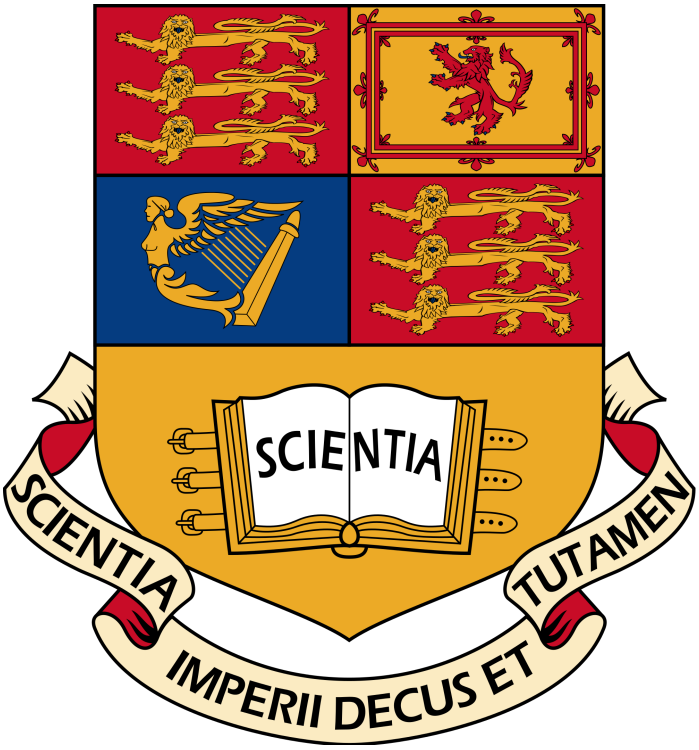
Active Delay Warning Transport App

Imperial College London BEng JMC Individual Project



Produced by

KARL CHAN

under the supervision of

DR. PETER MCBRIEN (supervisor)

DR. THOMAS HEINIS (2nd marker)

Last revised:

15th Jun, 2016

Imperial College London

TRANSPORT FOR LONDON

This project aims to provide better predictions on bus journey times in Central London, taking into account of real time delays, and informing users of alternative routes through an interactive, location-aware mobile app in the event of a delay.

The first stage involves collecting bus arrivals data from the TfL Unified API, and cleaning of noisy data. We then explore patterns in the historic bus journey times, such as periodicities and autocorrelation, in order to obtain clues of how to reuse them. For real time delays, we examine past delays for each pair of adjacent bus stops, and characterise them into lookup tables so that future delays can be quickly assessed.

Finally, we piece together a number of historic and real-time based prediction methods to battle against each other. Results suggest that historic travel times for weekdays and weekends should be treated differently, and that historic data considered in prediction generation should not be too distant. Real-time based methods are comparable to historic based prediction methods, and are significantly superior during holidays.

Acknowledgements

My sincere thanks to Dr. Peter McBrien, my supervisor, for his wholehearted support, guidance, time and patience throughout the project; CSG, in particular Mr. Thomas Joseph, for assisting me through all the technical difficulties; my parents for their unconditional love and support, for whom I am forever in debt; my friends who have been my inspiration; without all of you this project would not have come to life.

Contents

1. Introduction

1.1. What is this project about?

London is a metropolitan with more than 3.5 million vehicles on the road[1][1]. Traffic congestion in London has long been an issue, and it has even been reported as the 'most congested city in Europe'[2]. As a result, bus journey times can vary a lot depending the time of the day, particularly between rush hours and in the middle of the night.

With billions of bus journeys made on the transport network in London[2][3], it is clear that even the slightest amount of time that can be saved per bus journey will be of immense benefit to the population. If commuters could get hold of estimates of their travel time with less uncertainty, they could spend less time waiting for a bus, take alternative routes to avoid getting stuck in a traffic jam, be on time for work or meetings, thereby reducing their cost in commuting.

While there are many friendly-to-use apps that perform route planning on the bus network across Central London, such as Citymapper, Google Maps, and TfL[3] Route Planner (section 2.4), they all focus on forwarding real time live feeds of bus arrivals to users, but fail to utilise the potential of historic arrival times[4][4], which when under suitable analysis could potentially give better arrival predictions than live feeds standalone, and detect anomalies in current journey times.

Therefore, this project aims to analyse real time and historic data of bus arrivals, so as to give more accurate predictions of bus journey times. An easy to use interface should also be developed, so that should a delay be detected, affected users should be warned and alternative routes suggested.

1.2. Why is this problem difficult?

One of the challenges lie in efficient handling of large data sets. With more than 600 bus routes and 19000 stops in Central London, millions of rows and Gigabytes of data must be handled, and extensive computing resources are required to monitor, analyse data and service user requests real-time at the same time.

[1] According to Transport for London, 3,530,399 unique vehicles entered the Congestion Charging Zone during charging hours in December, 2015.

[2] According to Transport for London, 2.397 billion bus journeys were made in London in the year 2013-2014.

[3] From now on we will use TfL as shorthand for Transport for London.

[4] According to Shashi Verma, director of customer experience at TfL, "Citymapper and other transport apps don't do any further analysis of the data - it's all official information coming directly from London's transport system".

Since arrivals predictions of the entire bus network (nearly 100,000 arrival predictions[5]) come in every 10 seconds, the program must be able to absorb and classify all information within the 10 second window. Under limited resources[6] the program must be both computationally and memory efficient. We would also like to ultimately present a frontend app to users for real-time route planning and management, and hence latency between data processing and discovery of delays should be kept low.

Another difficulty lies in data analysis. While the heart of the project involves working with both historic and real-time data, gathering an organised and usable historic dataset has proved challenging. The live feeds of bus arrivals predictions from TfL are noisy in nature, and care has to be taken to filter out invalid predictions. Irregularities in historic data will also need to be carefully examined, such as breakdown, early termination or diversion of buses (section 4).

A handful of studies [5, 6, 7, 8, 9, 10, 11, 12] can be found in literature on bus arrivals predictions (section 2.5), where sophisticated statistical and machine learning techniques have been applied. Nevertheless, they are experimental in nature and are confined to individual bus routes where the geography along the routes are studied in-depth, and extra devices installed on buses to collect further data. This unfortunately is not possible in our situation, given the vast amount of bus routes to cover and limited resources available for this project. Therefore, this project will be one of a kind in achieving large scale monitoring and analysis of buses.

1.3. What we set out to achieve

This project begins with building a historic database of all bus arrivals in Central London. We monitor the current arrivals of all bus stops, filter out noisy predictions that we find invalid, before logging them to our database (section 4).

The next stage is then to analyse historic data, such as the average journey times in between two stops, and identify any trends with regards to the time of the day or day of the week. We start off by employing basic statistical analysis, looking at historic means, standard deviations, periods and autocorrelations, and compare them to our real-time measurements in search for delays (section 5). Based on initial results we devise prediction methods and evaluate them under appropriate error measures (section 6). Should the results be dissatisfactory, or time permitting, we will be looking into more advanced approaches as suggested in the literature (section 2.5).

The final stage is to produce an interactive app that allows route planning and delay notification, which should generate journey time predictions based on our analysis. Should a delay along the user's route be detected, he / she should also be informed of how much longer the journey is expected to take, the new estimated arrival time, and any quicker alternative routes that can be

[5] Assuming 19,000 stops and the top 5 arrivals predictions of each stop being monitored, this amounts to $19,000 \times 5 = 95,000$ arrivals predictions in total per request.

[6] Programs are distributed across 3 VMs provided by CSG - each with 4 cores and 8GB of RAM.

taken together with their estimated arrival times.

2. Background

2.1. The TfL Bus Network

The bus network in London is extraordinary in many ways. London as a compact and highly populated city is served by more than 600 bus routes. As a result, many have overlapping route sections, from high traffic areas in the city centre such as Oxford and Regent Street, to suburbs such like Tubbenden Lane in Orpington.

