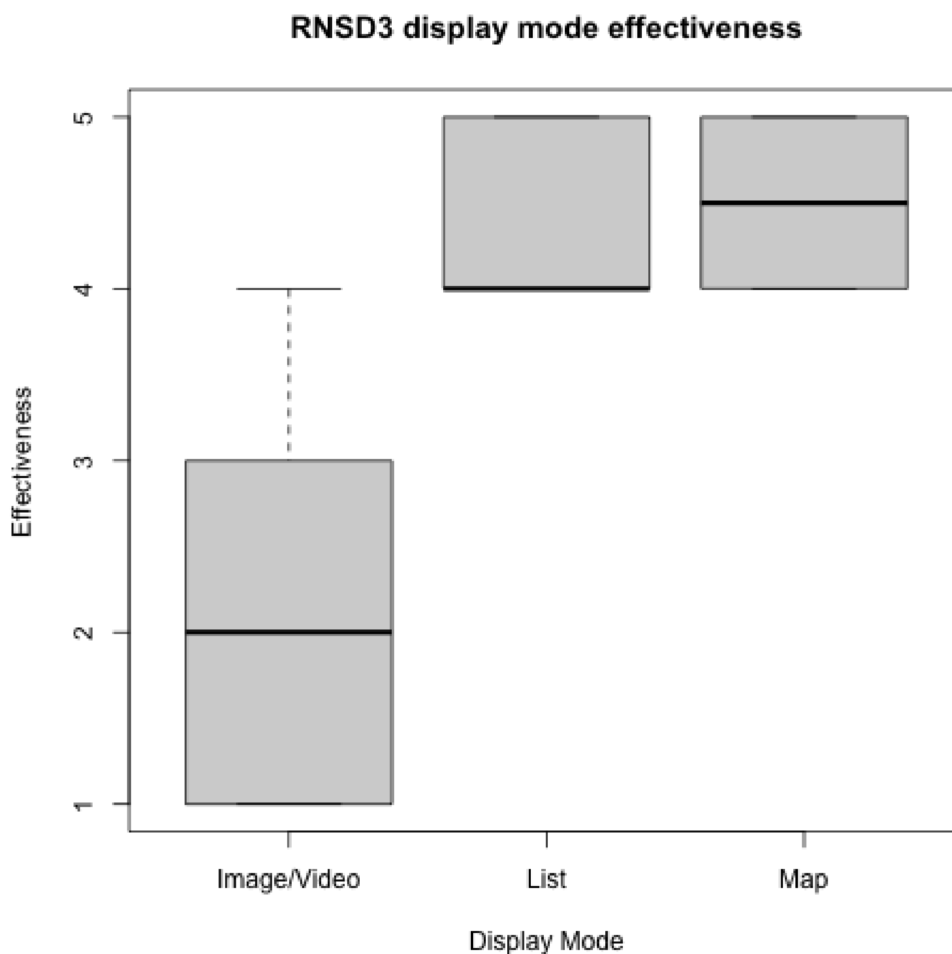


Figure 5.3: Boxplot of mode effectiveness scores, with higher values indicating better performance.

While six tests is not an ideally large sample size, a clear trend can nonetheless be seen. It is clear that the List and Map modes are far preferred to the Advert and Video modes. Every tester rated both the List and Map as either a 4 or 5 on the 5-point scale, with the Map mode having the slight advantage (however, I feel like I have not done enough testing to assert which of these two is prepared by the general public).

The variation in responses for the other two modes was much more varied, with the median value across both modes being 2 and the specific mean values of the Advert and Video being 2.67 and 2, respectively. This variation is mostly down to the presence of an outlier score of 4 for the Advert, with the tester in question noting that they saw the mode's usefulness for giving service alerts (such as delays or route changes) via the static images.

Overall, the result was not surprising. In practice, the two least popular modes replace content that is directly contextually relevant with static content whose relevance may be tangential at best. Of particular note are the two scores of 1 the Video mode received - This is likely due to the fact the video being shown in the demonstration was clearly an advertisement. In their written feedback, PID 2 specifically wrote:

Why do we need ads?

while in a similar manner, PID 4 stated:

I found the frequent changes in colours distracting. It would make my bus rides less comfortable since I would keep noticing sudden changes in the display.

This information is of particular significance to potential system operators (who may see a revenue opportunity with these display modes), as it shows that advertising on an RNSD3 or similar system is likely to generate a backlash from customers.

