Floating Buses: Optimizing bus routes and passenger allocations based on real-time collaboration.

By

Conor McKenna

Dissertation

Presented to

Trinity College, The University of Dublin

In fulfillment

Of the requirements

For the Degree of

Master of Science in Computer Science

Trinity College, The University of Dublin

Declaration

Abstract

Floating Buses: Optimizing bus routes and passenger allocations based on real-time collaboration.

By Conor McKenna
Master of Computer Science
Trinity College Dublin

Supervisor: Siobhán Clarke

With the current implementation of our public transportation systems, people are forced to adhere to strict timetables and fixed routes on their journeys, and are negatively impacted by the growing issue of vehicles reaching overcapacity, either having to stand or be refused entry completely. Conversely, buses have to make needless journeys to allow for the possibility of demand, with no way of knowing whether the trip is worthwhile or not until completion. This has a harmful effect on the environment, with wasted trips leading to harmful emissions being released for no reason. A better solution would be a more dynamic system that caters to passengers needs as they arise. This system wouldn't need to adhere to strict timetables and routes, but instead run when and where the need arises. It would be able to assure admission for each user who signs up for it, while also being able to guarantee that there are passengers available before setting off, cutting out needless journeys completely, reducing wasteful journeys that are harmful to the environment. This better system would allow more communication between passengers and drivers to create a more suitable transportation system for modern life. With modern technology keeping commuters connected 24/7, there is no need for people to still have to rely on outdated, inflexible modes of transportation.

The current state of the art is currently lacking an implementation and justification for a public transportation system with dynamically changing destinations based on live user input. There exist various proposals and prototypes of dynamic bus routing systems with dynamic stops, but few have publicly shown justifiable results for their implementations. Thomas Kearns et al. at the Illinois Institute of Technology (IIT) put forward a prototype solution of a dynamic bus routing system for the city of Chicago, proposing a system that created localised pickup and destination locations for groups of users based on an analysis of demographic data. This is a clever solution to bus routing, but doesn't account for stochastic demand, as user requirements can change daily. Electronics giant Philips proposed a system

using existing infrastructure, such as street lights, as dynamically allocated bus stops, but didn’t properly implement their system out of expectations that a robust IT architecture overhaul would be needed to bring the system to fruition. Other current public transportation routes only with fixed locations between its start and end points, and studies into the area of improving bus flow look at changing the route taken between these static in between points. Existing approaches to vehicle routing exist mainly in the domain of transporting packages to fixed locations, and existing ride-sharing applications focus purely on individual requests rather than servicing a larger community. Furthermore, existing bus route allocation methods only look at fixed routes and timetables, and seating allocation techniques don’t take dynamically changing capacity into account.

This study proposes a practical solution for a dynamic public transportation system, known as a floating bus. This floating bus system takes advantage of dynamic input via user requests from an Android application, to create floating bus stops and plot dynamic routes that cater to their assigned passengers needs. Requests are posted to the web application backend, sending the users email with their pickup coordinates, their destinations coordinates and the required number of seats for their journey. The system accepts requests and assigns them either to existing waypoints, which are aggregations of pick-up and drop-off coordinates from all requests received, or creates new ones to cater for them if they fall out of range of the existing ones. These are then used to create a route, which is then assigned to a free vehicle, or an existing route attached to a vehicle may be updated if one of its waypoints is updated to allow for a new request that lies within its aggregation radius. These waypoints are updated to be positioned at the nearest segment of real road network to function as position of the floating bus stops. Feedback and information is provided using Google Maps and Google Directions APIs, painting the ideal routes onto a Google Maps image, as well as displaying concentration of requests via a heatmap on top of the map.

The results show that a floating bus system has the potential to be a viable solution to creating a better city transportation system. Simulations were run against the system using mock data with different pickup and drop down locations and varying passenger amounts. This involved issuing multiple requests to the system at different locations across Dublin city for multiple vehicles to handle, where results showed that vehicles were assigned the most efficient routes based on their direction and capacity at each waypoint. The vehicles were then simulated to follow the set routes realistically, with passenger allocation changing at each floating bus stop. The metrics recorded from these simulations were time and distance, as well as the distance from a passengers desired pickup and drop off locations to their assigned pickup and drop off locations. passenger convenience. Using the distance travelled metric combined with average pollution output over time the environmental impact of the floating bus routes is also calculated. These are then compared with fixed route alternatives to show an improvement in delivering passengers to capacity while reducing the impact on the environment.

Summary

This project implements a dynamic bus routing system known as a Floating Bus, defined by its ability to generate Floating bus stops that are routed to dynamically. Floating bus stops are generated via user requests for their pickup and drop off destinations, giving the ability to be able to gauge demand for the service and reducing unnecessarily wasted journeys due to little demand or excessive demand (ie: when the bus is full). By knowing the locations to best serve user demand in advance, the floating bus service is able to improve on the public transportation service that serves every stop along its route in anticipation of demand.

To reduce the number of stoppages along the floating bus routes, user pickup and drop off locations are merged into communal floating bus stops, which are reflected in their request object after storing to a local database. These floating stops are then related to road network data gathered via Google APIs such as Directions and Distance Matrix. These help provide the ideal route for a floating bus vehicle and return usable road network data.

Results of the system show cases where a floating bus approach is a more efficient choice than a traditional static bus route, but also shows the pitfalls of poor vehicle and request allocation.

Acknowledgements

I would like to thank my academic supervisor Prof. Siobhán Clarke and Fatemeh Golpayegani for their support and assistance over the course of this project. Their advice was always helpful and pointed my work in the right direction.

I would like to thank my employers at VisibleThread for their support and ability to accommodate my studies while I was working with them part time.

Finally, I'd like to thank my family for their support and patience throughout the year.

Contents

Chapter 1 - Introduction

1.1 Research Question

This project seeks to answer the question, is it practical to create a smarter bus system than the current public transportation system with dynamic bus stops by taking advantage of modern connected smart devices? This project implements a system that dynamically adapts bus routes as new requests come into the system by dynamically creating and merging bus stop points.

1.2 Background and Motivation

As technology evolves and cities get smarter, we are faced with the ever growing issue of old systems being made redundant, often for very good reason. Technological advances that have yet to be incorporated into existing infrastructure make systems feel even more rigid and unsuitable for modern life anymore. A system we still have in place that continues to become more and more out of date is our main method of public transportation, namely the city bus service.

The inner city bus service is quickly becoming a relic of an age before our world of constantly connected devices and the convenience they bring. The public bus service follows strict routes and adheres to strict timetables exactly, and other than being occasionally late it follows these imposed rules exactly, regardless of consumer needs or demands. For example, a bus at full capacity will continue its route to completion, even if it would make more sense to deviate from the set route to reduce capacity sooner, which inconveniences passengers both on and off the bus. A bus will start and complete a route in the dead of night, even though there may be no passengers at all anywhere along the route, wasting a journey completely, causing needless harm to the environment, as well as wasting time and money making the trip. The current public bus system that has become an an inconvenience more than a convenience in many ways, having a negative impact on congestion, the economy, and the environment.

There are many reasons to pursue a better bus system than the one we have now. For example, traffic and losses caused by it. In 2007, congestion induced economic losses in Dublin which accounted for 4.1% of GDP (Gross Domestic Profits)[1]. On top of that, it is predicted that time lost due to aggravated traffic congestion in Dublin will rise to €2 billion by the year 2033[2].

As Dublin grows as a Smart city, so too does its smart infrastructure. With regards to public transportation, RTPI (Real Time Passenger Information) has been implemented into public transportation since 2008[3], which assists in letting potential passengers plan their routes according to the real time information about the public bus service. This technological advance in the domain of public transportation does not do enough in improving the system overall. Passengers still cannot guarantee space on their journeys, and may be left standing in the cold once the bus has reached maximum capacity, having to wait for another free one. They have no influence over the route the bus should take, and may have to walk a considerable distance either side of their journey to make it to their desired destination.

This project presents a dynamic routing system as an alternative to the traditional bus system, dubbed the floating bus. The name comes from the ability to create dynamic routes by creating floating, pop up bus stops that are navigated to by a servicing vehicle. These floating bus stops aren't based off of historical or demographic data, but live, real-time input from users via an Android application, and are related to real road network locations for stop locations. The floating bus system gives more control to the public on how a public transportation service is run. Utilizing the ever connected nature of modern smart devices, users can use the floating bus system to cater to their needs, within reason. The floating bus system enables users to create floating bus stops which dictate routes, allows them to reserve a seat, and receive accurate real time information and notifications of bus arrival to their assigned floating bus stop, in a way that suits them.

1.3 Project Overview

This dissertation compares and details existing research in fields relevant to the floating bus system in Chapter 2. The full implementation of the system is detailed in

Chapter 3, where the individual components of the system are broken down and explained before their use in the system is elaborated on. The results and evaluations of the various simulations and tests are shown in Chapter 4. Chapter 5 discusses how the floating bus system can be developed and iterated on in future, and how it could be utilized in a practical way in modern smart cities.

Chapter 2 - Background

This section gives an insight into the various fields of research studied before embarking on this project. It details the main findings of key papers and their approaches and findings, before comparing them to my own needs in this project. Various technologies are covered based on their suitability for the floating bus system.

After detailing some of the findings that are built upon in the floating bus system, a description of the theoretical background to the application is provided, giving further detail and insight into the reasoning behind the project.

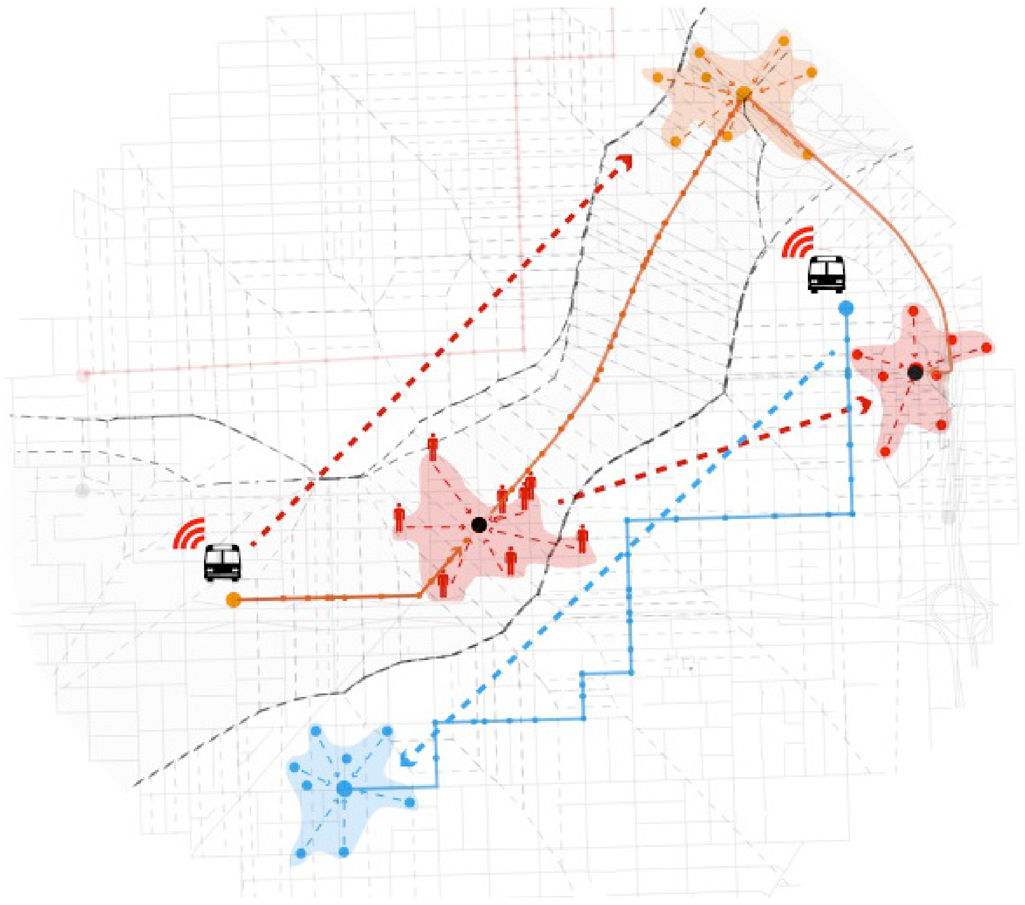
Finally, some background details to the different technologies and sensors used in this study is given for context into why they necessary in the implementation of the floating bus system.

2.1 State of the Art

The main issue with the current state of the art is that while proposals and prototypes of a dynamic bus service have been put forward, there is very little evidence that suggests whether a dynamic bus service would be more efficient or not over the traditional public bus transportation system.

2.1.1 Related Research

One particular relevant existing study found relating to dynamic bus routing is a prototype system put forward by Thomas Kearns et. al of Illinois Institute of Technology (IIT). They demonstrated a public bus routing system that navigated to dynamically generated locations based on demographic data.[13]



Example diagram from Thomas Kearns, Jordan Kanter and students Adam Weissert, Haidong Fei, and Li Gong (IIT 2013-2014)[13]

While Thomas Kearns et al. put forward an interesting prototype, there is little explanation about a practical implementation. From researching their work, there is little to be found as to why this dynamic approach is a practical solution, or to what problem they see it as an answer to. Their research project was a chosen finalist for the Louis Vuitton SPARK Award in 2014, and hence is worth researching the practical applications and benefits to the system. A video showing the prototype in action is available on the website "softsentience"[13].

Another notable related existing piece of work is the paper "A Real-Time Scheduling Method for a Variable-Route Bus in a Community" by Yan Fang et al.[14] This paper seeks to improve the efficiency of the bus system with regards to saving costs of running buses with a low level of customer demand. They propose a real-time scheduling method for assigning requests to variable routes. The paper looks at implementing a two-phase quick response approach and local optimization method to trade off computation time and solution quality for the routes. This paper shows quick response time and helps bridge the gap between static and dynamic routing, however the findings only consider one vehicle routing problem, rather than the multiple vehicle routing problem, leaving room for more research to be done in dynamic routing across a fleet of vehicles.

Electronics giant Philips is also researching the concept of redesigning how bus systems work with dynamically created stops, using existing infrastructure as bus stop markers[15]. The motivation for Philips to try and overhaul the bus system is to make a more flexible system, and to avoid experiences where passengers on near empty buses have to sit through the entirety of the route even when it's completely inefficient to service the route in full. Users submit their pickup and destination requests to the system via their smartphone, where they are allocated to a communal bus stop with other users looking to go to the same destination.

STOP BUS
Reimagining the bus service
flag anywhere, anytime
Bus towards Alexandra rd
105
App connects passengers in the same area who want to go to the same destination

Infographic of Philips' radical redesign - Stephen Dowling - BBC Future[16]

The main issue with Philips' proposal is the notion that implementing such a dynamic bus routing system would require a highly robust, overhauled IT architecture, and focuses too much on generating its stops at existing infrastructure to use as markers. A better solution could rely on road networks alone combined with constant communication with the passenger's smart device to signal pickups, meaning the only infrastructure to be relied upon is the road network itself.

Little information is publicly available about Dynamic Bus Systems studied so far, particularly with regards their implementation, so it's necessary to look at what other fields contribute to a dynamic bus routing system to better understand just how it should be implemented so that it can be found and discussed how it differs from the traditional bus system. A brief overview of a number of research areas of note when designing a public transport routing system with multiple vehicles and stochastic demand are as follows:

- Vehicle Routing Problem (VRP) and variants
- Ride Sharing
- Bus Allocation
- Seat Allocation

2.1.2 Vehicle Routing Problem

The Vehicle Routing Problem (VRP), first introduced by G. Dantzig and J. Ramser[4], is a generalization of the Travelling Salesman Problem (TSP). The VRP is defined as an undirected network, represented as a graph where

$$G = (V, E)$$

where

- V is vertex set $\{v_0, v_1, v_2, \ldots, , v_n\}$
- E is edge set $\{(v_i, v_j) : i \neq j, v_i, v_j \in V\}$

A depot is located at vertex v_0 and acts as the starting destination for the vehicle and as the destination or source of deliverables (goods, etc.). The remaining vertices correspond to customer locations which involve goods being collected or distributed to. Journeys are given a travel cost matrix D, where $D = \{d(v_i, v_j)\}$ is defined on E. Each vehicle in a fleet also has a homogenous load capacity Q, where each customer location v_i requires a load size q_i to be collected or delivered en route.

2.1.3 Ridesharing

Ridesharing is a model in which people share a ride, rather than generate a ride[11]. The concept of traditional ridesharing is defined by the Association for Commuter Transportation (ACT)[12] as pooling people from a common origin to a common destination. Ridesharing requires at a minimum two people involved, one driver and one or many passengers, each with their own individual start and end locations, within reason. The motivations for ridesharing are based on[14]:

- Emissions
- Fossil fuel use
- Efficiency
- Economic costs

Robert Geisberger et. al put forward a study into "Fast Detour Computation for Ride Sharing" as a way of showing that ride sharing services can be substantially improved using innovative route planning algorithms[14]. They generalize static algorithms for many-to-many routing approaches into a dynamic setting and develop additional pruning strategies. Their take on dynamic routing and allocation becomes relevant when assigning riders to floating buses.

2.1.4 Seat Allocation

The study of Seat allocation looks at ensuring capacity for transit vehicles is met as often as possible, and is neither exceeded or incomplete. This is done so to cater for as many passengers as possible per journey without needing to refuse any, or have any wasted space during journeys. It looks at optimizing journeys made by the number of people making it at once.

Yu Jiang et al. have a relevant study to the floating bus system in "Reliability-Based Transit Assignment for Congested Stochastic Transit Networks", which looks at risk-aversive stochastic transit assignment, where travel time, capacity and effects of congestion are all stochastic variables[15].

2.1.5 Bus Allocation

The area of Bus Allocation looks into efficient assignment of buses to different bus routes. This field of study seeks to minimize the number of wasted journeys per route due to overlapping buses and bus routes, and seeks to find the best way to manage a fleet of bus vehicles.

"The allocation of buses in heavily utilized networks with overlapping routes" by Anthony F. Han et. al looks at operating buses with overlapping routes and running at or close to capacity. A formulation of the problem is put forward which recognizes passenger route choice behavior, and seeks to minimize a function of passenger wait time and bus crowding with regards to the number of buses available with adequate capacity on each route to carry all passengers who would select it[16].

2.2 Technologies

Before embarking on creating the floating bus system, the appropriate nature of the application needed to be decided. Given the constantly connected nature of the floating bus service, it was decided early on to implement it as a RESTful web service application.

2.2.1 RESTful Web Services

RESTful web services are built to work best on the web. They are built using the Representational State Transfer (REST) architecture, where data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs), also seen as links on the web.[17]

The following principles of RESTful applications make it highly scalable and modifiable, and thus ideal for the floating bus application:

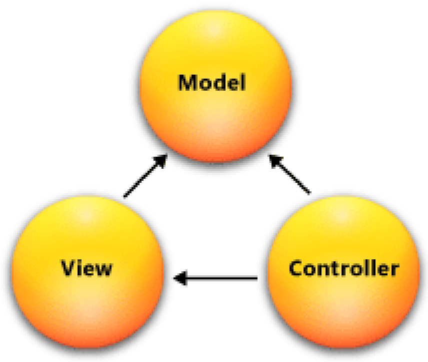
- Resource identification through URI:
    - Since resources are identified by URIs, different features and functionality of the floating bus system can be defined through different URI paths, and called upon when needed.
- Uniform Interface:

    - RESTful services utilize HTTP requests to communicate with clients. This allows for easy cross platform communication with the backend system, making it as easy for Android applications to communicate with the system as web browsers.
- Self-descriptive messages:
    - Content can be passed to and from the RESTful system in a variety of formats, including HTML, XML, JSON and more. This makes the transfer of information to and from the backend system much simpler, by the use of JSON objects.
- Stateful interactions through hyperlinks:
    - Every interaction with a resource is stateless, and therefore request messages are self-contained. Functionality of the backend system is tied to these hyperlinks, and is only executed when these hyperlinks are used.

Spring is an application framework built on Java used for implementing RESTful services. It uses the Model View Controller (MVC) architecture to allow for straight forward web service implementation.

The MVC design pattern is made up of three components:

- Model
    - Represents the data and implements the logic for the application’s data domain. Independent of controller and view.
- View
    - Displays the model data and communicates user actions to the controller. The View components are used to display the application's user interface (UI).
- Controller
    - Provides the model data to the view and processes actions on the back end of the application. Ultimately decides the view rendered to the UI.



MVC Design Pattern. Image provided by Microsoft: ASP.NET MVC Overview[9]

Communications between the Spring backend system and the Android application in this project are conducted using Apache's HTTPClient in the Android app. This allows the application to make necessary POST and GET HTTP Requests to the backend system to submit and retrieve information. JSON Objects are also sent between the two systems with needed information parsed on receiving on both sides.

Apache HTTPClient is a library that allows handling of HTTP requests. It has since been deprecated in current Android versions in lieu of various other libraries (eg: OkHttp), but is still usable and useful in this project. HTTP POST method requests send data to a server, with the data stored in the request body of the HTTP request[19]. The GET method is used to request data from a specified resource.

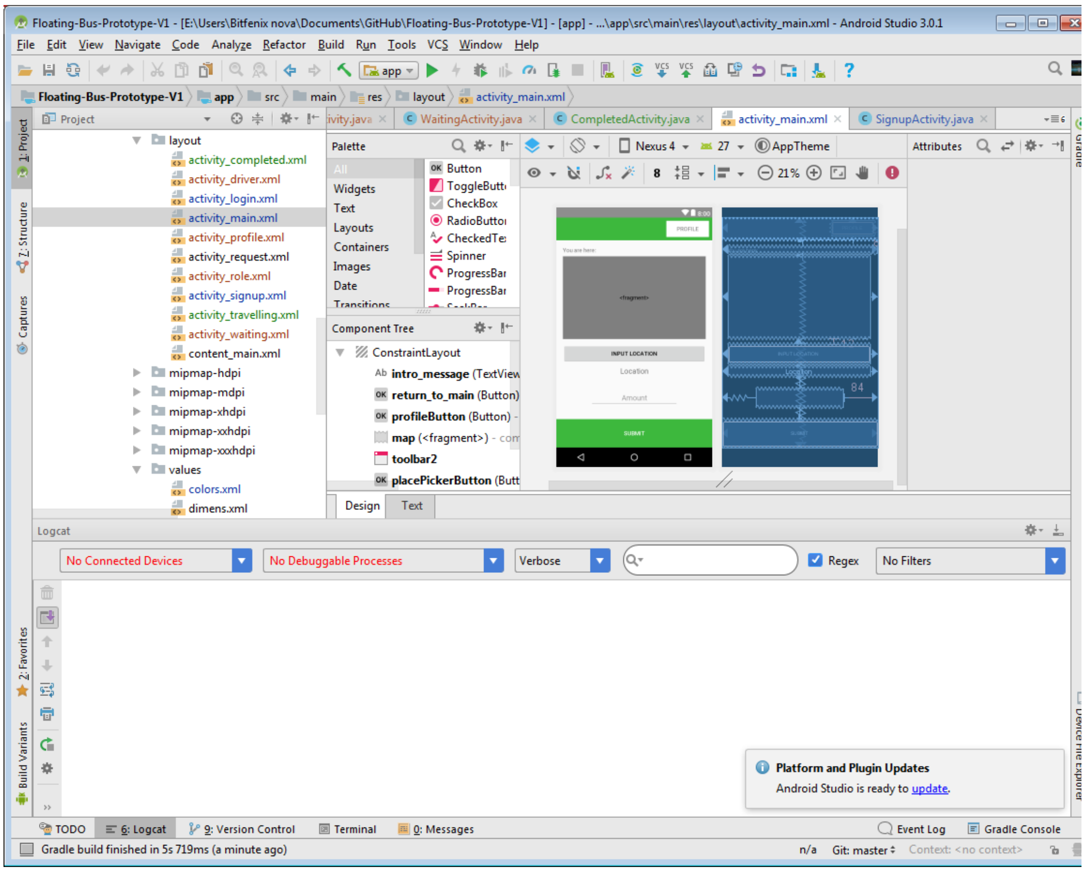
JavaScript Object Notation (JSON) is a widely used format for exchanging data between a client and a server[20]. JSON is a text based format, and is easily convertible to and from Javascript objects. It can also be easily parsed in Java.

The floating bus system was written as a RESTful web service due to the ability to use multiple different endpoints for executing different functions of the system. For example, adding a new request to the system is a simple matter of posting data to an "/addRequest" endpoint in JSON format. Once received, the functionality at this endpoint parses the JSON into a request object format and stores it to the database. This is explained later on, but is a good example of why the program was written this way, since it will be able to handle multiple different connections to devices at once like this.

2.2.2 Android Studio

Android is a widely used smartphone operating system developed by Google. Android applications are written in Java and are deployed as APK files to be installed on multiple different types of Android devices. Android studio is a tool suite provided by Google that allows applications to be developed for the Android platform.

Android was chosen for this project as a way of providing prospective users of the floating bus system a straightforward means of submitting their travel requests, and an efficient way of collecting their source locations automatically. Being built on Java and using Gradle as a dependency manager made it easy to add different dependencies as needed, such as Google Maps dependencies.


Floating-Bus-Prototype-V1 - [E:\Users\Bitfenix nova\Documents\GitHub\Floating-Bus-Prototype-V1] - [app] - ...\app\src\main\res\layout\activity_main.xml - Android Studio 3.0.1

Platform and Plugin Updates
Android Studio is ready to update.

Gradle build finished in 5s 719ms (a minute ago)

Android Studio Environment - Layout XML Editing

2.3 Google APIs

2.3.1 Google Directions API

The framework relied upon in this project is Google's Directions API. The API returns most efficient routes when calculating routes based on real road data, prioritizing travel time above all. Other factors prioritized are distance and number of turns required to reach the final destination. Google Directions allows for multiple origins and destinations along a route, allowing for deeper customization of the overall route to be calculated.[21]

2.3.2 Google Distance Matrix API

The Distance Matrix API is a service that provides travel distance, time and traffic information for a matrix of origins and destinations.[22] This is used in the floating bus system to determine various aspects about the route, which are particularly necessary when it comes to running simulations.

2.3.3 Google Maps JavaScript API

Google Maps JavaScript API is used for displaying map data on the front end system. Different visualisation libraries can be added to the map, and the JavaScript API can also provide directions itself based on origin and destinations provided.[23] In the floating bus implementation, the route followed is calculated by the Google Directions API on the backend of the system, and is drawn on the front end by the JavaScript API. One pitfall of the approach taken is that the ideal route is actually calculated twice like this, once on the backend for vehicle routing, storing and simulation purposes, and again on the frontend for display purposes. This means there can be a slight disconnect at times between the displayed route on the HTML control panel page (detailed later on), and with what the vehicle sees. This is verified via breadcrumbing the vehicle's trail to ensure they directions results match.

2.4 Databases

2.4.1 PostgreSQL

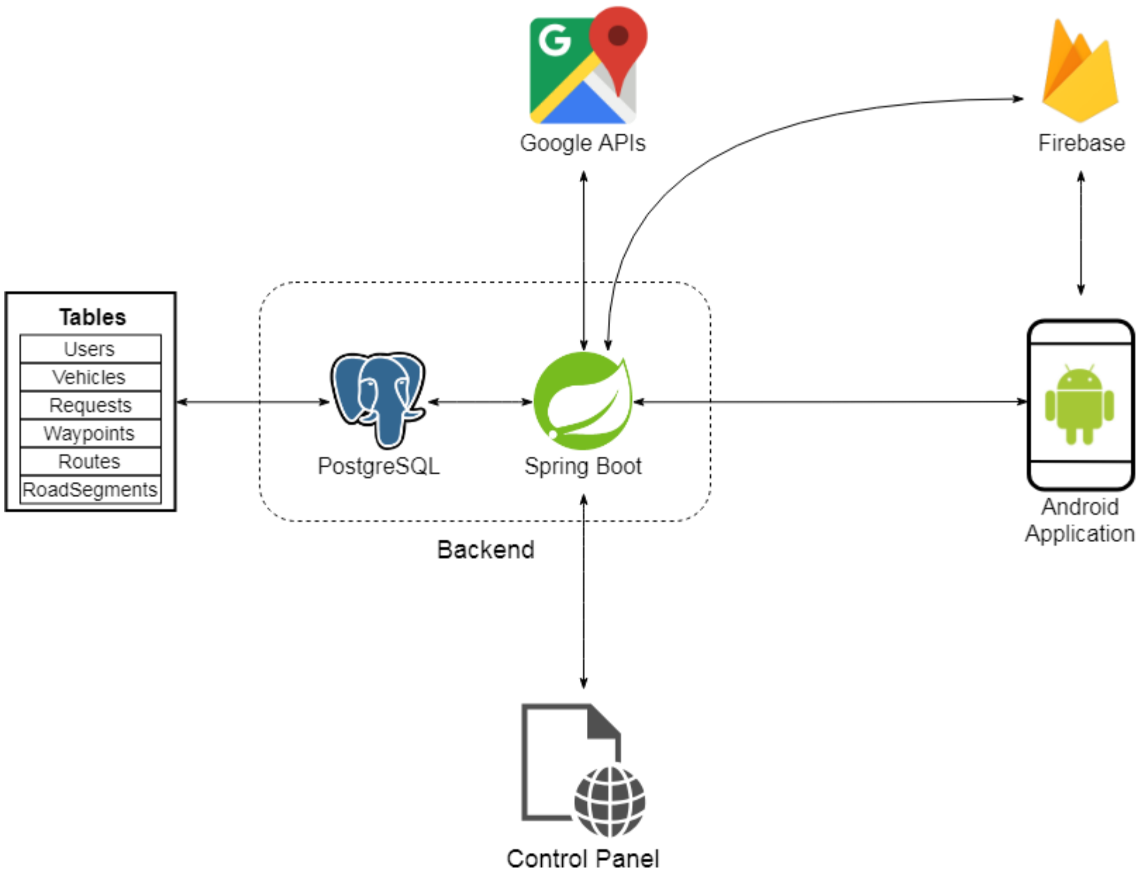
Postgres is an open source object-relational database management system. It presents a variety of different data types and integrity rules, and has the ability to be extended to include more (eg: PostGIS a spatial database extender for PostgreSQL[18]). PostgreSQL is easily integratable with Spring via Spring JPA.