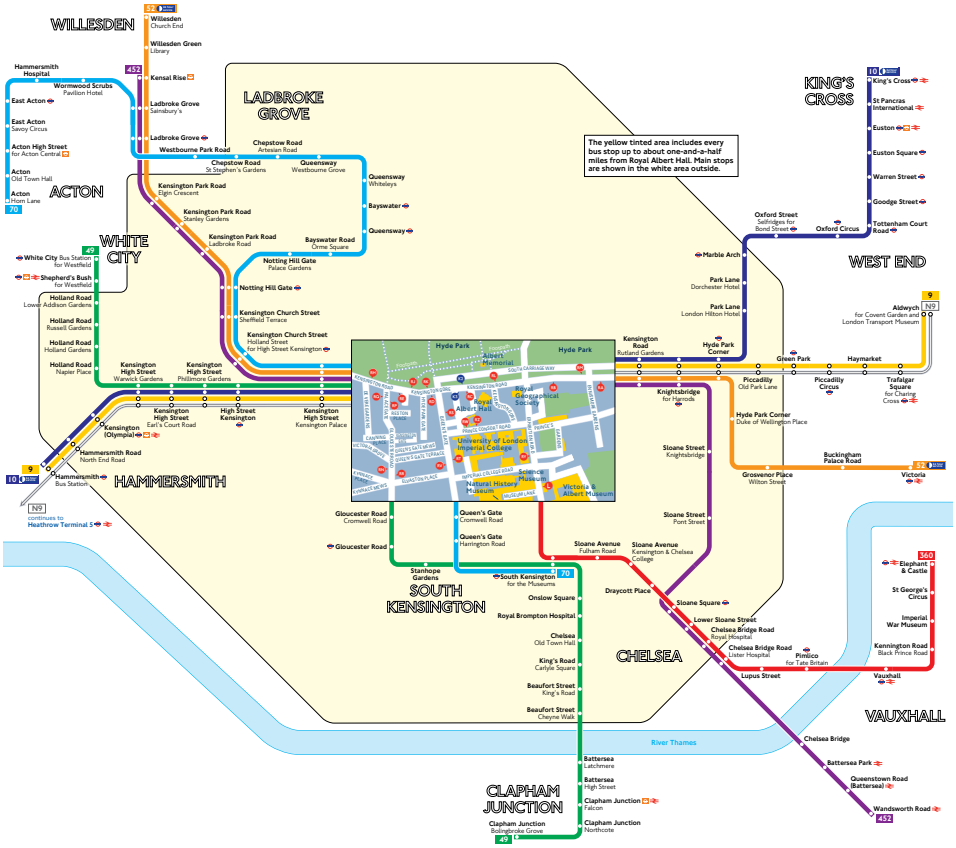


Figure 2.1.: Bus spider map showing bus routes that serve Royal Albert Hall. Only major stops are shown. It can be seen that routes 9 and 10 share the section between Hammersmith and Hyde Park Corner.
Courtesy of Transport for London. Retrieved from *https://tfl.gov.uk/maps_/bus-spider-maps*. Last accessed 28 May, 2016.

A notable example is routes 9 and 10, where they share a total of 18 stops between Hammersmith and Hyde Park Corner (figure 2.1). Within this section both routes share the same bus stops, and passengers commuting within the section will be indifferent to the route number, boarding whichever bus that comes first. The presence of shared route sections allows us to aggregate data from different bus routes to obtain a larger data set and perform more reliable analysis.

2.1.1. The iBus vehicle tracking system

All buses in Central London are equipped with the iBus Automatic Vehicle Location system[13]. It utilises various technologies including the Global Positioning System (GPS), odometers, speedometers, turn-rate sensors and rate gyro in order to pinpoint the location of buses in operation to an accuracy of 100 metres[14]. The data derived allows the TfL central system to automatically and continuously issue predictions for all bus stops and routes in Central London.

2.1.2. Open data

Traditionally, bus predictions have suffered from limited access, having only been circulated within TfL and emergency services. In 2011 TfL has taken the initiative to make bus predictions and route information accessible on the internet and through their unified API, details of which to be discussed in section 2.2. This has given rise to 3rd party apps such as CityMapper (section 2.4.1) that integrates everything in one place, so that users can have easy access to live bus information wherever they are on their mobiles. Since all buses in Central London are managed by TfL, effectively data of all bus routes serving London can be acquired.

The information contained in the developers' version of predictions is richer than what one would usually expect. Apart from the route number and expected time to arrival, they also contain the registration number of the bus, which makes it possible to track down a bus and log travel times[7].

[7] Formally, a prediction is a JSON entity that consists of the fields:

- `lineId` - The bus route that the vehicle is currently serving
- `direction` - The direction of the bus route that the vehicle is serving (either inbound or outbound)
- `vehicleId` - The registration number of the bus
- `naptanId` - The NaPTAN identifier of the stop that the bus is approaching
- `expectedArrival` - The expected arrival time of the bus at the stop
- `timestamp` - A timestamp generated by the TfL server referencing the time of creation of the prediction

2.2. TfL Unified API

Transport for London's unified API provides developers access to various information regarding the London transport network. These include[15]:

- Journey planning
- Status
- Disruptions and planned works
- Arrival/departure predictions
- Timetables
- Embarkation points and facilities
- Routes and lines
- Fares

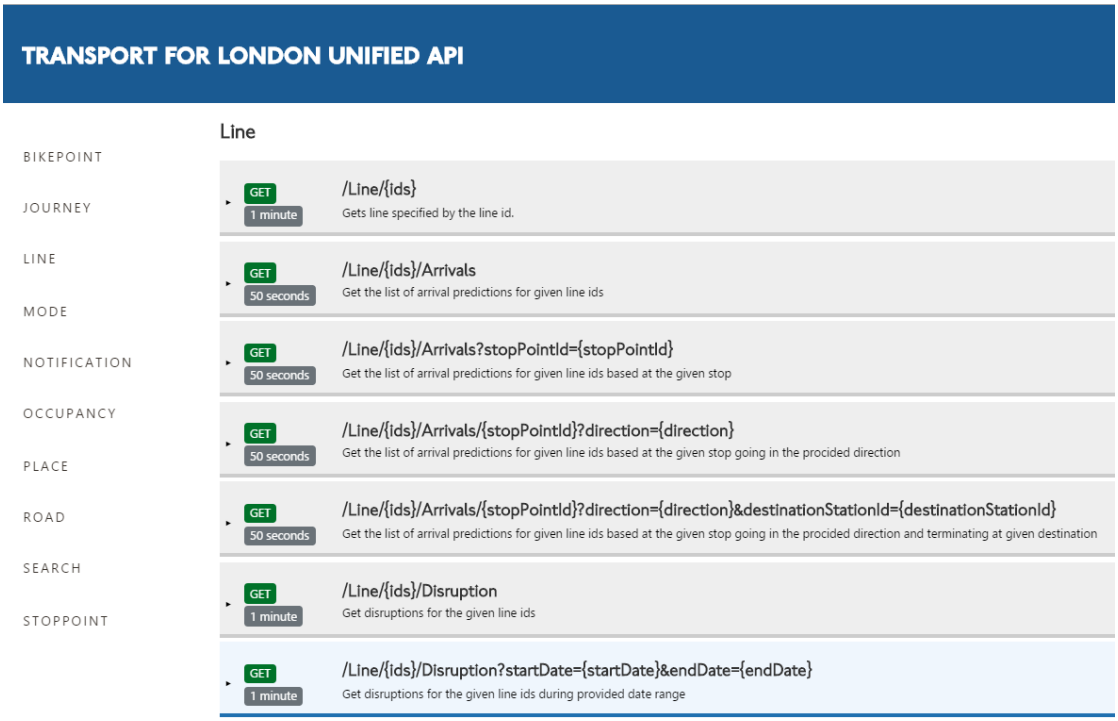


Figure 2.2.: Examples of requests supported by the TfL Unified API.