5.2.2 System Usability (RQ3)

		PID 1	**PID 2**	**PID 3**	**PID 4**	**PID 5**	**PID 6**
Q1	I think I would like to use this system frequently	1	2	1	1	1	1
Q2	I found the system unnecessarily complex	4	4	5	4	5	4
Q3	I thought the system was easy to use	1	2	1	1	1	1
Q4	I think that I would need the support of a technical person to be able to use this system	5	4	3	5	5	5
Q5	I found the various functions in this system were well integrated	2	2	2	2	2	2
Q6	I thought there was too much inconsistency in this system	4	4	4	5	3	5
Q7	I would imagine that most people would learn to use this system very quickly	1	2	2	1	1	1
Q8	I found the system very cumbersome to use	5	4	5	5	5	5
Q9	I felt very confident using the system	1	2	1	1	1	1
Q10	I needed to learn a lot of things before I could get going with this system	5	4	4	5	2	4
Score / 100		92.5	75.0	85.0	95.0	85.0	92.5

Figure 5.4: Table of SUS results and calculations for each participant, with final scores (higher scores indicate a better overall performance).

The SUS was originally created in 1986 by John Brooke [6] , and has since become a standard usability evaluation tool. The participant's responses can then be converted into a score from 0-100, with a higher value indicating greater usability. This is done via the algorithm outlined in Figure 5.5 [7].

1. 1 is subtracted from each response to an odd-numbered question.
2. For each response to an even-numbered question, the response is subtracted from 5.
3. Once the previous two steps are performed appropriately for each question, sum the outputs. Then, multiply this value by 2.5. This gives the final output.

Figure 5.5: SUS final score algorithm.

All six responses to this section produced high final scores, with a minimum of 75.0 and a maximum of 95.0. The mean score was 87.5. Considering the average published SUS score appears to be around 68 [7], this is a very good result. However, the study had been less pushed for time it would have been advisable to do some additional testing where users are required to sit further away from the screen, to ensure it is still legible at greater distances.

5.2.3 Qualitative Feedback (RQ3)

Despite the fact testers were not required to submit written feedback on the three modes they were shown, a good amount of responses were given nonetheless. Some of these have already been mentioned in Section 5.2.1, but there are other design critiques and suggestions related to the broad system design that are worth mentioning here as well. For instance, PID 1 stated:

> *I would look at adding some indicator to show passengers that the values cycle through. Otherwise they may be worried that the vehicle suddenly passed their stop. Some examples I've seen elsewhere are a timer circle that gives a feel for when the page will change. Or a series of dots with the page being displayed being a highlighted dot.*

In other words, there are few visual cues indicating an upcoming or current stop update - the system updates the list of stops seemingly spontaneously. Solutions to this could be a visual progress bar or similar element to indicate how long until the stops change and/or how far through the list of stops the vehicle currently is.

Chapter 6

Conclusions

6.1 Contributions

Real-time information as to the current location and upcoming route of a bus, ferry or other form of public transport is not always obviously available to commuters; often contained in unreliable websites or mobile applications that can be inaccurate and inconvenient to read and access for commuters on their journeys.

One common solution to this issue is a public display with a user interface showing upcoming stops along the current vehicle's route, however at present this technology is not available in New Zealand at an appropriate scale. This project sought to alleviate the issue by developing, testing and evaluating the performance of RNSD3 - an updated, deployment-ready version of the Next Stop Display system from Radiola Limited designed to implement the aforementioned behaviour. This resulted in two pilot programs demonstrating RNSD3's capabilities, running on simulations of two common Wellington bus and ferry routes.

RNSD3 distinguishes itself from its predecessor (RNSD2) through the Display Modes feature, which allows operators to switch the display between four different user interface configurations. These are used to display additional information. The **List** mode gives time predictions, the **Map** mode gives a 3D map of the surrounding area, and the **Advert** and **Video** modes allow operators to display static images and videos of their choosing on the screen. The fifth 'Stretched' mode is an implementation-specific mode that provides a stripped-down interface for a smaller, differently-proportioned screen (intended to be mounted on the inside of a ferry). RNSD3 also has some substantial backend improvements, allowing it to perform recovery procedures when an aspect of the system encounters an error. Finally, an external UDP server was built to streamline the monitoring of large numbers of deployed instances of the system.

The effectiveness of the four main modes was the focus of a subsequent user study, which assessed the perceived effectiveness of each mode as well as the overall usability of the system (via the System Usability Scale questionnaire). Due to time constraints resulting from COVID-19 disruptions, the sample size for this study was somewhat smaller than anticipated - however the results obtained indicated strong enough trends that some assertions can be made.

Firstly, the SUS results gave a mean usability score of 87.5 / 100 for the whole system - far above what is required for a system to be considered 'usable' (which is around 68) and thus indicating a very high degree of usability for potential customers. Secondly, the results of the display mode ratings show that the Map and List modes are perceived to be more effective than the Advert and Video modes. The main reasons for these results with the latter two modes seem to be a loss of supplementary information that directly relates to

upcoming stops, and an opposition to showing advertising and promotional material on the display (this assertion is partly derived from the qualitative feedback received, in addition to the numeric effectiveness data).