2.4.2 Firebase

Firebase is Google's mobile application development platform, and is integrated into the floating bus android application for authentication purposes. Users register through Firebase with an email and password, which is stored in Firebase's records. More about Firebase's implementation and functionality are discussed later on.

Chapter 3 - Implementation

This section details how the floating bus system was implemented and how it functions. Each of part of the system is discussed and explained in detail to understand the role each individual part plays in the overall system. The logic and reasoning behind design choices is also outlined. Here is a high level look at the floating bus system at a whole:



To be able to build an application that relies on routing techniques so heavily, it's crucial to first break down the individual components that make up routes and vehicles into manageable entities. First, the concept of a route has to be defined as something for the vehicle to be able to follow.

3.1 Backend Components

To be able to discuss the functionality of the backend and routing process appropriately, the various entities that make up the system must first be outlined to understand their purpose when it comes to plotting the floating bus routes. Different actors and their relationships are defined to better understand how the various parts work together in the application.

3.1.1 Vehicles

Vehicles act as the most important actors in the floating bus system, since without them there isn't anything available to service the requests as they come in. Vehicles are unique entities set up by users who declare their intention to use the system as drivers. The components that define a vehicle in full are as follows:

- Active
    - Is this vehicle currently in service or not. A vehicle that is currently inactive rightfully will not be assigned any incoming requests.
- Capacity and Max Capacity
    - Each vehicle has an upper seating limit for the number of passengers it can accommodate, ie: its max capacity. It's capacity refers to its current capacity, which is used in calculating whether the vehicle can take another request on board based on its remaining available space.
- Current Latitude and Longitude
    - The vehicle's current world space coordinates. These are updated frequently via the corresponding driver's Android application, or are manipulated via running simulations.

3.1.2 Routes

Routes are where all the information about a vehicle's path are stored. Route objects are made up of the following:

- Vehicle ID
    - A reference to the vehicle assigned to this route.

- Encoded Polyline
    - This is where the path of the route itself is stored as an encoded string. The process of populating this field is done by feeding a start point, an endpoint, and all destination points in between to Google's Directions API, which returns the path in encoded string format.
- Distance
    - The distance in meters from the start to the end of the route in terms of road networks, rather than euclidean distance between points.
- Time to complete
    - The duration of a journey along the route expressed in seconds.
- Traffic
    - The duration of the journey along the route taking traffic congestion into account. These results are based on current and historical traffic data, also expressed in seconds. The traffic models recorded are:
        - Best guess: Most realistic estimate
        - Optimistic: Returns lighter traffic estimates
        - Pessimistic: Returns heavy traffic estimates

3.1.3 Road Segments

Road segments are the individual pieces of a decoded polyline of a route. Each road segment is made up of two points in coordinate space, a start and an end point. While a route also has a start and and end point, they also include all the individual start and end points required to link the two (ie: road segments). Road segments are the lowest level of the makeup of a route and consist of the following:

- Route ID
    - The relevant route the segment belongs to.
- Order In Route
    - The order of the road segments to be followed when executing the route.
- Start Latitude and Longitude, End Latitude and Longitude
    - The coordinate points for the line the road segment is comprised of.

- Visited
    - A boolean parameter to state whether the vehicle has left this road segment before. Most important when running simulations with overlapping lines (ie: U-Turns, crossroads, etc.). This is explained in detail later on.

3.1.4 Requests

The motivation for the floating bus was to be able to cater to varying requests that are received by the system in real-time and to be able to adapt accordingly. With this in mind, it was important to create uniquely identifiable requests made per user to be able to manage them and allocate them correctly. Request objects are made up of the following:

- User Email
    - Each user can have one active request at any given time. The email used to sign up for the client application is used as an identifier in this case.
- User ID
    - Relates the request object to the relevant user.
- Destination Latitude and Longitude
    - Each request made by the user includes an end point in world space, whose coordinates are used in plotting new waypoints or merging with pre-existing ones.
- Source Latitude and Longitude
    - When the user submits their request, they are also submitting their current world space coordinates to be used in gauging their appropriate pickup waypoint.
- Amount
    - The number of passengers this request is intended for. Necessary for ensuring the most appropriate vehicle for them is chosen.
- Completed
    - An indicator that the user's destination has been met and that their request is no longer relevant to the routing process.

- Pickup and Dropoff Waypoint ID
    - Relates the start and end points of the request to their relevant waypoint objects.

3.1.5 Waypoints

The floating bus system defines points on the route as waypoints, including the end point. In other words, the floating bus system can place more waypoints along a route to develop and influence the directions taken in between the start and end points. The start point isn't treated as a waypoint, but is simply defined by the vehicle's starting coordinates.

Waypoint objects are a custom built data type that signify destinations along a route. Waypoints cater for request pickups and drop-offs, in that there are always two waypoints created for each request. The waypoint object is made up of more than just a coordinate in world space, and includes the following aspects:

- Route ID:
    - What route object it belongs to. Route objects are separate entities stored in a separate table.
    - Waypoints have a many to one relationship with route objects, while route objects have a one to one relationship with vehicles.
- Latitude, Longitude
    - World space coordinates. When waypoint objects are created first, they are created for each individual request. When compiling the route object in full, waypoints are combined together into a single waypoint if they fall within a set radius of each other. Their coordinate values consist of an average of the waypoints merged.
- Order In Route
    - It is important for the waypoints to be approached in a sensible order for the routing system to make sense. For example, say a drop-off waypoint was closer in world space than the pickup waypoint for the same request. It is imperative to the functionality of the floating bus that it treats these two waypoints differently, and approaches them in

the correct order so it has the passenger first to be able to drop them off.

- Completed
    - A true/false value to signify if this waypoint is still factored into route calculations. As a vehicle comes into range of a waypoint, it is marked as completed.

Though two waypoints are created for each request, they are often merged with other existing waypoints from other requests into one new waypoint. Many requests can have the same waypoints representing their respective pickup and drop off waypoints.

3.1.6 Users

The floating bus application consists of two types of users, one managed by Firebase, Google's online platform used for registering users, and one managed locally by the backend system. Firebase handles authentication and secure use of the application, while the backend system stores information about the user, such as their email, role and location data. The local user object is made up of the following:

- Email
    - The user's unique email.
- User Role
    - The user's role in the floating bus system. Available roles can be self defined from the client application, and can be either 'Passenger' or 'Driver'.
- Current Latitude and Longitude
    - The user's current world space coordinates.
    - The privacy implications of storing user location data are discussed further on.

The user role plays a big part in how the system tracks vehicles and requests. The main functionality involved is tracking users with the 'Driver' role, since their world space coordinates are then used to represent their vehicle's position. After a user registers with the android application, they have the ability to change their role from the default 'Passenger' to 'Driver', where they also specify the vehicle's capacity. This allows users to sign up as drivers for the floating bus route themselves, similar to Uber's ability to become a driver. This feature however would be unwise to maintain in practical applications of the floating bus outside of this research project due to health and safety, and legal implications.

3.2 Bus Allocation

Once a request is received, a pair of waypoints are generated for the request, namely a pickup waypoint and a drop off waypoint. This is done to be able to compare the request source and destination with pre-existing waypoints in the system for each active vehicle.

The system first gathers a list from the database of all active vehicles. An active vehicle is one that is currently "on-duty" and available to cater for incoming requests. A vehicle is set to inactive manually once the driver decides to finish their services. Completing a route does not end the vehicle's activity status, meaning the vehicle is still available for new requests even after catering to all of the requests it had already been assigned.

The process of choosing the most appropriate bus involves looking at a number of aspects of each vehicle, including:

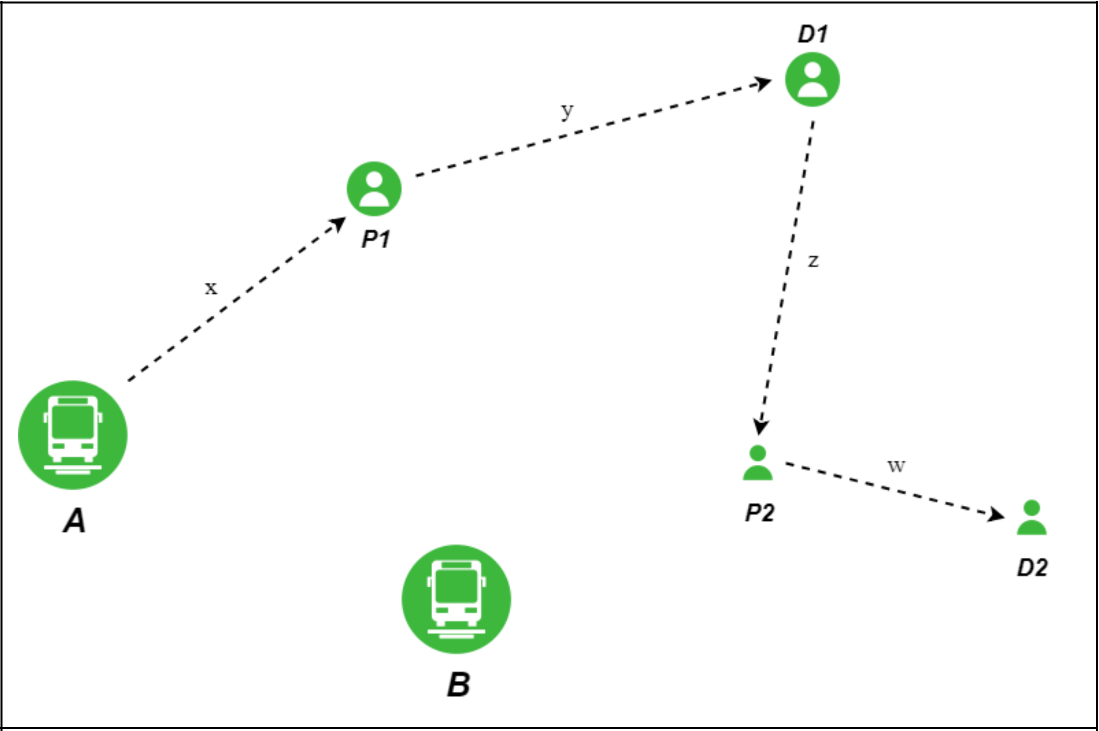
- Their current world space coordinates
- Their predefined waypoints on their assigned route
- Their capacity throughout

When allocating a new request to an available vehicle, there are a number of different aspects of the system that need to be taken into account to have it running as efficiently as possible.

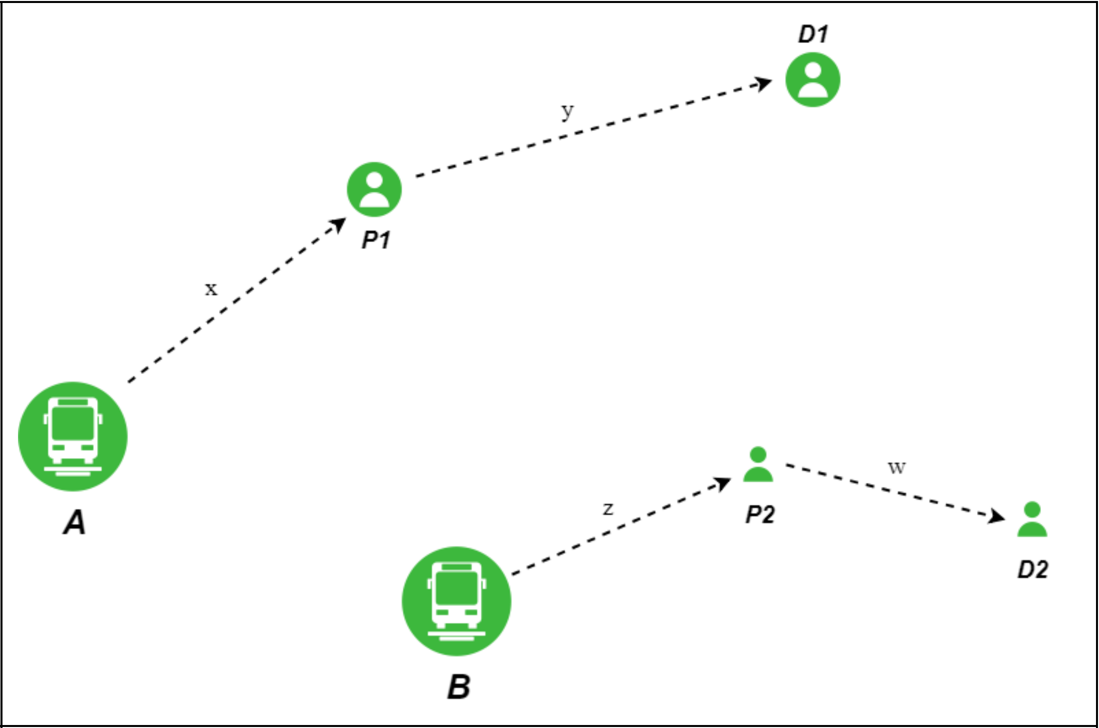
- Reducing the distance needed to be travelled to accommodate the request.
    - To achieve this, a request needs to be able to align to the nearest, most convenient waypoints in pre-existing routes, or with a nearby free vehicle. This is done to save one vehicle needing to travel great distances alone to meet all demands by itself, preferring that the load is shared out as much as possible with the other vehicles in the system. If the new request waypoints already align with a vehicle's

pre-existing waypoints however, without disrupting the route's capacity allowances, then it will be allocated there. In short, distance travelled is minimized by utilizing multiple vehicles. This can be seen illustrated below where

- A, B = Active Vehicles
- P1,P2 = Separate pickup request waypoints
- D1,D2 = Corresponding drop off waypoints

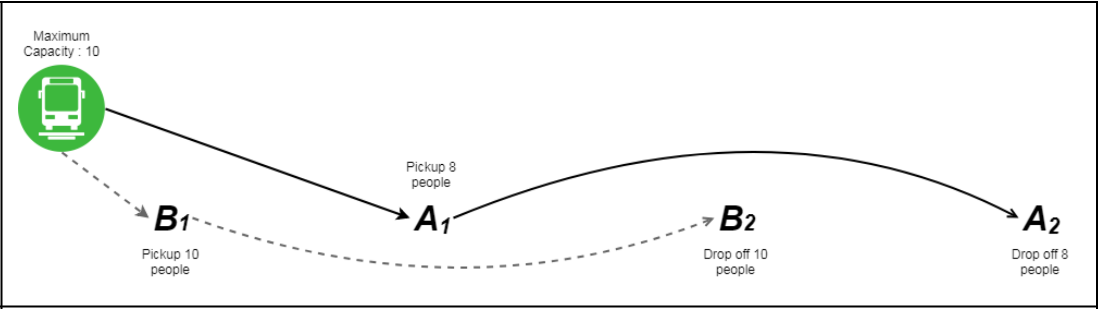


Undesired situation
Total Distance Travelled: (x+y+z+w)



Desired situation
Total Distance Travelled: (x+y) + (z+w)
Where
z = (B -> P2) < z = (D1 -> P2)

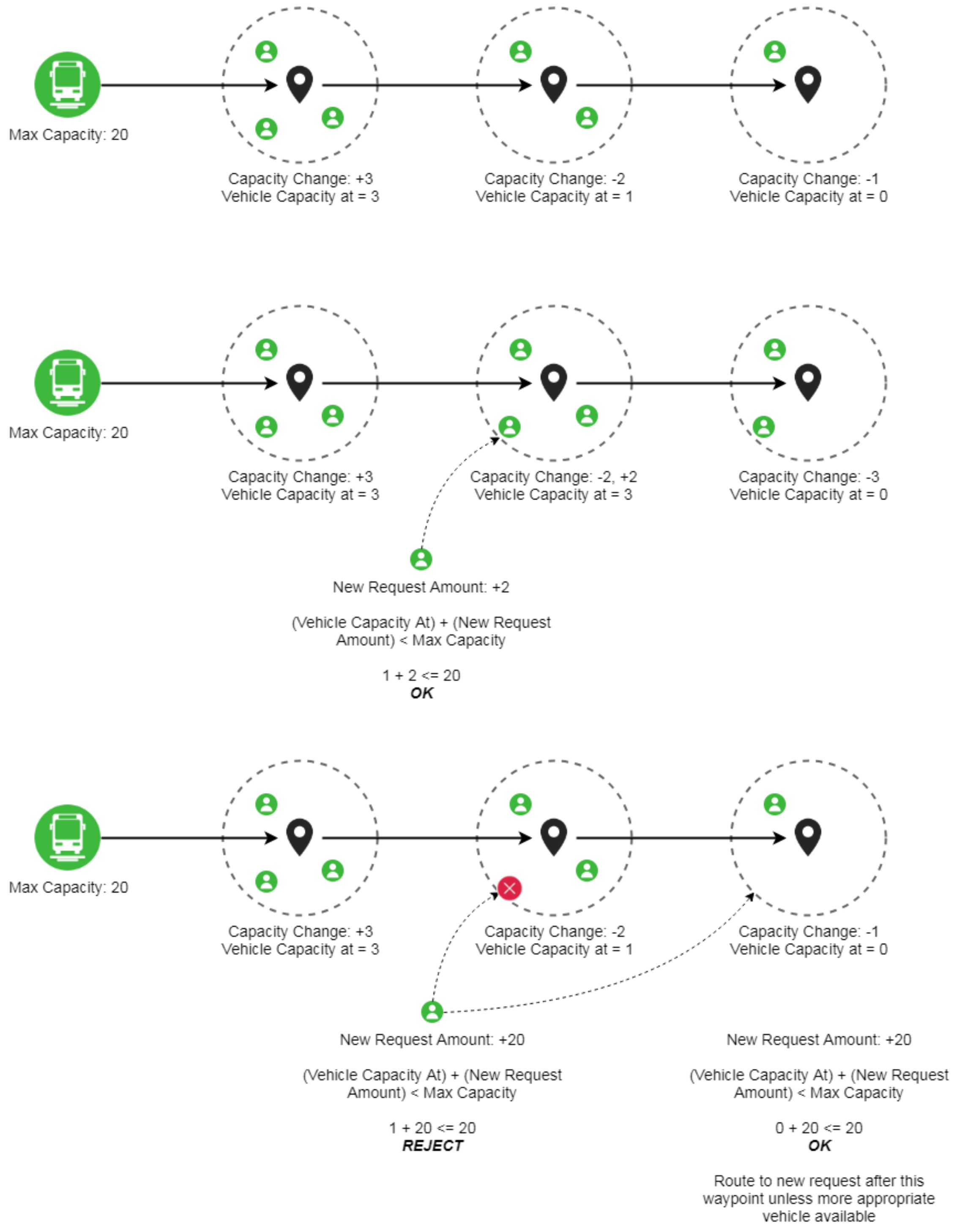
The main issue identified with assigning new requests to a vehicle is the notion of pushing out a request, or rather, invalidating either the new or an existing one due to unpredicted changes to the vehicle's capacity. The issue is illustrated below:



- Vehicle of max capacity 10.
- 'A' is a pre-existing request for the vehicle, from A1 to A2 for 8 people.
- 'B' is a new request, positioned closer to the vehicle, from B1 to B2, for 10 people.
- Who is serviced by this vehicle? How should both requests be accomodated?

To avoid this, when looking for the appropriate vehicle to assign to a request, the pickup and drop off waypoints for the request are generated and compared with the pre-existing routes of each vehicle. The closest pre-existing waypoints are then looked at and the nearest ones to the new request's pickup and drop off are chosen as the most ideal waypoints for this vehicle. The relevant drop off waypoint must always be equal or after the pickup waypoint's order in the route. This is because waypoints can appear in a route multiple times for different reasons, hence the capacity can vary for the same waypoint. The waypoints are also checked for their capacity allowance to make sure the incoming requests are eligible for this vehicle and won't exceed its maximum seating limit.

Once all of the vehicles with pre-existing routes are reviewed and the pickup and drop off waypoints of the most suitable vehicle have been identified, the system then looks at the vehicle without any predefined routes. These vehicle may have just entered the system or have completed their routes so far and are idle and waiting for new requests to be allocated to them. Since the aim is to get as many vehicles in the system covering as many requests to keep overall distance travelled low, the free vehicles are inspected for their proximity to the request pickup waypoint. If one of the free vehicles is better suited to adhere to this new request, it will be chosen as the most ideal candidate.



3.3 Routing

A route along a road network in it's most basic form is made up of a series of road segments, each with their own start and end points indicating when one segment ends and another begins. Overall, a route itself has a start and an end point, and contains a sequence of road segments to be followed to move between the two points.

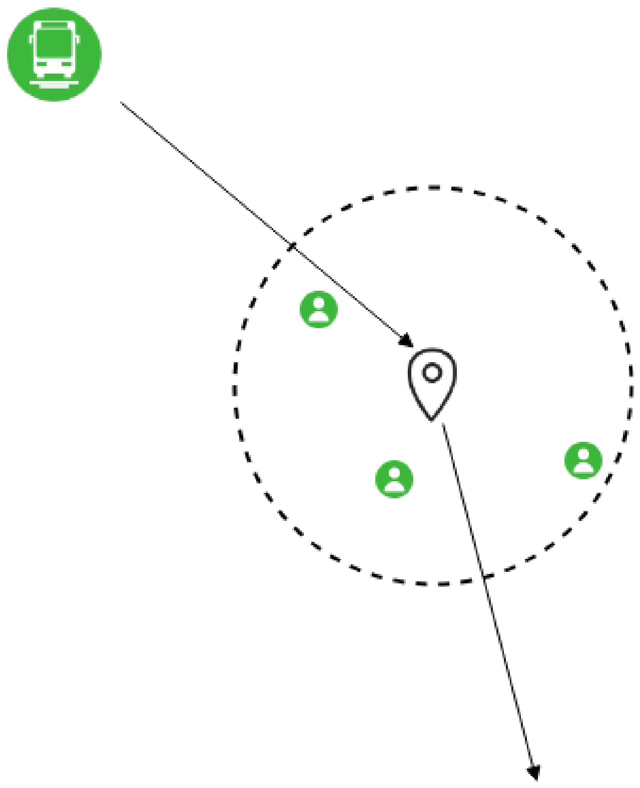
It’s important to note before explaining how the system sets the order of the waypoints in the route, that this does not claim to be the most efficient system available for routing a fleet of vehicles with multiple mid route destinations, but it is a logical routing system. The routing demonstrated is a practical implementation of what happens with a routing system that adheres to real time, stochastic requests where the waypoints have different roles (ie: pickup/drop off). Another reason is because road distance between floating bus stops is not factored when plotting the route, as the euclidean distance of these stops from each other is used instead. Limitations encountered while designing the system are discussed further on, such as encountering rate limits on Google’s Distance Matrix and Directions APIs.

After the most appropriate vehicle for the new request is found, the route is rebuilt using the new request on top of the old ones. In other words, the route is recalculated from all of the requests that have been assigned to this vehicle. This is done due to the way that waypoints are merged in the floating bus system, since new requests influence the position of a waypoint.

3.3.1 Floating Bus Stops

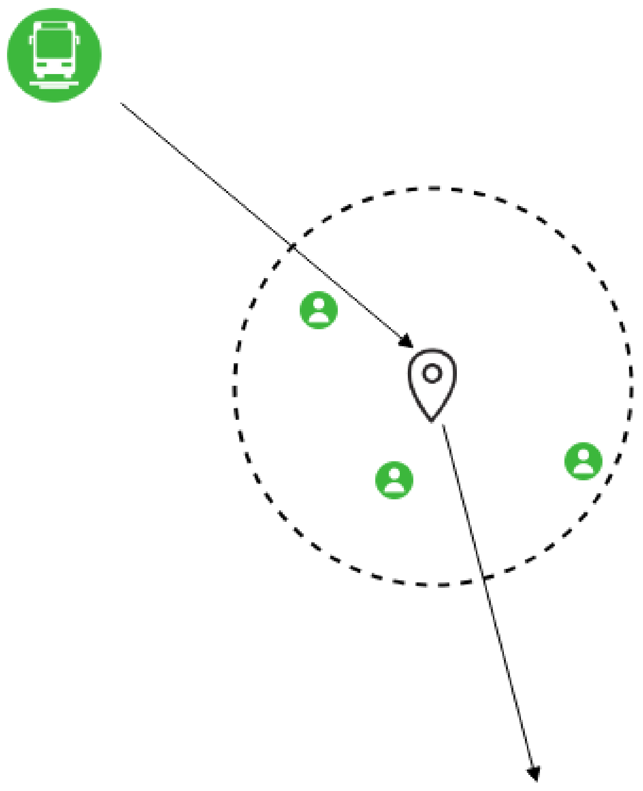
In the current system of public transportation, vehicles are assigned multiple stops along their fixed route. These stops have been carefully chosen and planned in advance by the government to be built in locations that best accommodate the passengers intending to avail of the service. These bus stops are always in the same place, and people become familiar with them by using them daily. These bus stops are also intended to cater to multiple passengers. However, they are not always the most convenient solution for passengers, and can be in awkward places, or some distance away from where they are.

The floating bus system presents a flexible bus stop option in the way the waypoints are used. Since the floating bus system targets multiple different requests at once, it would be far too inefficient to stop for each individual request. A better solution is to create new waypoint objects that are an amalgamation of multiple pick up and drop off requests, cutting down the need to repeatedly stop the journey for a series of nearby points. The reasoning behind the need to merge waypoints is illustrated below:



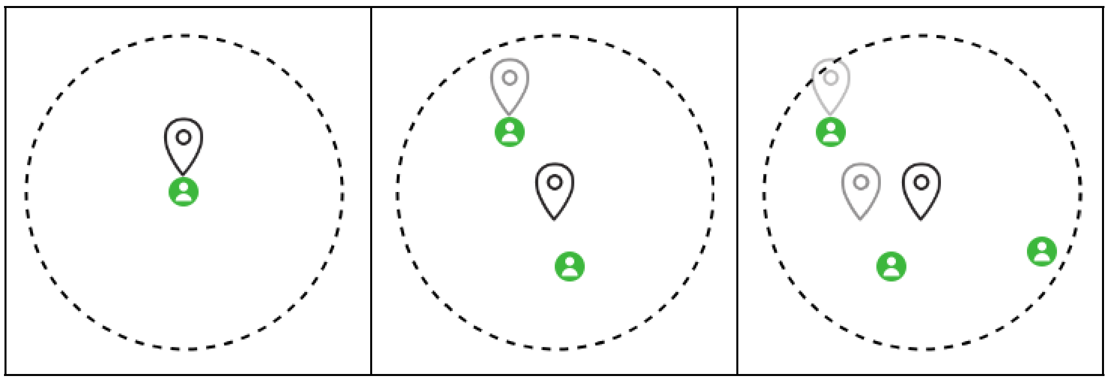
-Without Merging Request Waypoints

As previously stated, when a new request is added to the system, a pair of waypoints are generated, one for pickup and one for drop off. From the above, we can see that the floating bus servicing these requests has to make three stops in quick succession. The bus also has to redirect itself between each of the points multiple times to accommodate each of them. This leads to increased overall journey time due to additional stoppages, and increases the distance travelled due to the bus taking every turn along the way to meet the requests.