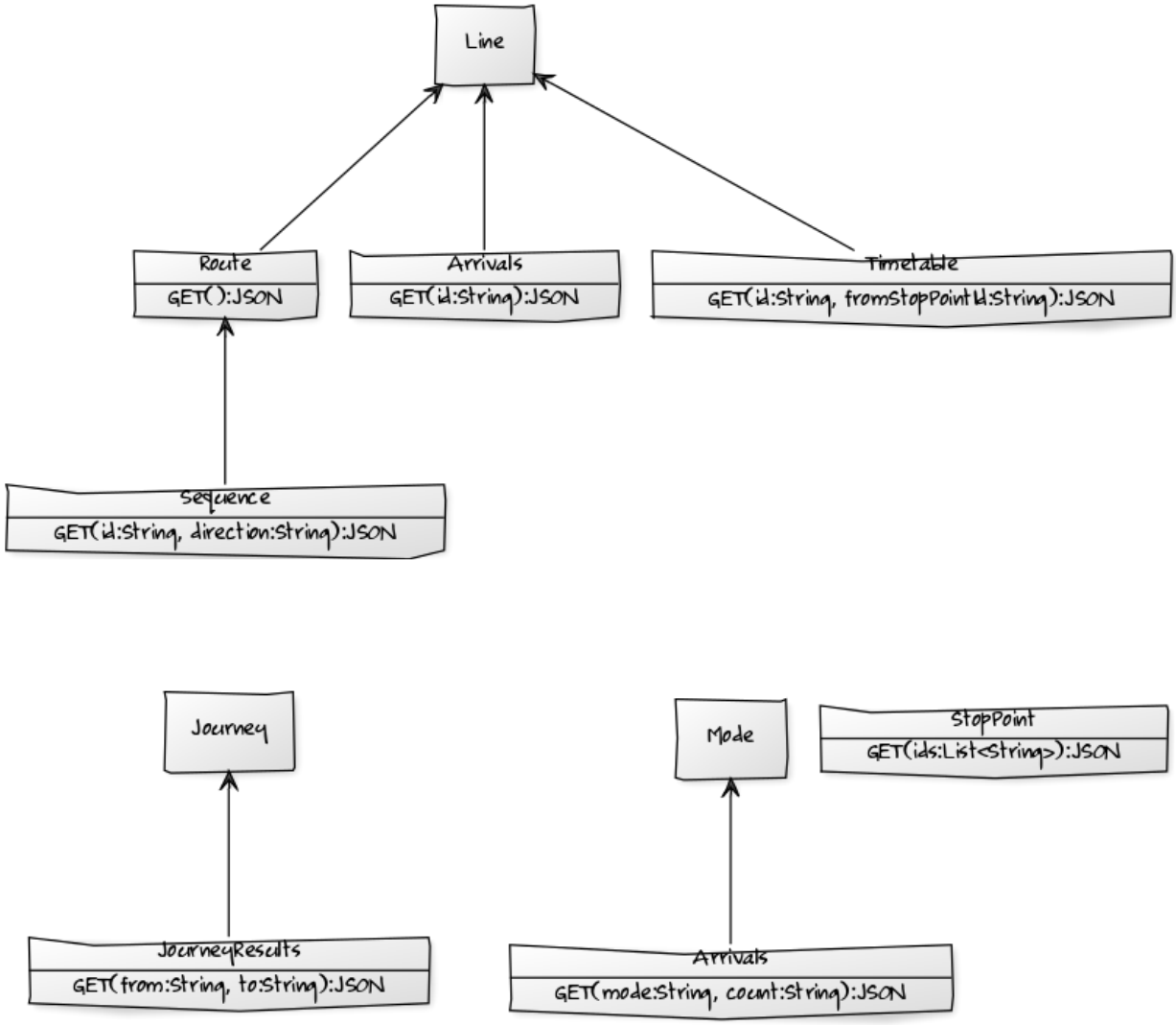


Figure 2.3.: UML class diagram of selected requests supported by the TfL Unified API that we will be using.

Responses are returned in JSON format, and data is cached on their system[16] and updated only periodically, from anywhere between 10 seconds (e.g. live predictions) to a day (e.g. route information), depending on the query as specified by the API.

Figure 2.3 shows the UML class diagram of the relevant TfL API calls that we will be using. The TfL Journey Planner (section 2.4.3) can invoked by specifying the initial and final coordinates, and detailed information including path coordinates, walking and transport instructions, intermediate stops along the route, expected journey times etc. are returned. Route information including sequence of stops and timetables can be obtained, and arrivals information can be acquired by bus route or mode, as we will go into great detail in section 4 - *Implementation*.

We will be making use of this API to obtain bus information and arrivals predictions throughout the remainder of the project.

2.3. Inferring bus arrivals from predictions

2.3.1. Procedure

We can infer bus arrivals at stops by keeping track of the changes in states of predictions. The key is in the disappearance of predictions as a bus travels past a stop. If we continuously monitor the states of predictions referring to a fixed bus and stop pair, assuming that bus predictions are continuously updated when buses are on the road, we can take the final state before disappearing as the most accurate, and therefore treating its expected arrival time as the actual arrival time at the stop.

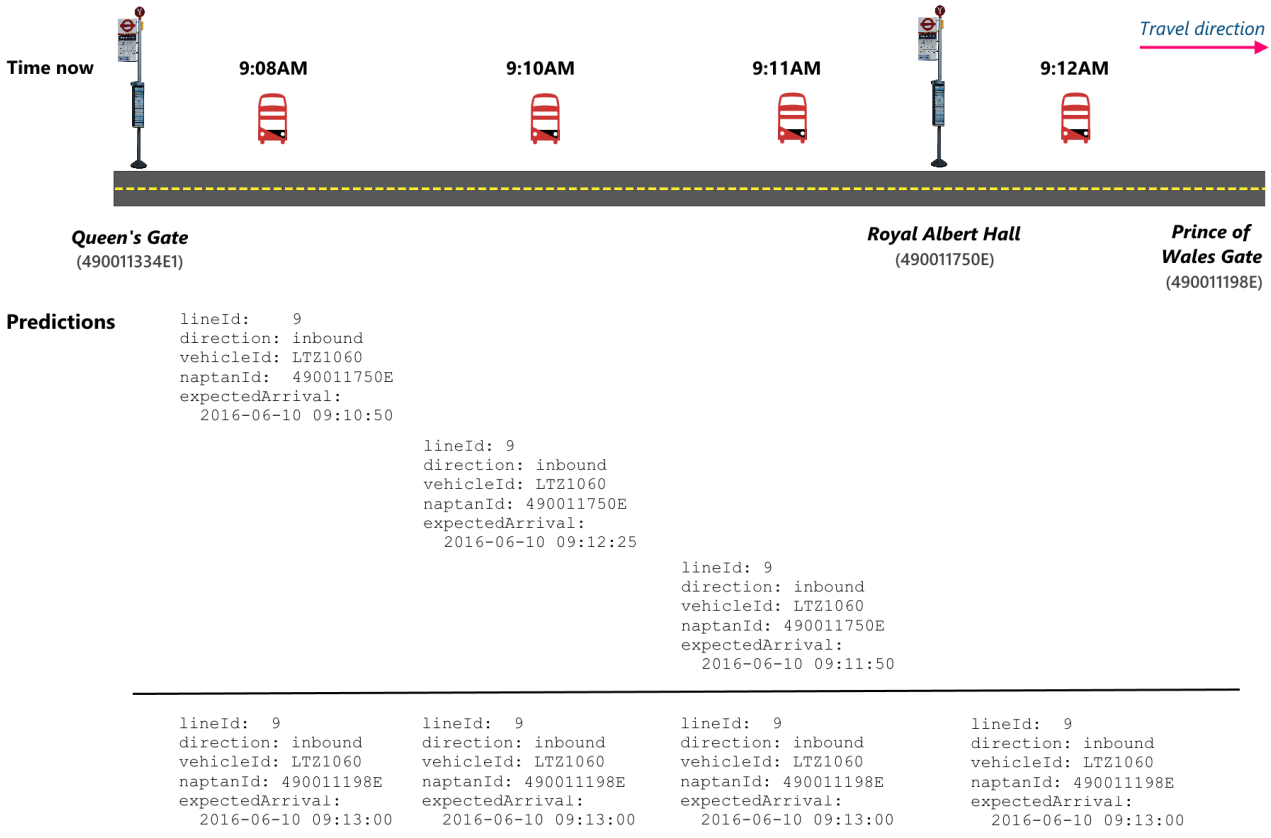


Figure 2.4.: Examples of changes in states of bus predictions. Prediction for Royal Albert Hall disappears after the bus travels past it.