These findings have some substantial implications. The high SUS scores indicate that RNSD3 would be seen as a useful and intuitive addition to a public transport system, despite the fact most people will probably not have seen a comparable system before. The disparity in effectiveness scores (and the given reasons for this) should not be discounted, however. Potential clients (such as councils) may be drawn to the revenue or promotional potential of using the Advert and/or Video modes for advertising, but the study results show a decision like this is unlikely to be well received by the public. The work done on this project also made a substantial contribution to Radiola Limited's commercial bid tender with GWRC. As of the writing of this report, the outcome of this tender has not been confirmed.

6.2 Future Work

While the testing conducted so far indicates the system fulfils its requirements well, there are a number of worthy considerations that have not been investigated. A notable example of this is how readable the screen is when viewed from different angles and greater distances; as it will be when used on a bus, train or ferry. During the planning of the project this issue was apparent, and a brief plan for a user study that would test it was produced. This proposed study centred around laying out a set of seats according to the seating plan of local buses in Wellington, and placing RNSD3 screens and horizontal and vertical positions according to where they would be mounted - in effect, simulating the experience of sitting on an actual bus. Participants would then be randomly assigned a seat and asked a series questions about what the screen was showing (i.e. what the current next stop is). The accuracy of these answers would then indicate the screen's readability. Due to the amount of time and resources this study would take to set up, however, it was decided to pursue the (simpler) study discussed in Chapter 5. Thus, the first suggestion to those interested in further development of this system is to perform a further study along the lines of this concept. Poor results in this would indicate significant changes to the user interface will be required.

From an implementation perspective, the first recommended area of focus is an external system designed to complement the UDP server; allowing administrators to quickly connect to active RNSD3 devices (via SSH or a similar protocol) based on an ID value of the vehicle the device is deployed on. A tool such as this, combined with the existing UDP server program, would allow for rapid identification and fixing of issues with individual devices running on a transport network.

Appendix

User Study Qualitative Responses

PID 1

- **List Mode:** I would look at adding some indicator to show passengers that the values cycle through. Otherwise they may be worried that the vehicle suddenly passed their stop. Some examples I've seen elsewhere are a timer circle that gives a feel for when the page will change. Or a series of dots with the page being displayed being a highlighted dot.
- **Map Mode:** think having times would still be useful as it will be difficult for passengers to estimate arrival times. However most passengers really just want to know when their stop is next. So times aren't that vital. I'd also include a timer icon to show when the page of stops will change.
- **Advert Mode:** *no response*
- **Video Mode:** I would look at including times here to highlight the movement of the vehicle. Otherwise it may as well be a vinyl schedule stuck to the screen with the ads playing. And another one for a timer to show when stops displays will change.

PID 2

- **List Mode:** Liked times to destinations displayed.
- **Map Mode:** Liked seeing where you are in relation to next stops and find destination.
- **Advert Mode:** *no response*
- **Video Mode:** Why do we need ads?

PID 3

- **List Mode:** Informative but would pair very well with the map.
- **Map Mode:** Superb!
- **Advert Mode:** Useful for PSAs, and other events.
- **Video Mode:** *no response*

PID 4

- **List Mode:** Changes in time for time prediction was perhaps too frequent. Good it updates in real time but I would have preferred an average if the prediction is not stable.
- **Map Mode:** Great visualization. Perhaps the map could rotate less often. I felt it would have been more clear if only the bus' marker was changing orientation.
- **Advert Mode:** *no response*
- **Video Mode:** I found the frequent changes in colours distracting. It would make my bus rides less comfortable since I would keep noticing sudden changes in the display.

PID 5

- **List Mode:** I liked seeing the estimate. The order of the stops was clear.
- **Map Mode:** The map was eye catching and showed my position effectively, though there was no time estimate.
- **Advert Mode:** The image display showed the order of the stops well, but the display refresh was too fast for me *(illegible)* and the time estimate would have improved it.
- **Video Mode:** *no response*

PID 6

- **List Mode:** *no response*
- **Map Mode:** *no response*
- **Advert Mode:** *no response*
- **Video Mode:** *no response*

Bibliography

[1] K. Dziekan and K. Kottenhoff, "Dynamic at-stop real-time information displays for public transport: effects on customers," *Transportation Research Part A Policy and Practice*, vol. 41, July 2007.

[2] "Arriving now: Digital screens providing real-time service information aboard cta buses," December 2018. https://www.transitchicago.com/arriving-now-digital-screens-providing-real-time-service-information-aboard-cta-buses/.

[3] A. Ng, D. Medeiros, M. McGill, J. Williamson, and S. Brewster, "The passenger experience of mixed reality virtual display layouts in airplane environments," *2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2021.

[4] B. Johnson, "What makes reasonml so great?," September 2018. https://blog.logrocket.com/what-makes-reasonml-so-great-c2c2fc215ccb/.

[5] J. Nielsen, *Usability Engineering*. Morgan Kaufmann Publishers Inc., November 1994.

[6] usability.gov, "System usability scale (sus)." https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html.

[7] J. Sauro, "Measuring usability with the system usability scale (sus)," February 2011. https://measuringu.com/sus/.