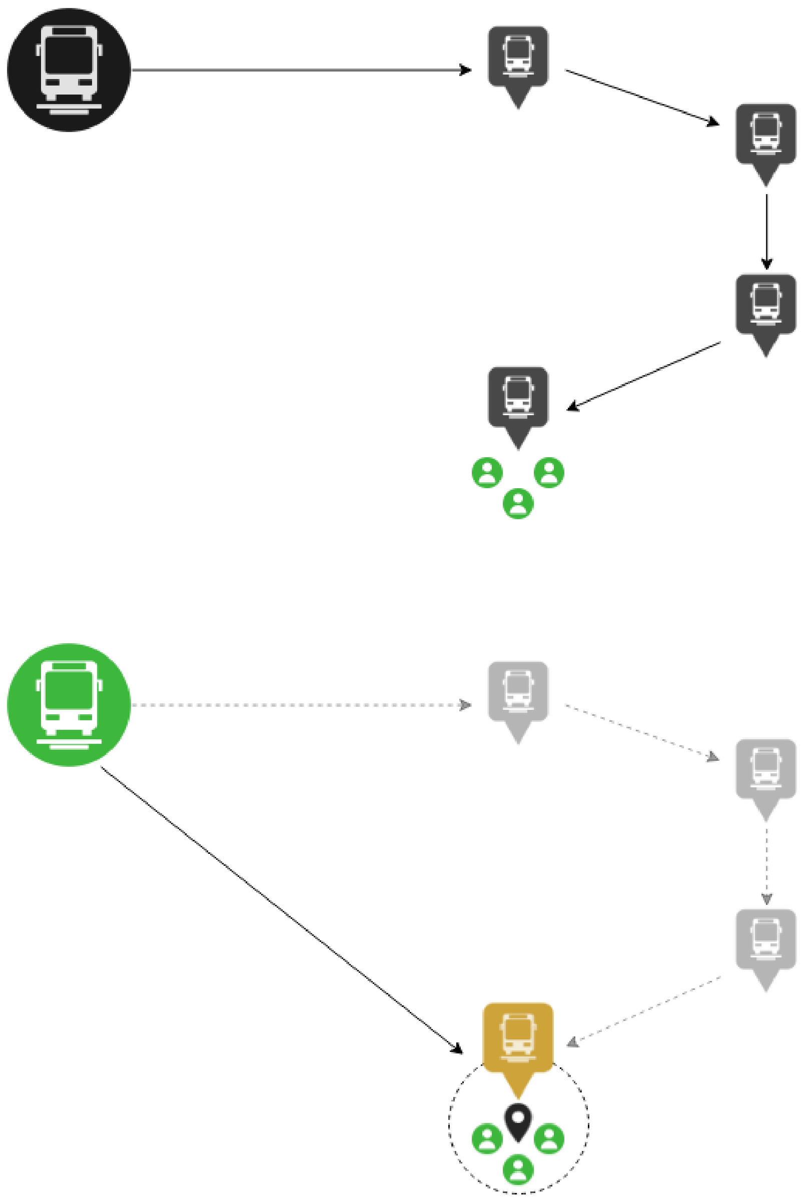
-Merging Request Waypoints

Here is the current approach employed by the floating bus system, where the waypoints created are merged into one within a certain radius. We can see that instead of making three unnecessary stops, the floating bus now only has to make one. This hugely cuts down on time spent accommodating each request in an area individually, especially as the number of request waypoints grows. The bus route is also set according to this merged waypoint, instead of the three individuals, cutting down on travel distance in the overall route. The steps to merge waypoints into one floating bus stop look like this:



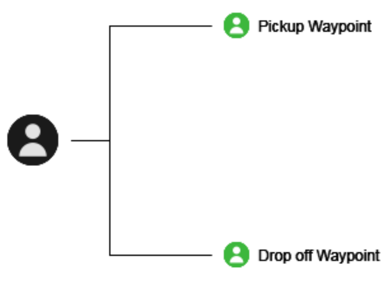
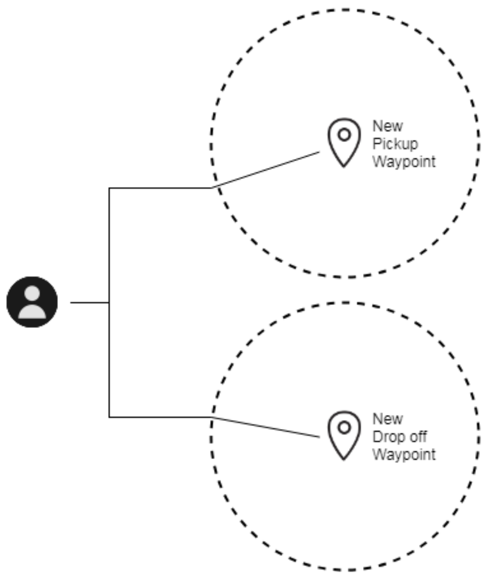
First request comes in, no need to merge.	Second request comes in, merged waypoint set to average of latitude and longitude coordinates of the two points.	Third request comes in, merged waypoint set to average of latitude and longitude coordinates of the three points.

Since floating bus stops are generated based off user demand, routes can be plot with this in mind and can reduce unnecessary wasteful detour on its journey, similar to how a static bus may have no demand but it still needs to service each stop in anticipation of demand. The overall benefit to knowing the demand for these floating bus stops in advance is illustrated below:



To merge the waypoints, the system identifies one to start at, and then using the Euclidean distance formula to detect if any other waypoints fall within the set radius of the system. The merge radius is set to *0.008983d/2,* which represents half of a kilometer. The value *0.008983d* representing one kilometer as a double is calculated using 1 degree of latitude of a line of longitude, which is represented as *360/40,075km* (the circumference of the earth). The merge radius number is changed when emulating static bus routes in the system, which will be discussed in the simulations section.

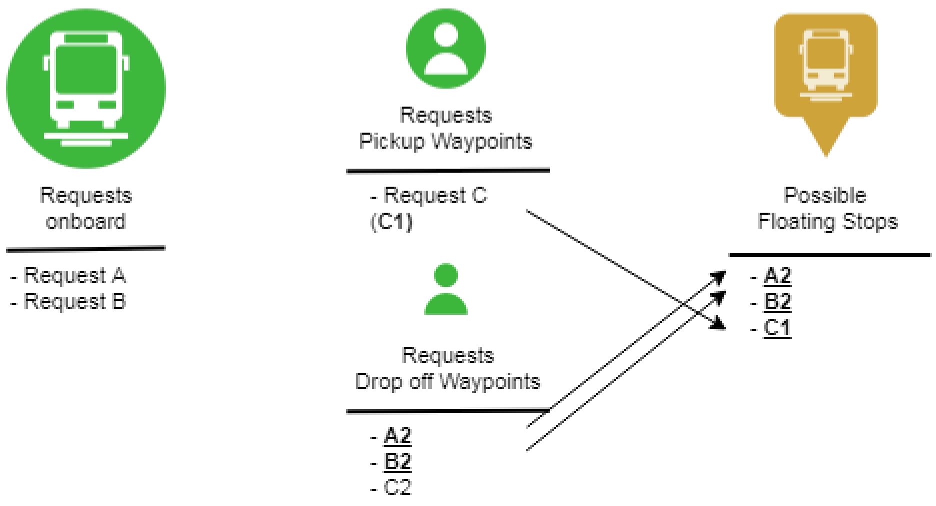
Once the waypoints are merged together, the requests that they affect need to be updated to reflect their new pickup and drop off waypoints, since their previous ones no longer exist and will not be catered for. To do this, each request allocated to the vehicle for pickup, or that is onboard and awaiting drop off, is iterated through and their previous waypoints are gotten and compared to the new ones. Based on the euclidean distance of the nearest of the merged waypoints for the request's pickup or drop off are then assigned to the request.

	
Waypoints generated as request is received.	Request updated to recognize newly merged waypoints as its pickup and drop off locations.

3.3.2 Plotting The Route

Now that we have the all of the destination waypoints for the route merged or created, the running order of the waypoints needs to be set in a logical and efficient manner. This means that the system needs to be able to differentiate between the different roles played by each waypoint for each request.

First, the system needs to ensure that the vehicle either has passengers or can get them first, before considering any waypoints used for drop offs. To do this, two lists of waypoints are created, allPickupWPs and allDropoffWPs. The lists are compiled as follows:



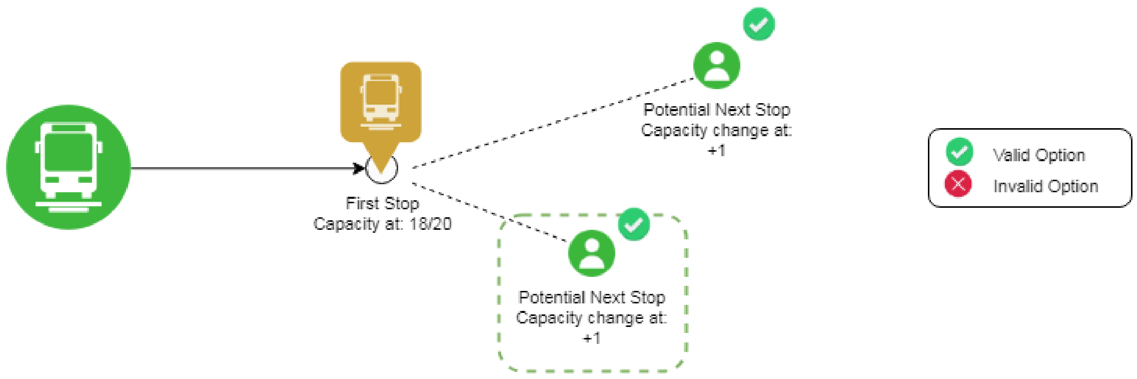
- The lists are compiled based on the requests allocated to the vehicle, their assigned waypoints in particular. As previously stated, each request has two waypoints, a pickup and a drop off waypoint, and both are assigned to each request after the merging process.
- The system first looks at all of the uncompleted requests tied to the vehicle. The requests are checked for their isPickedUp boolean parameter, which if true means that the requests are currently being serviced by the vehicle. This means that their pickup waypoints no longer apply when routing, and their drop off waypoints are a valid option for the next floating bus stop to be made. If they are not picked up, then their pick up waypoints are still valid

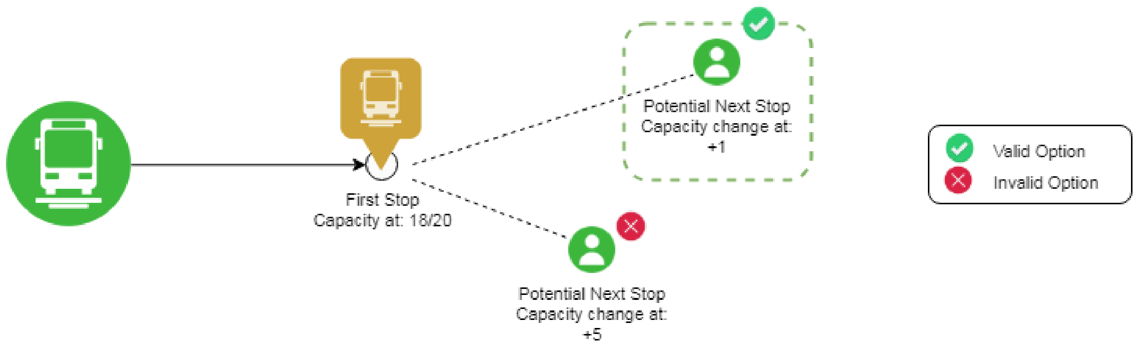
and their drop off waypoints are ignored for now. Each list is populated based on the relevance of each request's pickup or drop off waypoints.

- Once each list has been compiled, we amalgamate them into one list, allPossibleChoices, since we now have a set of valid waypoints that the bus can service appropriately. Before beginning the routing procedure for all of the waypoints, the first waypoint needs to be chosen from the list of possible choices, based on its proximity to the floating bus vehicle and it's adherence to the vehicle capacity limits (ie: going to a pickup request won't exceed the max seating limit). Since the route is calculated fresh with each new request assigned, the first waypoint also represents the "next" waypoint for a vehicle that's already in transit. The steps involved in choosing the first waypoint in the route is as follows:
    - The possible waypoints list is iterated through and the total capacity change at each is calculated from the requests tied to each. This means that for each waypoint, the requests that have this particular waypoint as a pickup waypoint and the requests that have it as a drop off waypoint are used in calculating the net capacity change.
    - Next, the Euclidean distance from the vehicle's current position to the waypoint is calculated. This is done rather than using Google's Distance Matrix API we make a request for the road network distance between the vehicle's current position and the current waypoints coordinate position. This is due to a number of reasons:
        - Distance calculations can take a longer time to complete.
        - To avoid meeting the Distance Matrix API rate limit too quickly. For example, if one vehicle had five potential waypoints to compare road distances, that would be 5/2,500 requests alone. Then say the number of vehicles doing the same increases, and the request is made of the API each time a floating bus request is received, the limit would be very quickly reached and the system would be rendered inoperable without API access.
    - If the distance to the waypoint is shorter than the previously accepted closest waypoint, the new waypoint is used instead as the leading candidate. Once the overall closest waypoint has been found and set

as the first waypoint in the route, it is removed from the list of remaining possible waypoint choices.

- Before deciding on the next waypoint in the running order of the route, the system checks if the first waypoint chosen acts as a pickup waypoint. It does this by checking with all the requests for the vehicle for their assigned pickup waypoint. Since multiple different requests can have the same pickup but different drop off waypoints, their drop off waypoints are added to the possible choices list as new potential candidates to appear next in the running order. This process is repeated for each pickup waypoint that is added to the route, so that no request's drop off stop appears before the user has been collected.
- The remainder of the route is calculated based on capacity and Euclidean distance from the previous waypoint accepted into the route. This is done repeatedly until the allPossibleChoices list is empty (ie: all the waypoints in the route have been catered for).





- As a side effect from adding each request's drop off waypoint as their pickup is added to the route, there are multiple duplicate waypoints in the route. Hence the route is then cleaned up of all duplicates found, and the requests are then updated to reflect their waypoints in terms of the cleaned up list.
- For the final step in calculating the route, the Waypoint, Request, Route and Vehicle objects are saved to the database. A new route object is created and related to the current vehicle. The waypoints

capacity value is updated to reflect the expected capacity at each stop based on the pickups and drop offs they cater for. The waypoints are then also related to the route and vehicle objects before being stored to the database. The requests are stored too after assigning them their waypoint IDs (since the waypoint objects themselves aren't stored, their IDs are used as a foreign key to relate to the requests table) and assigned vehicle ID. Finally, the vehicle is assigned it's newly created route's ID.

3.3.3 Obtaining Route Information

After obtaining the ideal running order of the floating bus stops for the most appropriate vehicle, the system needs to obtain route information about how to service these stops in terms of actual road network data. The information gathered and stored about the route is the road network graph between the vehicle's current position as the origin, it's final destination floating stop and all floating bus stops in between. More information about this path is also stored, such as the distance it takes from start to finish, the predicted travel time and travel times with different traffic estimates. All of this information is obtained by communicating with two main Google APIs, the Directions API and the Distance Matrix API. Another key part of this step is collecting road segments as objects to store in the database, to relate with the corresponding waypoints created.

To be able to use Google's APIs, a unique API key is needed. This is generated by signing up for a Google developer account and selecting the requested APIs for use. Google Maps APIs share the same API key, so only one needs to be generated. Before being able to utilize any of these API services in the backend system, a GeoApiContext object is created, which contains the API key.

The APIs are accessed like so:

1. From the previously described routing step, we gather the waypoints created in the order they were set. Using their generated latitude and longitude values, a new list of Google's LatLng objects were created. These will be used for the list of destinations along the route that represent the generated floating bus stops.

2. A departure time is set to current date time, since the route information needed is based on the current conditions of the overall route. This affects the results returned for directions and traffic data, as this is based on historical data as well as current.
3. The Distance Matrix API is then queried for route traffic information. A new DistanceMatrixAPIRequest object is created with the context object previously created. A response DistanceMatrix object with the route details is returned for each request made. A request takes the following parameters:
    - Origin(s): Position of one or more starting locations. Can be an array of Strings, Google LatLngs or Google Place objects. The vehicle’s current position LatLng is used as the origin for the route calculations.
    - Destination(s): Desired location(s) to be passed in the overall route. Can also be an array of Strings, Google LatLngs or Google Place objects. The floating bus stops are used as an array of LatLngs for route calculations.
    - Travel Mode: Specifies the mode of transportation to be routed. The different modes include: Driving, Transit, Walking, Bicycling. The vehicle default is set to Driving mode, but can be specified as Transit when creating a vehicle via a JSON file. Unfortunately Transit mode doesn’t provide routing information with bus corridors included, hence standard Driving mode is used as default.
    - Transit Modes: Provides additional routing specifications for more than one type of routing if Travel mode is set to Transit. Transit Modes include Bus, Rail, Subway, Train and Tram.
    - Traffic Model: Specifies the route’s congestion when calculating route time for a vehicle in Driving Travel mode. Results returned are based off of historical conditions and live traffic conditions. The three options available are: Best Guess (best estimate of travel time), Pessimistic (heavy traffic conditions, normally exceeding actual travel time) and Optimistic (lightest traffic conditions, normally shorter than the actual travel time).

- Transit Routing Preference: An additional parameter for Transit vehicles. Specified to “Fewer Transfers” for the floating bus system.
- Departure Time: Specifies when the route is intended to be taken. This becomes particularly important to have when calculating traffic time to gather live traffic results.

4. A unique DistanceMatrixAPIRequest is made for each of the three traffic modes at the same time to gather full route data. A DistanceMatrixAPIRequest object returns route information in a rows, which are arrays of DistanceMatrixResponseRow objects corresponding to each origin. Rows contain elements, which contain the information needed such as duration, distance and duration in traffic for each origin/destination pair. To get the total route information, the elements in each row are iterated through and compiled into a single value for each matrix for the overall route. In the end, we have total distance, duration and duration in traffic for the overall route. The traffic recordings are stored separately in three different variables, and stored to depending on the relevant Traffic model.
5. After getting the route’s numerical details, the system needs get a plot of the route in a way that can be stored. To do this, Google Directions API is now queried using the same origin and floating bus stops as were used in the Distance Matrix calculations. A DirectionsResult object is returned from a query to the API, which takes the following parameters:
   - Origin: A single starting location from which to calculate directions. The origin can be specified as a String, Google LatLng or Google Place object. Since the floating bus system works off of latitude/longitude values, the vehicle’s current position is converted to a LatLng and is used as the starting origin for the calculated directions.
   - Waypoints: An array of mid route LatLngs that alter the way the directions of the route are returned. The floating bus stops are used here to best specify the ideal route to be returned to accommodate them.

    - Destination: The final stopping position on the route. The last floating bus stop in the route is used as the final destination for the current route.
6. The Directions API returns a DirectionsResult with an array of routes of type DirectionsRoute. Since we only have one origin and destination per request, our only route is stored at index zero. The DirectionsRoute object returns a graph line of the route as an encoded polyline made up of all the LatLng coordinates on the route for each section of the road.
7. Now that the system has collected the information about the route, it now stores the route as an entity in the database. Any previous routes tied to this vehicle are overwritten with the new route object. A Route object consists of the following:
    - Encoded Polyline (String)
    - Distance Metres (long)
    - Time to Complete (long)
    - Traffic Time Best Guess/Optimistic/Pessimistic (long)
8. Next, the database is populated with all of the road segments that make up the route from the directions received from the Directions API. To do this, the encoded polyline is decoded into a list of LatLng values that represent each road segment's start point. By using each LatLng as a start point, we can use the following LatLng as its end point to build a road segment object. This is done for every LatLng in the route to get a full representation of the optimal route of the road network returned from the Directions API in its most basic form.
9. As a final step, the waypoints that were used to plot the route are related to the road segments that make up the route. This is done to relate pickup locations to their points on the route that best fit as a place for them to act as a floating bus stop. For example, if a request is made from the center of a large park, the floating bus stop will be mapped to the nearest road segment to where the request was made from. The road segment is broken up into three different points of contact to be compared with for their distance to the floating bus waypoint, namely its start coordinates, midpoint coordinates and

its end coordinates. This is done for each road segment until the closest point out of the road network is found. Once found, the floating bus stop's position is updated to the midpoint coordinates of the relevant road segment. The waypoints are saved to the database.

3.4 Backend System

This section gives a technical overview of how the above was implemented in context of the tools and design architecture used. A Java application built using Spring's web application framework, Spring Boot, is used to manage the floating bus system. A local PostgreSQL database is

3.4.1 Controllers

As stated above, the Controller in the MVC model is used to process logic, render views and add to the model. In the floating bus system there exist multiple different controllers, each serving a specific function. These are as follows:

- Route Controller
    - The core functionality of the floating bus routing and passenger assignment is carried out.
- Registration Controller
    - Aside from user registration and authentication being handled by Firebase, a copy of each user is registered and stored in the backend system. These endpoints are reached at the same time as the user is registered in Firebase.
- Vehicle Controller
    - New vehicles are added to the system and world space, specifying their current coordinates and maximum capacity.
- Simulation Controller
    - Extrapolates vehicles along their defined polyline routes and emulates the operations carried out on each vehicle as they pass waypoints and complete their routes.
- Front End Controller

    - Renders HTML views required to view simulations and vehicle progress. Adds required data from the backend to the model for use in Thymeleaf's template system.
- Ajax Controller
    - Used to control backend operations and behaviour from the front end view. Enables non-disruptive use over the floating bus control panel.

3.4.2 Scheduled Components

There are a number of scheduled components running in the background of the floating bus system that perform a number of important functions for both practical use with the system and for testing/simulation purposes. These scheduled components run specified functions on a loop with a fixed delay once the application is started.

The scheduled components in the floating bus system play an important role when it comes to both running simulations and monitoring vehicle progress. The various scheduled components in the system are the following:

- Vehicle Monitor
    - This component is used to track vehicle progress along a route, in particular paying attention to whether waypoints and requests along the route have been completed by their road segment. The functions performed are as follows:
    - **WaypointBehaviour:** Finds all requests by vehicle and checks against their pickup and drop off waypoints to see if their corresponding road segments have been completed. Once true, the request is set to picked up or completed as appropriate, and the vehicle's capacity is altered to reflect the request's pickup or drop off amount.
    - **MonitorRouteCompletion:** Checks vehicle's position on the route polyline by its individual road segments. If the vehicle is found on the segment, it sets the segment's visited parameter to true and saves it back to the database.

    - The Vehicle Monitor only operates on active vehicles in the system, so dormant vehicles aren't factored into these functions.
- Simulation Controller
    - The simulation controller has two ways of running simulations, either on all vehicles at once, or one at a time. This is where the functionality for extrapolating test vehicles along a route is kept. The simulation functionality is nearly identical for simulating all at once and simulating one at a time, the main different being how one iterates through every vehicle and route. The way the simulations are run is discussed later on.

3.4.9 Simulations

The Simulation Controller is a scheduled component that is set to run every one thousand milliseconds (ie: every second). Simulations can be triggered to run from the front end control panel, and can be run on all vehicles simultaneously or on a per vehicle basis. The steps involved are as follows for per vehicle simulations:

1. The specified vehicle is obtained via the id provided. Its current coordinates are used to create a LatLng object representing its position in world space.
2. The vehicle's associated route id is used to pull the route object from the database, containing the necessary information needed to run the simulation, for instance:
    - EncodedPolyline : String
    - DistanceMetres : long
    - TimeToComplete: long

   The base simulation speed is then calculated using the DistanceMetres and TimeToComplete, which is stored in seconds, in the Distance Speed Time Formula:

$$D = S*T$$

where:

- D = Distance Travelled (metres)
- S = Speed (metres/second)
- T = Time (seconds)

For the purposes of calculating the simulation speed, the formula is used as so:

$$S = D / T$$

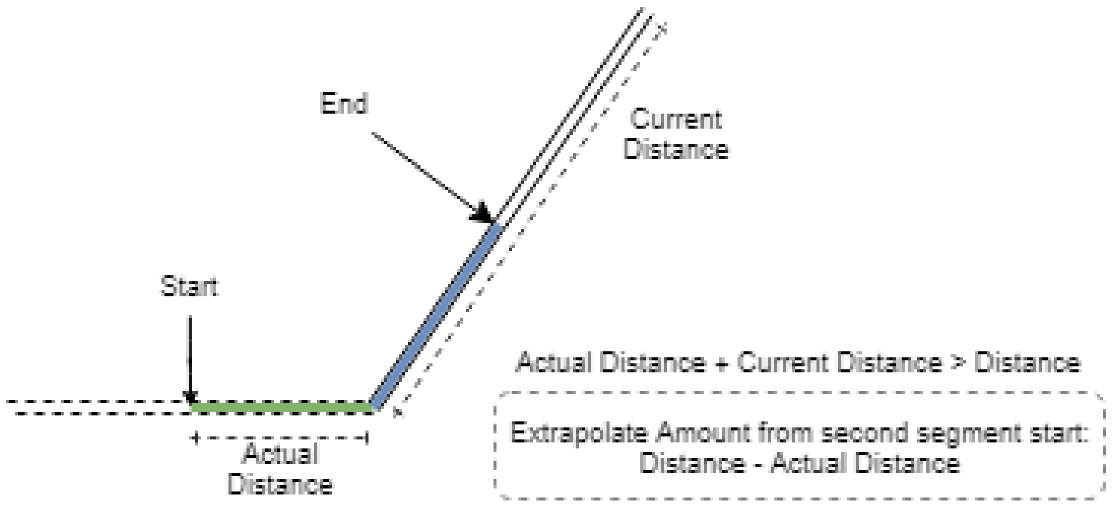
Giving the vehicle's average speed for the route.

3. The encoded polyline stored in the route is decoded from a string into a list of LatLng objects. This is done via Google Maps Model library. Once decoded, we pass the list of LatLngs, the vehicle's position, the average speed and the floating bus route object into the extrapolate function.
4. The extrapolate function does exactly what we expect, and that is to extrapolate the vehicle's position along the given polyline (ie: road network, not as the bird flies). The steps taken to extrapolate along the line are as follows:
    - First check if the vehicle is on the line at all. If not, move it to the start of the route.
        - Note: This is an acceptable measure to take in running simulations, since we expect it to follow the route exactly. In real life, it can't always be expected for a vehicle to follow a set route exactly, but as long as they reach the road segments that contain the route's set waypoints the route can still be completed.
    - Next check is aimed towards the meeting the end of the route, by checking if the vehicle is within a specified radius the position is progressed to the end of the route. This is done to ensure route waypoint completion.
    - Next part is the actual extrapolation of the vehicle's position along the line. This is done by segmenting the route's list of LatLngs into road segments based on the LatLngs that represent the start and end points of each segment. A segment, separate to the road segments object, is created as a list of two LatLngs, the start and end LatLngs.
    - To begin the extrapolation, the system first finds the starting segment. It does this by checking what segment path that the vehicle is currently positioned on using the PolyUtil isLocationOnPath() function. Once found, the foundStart boolean is set to true and the

system knows where to extrapolate from. As part of this step, the road segments proceeding the start are set to visited as a precaution for any missed waypoints in previous road segments (an issue that arises when the simulation speed is turned up). The current distance is calculated using the vehicle's position on the segment to the segment's end, and if it's greater than the specified distance to extrapolate (in this instance, the average speed) we simply extrapolate the vehicle forward by the specified distance, since it still contains the vehicle after moving it. If the current distance is less than the distance specified, this segment has been completed and is set to visited.

    - Note: the calculated average speed is used as the distance parameter as it is calculated as a metres per second value, and this scheduled component is run every one thousand milliseconds (ie: every second).

- If the current distance is less than the distance, it is added to actual distance, and the current distance is re-calculated using the next segment's start and end points. If the new current distance plus the actual distance together is greater than the specified distance, we extrapolate from the new segment's start point by the distance minus the actual distance (ie: the last segment's remainder). This is illustrated below:


End
Current Distance
Start
Actual Distance + Current Distance > Distance
Extrapolate Amount from second segment start: Distance - Actual Distance
Actual Distance

5. The SphericalUtil class is used to calculate the heading and subsequent location of the vehicle along the line, which is returned as the extrapolated LatLng position.