Figure 2.4 illustrates an example of the technique. Suppose we observe a prediction at 9:08 AM referring to bus LTZ1060 serving route 9 in the inbound direction, which has just departed Queen's

Gate and is expected to approach Royal Albert Hall[8] at 09:10:50. 2 minutes later at 9:10 AM, we observe changes to the prediction, where the expected arrival time has now been altered to 09:12:25. After another minute, we observe another update to the prediction, with expected arrival time 09:11:50. Eventually at 9:12 AM, the prediction no longer appears, and we are left with predictions of the vehicle pointing to stops further down the line (on this occasion Prince of Wales Gate[9]). We therefore infer that bus LTZ1060 has arrived at Royal Albert Hall at 09:11:50.

By repeating the above process for the next stop, we can obtain an estimate of the arrival time at the next stop. Taking the difference of the two timestamps (arrival times), we can deduce the journey time between the two stops. Generalising to all buses in operation and stops in Central London we can then build a database of historic bus travel times (section 4.2).

2.3.2. Anomalies

Unfortunately, care has to be taken when applying the above procedure, as anomalies have been found to arise.

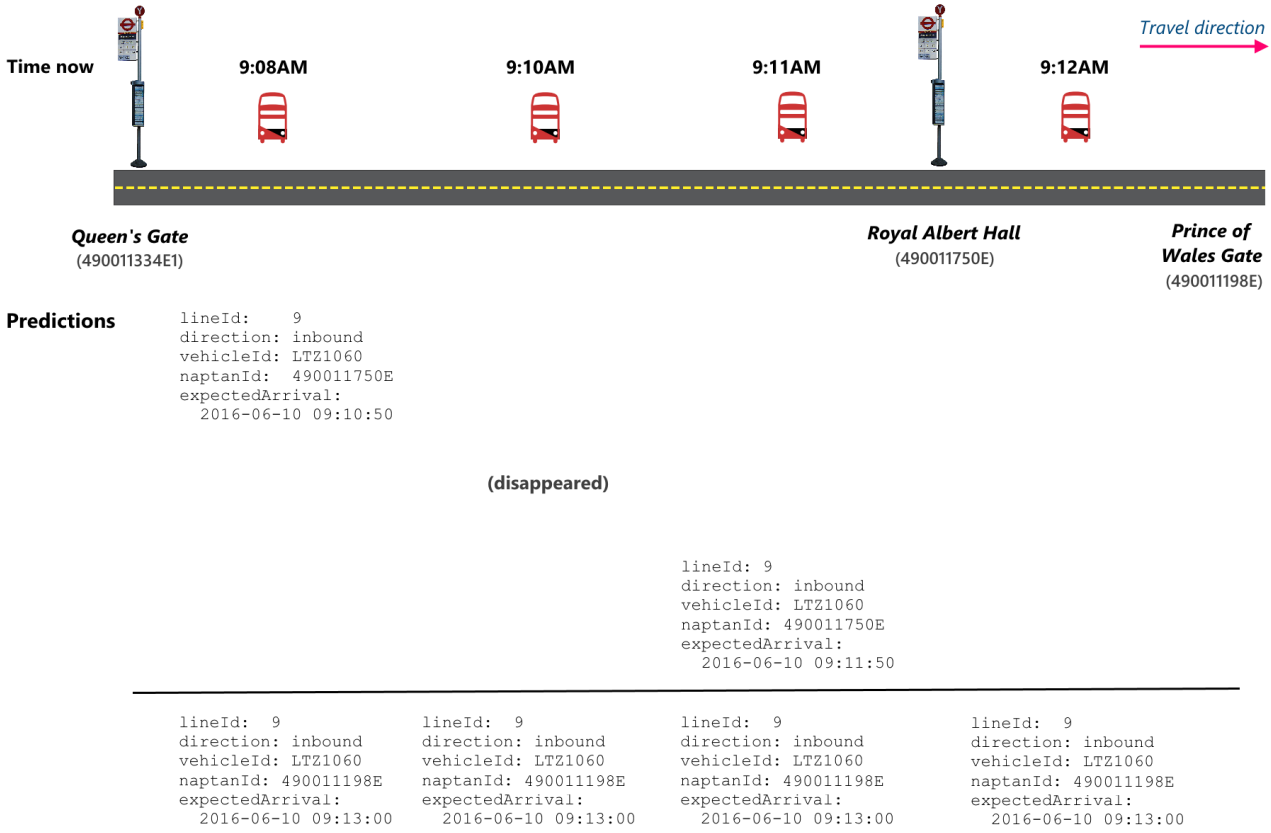


Figure 2.5.: Anomaly 1 - Temporary disappearance of predictions

[8] Identified by `naptanId` 490011750E.
[9] Identified by `naptanId` 490011198E.

One is the temporary disappearance of predictions. Consider the situation in figure 2.5, a variation of the example given previously in figure 2.4, with the prediction for Royal Albert Hall temporarily disappearing at the 9:10 AM mark, then reappearing at 9:11 AM. The above procedure would lead us to believe that the bus has arrived Royal Albert Hall at 09:10:50, and proceeded to the next stop thereafter.

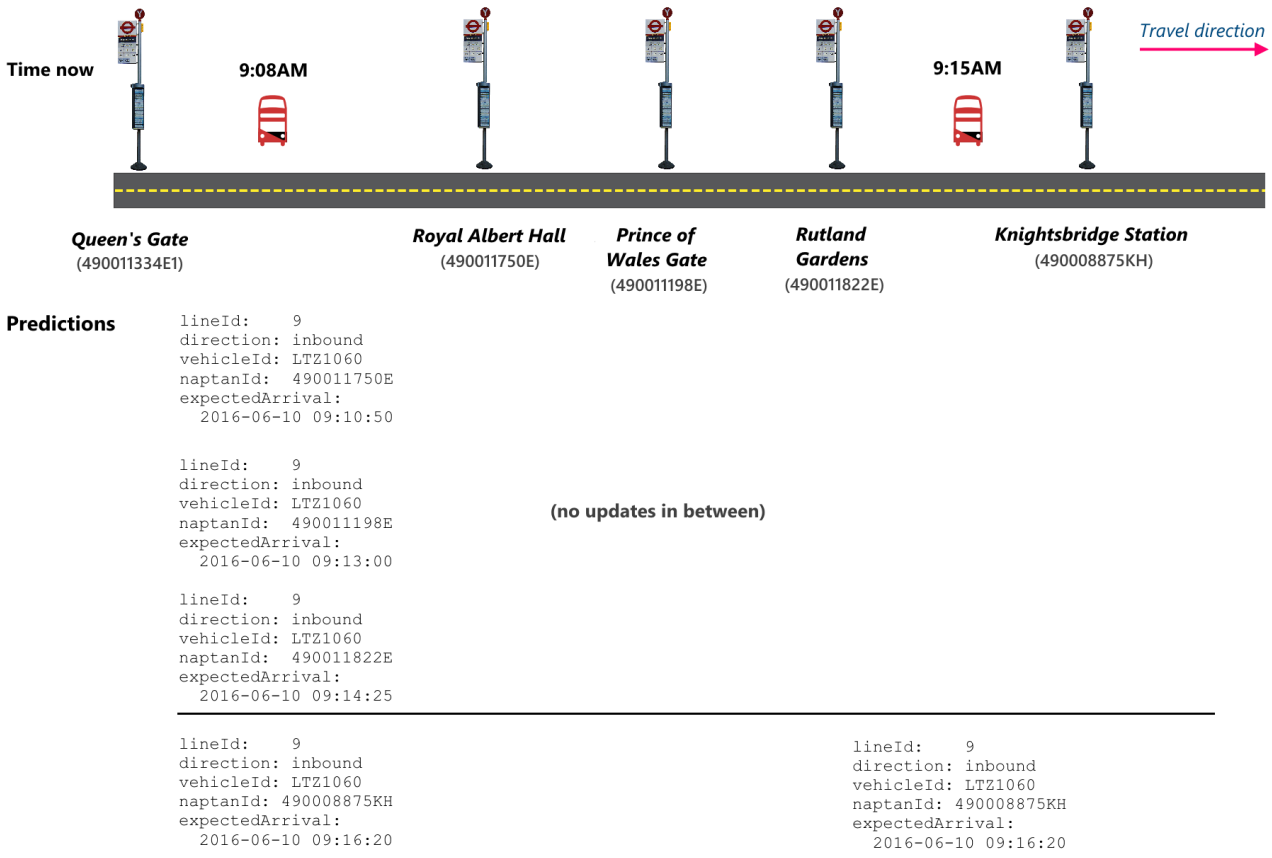


Figure 2.6.: Anomaly 2 - Batch update of predictions

Second is the batch update of predictions. Figure 2.6 depicts one such situation, in which the source of predictions have failed to refresh for a long time, and therefore the bus under study has travelled a few stops down the route, before the next batch of predictions coming in, to suggest the current location of the bus. In this example, incremental disappearance of predictions for the stops in between - Royal Albert Hall, Prince of Wales Gate and Rutland Gardens is not observed, but rather batch disappearance occurs at the 9:15 AM mark.

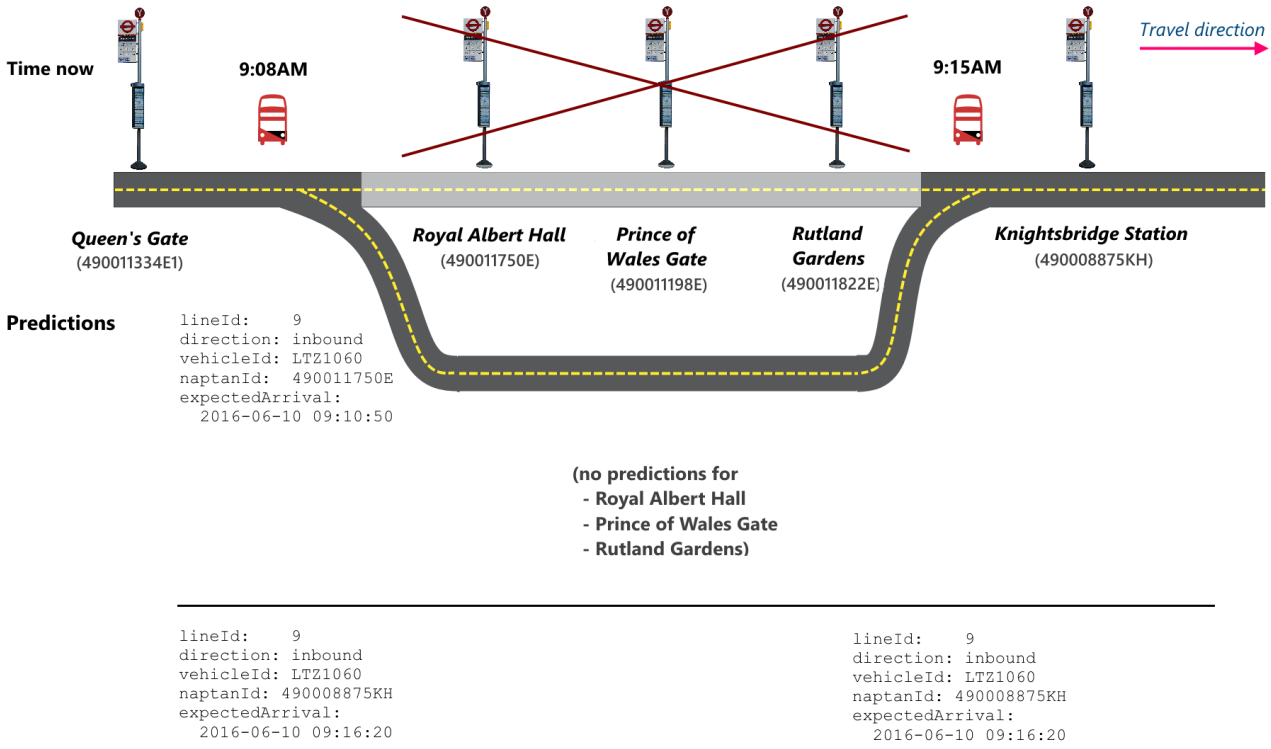


Figure 2.7.: Anomaly 3 - Missing predictions

Third is missing predictions. Owing to diversion or other causes, predictions for a certain section along a bus route may not be present. Figure 2.7 shows one such case, where predictions for stops between Queen's Gate and Knightsbridge Station, i.e. Royal Albert Hall, Prince of Wales Gate and Rutland Gardens, are non-existent.

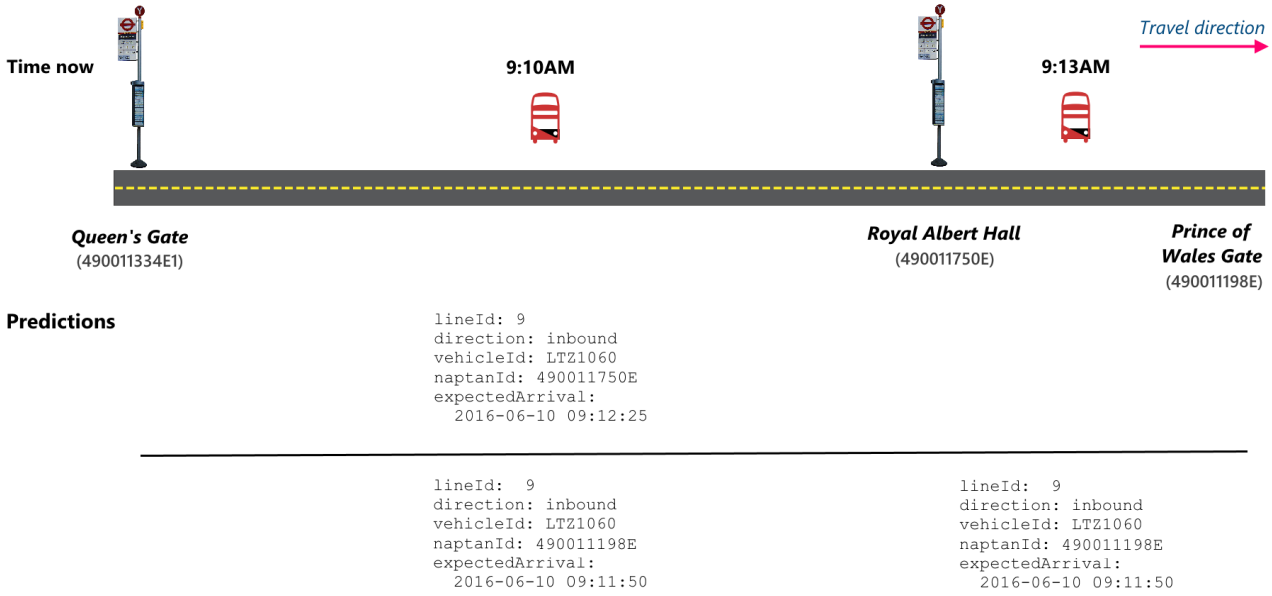


Figure 2.8.: Anomaly 4 - Inconsistent predictions

Fourth is inconsistent predictions in time. As predictions are generated on the basis for individual stops, there are no guarantees that the expected arrival times for stops down a route are increasing, as one might expect. In fact, it was found that the expected arrival times at later stops could be earlier than those of previous stops. Figure 2.8 illustrates an example, where the expected arrival time for Prince of Wales Gate (09:11:50) comes before that of Royal Albert Hall (09:12:25), although the bus is routed to first visit Royal Albert Hall.

We will look at addressing these anomalies in section 4.2.2 - *noise removal*.

2.4. Existing Journey Planners

There are a number of journey planners available in the market that allows for route planning on London's public transport network. We introduce three popular apps - CityMapper, Google Maps and TfL Journey Planner.