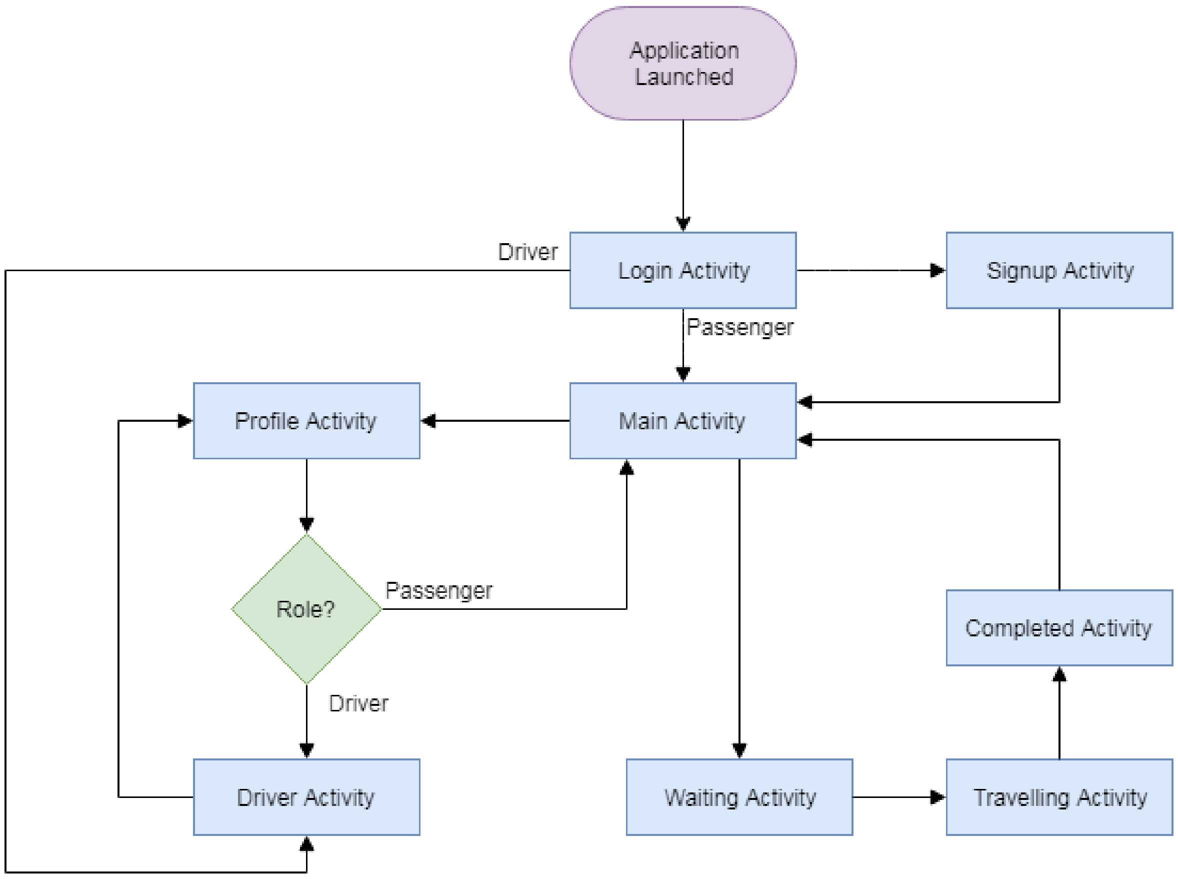
NB: SphericalUtil and PolyUtil classes are provided by Google's Android Map Utils.

3.5 Android Application

One of the main differences of the floating bus system compared to traditional public bus systems is its ability to take dynamic requests at any time and to be able to plot its route accordingly. This feature is only made possible thanks to the ever connected nature of modern smart devices, such as Android smartphones. To be able to generate the requests that influence the floating buses, an Android application was implemented, making it easy for users to be able interact with the floating bus system.

3.5.1 Activities

Android application development consists of defining and managing activities where app functionality is stored. An activity represents a single screen of the user interface, hence multiple activities are defined for multiple different screens. The overall flow of the application looks like this:



The various activities are as follows:

- Login Activity
  - This is the first screen presented to the user on the first time the application is opened. The user enters their pre-registered email address and password, and the app connects with Firebase to verify their credentials. Upon successful validation, the user is passed onto the next activity, Main Activity.
  - If a user has logged into the application before, they are immediately passed to either the Main Activity or Driver Activity of the application. This is determined by checking the SharedPreferences and existing Firebase instance of the application before rendering the layout xml for the login activity.
    - SharedPreferences is a way of storing user data locally to be stored, modified and accessed in the application later.

        - To be able to re access this activity's page (ie: not be immediately ushered on to the next activity), the application data can be cleared in the Android system settings.
    - If the user hasn't signed up to the application before, they are able to follow a link through to the Signup Activity to register an account to use the application.
- Signup Activity
    - New users to the application can register to use the application using a valid email and password. Basic rules are enforced to ensure forms are filled out correctly, but for the purposes of this study are very lenient. For practical use, tighter password rules and a valid email regex checker should be added.
    - Once the basic rules are met, the email and password are sent to Firebase's authentication service.
    - A local copy of user details is locally stored in the backend system. This is done by running a special command from the backend system itself, though future implementations should add the user from the Android application at the same time as Firebase registration.
- Profile Activity
    - This is the screen that allows users to specify their role in the floating bus system. The screen is presented as a profile page, where users can see their email address, role and max capacity of their vehicle if set to 'Driver'.
        - To be able to access the max capacity field, the user must declare their role as 'Driver'.
    - Depending on the user's chosen role, they will be redirected to either "Main Activity" or "Driver Activity". A post will be made to the backend system signalling the user's updated role and makes changes accordingly (ie: creates a vehicle and assigns the 'Driver' user to it, or sets the vehicle to inactive and removes it from the system. A user can only change back to 'Passenger' once their route no longer has any active requests tied to it.)

- Main Activity
    - This is the activity that ‘Passenger’ role users are directed to after login. This screen shows the user’s current latitude and longitude coordinates on a Google maps viewer, and allows them to use a Google Place Picker to set a request destination. An input is also provided for inputting the amount of passengers this request requires.
        - Google Place Picker is a simple way of choosing a location from Google maps using an address in plain English (for example, “Trinity College, Dublin”). The place picker object returns with the required LatLng coordinate data required for the request’s destination, while making it easier for the user to set a destination without worrying about the LatLng of their desired destination.
    - In the background of this activity, the user location is collected and used to display their position on the map. In addition to this, their current LatLng position is used when making the request to the backend as the start LatLng coordinates.
        - Android requires that permissions are requested of the user before using their location data.
- Driver Activity
    - This activity is exclusive to users that have assigned themselves to the ‘Driver’ role. Users are shown a polyline of their assigned route on a map, with waypoint markers along the way.
    - In the background of this activity, the user’s location data is posted to the backend system, where it is used for the user’s assigned vehicle position. Information is also fetched from the backend about the vehicle’s route so that new routes or waypoints are reflected to the driver.
- Waiting Activity
    - This activity is exclusive to users with the ‘Passenger’ role that have submitted a request for a ride. Users are shown a map with their assigned vehicle’s polyline route, along with different markers that

    indicate the vehicle’s position, their pickup waypoint location, and their current location.
  - HTTP Get requests are run in the background of this activity, fetching information about the user’s assigned vehicle and pickup waypoint and refreshing their positions on the map as they/if they update. The vehicle’s polyline is also refreshed if it changes. The request’s picked up status is also checked.
- Travelling Activity
  - This activity occurs one the request has been picked up by the vehicle. The user is displayed a map fragment with their assigned vehicle’s path and current position constantly refreshing. The request’s completed status is also monitored in this activity.
- Completed Activity
  - This is the final screen in the request flow, which is shown once the request is completed and the user is at their desired destination. The user’s generated fare is displayed, and the user can now return to the Main Activity if they wish to place another request.

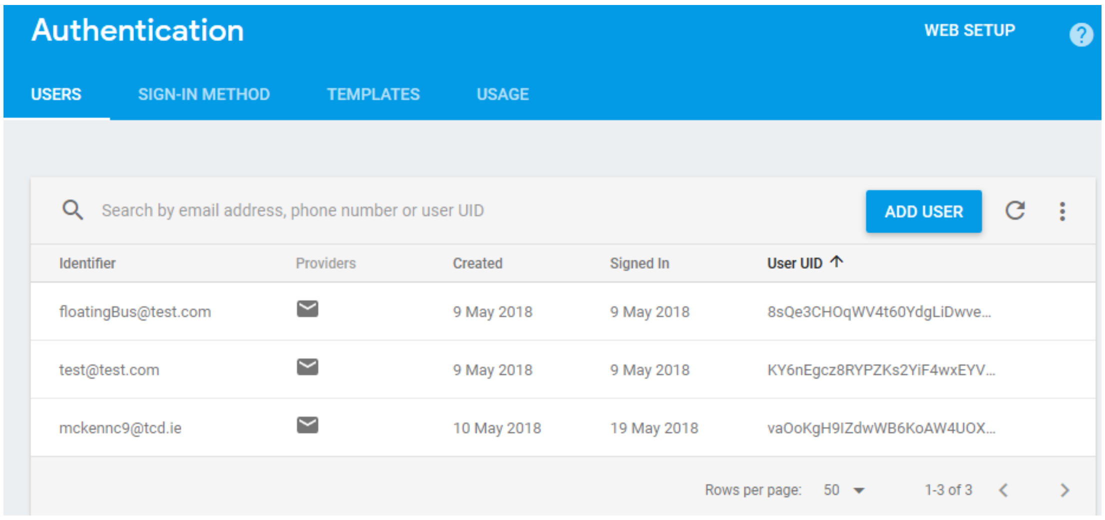
3.5.2 Functionality

This section describes the main functionality of the Android application including how it communicates with the different services at play in this system, including the floating bus system itself. The Android application requires constant internet access to function correctly, and is the epitome of the ever connected technology that the floating bus system takes advantage of.

3.5.2.1 Firebase

Firebase is Google’s mobile and web application development platform. It provides a wide range of services for app development, including authentication, database and analytics services.

Firebase is used in the floating bus Android application for user authentication, for both login and registration. The Firebase SDK is integrated into the app project, and provides an email and password authentication service. The SDK handles communications between the application and the Firebase online service.



Identifier	Providers	Created	Signed In	User UID ↑
floatingBus@test.com	✉	9 May 2018	9 May 2018	8sQe3CHOqWV4t60YdgLiDwve...
test@test.com	✉	9 May 2018	9 May 2018	KY6nEgcz8RYPZKs2YiF4wxEYV...
mckennc9@tcd.ie	✉	10 May 2018	19 May 2018	vaOoKgH9lZdwWB6KoAW4UOX...

Rows per page: 50 1-3 of 3

3.5.2.2 Submitting a Request

Communication with the backend system involves using HTTP requests to pass information between the two systems. Data is passed to and forth using JSON data format. A good example of sending data from the Android app to the backend system can be seen when making a request for a ride.

When a user logins into the application successfully for the first time, and any time they are set to 'Passenger', they are presented with the Main Activity screen where they are able to make pickup requests. The body of a request is made up of the following:

- User Email
- Current Coordinates
- Destination Coordinates
- Passenger Amount

To allow for ease of use when picking a destination to travel to, a PlacePicker widget is used. The PlacePicker is part of the Places SDK for Android, and provides a UI dialog with an interactive map and list of nearby locations. Users can type their destination into the search bar to find the exact location they're looking for. Once confirmed, a PlacePicker object is created with the LatLng coordinates needed when making the request. This tool saves the user from having to input difficult to remember coordinate data, while still allowing the application itself to receive that data. The PlacePicker widget is seen below:

After confirming the destination location of their request, the user is returned to the Main Activity view to fill out the final field, the "Amount" field, representing the number of passengers that will accompany them on this request. This is important for checking against the capacity of the floating buses available.

Once the request is ready to be sent, the user taps the submit button. On the backend of the application, a JSON object is made for the request to be sent to the floating bus system backend. The request object resembles the following sample JSON object:

```
{
  "amount": "1",

  "sourceLat": "53.3352318",
  "sourceLong": "-6.228456899999969",

  "destLat": "53.33505290000001",
  "destLong": "-6.255541900000026",

  "address": "Aviva Stadium, Lansdowne Rd, Dublin 4",

  "email": "test@test.com"
}
```

In the above, the 'sourceLat' and 'sourceLong' values are taken from the user's current world space coordinates, and the 'destLat' and 'destLong' values are obtained from the PlacePicker object. While the 'address' is also obtained from the PlacePicker and sent with the request, it isn't used when creating waypoints on the backend, but is stored to the database. The reason for this is discussed in the future work section of this report.

Once the request has been successfully received, the backend system returns a response 200 OK to signify no errors occurred when accepting the request. The user is then redirected to the Waiting Activity.

While on the Waiting Activity screen, there is constant communication between the application and the backend system. The application performs GET requests to the floating bus system so to get updated information about the user's assigned vehicle and pickup waypoint. The user's location is re-obtained and displayed on the map so to monitor their progress as they move closer to their assigned waypoint.

To manage requests made to the backend server, a set of "helper" classes were made for POST and GET requests, with an additional helper class for vehicle/request specific information.

3.5.2.3 Standby Activities

Once a request has been made, the user is ushered onto a new Waiting Activity screen where the application polls the backend system to monitor the request's picked up status. Users may cancel their request at this point, leading to the vehicle's route being recalculated.

After being picked up by the vehicle, the user is brought to the Travelling activity where the application once again polls the backend system for route information and request completed status.

Once the request has been marked as completed, the user is brought to the Completed Activity page where they are shown their fare for their journey. This is calculated using the euclidean distance between their pickup and drop off waypoints rather than their road network distance due to the highly variable nature of the floating bus routes. The user may now proceed to make new requests to the system if needs be.

3.5.2.4 Driver Activity

A unique feature of the floating bus system implementation is the ability for users to sign up as Drivers to follow routes and service customers. This activity updates and shows the vehicle’s calculated route and stops along its route to be serviced. The vehicle’s position is tracked using the user’s geolocation.

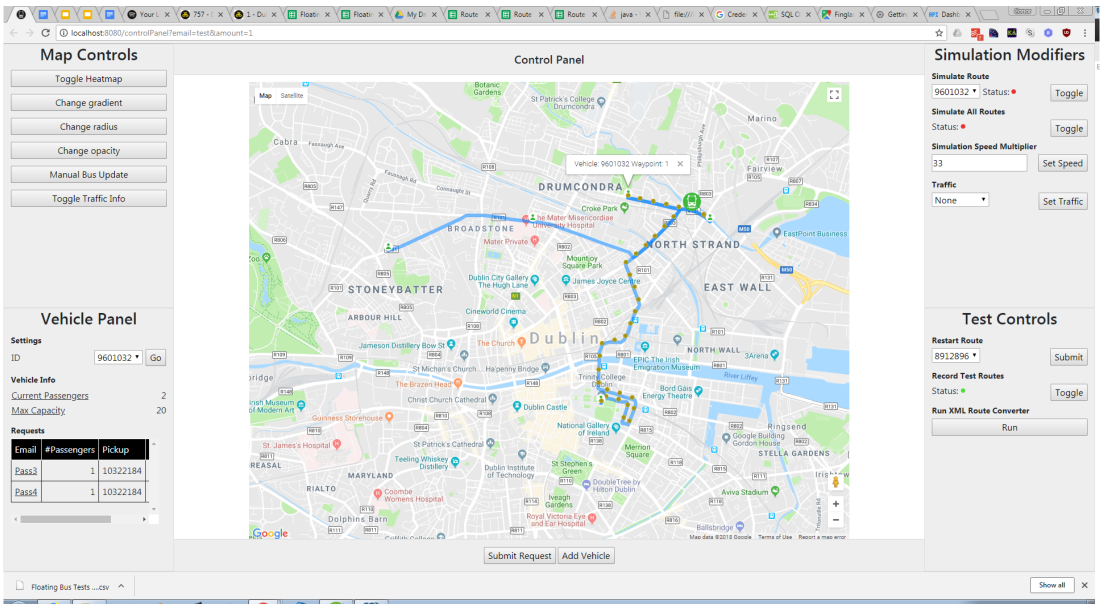
Chapter 4 - Evaluation

In this section the practicality of the floating bus system is shown and discussed. The practicality of the system is largely based off of how it compares to existing systems in the public transport domain. This means that the floating bus system will be compared with the public bus service, based off of Dublin Bus routes and stop locations. The evaluation of the floating bus system's practicality is based off the following metrics:

- Distance Travelled
- Distance from start point and distance to desired destination
  - This is a metric used for gauging passenger convenience, and looks at the distance users have to travel to get to their allocated floating bus stop waypoint versus the corresponding static bus stop.
- Time to complete
  - Including time taken with different traffic estimates
- Emissions
  - Based on average CO2 emissions per kilometer travelled.

4.1 Implementation Testing

The functionality of the floating bus system was tested rigorously with multiple different scenarios to see how the system performed as expected under certain circumstances. The implementation of the system was tested using the control panel interface, which allowed for the creation of new vehicles and requests. Simulations are also run from this view, with emulated pickups and drop offs, allowing for a full understanding of how the floating bus works in a practical setting. Vehicle movement was "breadcrumbed" to verify the integrity of the route.



Vehicle Panel

Settings

ID 9601032 Go

Vehicle Info

Current Passengers 2

Max Capacity 20

Requests

Email	#Passengers	Pickup
Pass3	1	10322184
Pass4	1	10322184

Simulation Modifiers

Simulate Route

9601032 Status: ● Toggle

Simulate All Routes

Status: ● Toggle

Simulation Speed Multiplier

33 Set Speed

Traffic

None Set Traffic

Test Controls

Restart Route

8912896 Submit

Record Test Routes

Status: ● Toggle

Run XML Route Converter

Run

Control Panel Interface

4.2 Test Cases

To be able to properly evaluate the floating bus functionality, it was important to run all manner of test situations. The results gathered were highly dependent on the route and number of requests assigned to each vehicle, and could be better or worse than their traditional counterparts.

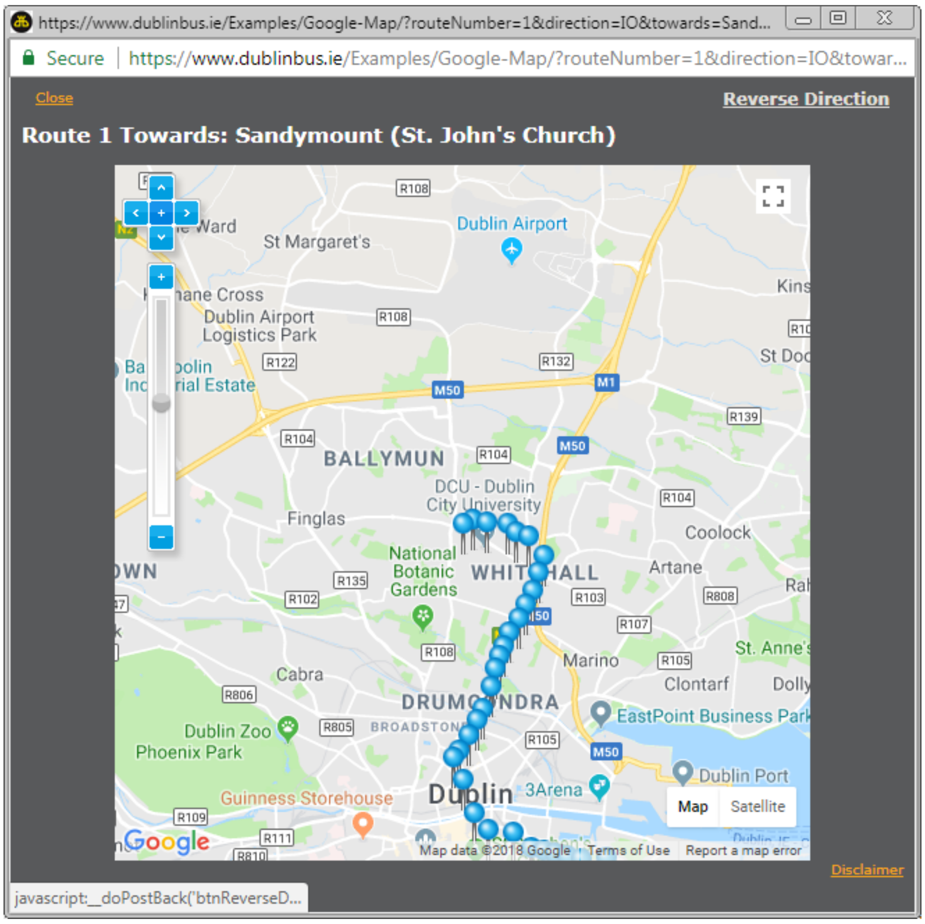
The first set of test cases look at situations where the floating bus is a more viable solution to the static bus route in terms of economic and environmental reasons. These test cases compare how a set of requests can be handled by fewer floating buses than multiple static bus routes. The static buses are based off of real Dublin bus routes manually input based on route information from the official Dublin bus website.

4.2.1 Test Data Collection

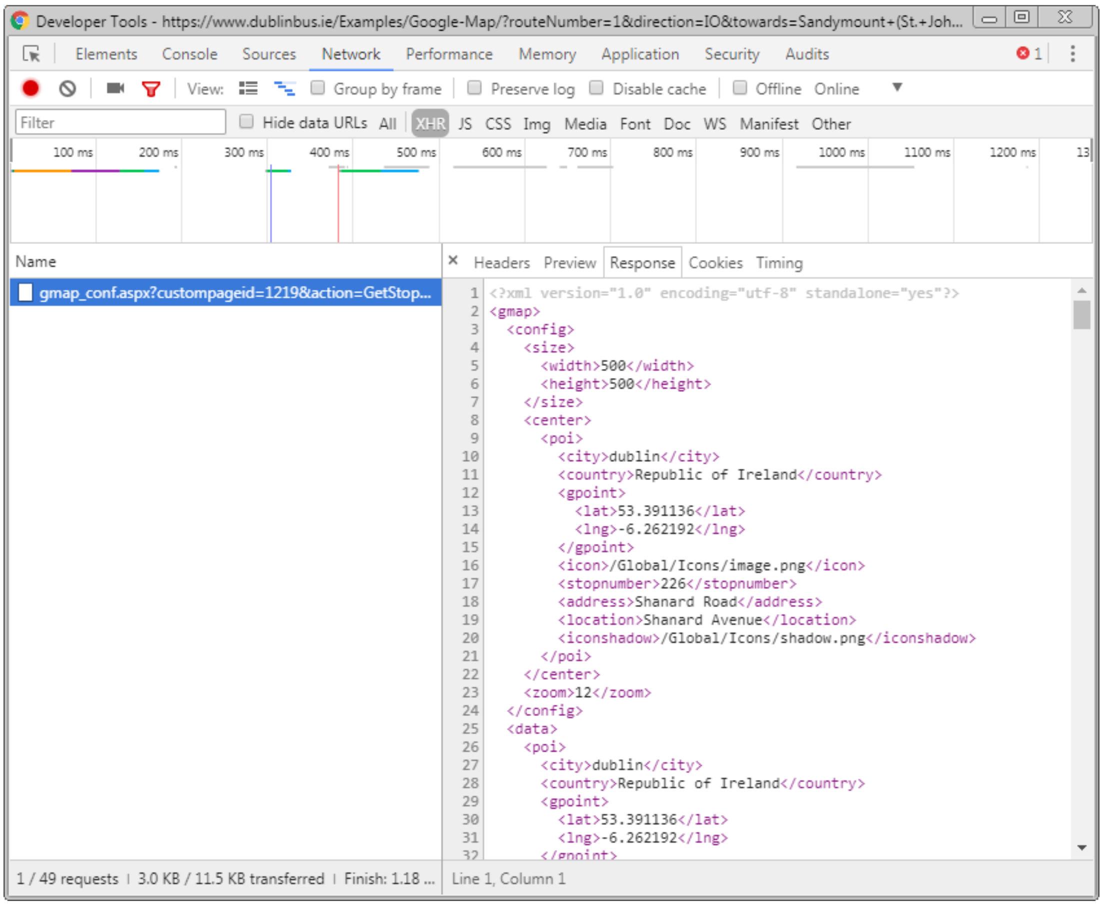
The floating bus system is evaluated by comparing various use cases for certain requests against their static route alternatives. This means that the Dublin bus route information had to be gotten and compared with how a floating bus would operate under the same user demands.

Since the static bus option doesn't adhere to user input, the default routes are compared with floating versions. To be able to do this, it was important to get actual Dublin bus route information for stops and the running order they run in. To do this, the Dublin bus route information needed to be extracted from the Dublin bus website itself. The steps taken to acquire the route info for a sample Dublin bus route are as follows:

1. Open Dublin Bus route viewer:



2. Open Chrome developer tools and open the Network tab.
3. Select XHR files to show and refresh the route viewer window. This will reload the window prompting the XML file with the route information to be resent and accessed.



gmap_conf.aspx?custompageid=1219&action=GetStop...

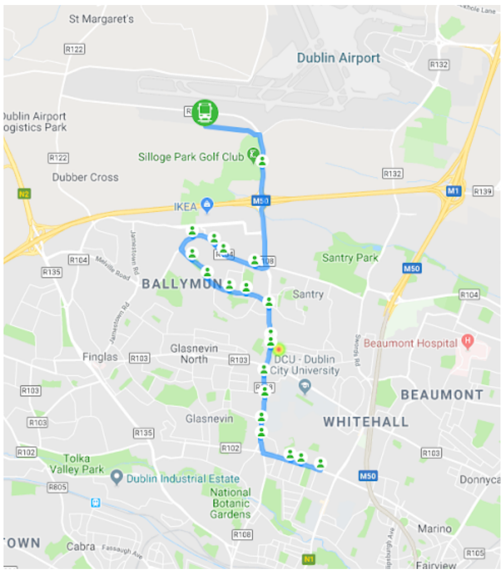
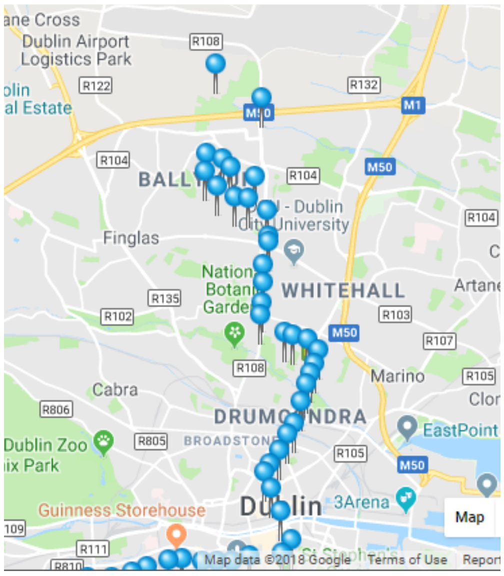
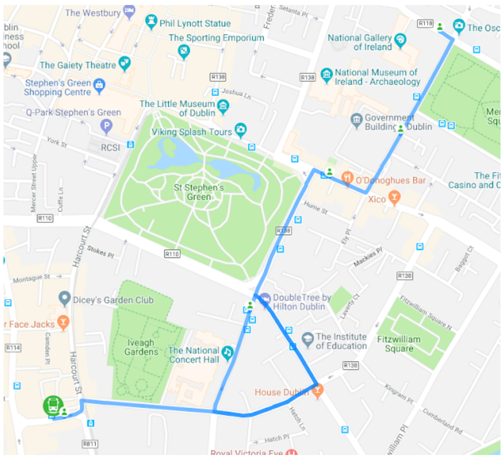
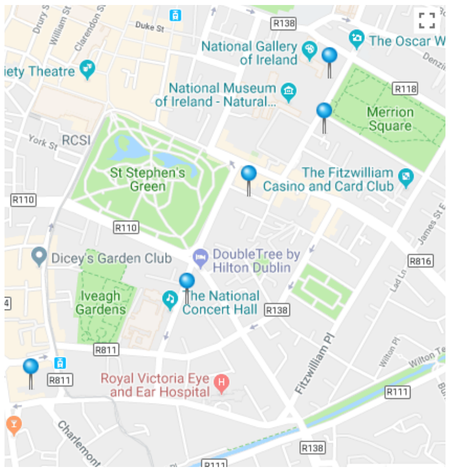
```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<gmap>
  <config>
    <size>
      <width>500</width>
      <height>500</height>
    </size>
    <center>
      <poi>
        <city>dublin</city>
        <country>Republic of Ireland</country>
        <gpoint>
          <lat>53.391136</lat>
          <lng>-6.262192</lng>
        </gpoint>
        <icon>/Global/Icons/image.png</icon>
        <stopnumber>226</stopnumber>
        <address>Shanard Road</address>
        <location>Shanard Avenue</location>
        <iconshadow>/Global/Icons/shadow.png</iconshadow>
      </poi>
    </center>
    <zoom>12</zoom>
  </config>
  <data>
    <poi>
      <city>dublin</city>
      <country>Republic of Ireland</country>
      <gpoint>
        <lat>53.391136</lat>
        <lng>-6.262192</lng>
      </gpoint>
```

1 / 49 requests | 3.0 KB / 11.5 KB transferred | Finish: 1.18 ... Line 1, Column 1

4. Select all from the Response tab and save to an XML file.

From inspection of the route information, it can be seen that the stop information is already in the correct running order according to the corresponding timetable's stop order. Once the route data has been saved to an XML file, it then needs to be parsed into a format usable by the floating bus test system. To do this, a function was written that takes the XML file and parses the information into a usable CSV format. A JSON file for the bus route vehicle is also created at the first stop in the route. These files are then be moved to the test folder. To be able to parse more routes, the above is repeated and the XML files are moved into the bus routes folder to be parsed for use.