2.4.1. CityMapper

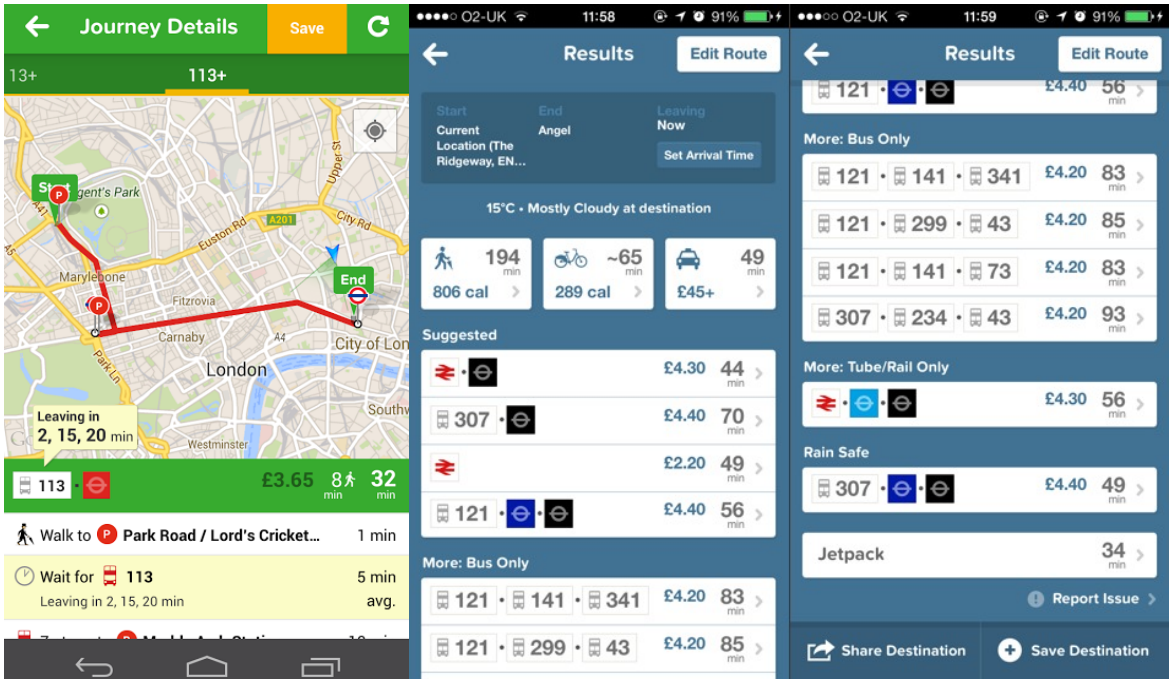


Figure 2.9.: Screenshots of the CityMapper route planning (left) and results page (right) on iOS. Photo courtesy of *http://cdn4.aptoide.com/imgs/2/0/c/20c57d4d937143ac0c8df7437ac70ad3_screen_384x640.png* (left) and *http://cdn.appstorm.net/iphone.appstorm.net/iphone/files/2013/09/citymapper-route-plan1.jpg* (right). Last accessed 7 Feb 2016.

CityMapper[10] is a journey planning app that is in operation in many famous international cities, some of which are London, New York, Paris, Rome, Singapore and Hong Kong[17]. It runs on iOS, Android and the web, and supports point-to-point journey planning, so that when a user specifies the origin and destination, CityMapper will search for a suitable route through the public transport network, and return the details of the route, including travel instructions, expected journey times and fares. It is highly regarded for its intuitive interface, user-friendliness and accurate routing information[18].

Predictions are solely repeated[4] from the TfL live feeds in real time, and so further analysis is not performed to improve the accuracy of predictions. As such, predictions are subject to noise. This is an area we will be working on, and given that CityMapper has a solid and proven front-end, we look up to CityMapper as the benchmark in design of our app.

[10] Startup company founded by Azmat Yusuf in London, 2011.

2.4.2. Google Maps

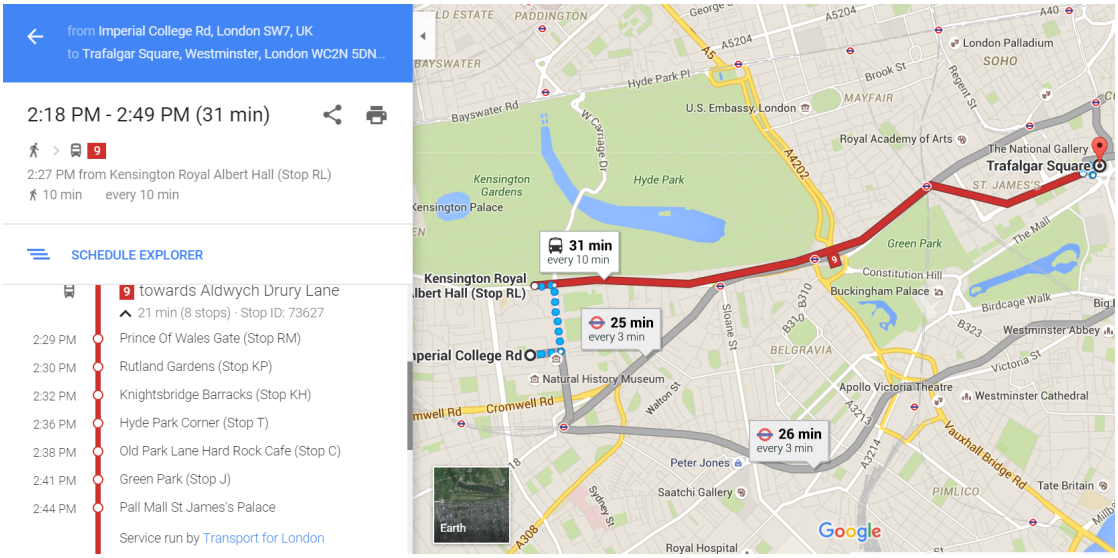


Figure 2.10.: Screenshot of the Google Maps route planner on Google Chrome, from Imperial College to Trafalgar Square.

Google Maps[11] is a long-standing mobile and desktop app that supports route planning in most cities around the world. As it aims for general availability around the world, trade off with city specific features is inevitable, such as journey fares, tips on which underground carriage to take, etc. It is widely used by the public since it is preinstalled and easily accessible on Android devices.

Its source(s) of arrivals information for London is undocumented. Data is likely to come from the TfL live feeds, as some of the stop details (e.g. Stop ID) as shown in figure 2.10 are internal to TfL and unlikely necessary for the public. If such, arrivals predictions are simply forwarded to the public, so noise will be present as in the case of CityMapper. We hope to address this issue in our project to provide more accurate estimates.

[11] *http://maps.google.co.uk*. Last accessed 11 Feb, 2016.

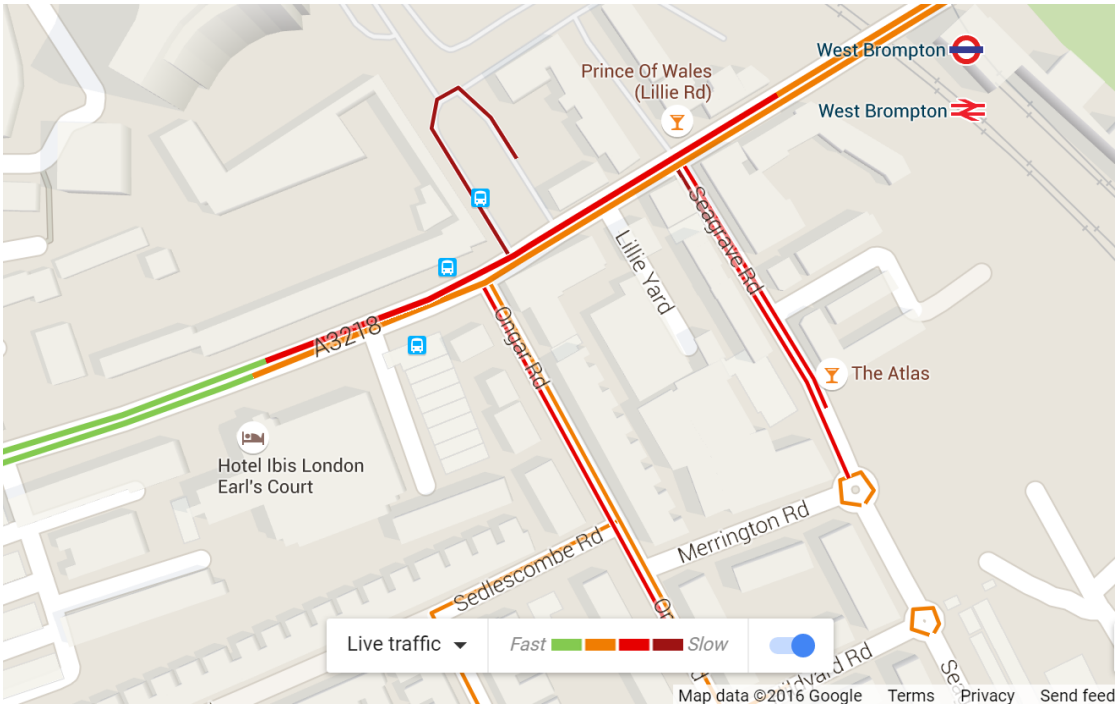


Figure 2.11.: Screenshot of the Google Maps traffic page on Google Chrome, on traffic conditions near West Brompton. Darker areas (e.g. red) indicate heavy traffic, and hence a user could expect delays.