To best represent the static bus route alternatives, waypoint merging is turned down to 1/10th of a kilometre, so that only overlapping waypoints at each static bus stop are merged into one. Also due to Google's limitations on the number of route waypoints allowed, the routes are incomplete, but are enough to show solid results in cases. The routes were then inspected manually to ensure that the returned directions weren't completely unrealistic to a real bus route. This is because Google Directions doesn't allow for bus lanes in Driving vehicle's directions, hence it was necessary to make sure that the floating bus and static buses were being compared as fairly as possible. Below is an example of a realistic route (as compared with the Dublin Bus route viewer), and a totally unrealistic route (that is rejected for testing).

 Dublin Airport BALLYMUN Santry DCU - Dublin City University BEAUMONT WHITEHALL Glasnevin Finglas Marino	 Dublin Airport Logistics Park Finglas WHITEHALL Cabra DRUMCONDRA Dublin Zoo Guinness Storehouse Dublin 3Arena EastPoint Marino Map
Acceptable, follows same shape and pattern as stated.	
 St Stephen's Green National Gallery of Ireland DoubleTree by Hilton Dublin Iveagh Gardens The National Concert Hall Fitzwilliam Square	 National Gallery of Ireland National Museum of Ireland - Natural... Merrion Square St Stephen's Green The Fitzwilliam Casino and Card Club DoubleTree by Hilton Dublin Iveagh Gardens The National Concert Hall Royal Victoria Eye and Ear Hospital
Unacceptable, ignoring bus lanes leads to additional loop due to meeting one way roads along a normal driving route.	

With the limitations on the number of waypoints allowed, we end up using route data with only a small section of the overall route like so:

Id	S Lat	S Lng	D Lat	D Lng	S Address	D Address	Amount	email
1	53.418164	-6.277027	53.413193	-6.265336	6235	327	1	Route 13 Towards - Grange Castle.csv_2
2	53.413193	-6.265336	53.401122	-6.266943	327	6016	1	Route 13 Towards - Grange Castle.csv_3
3	53.401122	-6.266943	53.402647	-6.273342	6016	6017	1	Route 13 Towards - Grange Castle.csv_4
...								...
17	53.380416	-6.265649	53.377693	-6.259649	39	116	1	Route 13 Towards - Grange Castle.csv_18
18	53.377693	-6.259649	53.377366	-6.257526	116	117	1	Route 13 Towards - Grange Castle.csv_19

*Note: See Appendix B for Route 757's formatted and unformatted data. S(ource) Address and D(estination) Address are labelled as bus stop numbers for static routes.

To be able to test against the floating bus system fairly, we create a new floating bus vehicle and demonstrate the ways in which it would perform better than the static bus route, and where it would perform worse.

4.2.2 Measuring Passenger Quality of Life

The first test is checking proximity of the stop location to the request's actual starting and ending locations. This represents how a passenger may have to walk some distance before getting embarking on their route, or to reach their goal. The test data for this case is created using request coordinates in close proximity to those of the Dublin bus stops. For the above test, the following floating bus requests were used:

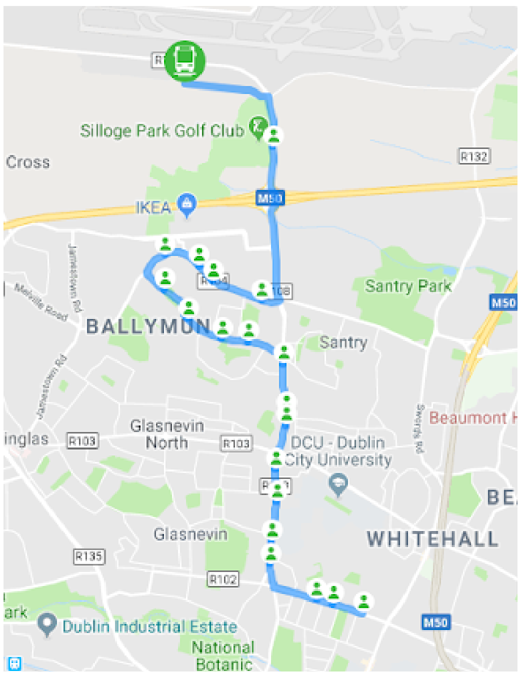
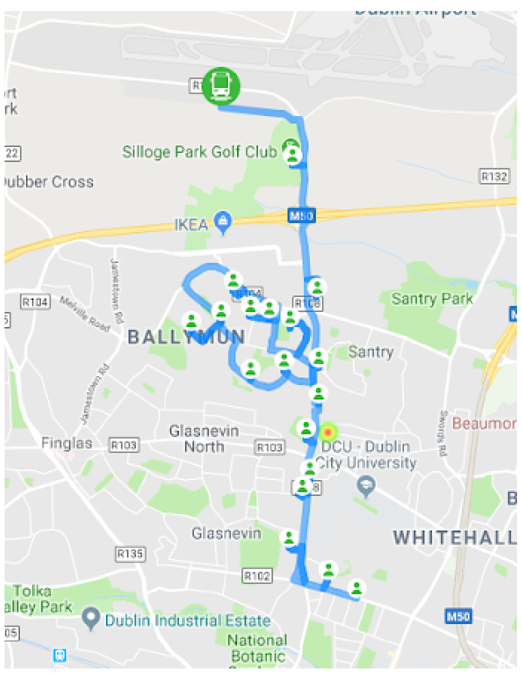
Id	S Lat	S Lng	D Lat	D Lng	S Address	D Address	Amount	email
1	53.42916	-6.301042	53.40231	-6.26362	Sandy Hill	Northwoord Business Campus	1	Route 13 Towards - Grange Castle.csv_2 Floating1
2	53.40231	-6.26362	53.399593	-6.267876	Northwoord Business Campus	Balcurris Park	1	Route 13 Towards - Grange Castle.csv_2 Floating2
3	53.399593	-6.267876	53.400815	-6.270323	Balcurris Park	St Joseph's Junior School	1	Route 13 Towards - Grange Castle.csv_2 Floating3
...								...
17	53.379162	-6.262108	53.378842	-6.261947	Andersons Food Hall & Cafe	The Rise Pharmacy	1	Route 13 Towards - Grange Castle.csv_2 Floating17
18	53.378842	-6.261947	53.377034	-6.257332	The Rise Pharmacy	Computer Ambulance	1	Route 13 Towards - Grange Castle.csv_2 Floating18

To be able to compare this with the static bus implementation, the following columns were added to the initial data set:

Start Lat	Start Lng	End Lat	End Lng	Start Address	End Address
53.42916	-6.301042	53.40231	-6.26362	SandyHill	Northwoord Business Campus
53.40231	-6.26362	53.399593	-6.267876	Northwoord Business Campus	Balcurris Park
53.399593	-6.267876	53.400815	-6.270323	Balcurris Park	St Joseph's Junior School
...					...
53.379162	-6.262108	53.378842	-6.261947	Andersons Food Hall & Cafe	The Rise Pharmacy
53.378842	-6.261947	53.377034	-6.257332	The Rise Pharmacy	Computer Ambulance

The start and end coordinates are then compared with the source and destination (ie: slat, slng) coordinates respectively. The start and end coordinates are used in place of the source and destination coordinates for the floating bus route when making requests, and are updated to reflect each of the resulting waypoint's final coordinates before measuring the distance between them.

The end routes look like this for the static option and floating bus implementation respectively:

	
Dublin Bus Route 13 (Sub section)	Floating Bus Route

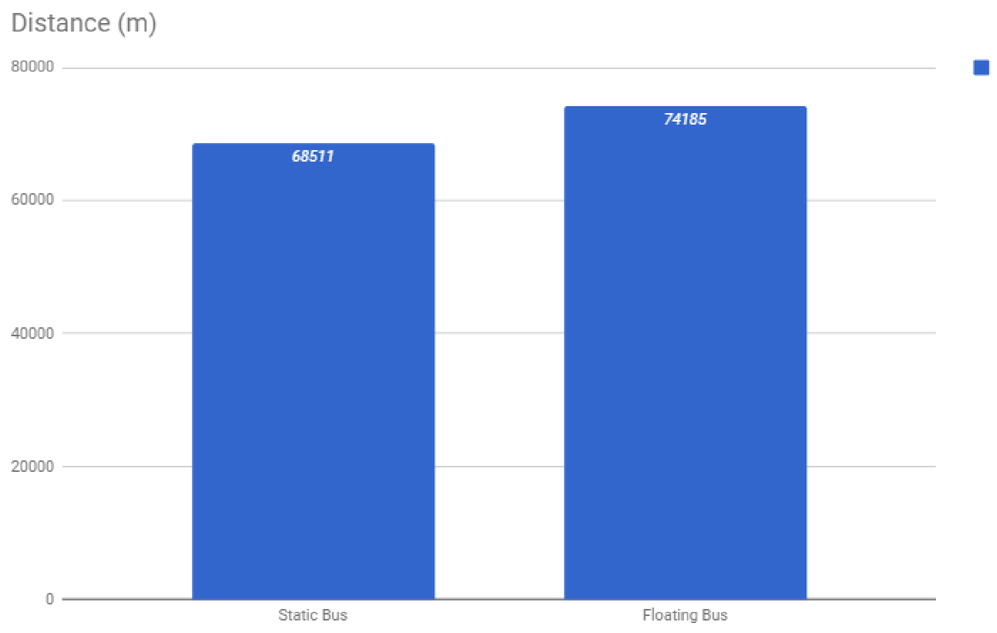
The route info for each is as follows:

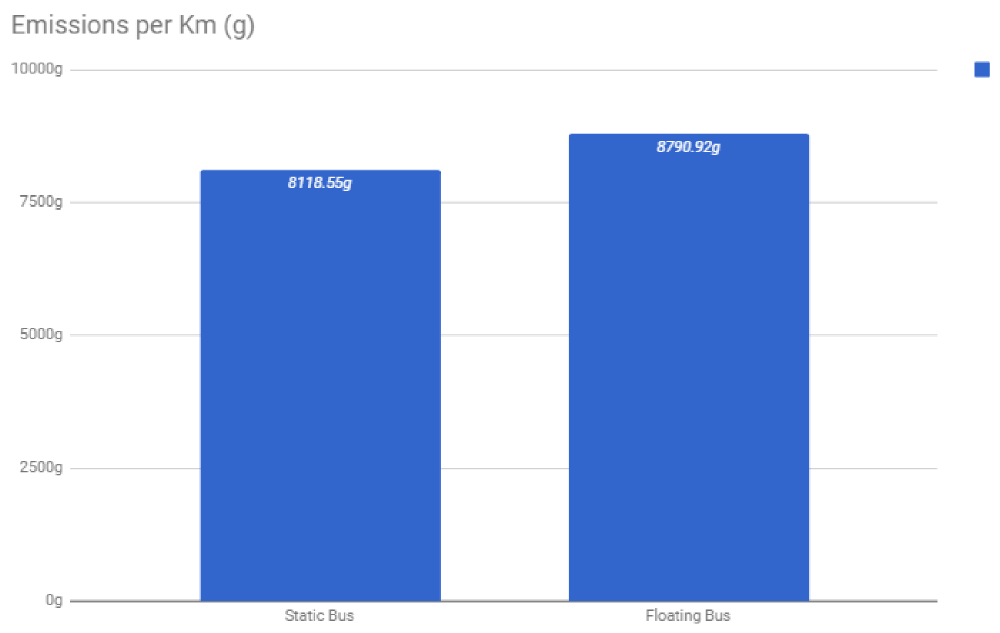
id	distance_metres	time_to_complete	traffic_time_best_guess	traffic_time_optimistic	traffic_time_pessimistic	vehicle_id
8912898	68511	9045	6804	6516	7829	9142274
9142272	74185	10206	10650	8375	15843	9371648

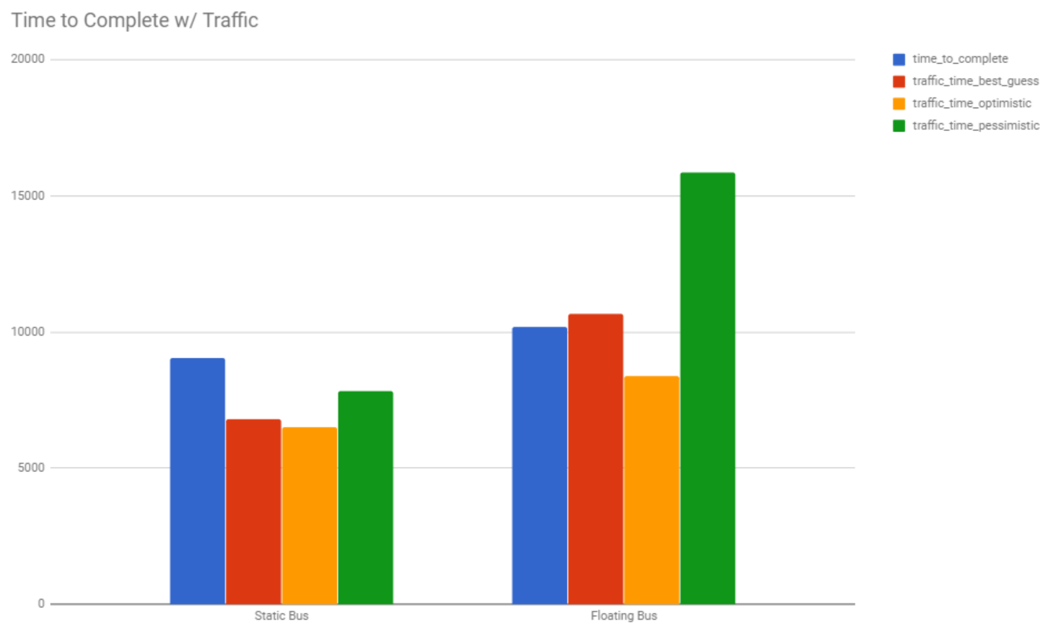
Where:

- Vehicle ID: 9142274 is Dublin Bus Route 13
- Vehicle ID: 9371648 is Floating Bus

The results can be compared using the following bar charts:



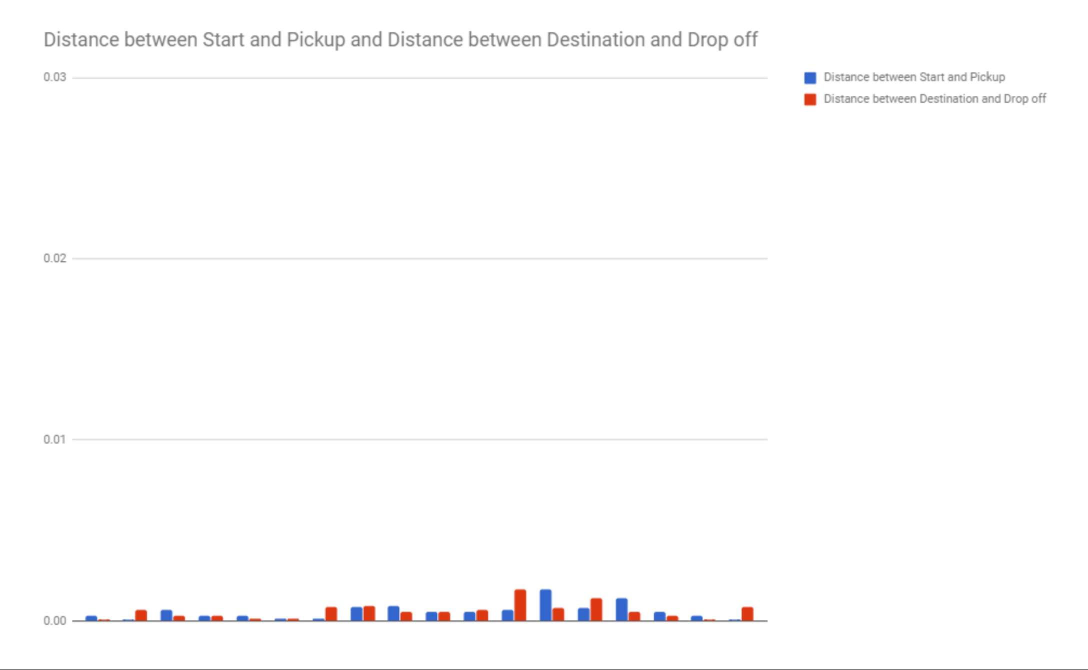




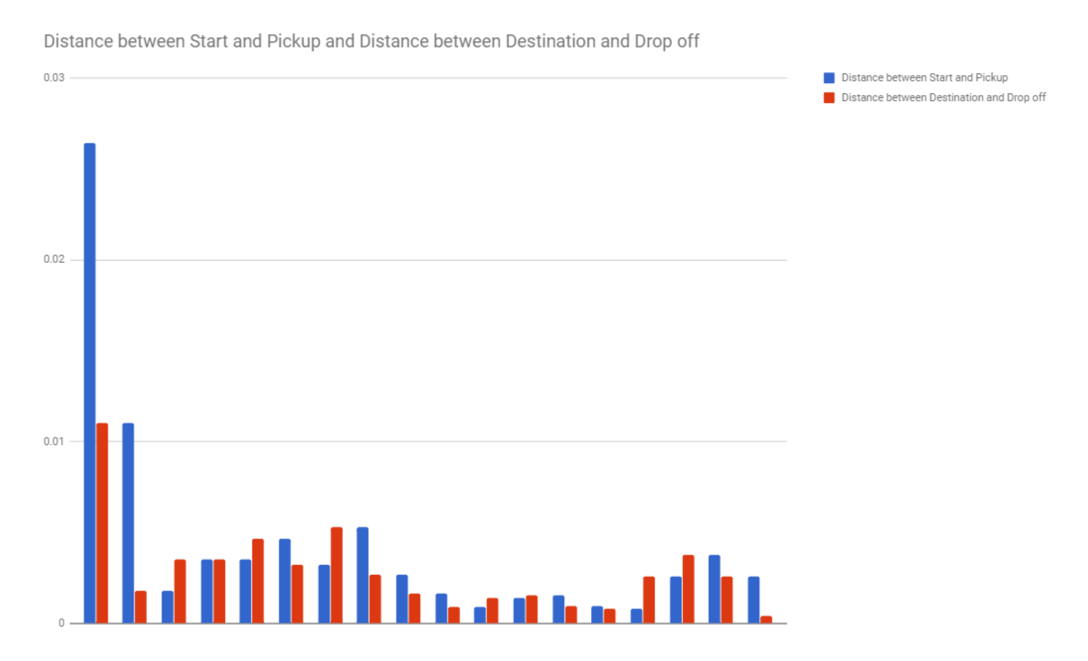
Measuring passenger convenience was done by comparing the distance from a request's source coordinate location and requested destination coordinate location with where each request's pickup and drop off waypoint finalized after merging and being related to the relevant road segment. The results for each type of bus are as follows for the above:

Static Bus		**Floating Bus**	
Distance between Start and Pickup	Distance between Destination and Drop off	Distance between Start and Pickup	Distance between Destination and Drop off
0.02641272877	0.01101745638	0.0002596150997	0.00003605551275
0.01101745638	0.001791181175	0.00003605551275	0.0006079884867
0.001791181175	0.003531371547	0.0006079884867	0.0002659116395
0.003531371547	0.003519961648	0.0002659116395	0.0002662273465
0.003519961648	0.004630753934	0.0002662273465	0.0001339888055
0.004630753934	0.003227327842	0.0001339888055	0.0001272635062
0.003227327842	0.005285692859	0.0001272635062	0.0007544328996
0.005285692859	0.002692096766	0.0007544328996	0.0007975688058
0.002692096766	0.001612904213	0.0007975688058	0.0004722139346
0.001612904213	0.0008709087208	0.0004722139346	0.0004920254058
0.0008709087208	0.001385144397	0.0004920254058	0.0005783165223
0.001385144397	0.001555394805	0.0005783165223	0.001757960466
0.001555394805	0.0009502720663	0.001757960466	0.0007060028329
0.0009502720663	0.0008161648118	0.0007060028329	0.00123618971
0.0008161648118	0.002550077842	0.00123618971	0.000463876061
0.002550077842	0.003756487322	0.000463876061	0.0002921951403
0.003756487322	0.002569242106	0.0002921951403	0.00008045495634
0.002569242106	0.0003845256818	0.00008045495634	0.0007611734362

Since each request in the static implementation has no flexible pickup option, the distance from a start point to the nearest fixed bus stop is used. The floating bus stops generate closer to source and destination coordinates and hence cut down the distance value required by the user to travel when embarking and disembarking on the floating bus service. Comparing the results on the same scale gives us these charts:



Floating Bus



Static Bus

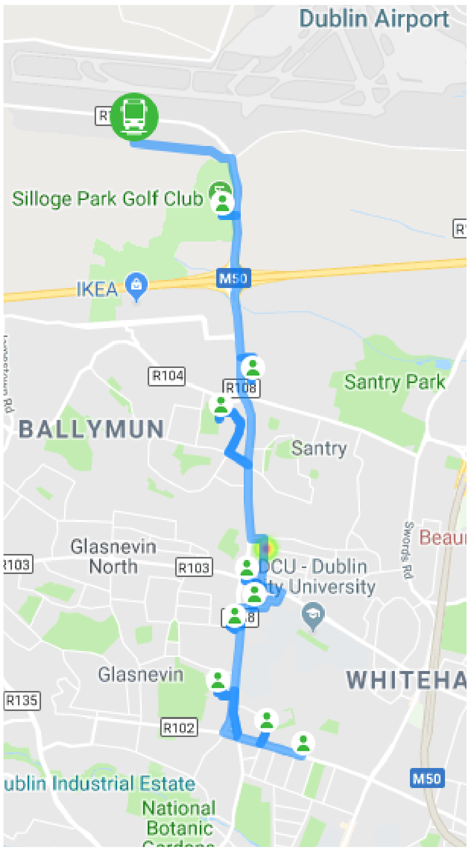
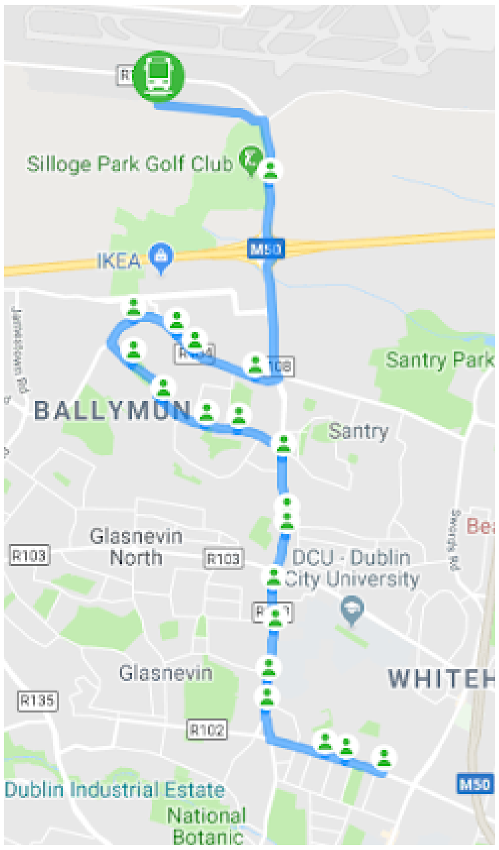
At a glance, it's easy to see that the floating bus is a much more convenient option for people who prefer a service that goes door to door. There is a trade off when it comes to the vehicle's overall distance travelled and time taken to complete the route, so users may be inconvenienced by longer journey times. However, these results show good potential in the floating bus service for applications such as school bus services, or for catering to passengers with disabilities that find it easier to be picked up closer to their requested locations.

4.2.3 Routing Changes Based on Demand

This test case shows how the floating bus can be both more efficient and less efficient than a static bus route depending on user demands. To show this, the same route as above was used with various requests removed or added to show how the floating bus efficiency can change between journeys.

First we compare a journey where we have lower demand for the route. The set of requests are as follows:

Id	S Lat	S Lng	D Lat	D Lng	S Address	D Address	Amount
1	53.413157	-6.267256	53.40231	-6.26362	Silloge Park Golf Club	Northwoord Business Campus	1
2	53.40231	-6.26362	53.399593	-6.267876	Northwoord Business Campus	Balcurris Park	1
3	53.390477	-6.265004	53.387826	-6.264093	Ballymun Library	Our Lady of Victories Church, Glasnevin	1
4	53.387826	-6.264093	53.385712	-6.265609	Our Lady of Victories Church, Glasnevin	Eurospar	1
5	53.385712	-6.265609	53.381855	-6.268021	Eurospar	North Dublin National School Project	1
6	53.381855	-6.268021	53.379162	-6.262108	North Dublin National School Project	Andersons Food Hall & Cafe	1
7	53.379162	-6.262108	53.378842	-6.261947	Andersons Food Hall & Cafe	The Rise Pharmacy	1
8	53.378842	-6.261947	53.377034	-6.257332	The Rise Pharmacy	Computer Ambulance	1

	
Floating Bus Route	Static Bus Equivalent Route

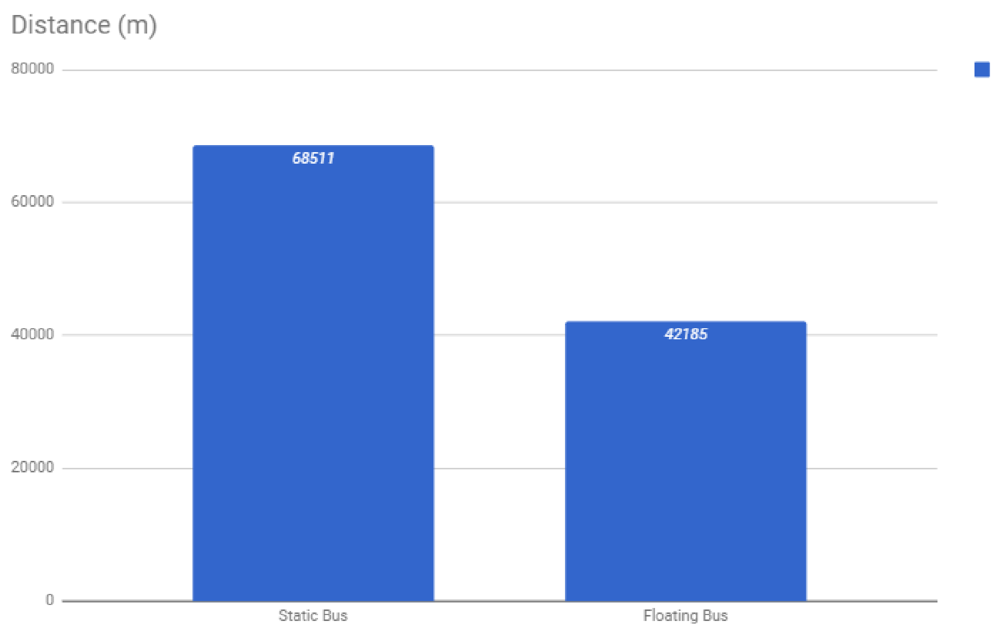
Here we can see that by knowing the demand in advance, the floating bus can follow a more optimal route than its static bus equivalent. Being able to make decisions like these when it comes to plotting routes can cut down on wasteful portions of trips, as can be seen above. The route information returned for the above is:

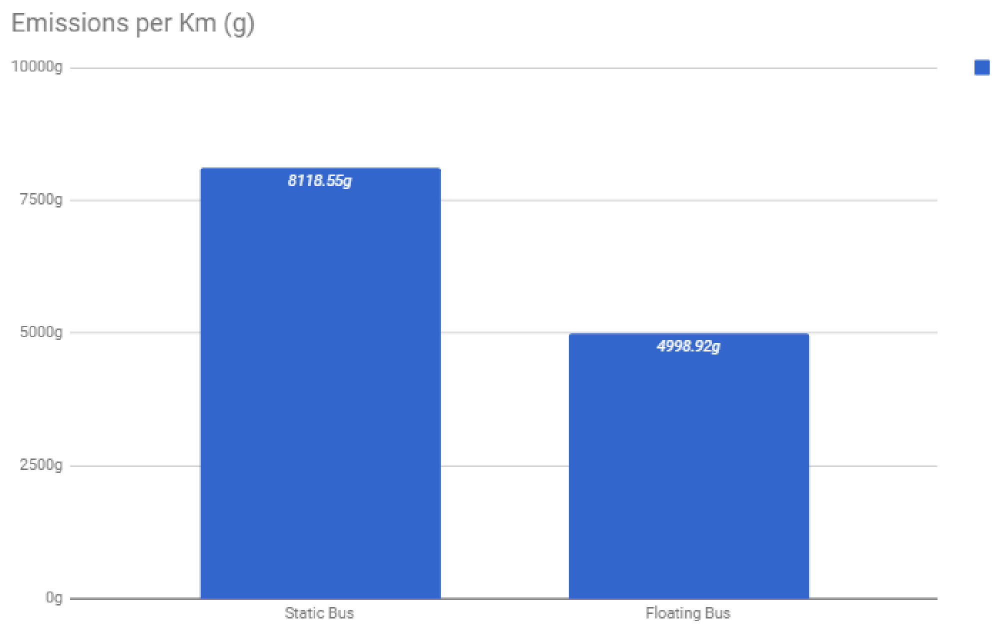
id	distance_ metres	time_to_c omplete	traffic_time_be st_guess	traffic_time_o ptimistic	traffic_time_ pessimistic	vehicle_id
8912898	68511	9045	6804	6516	7829	9142274
9207808	42185	5901	5905	4891	9520	9437184

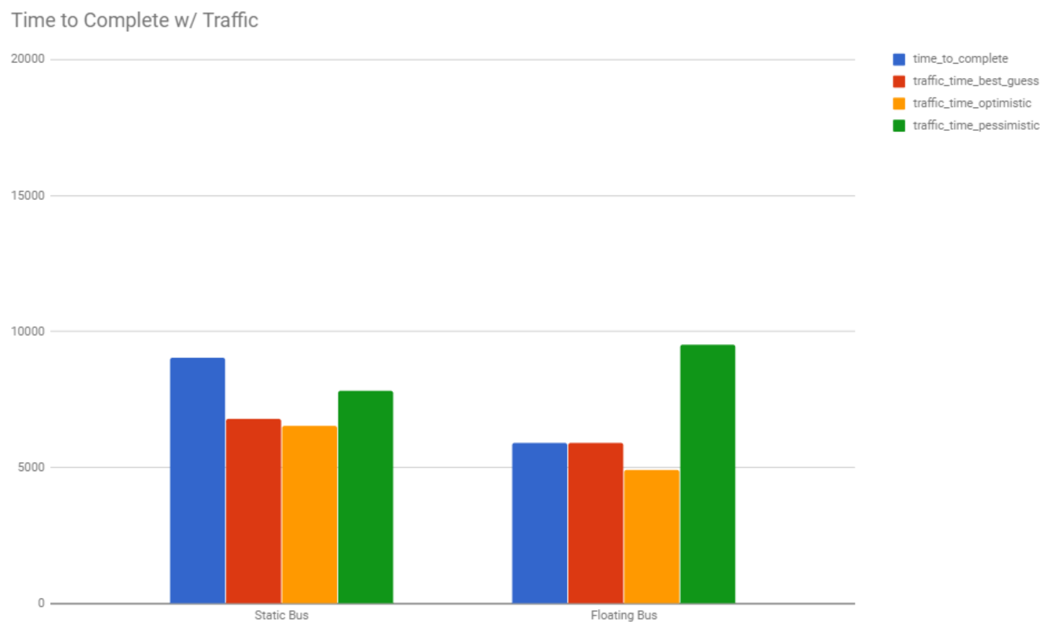
Where:

- Vehicle ID: 9142274 is Dublin Bus Route 13
- Vehicle ID: 9437184 is Floating Bus

The results can be compared using the following bar charts:

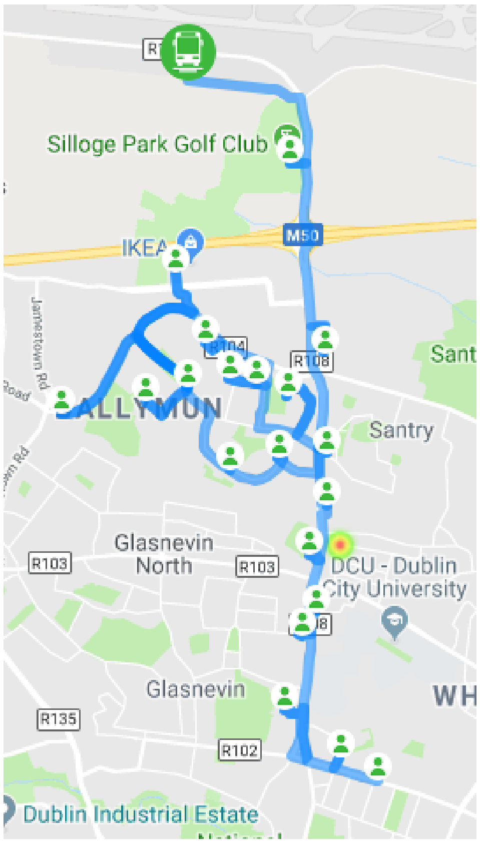
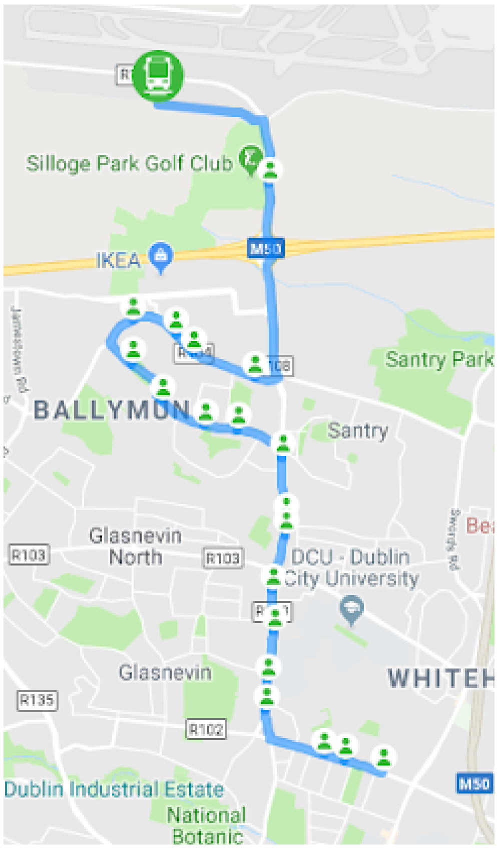

Distance (m)
80000
60000
40000
20000
0
68511
42185
Static Bus
Floating Bus


Emissions per Km (g)
10000g
7500g
5000g
2500g
0g
8118.55g
4998.92g
Static Bus
Floating Bus



Here we can see a marked improvement in the overall route needed to be taken when demand for the bus service is lower in all areas but time to complete with pessimistic traffic. Some segments of the road network taken by the floating bus are likely affected worse by heavy traffic conditions.

The floating bus doesn’t always create more efficient routes by itself however, depending on a higher concentration of stops and at further distances from the related static bus route the service performs worse over all. An example of a less efficient floating bus can be seen below:

 Silloge Park Golf Club IKEA M50 R104 R108 Sant BALLYMUN Santry Glasnevin North R103 R103 DCU - Dublin City University Glasnevin WH R135 R102 Dublin Industrial Estate	 Silloge Park Golf Club IKEA M50 Santry Park BALLYMUN Santry Glasnevin North R103 R103 DCU - Dublin City University Glasnevin WHITEH R135 R102 Dublin Industrial Estate National Botanic M50
Floating Bus Route	Static Bus Equivalent Route

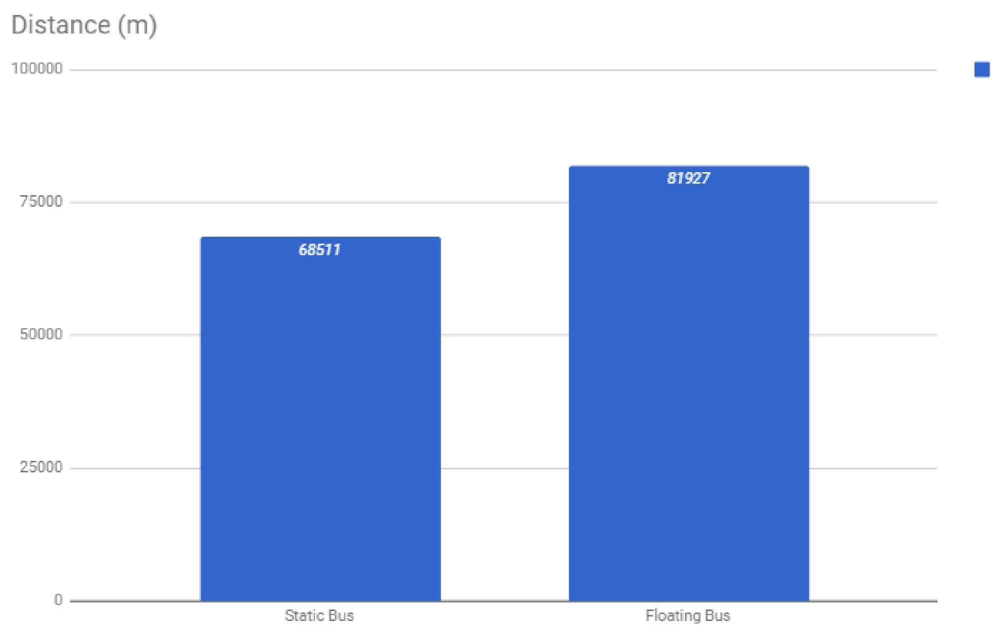
The results for the above routes are as follows:

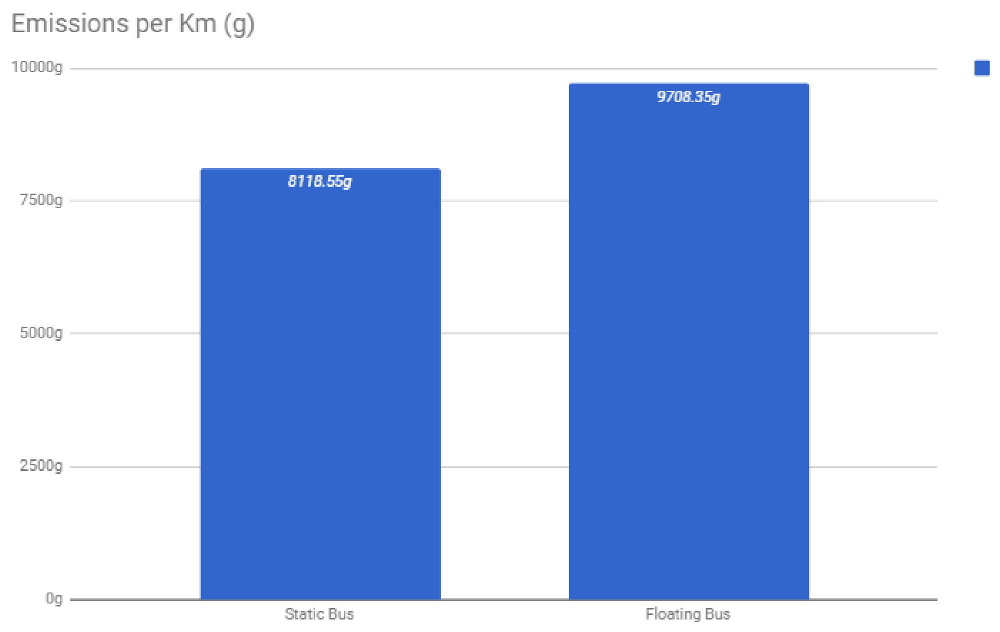
id	distance_metres	time_to_complete	traffic_time_best_guess	traffic_time_optimistic	traffic_time_pessimistic	vehicle_id
8912898	68511	9045	6804	6516	7829	9142274
9371648	81927	11191	11517	9345	17204	9601024

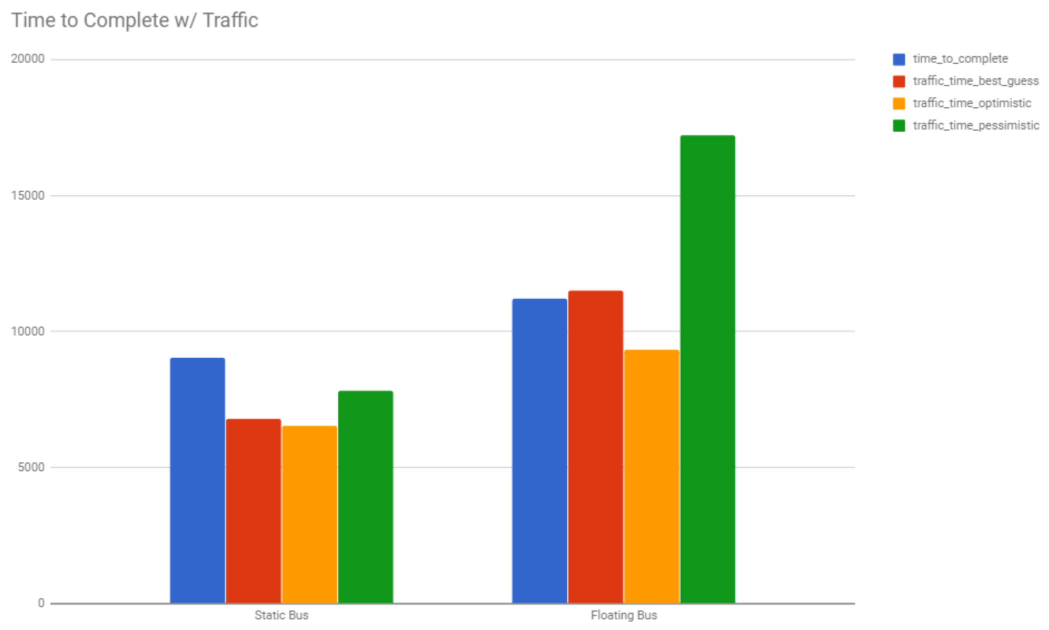
Where:

- Vehicle ID: 9142274 is Dublin Bus Route 13
- Vehicle ID: 9601024 is Floating Bus

The results can be compared using the following bar charts:


Distance (m)
100000
75000
50000
25000
0
68511
81927
Static Bus
Floating Bus


Emissions per Km (g)
10000g
7500g
5000g
2500g
0g
8118.55g
9708.35g
Static Bus
Floating Bus

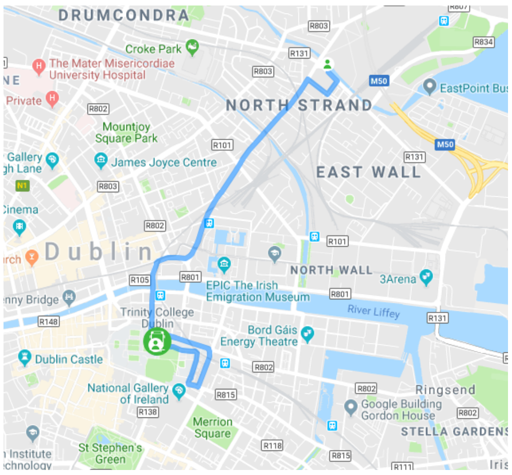


Here we can see the floating bus is taking a lot more time and going to greater distances to service the high volume of requests that are further away. This is something that could be solved by allocating these requests to another vehicle if possible.

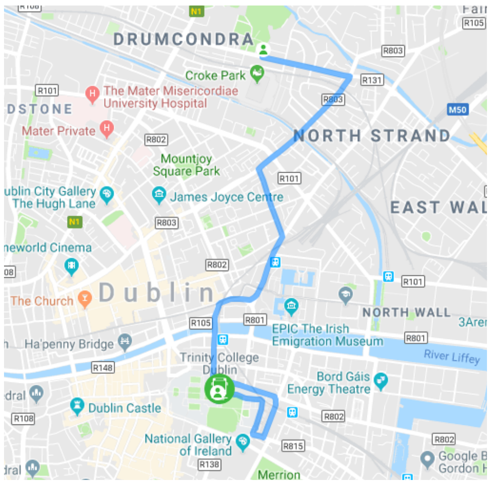
4.2.4 Number of Floating Bus Stops

In this section it is evaluated whether it is more efficient to have a single or multiple floating buses servicing the same requests. This is done by comparing how multiple static buses would service these requests all originating from a mutual pickup point to multiple separate destinations.

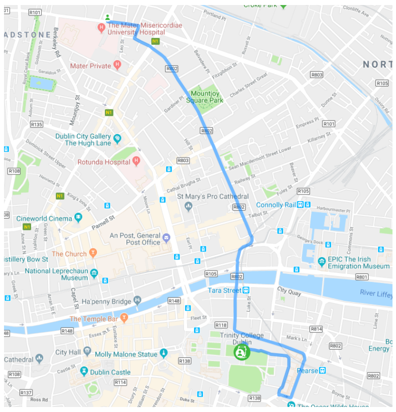
Trinity College Dublin was used as the starting location, and various stops were chosen around it. Individual requests were assigned as the static routes, and all of the requests together were assigned for the floating buses. The routes compared are as follows:



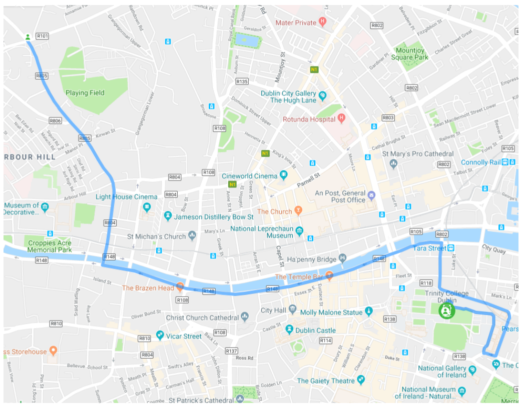
Individual Request #1



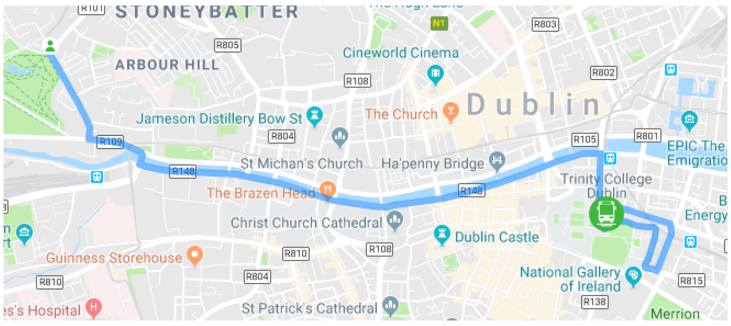
Individual Request #2



Individual Request #3

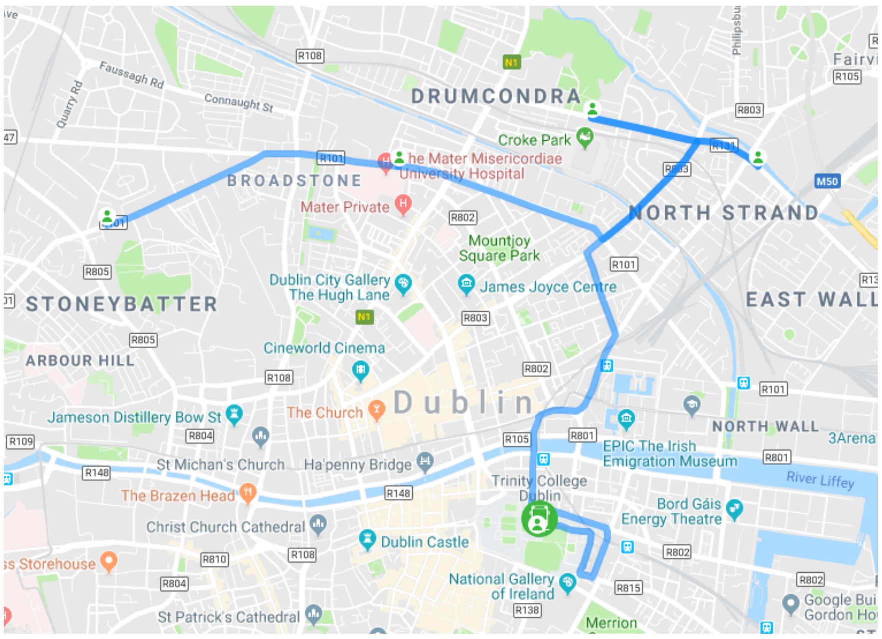


Individual Request #4

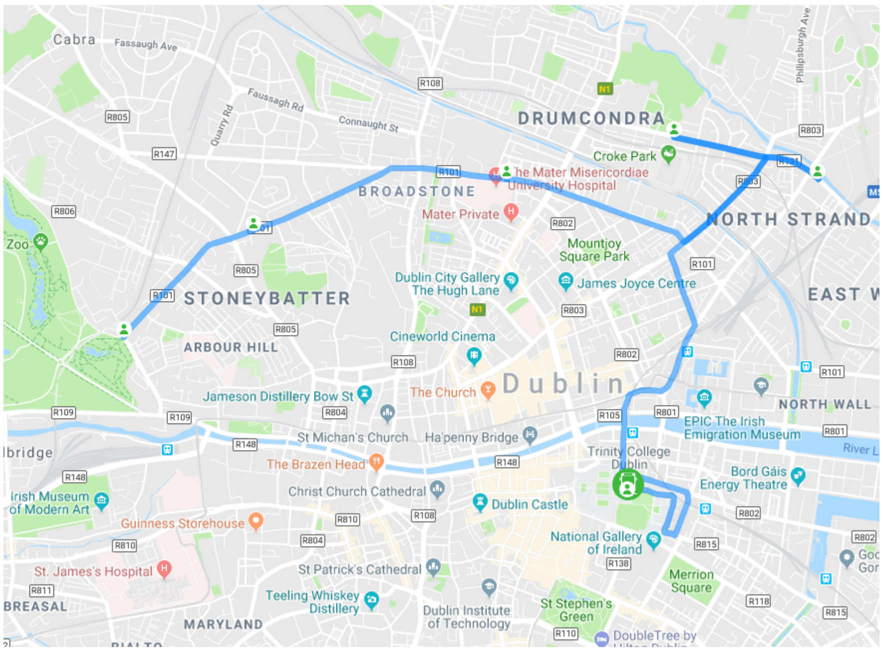


Individual Request #5

Compared with:



For ⅘ requests



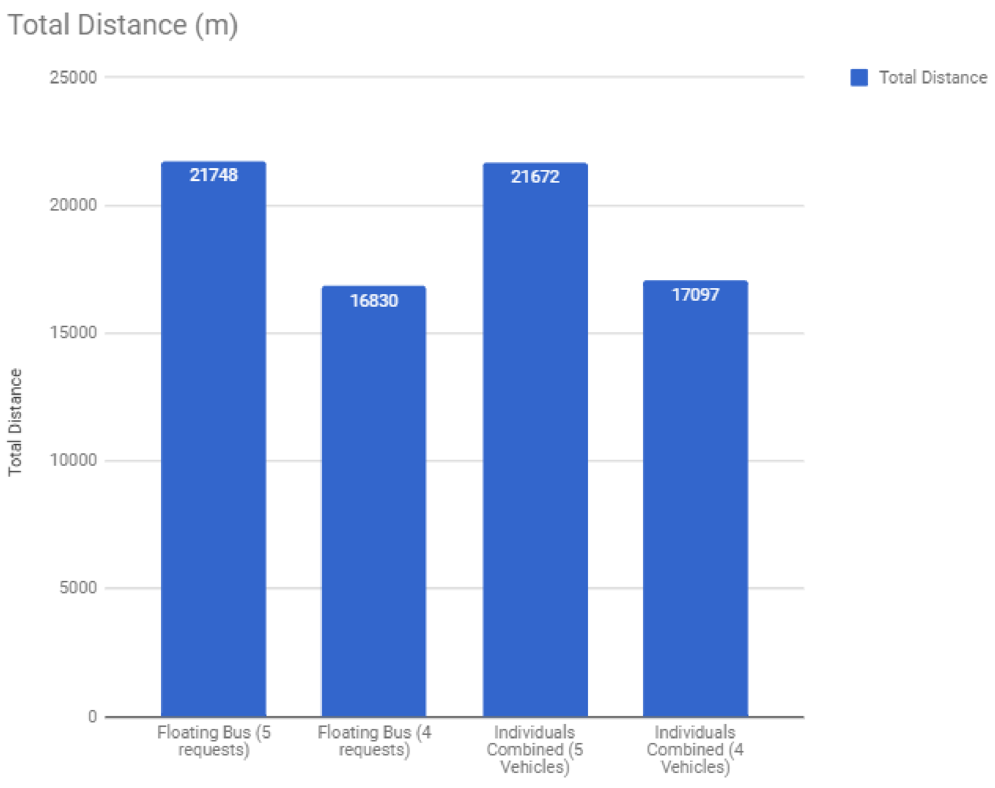
For 5/5 requests

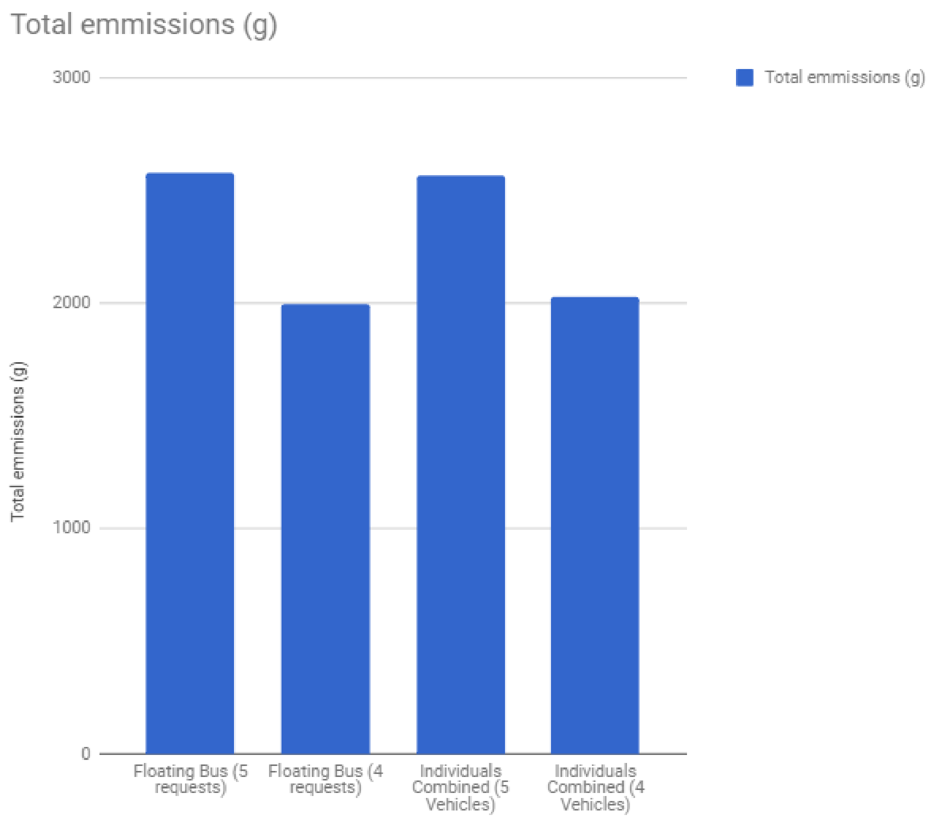
The route info obtained for the above is as follows:

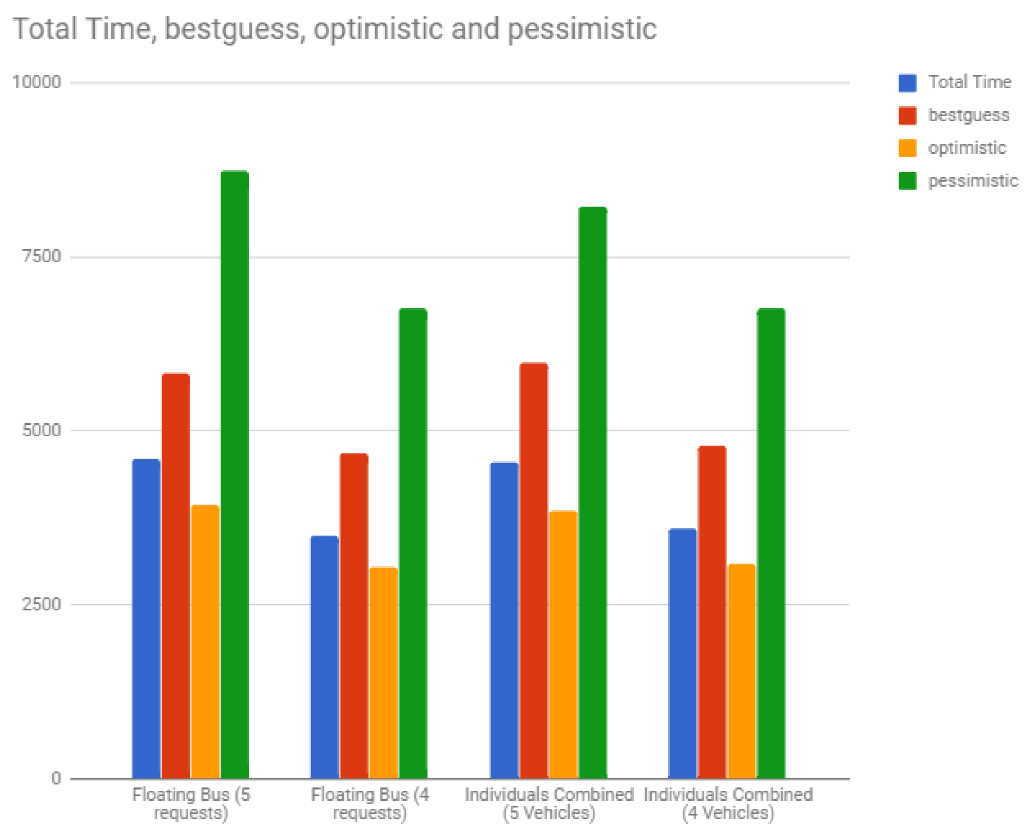
id	distance m	time	bestguess	optimistic	pessimistic	vehicleid	
9371657	21748	4585	5829	3932	8732	9601033	Floating bus 5
9371656	16830	3484	4677	3037	6763	9601032	Floating bus 4
9371655	4848	1034	1389	870	1902	9601031	Individ #1
9371654	4790	1020	1439	885	1984	9601030	Individ #2
9371653	3356	746	1006	668	1461	9601029	Individ #3
9371652	4103	795	953	662	1414	9601028	Individ #4
9371651	4575	951	1178	769	1472	9601027	Individ #5

The route information is combined for the individual buses to compare with the floating buses that service the same requests:

	Total Distance	Total Time	bestguess	optimistic	pessimistic	Total emmissions (g)
Floating Bus (5 requests)	21748	4585	5829	3932	8732	2577.138
Floating Bus (4 requests)	16830	3484	4677	3037	6763	1994.355
Individuals Combined (5 Vehicles)	21672	4546	5965	3854	8233	2568.132
Individuals Combined (4 Vehicles)	17097	3595	4787	3085	6761	2025.9945


Total Distance (m)
Total Distance
25000
20000
15000
10000
5000
0
21748
16830
21672
17097
Floating Bus (5 requests)
Floating Bus (4 requests)
Individuals Combined (5 Vehicles)
Individuals Combined (4 Vehicles)


Total emmissions (g)
Total emmissions (g)
3000
2000
1000
0
Floating Bus (5 requests)
Floating Bus (4 requests)
Individuals Combined (5 Vehicles)
Individuals Combined (4 Vehicles)



From the above data, we can see there is a point when using one floating bus to service multiple requests becomes more inefficient than having multiple vehicle's servicing one request each. The distance travelled overall becomes more than the cumulative distance of each of the individuals. Using this knowledge, we can see that it is definitely within our interests to keep as many vehicles moving along the route and to not leave nearby suitable vehicle idle for new requests.