A key feature worth mentioning is the '*Traffic*' page, in which Google crowdsources data from smartphone users[19] on the road to construct a real-time map of road conditions (figure 2.11). Roads are highlighted with different colours to indicate their respective actual speeds of traffic, and hence users could infer delays visually. An unfortunate limitation is that Google does not provide numerical estimates of delays, making this information hard to reuse for our purposes of generating live predictions. Nevertheless, it will serve later as a useful tool for checking whether we have correctly identified delays, as our conclusions should in theory coincide.

2.4.3. TfL (Transport for London) Journey Planner

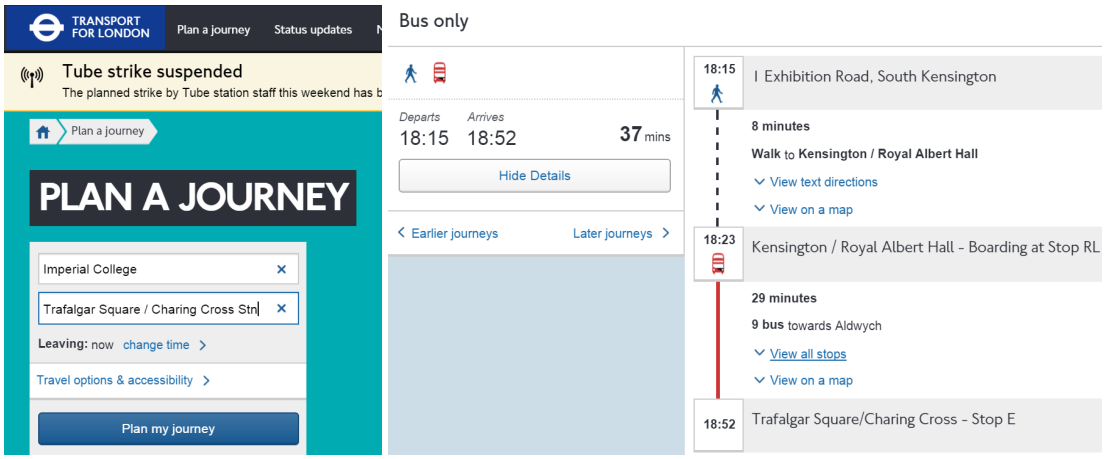


Figure 2.12.: Screenshot of TfL Journey Planner on Google Chrome, from Imperial College to Trafalgar Square.

TfL Journey Planner is the official webapp[12] run by Transport for London to plan journeys in Central London on the public transport network managed by Transport for London. Despite having a comparatively basic graphical user interface, it pretty much matches CityMapper and Google Maps in terms of route planning features.

Similar to CityMapper, it utilises the TfL live feeds to generate current predictions along the user's route, but does not make use of historic data to improve predictions. It is also susceptible to noise since it does not perform filtering on live predictions[4].

2.5. Bus Prediction Techniques in Literature

A number of studies have been performed on historic and real-time bus arrivals data to achieve better predictions in journey times. The techniques employed include the following machine learning and statistical methods:

Machine learning methods	Statistical methods
Artificial neural network (ANN)	Regression model
Support vector machine (SVM)	Time series
k-th nearest neighbour (k-NN)	Kalman filter

[12] *https://tfl.gov.uk/plan-a-journey/*. Last accessed 7 Feb, 2016.

2.5.1. Artificial Neural Network (ANN)

Artificial neural networks are a family of machine learning models that resemble biological neural networks (e.g. the brains of animals), consisting of inter-connected neurons. ANNs are powerful in the sense that they can identify non-linear relationships between predictor variables and response variables.

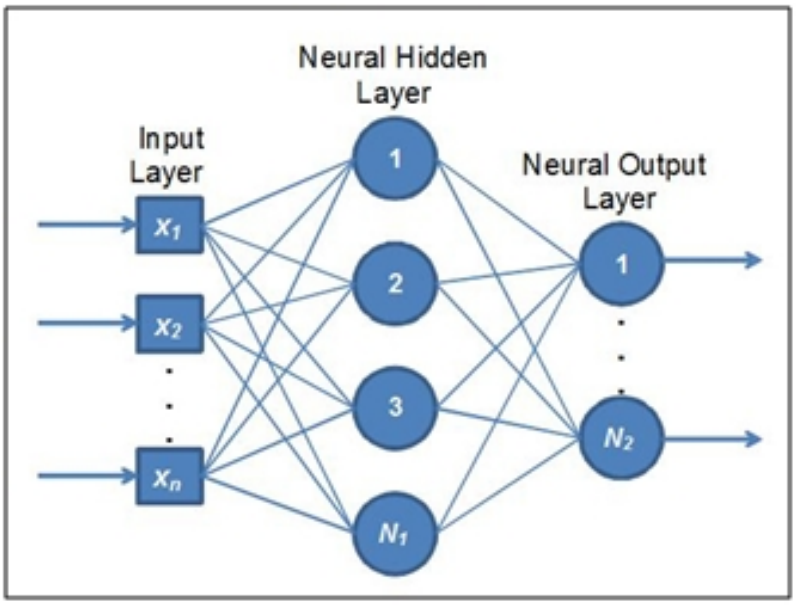


Figure 2.13.: An illustration of the artificial neural network with 3 input neurons, 4 hidden neurons and 2 output neurons. Photo courtesy of *http://engineering.electrical-equipment.org/wp-content/uploads/2013/05/Artificial-Neural-Network-Based-Power-System-Restoration-2.jpg*. Last accessed 8 Feb 2016.

An artificial neural network consists of

- an input layer, having the same number of neurons as the number of predictor variables, so that each neuron corresponds to each predictor variable and accepts corresponding possible values as inputs;
- one or more hidden layers within which neurons in a layer are interconnected with all neurons in adjacent layers; and
- an output layer, in which each neuron take a possible value of the response variable, so that each will receive a probability representing the likelihood of it being true.

The backpropagation algorithm is then applied over a set of training data, iteratively adjusting the "weights" of connections in between neurons in order to minimise the errors over the input-output neurons, thereby learning the relationship between the predictor and response variables.

Chien et al.[8] studied the bus arrivals times of route 39 in New Jersey, the United States. Both link-based and stop-based approaches were investigated, where stop-based journey times refer to journey times measured directly between adjacent stops, and link-based journey times refer to the journey times between adjacent stops evaluated as the sum of individual chunks between road intersections (figure 2.14).