Chapter 5 - Conclusions

5.1 Findings

From the evaluations of the system, we can see that a dynamic bus routing system has the potential to be a practical solution to improving the aspects we set out to prove. Being able to shorten distance from source and destination locations to their relevant floating bus stops increases the passenger's quality of life by accommodating their requests closer to their sources. We can see that with proper management of the buses the system can cut down on unnecessary distances spent on routes without capacity, but also that an abundance of requests can lead to inefficient routing overall when compared with similar static routes. Finally, it can be seen that there is a turning point where having a single vehicle operating for all requests becomes more inefficient than having multiple vehicles servicing the same requests, meaning that by adding additional floating buses, we can still reduce the overall number of vehicle on the road and still create a more efficient means of servicing the requests received.

5.2 Implementation Issues

The biggest obstacle to implementing the floating bus system was working with Google's API services, with particular issues encountered with regards to hitting query limits. The standard usage limits for both the APIs used in this project are as follows:

Directions API

- 2,500 free directions requests per day.
    - This becomes an increasingly apparent issue as more floating buses are added to the system, and as more requests are made to them, causing their routes to be recalculated each time.

- Up to 23 waypoints allowed in each request, including the origin and final destination.
    - This limit required that floating buses had to deny any additional requests that would exceed this limit for their route. The merging of waypoints in this project helped avoid this issue often, but it is still a concern for vehicles with larger capacity that accomodate a large number of unmerged waypoints.
- 50 requests per second.
    - While this limitation wasn’t encountered during the course of this project, it could become problematic when a large number of floating buses need to make directions requests at the same time.

Distance Matrix API

- 2,500 free elements per day.
    - In the Distance Matrix API, an element is defined as the number of origins multiplied by the number of destinations in a request. For example, a floating bus with its current location as the origin with 5 floating bus stops is five elements.
    - This becomes an issue quickly as traffic data is collected for each routing request, specifically because to collect the three different categories of traffic data, a separate request has to be made for each traffic model to get their results. Taking the above example into account, one route calculation with five elements becomes fifteen elements after all traffic calculations are completed.
- Maximum of 25 origins or 25 destinations per request.
    - Similar to the Directions API, however the distance matrix uses origin/destination pairs, meaning for every sub section of the overall route between destinations, the previous destination is used as an origin (ie: A to B, B to C, where B is destination then origin point). This is alleviated in the same way as Directions thanks to the merging process and upper destination limits.
- 100 elements per request and 100 elements per second.

    - This limit wasn’t encountered, and is unlikely to occur thanks to the measures taken in preventing the destination limit exceeding the previous limit’s amount. Since there is only ever one origin in a floating bus route request, the number of elements per request is equal to the number of destinations.

Maps JavaScript API

- 25,000 map loads per 24 hours.
    - The map load limit is a negligible concern for the floating bus system, since the only use of it is in the control panel HTML page.

5.3 Future Work

There is a huge amount of scope for future work in the area of Dynamic Bus Routing. Now that we have numerical figures that can be used to represent how well the floating bus system works compared to the traditional public bus service, we can start to identify areas that could improve the overall system and have a baseline with which to compare modifications to the overall system with.

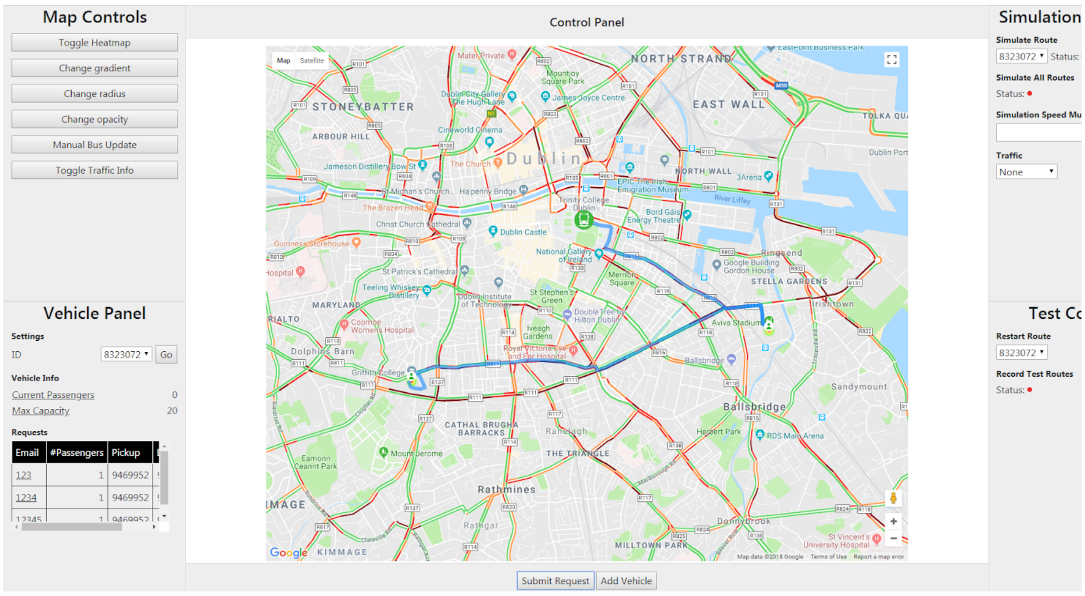
Future work projects that I see being highly beneficial to the area in a practical sense are:

- Integrating existing public architecture into generation of floating bus stops.
    - While the floating bus right now works similarly to a taxi service in that it can stop on any road segment it needs to, there are legal issues with the creation of bus stops that need to be accounted for. Gardaí and Council approval is needed before new traditional bus stops can be implemented, and the same requirement could be required for floating buses if implemented.
- Improved Routing
    - Routing could certainly be improved by implementing bus lanes and removing the various Google API limitations when plotting routes.

Routing could also be improved by using road network distance over using Euclidean distance between waypoints.

Appendix A

User Interface: Control Panel



Settings

ID 8323072 Go

Vehicle Info

Current Passengers	0
Max Capacity	20

Requests

Email	#Passengers	Pickup
123	1	9469952
1234	1	9469952
12345	1	9469952

Simulate Route: 8323072 Status:

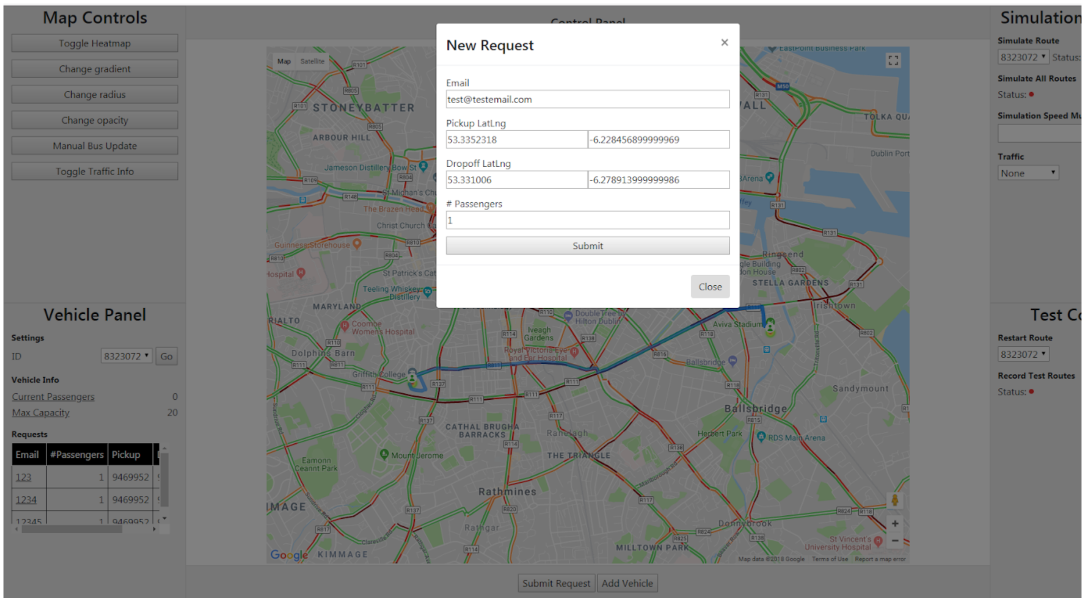
Simulate All Routes Status: ●

Traffic: None

Restart Route: 8323072

Record Test Routes Status: ●

Control Panel: Overview



New Request

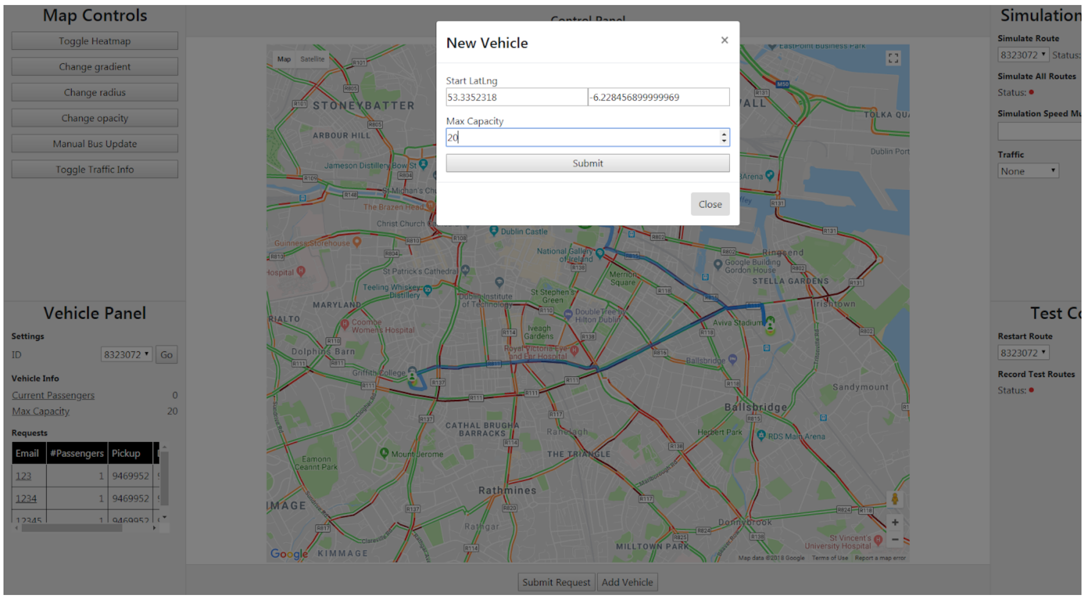
Email: test@testemail.com

Pickup LatLng: 53.3352318 -6.228456899999969

Dropoff LatLng: 53.331006 -6.278913999999986

Passengers: 1

Control Panel: Submitting a new request



New Vehicle ×

Start LatLng: 53.3352318 | -6.228456899999969

Max Capacity: 20

Submit

Settings

ID 8323072 Go

Vehicle Info

Current Passengers 0

Max Capacity 20

Requests

Email	#Passengers	Pickup
123	1	9469952
1234	1	9469952
12345	1	9469952

Simulate Route: 8323072 Status:

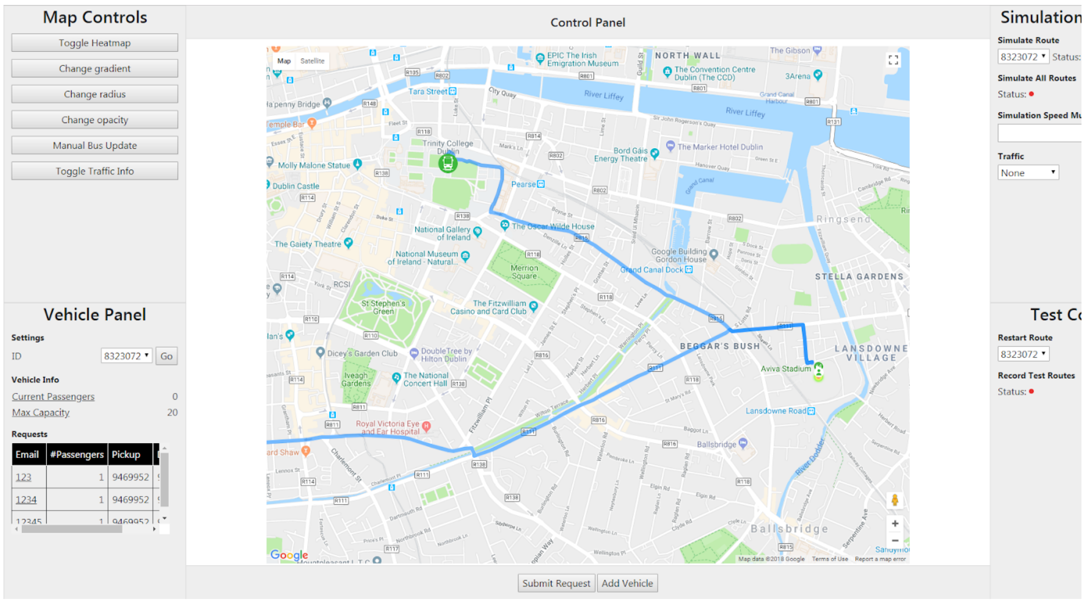
Simulate All Routes Status: ●

Traffic: None

Restart Route: 8323072

Record Test Routes Status: ●

Control Panel: Creating a new Vehicle



Settings

ID 8323072 Go

Vehicle Info

Current Passengers 0

Max Capacity 20

Requests

Email	#Passengers	Pickup
123	1	9469952
1234	1	9469952
12345	1	9469952

Simulate Route: 8323072 Status:

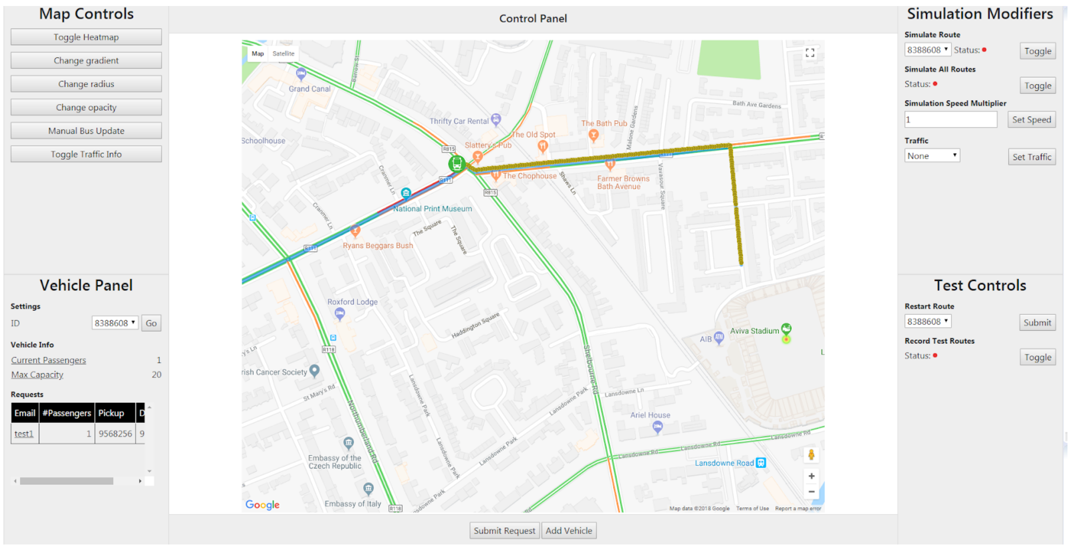
Simulate All Routes Status: ●

Traffic: None

Restart Route: 8323072

Record Test Routes Status: ●

Control Panel: Viewing a Vehicle's Route



Settings

ID 8388608

Vehicle Info

Current Passengers 1

Max Capacity 20

Requests

Email	#Passengers	Pickup	D
test1	1	9568256	9

Simulate Route

8388608 Status: ●

Simulate All Routes

Status: ●

Simulation Speed Multiplier

1

Traffic

None

Restart Route

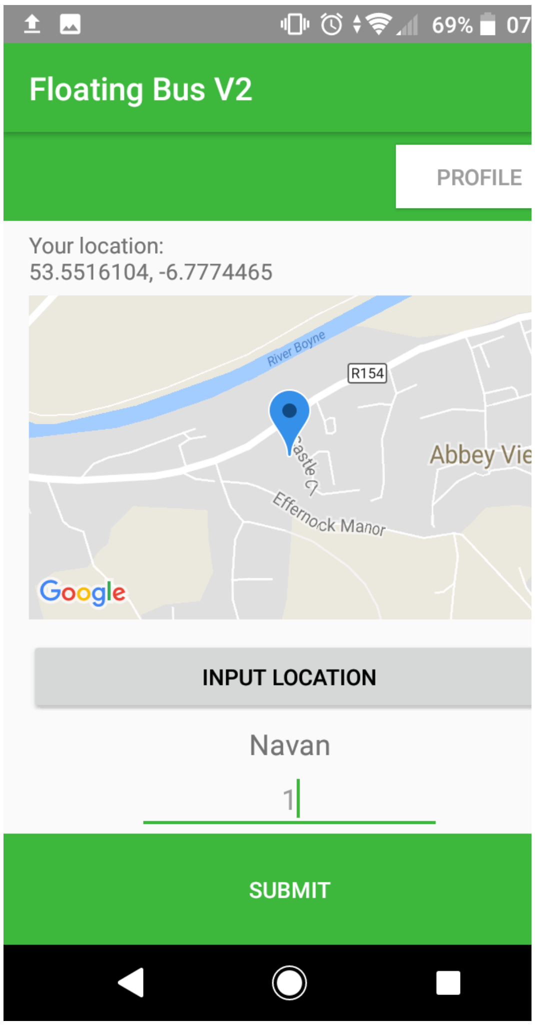
8388608

Record Test Routes

Status: ●

Control Panel - Vehicle Mid Simulation w/ breadcrumbing

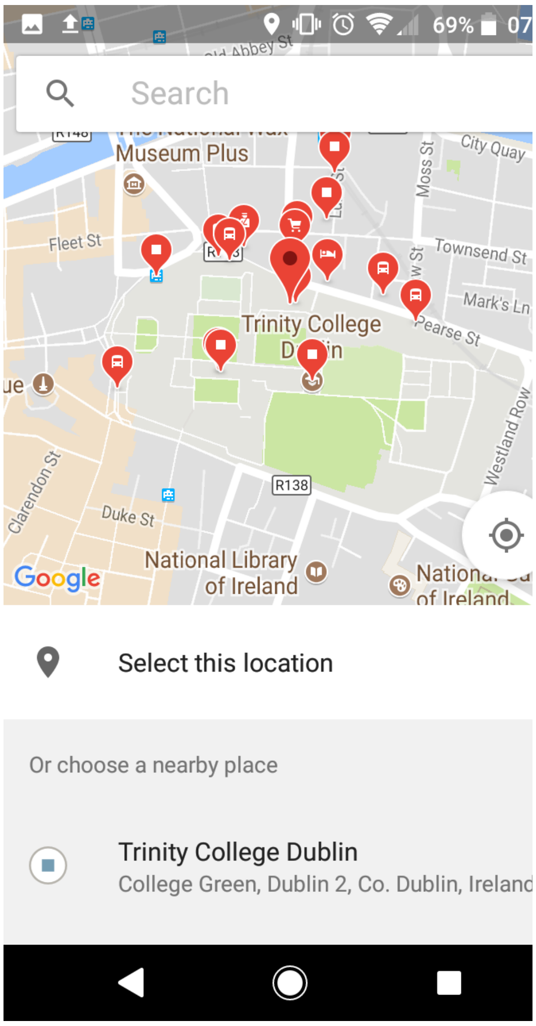
User Interface: Android Client



Your location:
53.5516104, -6.7774465

Navan

1

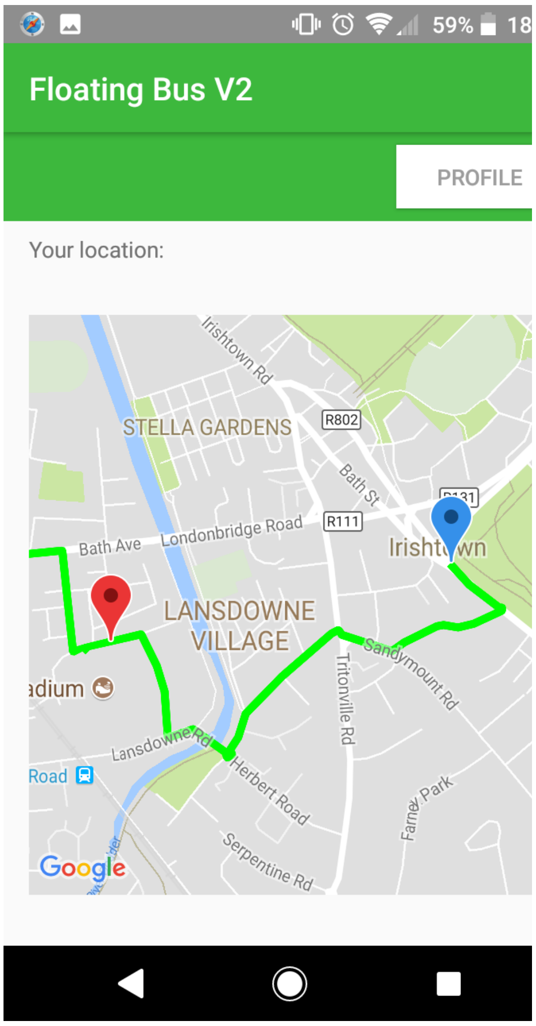


Or choose a nearby place

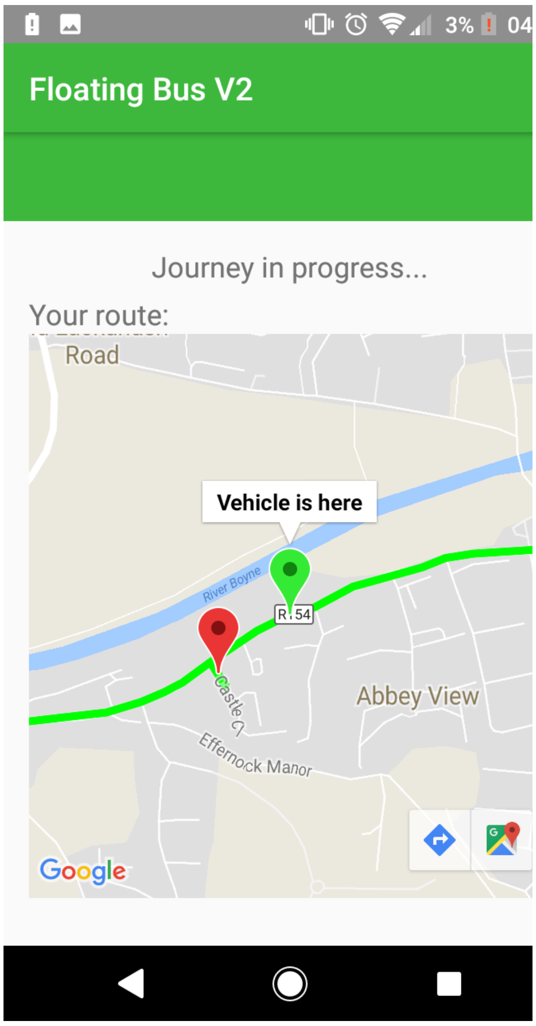
Trinity College Dublin
College Green, Dublin 2, Co. Dublin, Ireland

Main Activity

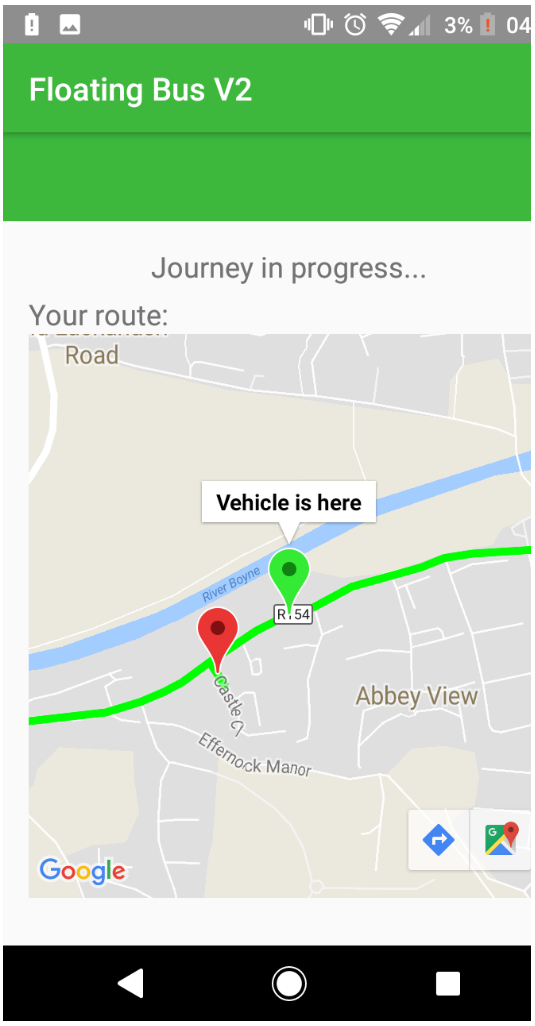
Place Picker Widget



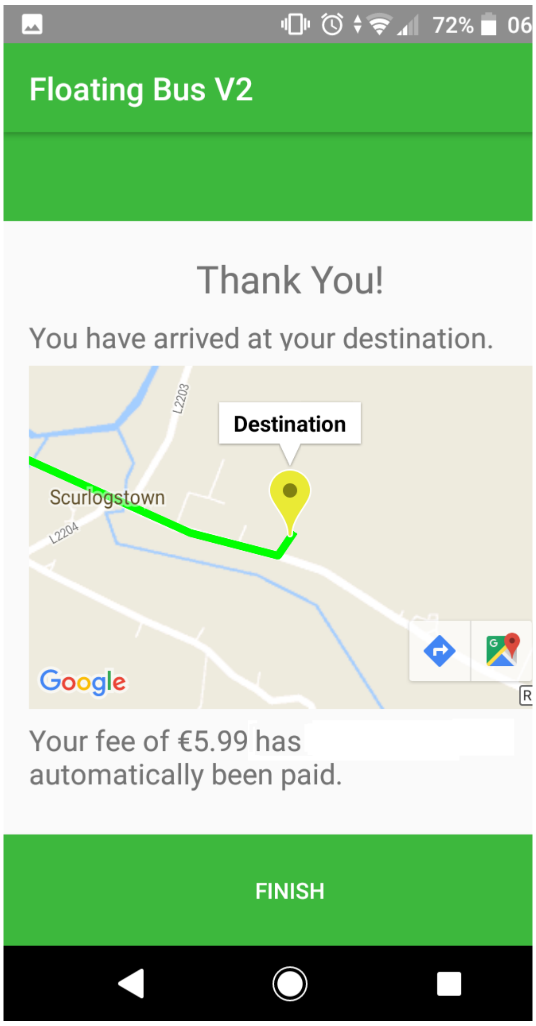
Driving Activity



Travelling Activity



Travelling Activity

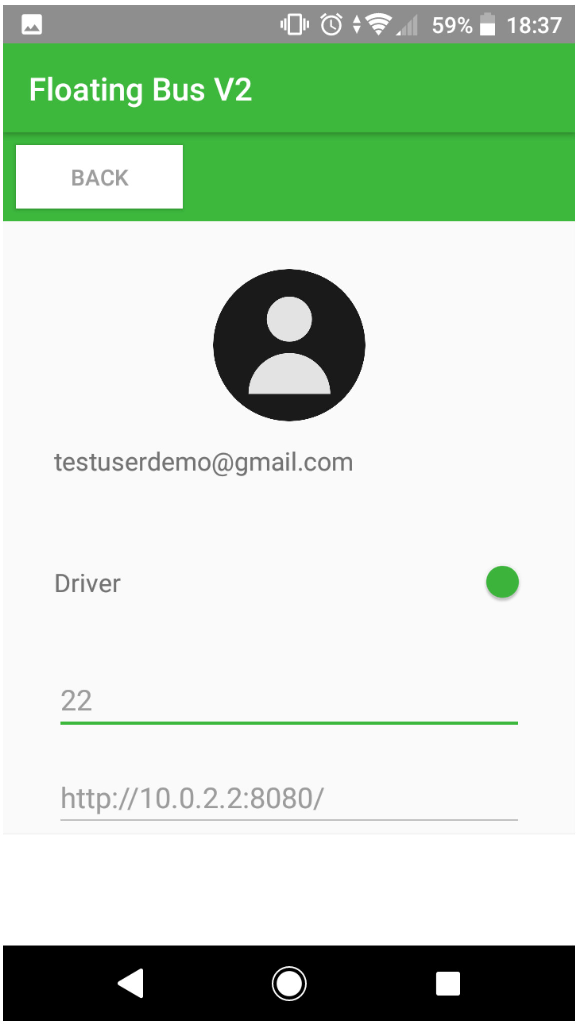


Thank You!

You have arrived at your destination.

Your fee of €5.99 has automatically been paid.

Completed Activity



testuserdemo@gmail.com

Driver

22

http://10.0.2.2:8080/

Profile Activity

Appendix B

Sample Dublin Bus Route Data XML

Route 757 Towards - Dublin Airport

```xml
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<gmap>
  <config>
    <size>
      <width>500</width>
      <height>500</height>
    </size>
    <center>
      <poi>
        <city>dublin</city>
        <country>Republic of Ireland</country>
        <gpoint>
          <lat>53.33378</lat>
          <lng>-6.263919</lng>
        </gpoint>
        <icon>/Global/Icons/image.png</icon>
        <stopnumber>1074</stopnumber>
        <address>Charlotte Way</address>
        <location>Camden Court Hotel</location>
        <iconshadow>/Global/Icons/shadow.png</iconshadow>
      </poi>
    </center>
    <zoom>12</zoom>
  </config>
  <data>
    <poi>
      <city>dublin</city>
      <country>Republic of Ireland</country>
      <gpoint>
        <lat>53.33378</lat>
        <lng>-6.263919</lng>
      </gpoint>
      <icon>/Global/Icons/image.png</icon>
      <stopnumber>1074</stopnumber>
      <address>Charlotte Way</address>
      <location>Camden Court Hotel</location>
      <iconshadow>/Global/Icons/shadow.png</iconshadow>
    </poi>
    <poi>
      <city>dublin</city>
    <country>Republic of Ireland</country>
    <gpoint>
      <lat>53.335893</lat>
      <lng>-6.25752</lng>
    </gpoint>
    <icon>/Global/Icons/image.png</icon>
    <stopnumber>6074</stopnumber>
    <address>Earlsfort Tce</address>
    <location>Leeson Street</location>
    <iconshadow>/Global/Icons/shadow.png</iconshadow>
  </poi>
  <poi>
    <city>dublin</city>
    <country>Republic of Ireland</country>
    <gpoint>
      <lat>53.338513</lat>
      <lng>-6.255004</lng>
    </gpoint>
    <icon>/Global/Icons/image.png</icon>
    <stopnumber>748</stopnumber>
    <address>Merrion Row</address>
    <location>Huguenot Cemetery</location>
    <iconshadow>/Global/Icons/shadow.png</iconshadow>
  </poi>
  <poi>
    <city>dublin</city>
    <country>Republic of Ireland</country>
    <gpoint>
      <lat>53.340067</lat>
      <lng>-6.25192</lng>
    </gpoint>
    <icon>/Global/Icons/image.png</icon>
    <stopnumber>2905</stopnumber>
    <address>Merrion Sq West</address>
    <location>Natural History Museum</location>
    <iconshadow>/Global/Icons/shadow.png</iconshadow>
  </poi>
  <poi>
    <city>dublin</city>
    <country>Republic of Ireland</country>
    <gpoint>
      <lat>53.341417</lat>
      <lng>-6.251671</lng>
    </gpoint>
    <icon>/Global/Icons/image.png</icon>
    <stopnumber>494</stopnumber>
    <address>Clare Street</address>
    <location>Lincoln Place</location>
    <iconshadow>/Global/Icons/shadow.png</iconshadow>
  </poi>
```

```xml
    <poi>
      <city>dublin</city>
      <country>Republic of Ireland</country>
      <gpoint>
        <lat>53.343586</lat>
        <lng>-6.249726</lng>
      </gpoint>
      <icon>/Global/Icons/image.png</icon>
      <stopnumber>495</stopnumber>
      <address>Westland Row</address>
      <location>Pearse Station</location>
      <iconshadow>/Global/Icons/shadow.png</iconshadow>
    </poi>
    <poi>
      <city>dublin</city>
      <country>Republic of Ireland</country>
      <gpoint>
        <lat>53.345022</lat>
        <lng>-6.254255</lng>
      </gpoint>
      <icon>/Global/Icons/image.png</icon>
      <stopnumber>7588</stopnumber>
      <address>Pearse Street</address>
      <location>Tara Street</location>
      <iconshadow>/Global/Icons/shadow.png</iconshadow>
    </poi>
    <poi>
      <city>dublin</city>
      <country>Republic of Ireland</country>
      <gpoint>
        <lat>53.349758</lat>
        <lng>-6.252437</lng>
      </gpoint>
      <icon>/Global/Icons/image.png</icon>
      <stopnumber>4717</stopnumber>
      <address>Amiens Street</address>
      <location>Bus Aras</location>
      <iconshadow>/Global/Icons/shadow.png</iconshadow>
    </poi>
    <poi>
      <city>dublin</city>
      <country>Republic of Ireland</country>
      <gpoint>
        <lat>53.348064</lat>
        <lng>-6.247145</lng>
      </gpoint>
      <icon>/Global/Icons/image.png</icon>
      <stopnumber>2499</stopnumber>
      <address>Custom House Quay</address>
      <location>Jurys Hotel</location>
    <iconshadow>/Global/Icons/shadow.png</iconshadow>
  </poi>
  <poi>
    <city>dublin</city>
    <country>Republic of Ireland</country>
    <gpoint>
      <lat>53.34779</lat>
      <lng>-6.242869</lng>
    </gpoint>
    <icon>/Global/Icons/image.png</icon>
    <stopnumber>7216</stopnumber>
    <address>North Wall Quay</address>
    <location>Guild Street</location>
    <iconshadow>/Global/Icons/shadow.png</iconshadow>
  </poi>
  <poi>
    <city>dublin</city>
    <country>Republic of Ireland</country>
    <gpoint>
      <lat>53.347373</lat>
      <lng>-6.236342</lng>
    </gpoint>
    <icon>/Global/Icons/image.png</icon>
    <stopnumber>2501</stopnumber>
    <address>North Wall Quay</address>
    <location>New Wapping Street</location>
    <iconshadow>/Global/Icons/shadow.png</iconshadow>
  </poi>
  <poi>
    <city>dublin</city>
    <country>Republic of Ireland</country>
    <gpoint>
      <lat>53.346855</lat>
      <lng>-6.228846</lng>
    </gpoint>
    <icon>/Global/Icons/image.png</icon>
    <stopnumber>7623</stopnumber>
    <address>East Wall Rd</address>
    <location>3 Arena</location>
    <iconshadow>/Global/Icons/shadow.png</iconshadow>
  </poi>
  <poi>
    <city>dublin</city>
    <country>Republic of Ireland</country>
    <gpoint>
      <lat>53.426784</lat>
      <lng>-6.240497</lng>
    </gpoint>
    <icon>/Global/Icons/image.png</icon>
    <stopnumber>7401</stopnumber>
```

```xml
      <address>Dublin Airport</address>
      <location>Terminal 2</location>
      <iconshadow>/Global/Icons/shadow.png</iconshadow>
    </poi>
    <poi>
      <city>dublin</city>
      <country>Republic of Ireland</country>
      <gpoint>
        <lat>53.428116</lat>
        <lng>-6.244116</lng>
      </gpoint>
      <icon>/Global/Icons/image.png</icon>
      <stopnumber>3665</stopnumber>
      <address>Dublin Airport</address>
      <location>Terminal 1</location>
      <iconshadow>/Global/Icons/shadow.png</iconshadow>
    </poi>
  </data>
</gmap>
```

Sample Dublin Bus Route Data Formatted CSV

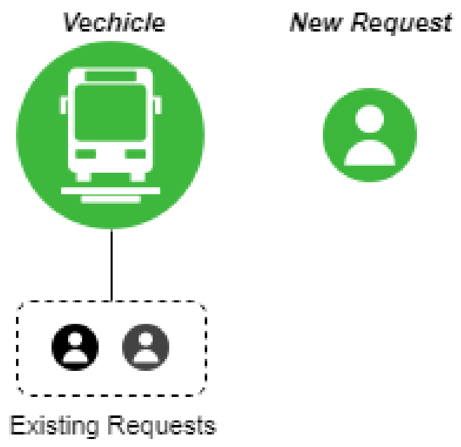
Route 757 Towards - Dublin Airport

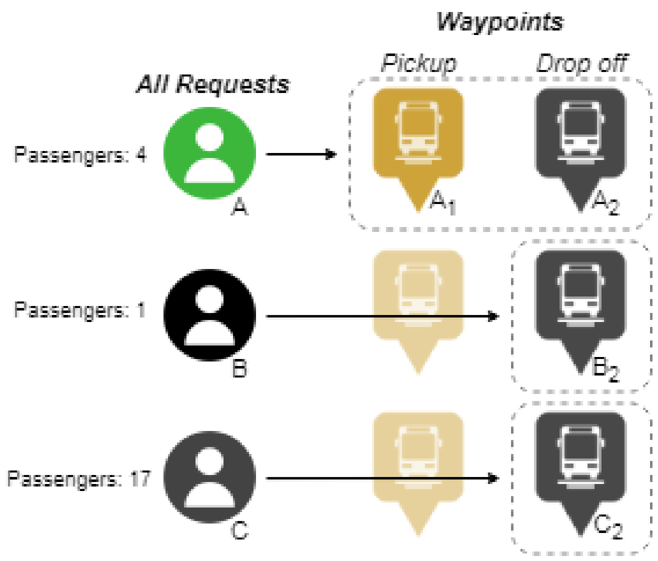
Id	S Lat	S Lng	D Lat	D Lng	S Address	D Address	Amount	email
1	53.33378	-6.263919	53.33378	-6.263919	1074	1074	1	Route 757 Towards - Dublin Airport.csv_1
2	53.33378	-6.263919	53.335893	-6.25752	1074	6074	1	Route 757 Towards - Dublin Airport.csv_2
3	53.335893	-6.25752	53.338513	-6.255004	6074	748	1	Route 757 Towards - Dublin Airport.csv_3
4	53.338513	-6.255004	53.340067	-6.25192	748	2905	1	Route 757 Towards - Dublin Airport.csv_4
5	53.340067	-6.25192	53.341417	-6.251671	2905	494	1	Route 757 Towards - Dublin Airport.csv_5
6	53.341417	-6.251671	53.343586	-6.249726	494	495	1	Route 757 Towards - Dublin Airport.csv_6

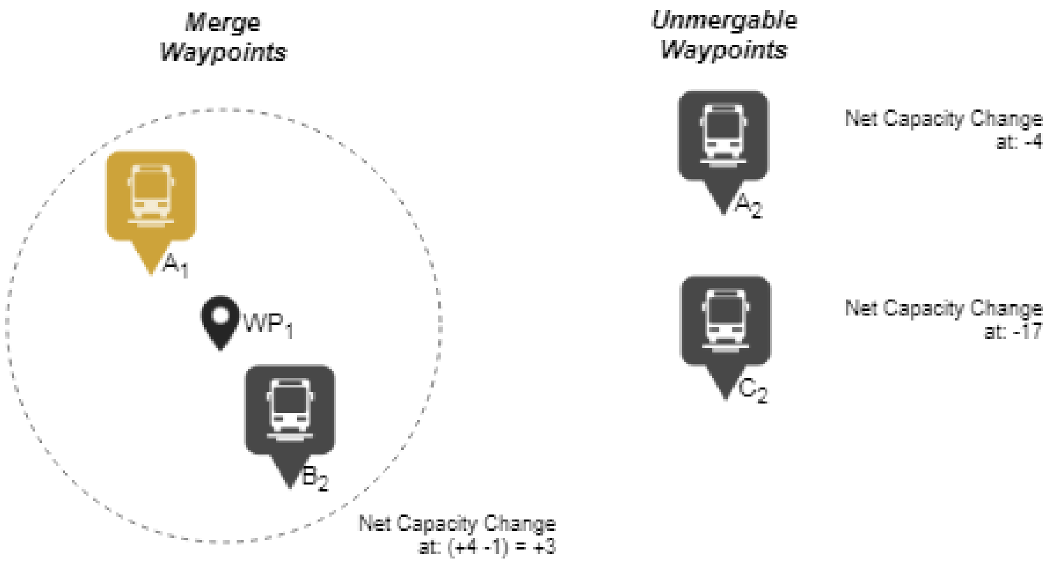
7	53.343586	-6.249726	53.345022	-6.254255	495	7588	1	Route 757 Towards - Dublin Airport.csv_7
8	53.345022	-6.254255	53.349758	-6.252437	7588	4717	1	Route 757 Towards - Dublin Airport.csv_8
9	53.349758	-6.252437	53.348064	-6.247145	4717	2499	1	Route 757 Towards - Dublin Airport.csv_9
10	53.348064	-6.247145	53.34779	-6.242869	2499	7216	1	Route 757 Towards - Dublin Airport.csv_10
11	53.34779	-6.242869	53.347373	-6.236342	7216	2501	1	Route 757 Towards - Dublin Airport.csv_11
12	53.347373	-6.236342	53.346855	-6.228846	2501	7623	1	Route 757 Towards - Dublin Airport.csv_12
13	53.346855	-6.228846	53.426784	-6.240497	7623	7401	1	Route 757 Towards - Dublin Airport.csv_13
14	53.426784	-6.240497	53.428116	-6.244116	7401	3665	1	Route 757 Towards - Dublin Airport.csv_14

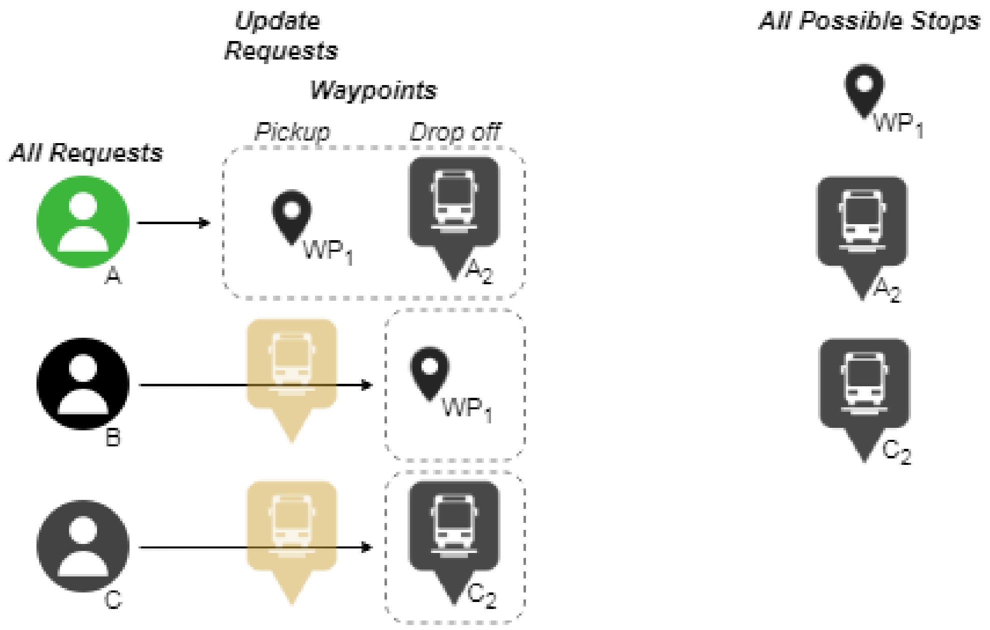
Appendix C

Full Routing and Merging Process Illustrated

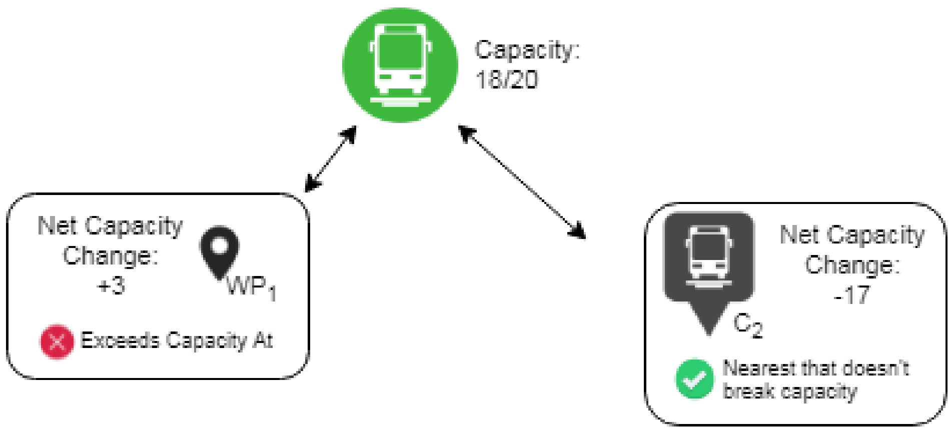




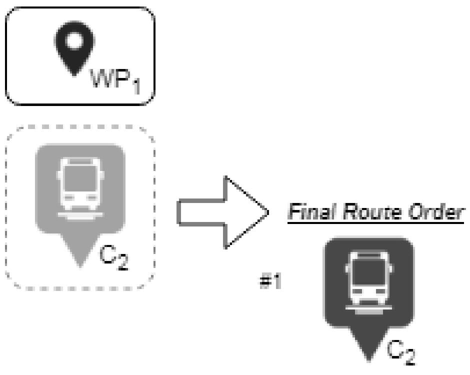

Merge Waypoints
A1
WP1
B2
Net Capacity Change at: (+4 -1) = +3
Unmergable Waypoints
A2
Net Capacity Change at: -4
C2
Net Capacity Change at: -17

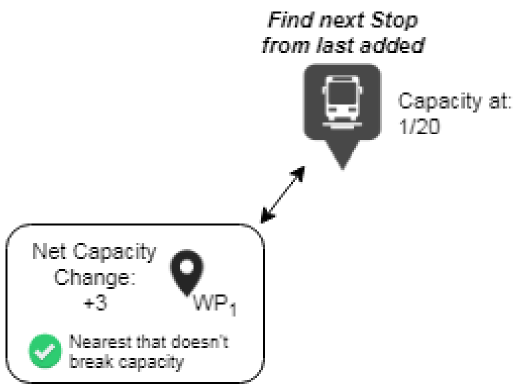

Update Requests
Waypoints
Pickup
Drop off
All Requests
A
WP1
A2
B
WP1
C
C2
All Possible Stops
WP1
A2
C2

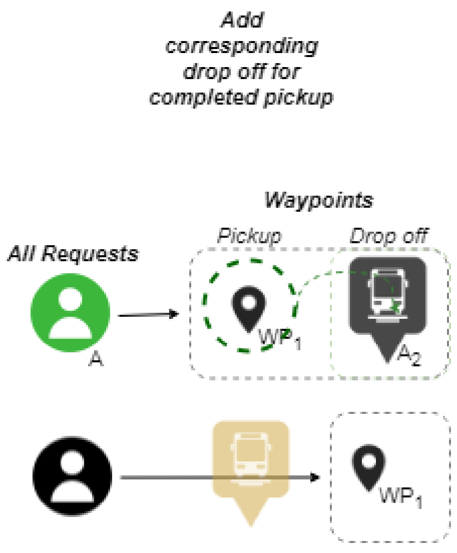
Find First Stop

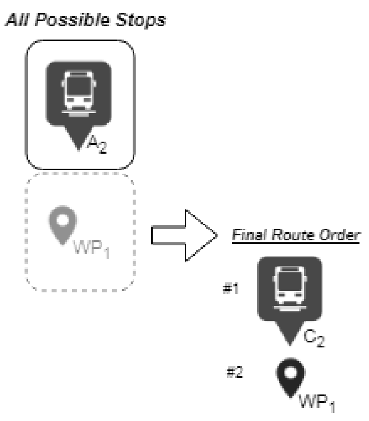


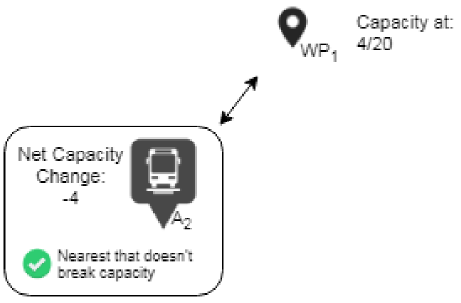
All Possible Stops

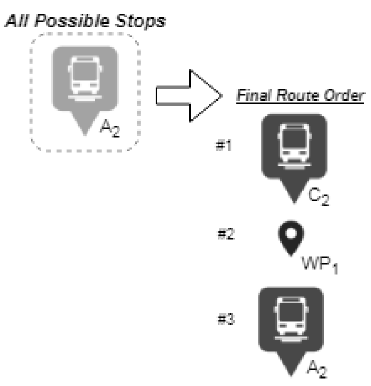


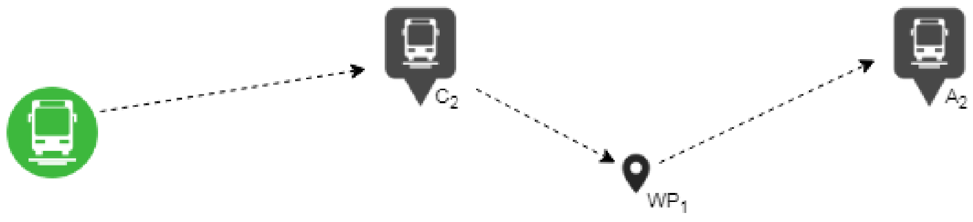

Find next Stop from last added
Capacity at: 1/20
Net Capacity Change: +3
WP_1
Nearest that doesn't break capacity


Add corresponding drop off for completed pickup
Waypoints
Pickup
Drop off
All Requests
A
WP_1
A_2
WP_1


All Possible Stops
A_2
WP_1
Final Route Order
#1
C_2
#2
WP_1


WP_1
Capacity at: 4/20
Net Capacity Change: -4
A_2
Nearest that doesn't break capacity


All Possible Stops
A_2
Final Route Order
#1
C_2
#2
WP_1
#3
A_2


C2
WP1
A2

Bibliography

[1] Prof. Siobhán Clarke “Trinity Smart and Sustainable Cities Research Centre”, Trinity College Dublin - https://goo.gl/JXJwfz (Accessed 14th May 2018)

[2] Sarah Burns “Time lost in Dublin traffic costs economy €350m per year”, The Irish Times, Sat, Apr 29, 2017, 11:46

[3] Real Time Passenger Information - http://smartdublin.ie/smartstories/real-time-passenger-information/ , 04/08/2017 (Accessed 14th May 2018)

[4] "The Vehicle Routing Problem with Simultaneous ... - ScienceDirect.com." https://www.sciencedirect.com/science/article/pii/S1877705811024805/pdf?md5=c55979137e9970cf839d2c4fd51971aa&pid=1-s2.0-S1877705811024805-main.pdf&_valck=1 (Last accessed 3 Feb. 2018)

[5] Dynamic Bus Routing - Thomas Kearns, Jordan Kanter and students Adam Weissert, Haidong Fei, and Li Gong as part of the Urban Data Model Prototype Studio at IIT in 2013-2014 - https://softsentience.wordpress.com/2016/04/30/dynamic-bus-routing/ (Last Accessed 23rd May 2018)

[6] Fang Y., Hu X., Wu L., Miao Y. (2010) A Real-Time Scheduling Method for a Variable-Route Bus in a Community. In: Phillips-Wren G., Jain L.C., Nakamatsu K., Howlett R.J. (eds) Advances in Intelligent Decision Technologies. Smart Innovation, Systems and Technologies, vol 4. Springer, Berlin, Heidelberg

[7] “Radically rethinking the bus system” by Stephen Dowling 28 March 2013 BBC Future - http://www.bbc.com/future/story/20130327-new-bus-stop-for-flexible-travel (Last Accessed 23rd May 2018)

[8] “Infographic: A smarter future for bus travel” by Stephen Dowling 28 March 2013 BBC Future - http://www.bbc.com/future/story/20130327-building-bus-stops-via-smartphone (Last Accessed 23rd May 2018)

[9] ASP.NET MVC Overview - Microsoft Docs https://docs.microsoft.com/en-us/previous-versions/aspnet/web-frameworks/dd381412(v=vs.108) (Last Accessed 24th May)

[10] PostGIS - Spatial and Geographic objects for PostgreSQL - https://postgis.net (Last Accessed 24th May)

[11] ACT Issues Policy Guidance Regarding the Definition of Ridesharing - http://actweb.org/wp-content/uploads/2014/12/ACT_PolicyStatement_Definition_of_Ridesharing_for_State_and_Local_Ordinances.pdf (Last accessed 24th May 2018)

[12] "Defining "Ridesharing:" A Guide for Reporters, Legislators, and Regulators - http://actweb.org/wp-content/uploads/2014/11/Ridesharing-Definition-Release_091714v2.pdf (Last accessed 24th May 2018)

[13] Introduction to Ridesharing: Overview of definitions and setting the stage - http://actweb.org/wp-content/uploads/2014/12/ACT_Powerpoint_Ridesharing_Shaheen.pdf (Last accessed 24th May 2018)

[14] "Fast Detour Computation for Ride Sharing" - Robert Geisberger, Dennis Luxen, Sabine Neubauer, Peter Sanders, and Lars Volker - Karlsruhe Institute of Technology, 76128 Karlsruhe, Germany

[15] "Reliability-Based Transit Assignment for Congested Stochastic Transit Networks" - by W. Y. Szeto, Muthu Solayappan, Yu Jiang

[16] "The allocation of buses in heavily utilized networks with overlapping routes" - by Anthony F. Han, Nigel H.M. Wilson

[17] "What Are RESTful Web Services?" - Oracle - "https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html (Last accessed 24th May 2018)

[18] "Spring Framework Documentation" - https://docs.spring.io/spring/docs/current/spring-framework-reference/index.html (Last accessed 24th May 2018)

[19] "w3schools.com HTTP Request Methods" - https://www.w3schools.com/tags/ref_httpmethods.asp (Last accessed 24th May 2018)

[20] "w3schools.com JSON - Introduction" - https://www.w3schools.com/js/js_json_intro.asp (Last accessed 24th May 2018)

[21] "Google Maps Platform Directions API" - https://developers.google.com/maps/documentation/directions/intro (Last accessed 24th May 2018)

[22] "Google Maps Platform Distance Matrix API" - https://developers.google.com/maps/documentation/distance-matrix/intro (Last accessed 24th May 2018)

[23] "Google Maps Platform Maps JavaScript API" - https://developers.google.com/maps/documentation/javascript/tutorial (Last accessed 24th May 2018)