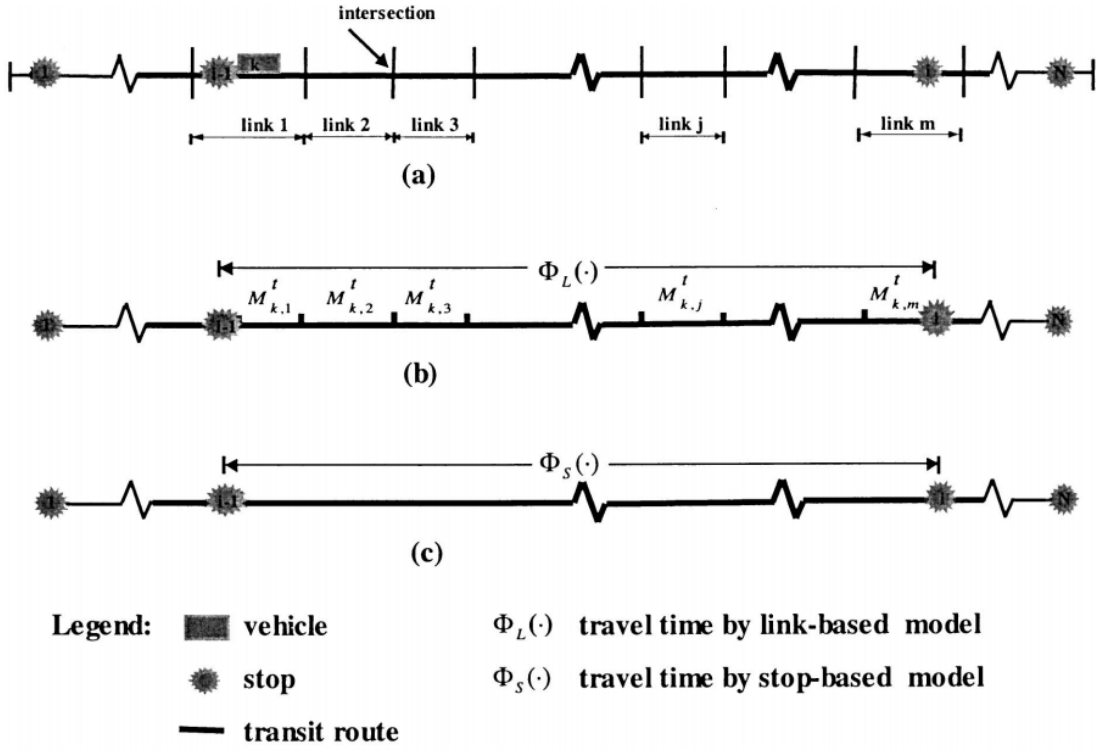


Figure 2.14.: Transit route segment with link-based and stop-based travel times: (a) segment between stops; (b) link-based travel time; (c) stop-based travel time.
Reproduced from Chien et al.[8], figure 1.

The ANN models for the respective link-based and stop-based approaches were built, together with their "enhanced" counterparts which were adaptive to real-time situations. It was found that without integrating the adaptive algorithm, the stop-based ANN outperformed the link-based ANN, particular between stops with multiple intersections. But when it comes to the adaptive ANNs, both stop-based and link-based ANNs were easily outperformed, especially in multistop predictions.

Although ANNs have been shown promising in this respect, there may be limitations in employing ANNs for the purpose of this project. One such limitation is the availability of predictor variables. Bus travel distance, link traffic volume, link speed, link delay, link queue time and passenger demands at stops were made predictor variables for the link-based and stop-based models, whereas it may be difficult to obtain these information for Central London, in particular the traffic volume

and passenger demands at stops, which are not made available by the TfL API.

2.5.2. Support Vector Machine (SVM)

Support vector machines are supervised machine learning models that performs classification and regression on data. It classifies data by mapping predictor variables into higher dimension spaces, and seeking hyperplanes that best separate data of one category from another. Unlike ANNs which are prone to overfitting, SVMs always lead to the unique and globally optimum solution, and therefore achieving a higher generalisation performance than ANNs. One main drawback is the lengthy training process, as the learning algorithm runs in between $O\left(n^2\right)$ and $O\left(n^3\right)$ time, where n is the size of training examples[5].

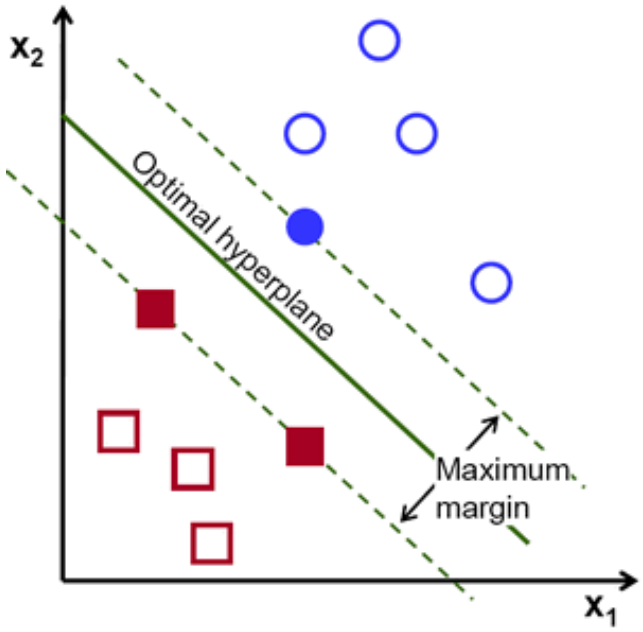


Figure 2.15.: Optimal hyperplane generated by the support vector machine in classifying between the two categories of data - red and blue.
Photo courtesy of *http://docs.opencv.org/2.4/_images/optimal-hyperplane.png*. Last accessed 8 Feb, 2016.

Bin et al.[5] performed a study on the arrival times of transit route 4 in Dalian, China, generating predictions using SVM and comparing them to ANN. The training data was classified into four categories, depending on whether one belonged to peak-hours and whether it was sunny at that time. It was found that SVM outperformed ANN by $5-7\%$ across all 4 categories, and that predictions generated by SVM were relatively stable compared to ANN as the RMSE[13] graphs of SVM showed less hikes than those of ANN.

[13] Abbreviation for root-mean-square error.

Similar to ANNs, SVMs are computationally intensive to train, and therefore considerable training process may need to be undergone before good results are observed. They are also "black boxes", making results difficult to understand and explain, particularly when compared to regression or time series methods.

2.5.3. k-th Nearest Neighbour (k-NN)

The k-th nearest neighbour algorithm is a lazy-learning machine learning algorithm for classification and regression. Training data is simply stored in Euclidean space, and any generalisation beyond the training data is deferred until a query instance is presented. The algorithm will then search through the k instances closest to the query instance in terms of a pre-defined distance function, and assigns to the query instance a classification by majority vote.

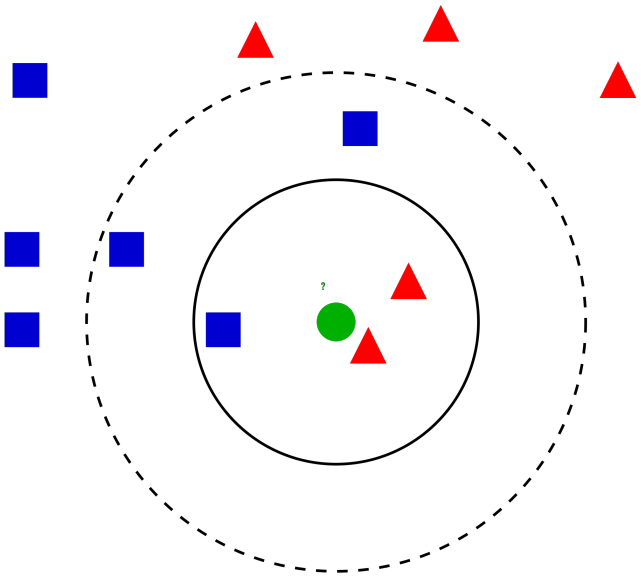


Figure 2.16.: Example of k-NN classification. The query instance (green circle) should be classified as red triangles or blue squares. If $k = 3$ (solid line circle) the former is assigned, whereas if $k = 5$ (dashed line circle) the latter is assigned. Photo courtesy of *https://upload.wikimedia.org/wikipedia/commons/thumb/e/e7/KnnClassification.svg/2000px-KnnClassification.svg.png*. Last accessed 9 Feb, 2016.

Chang et al.[7] built a nearest neighbour non-parametric model (NN-NPR) to forecast multiple-interval bus travel times of an intra-city route in Seoul, South Korea, and it was found to outperform the historic-average model. However, it was also noted that their original k-NN forecasting algorithm had longer execution time than other prediction techniques. Yu et al.[6] performed a study in Hong Kong comparing multiple prediction techniques such as SVM, ANN, k-NN and linear regression in bus arrival times, but to find that k-NN were only as good as the linear regression model, and being beaten by both SVM and ANN.

2.5.4. Regression model

Regression models are used in statistics to fit relationships between covariates (called predictor variables) and the outcome (called response variable). Linear regression is a common approach, in particular the generalised linear model is popular, in which the response variable is expressed as a function of the linear combination of predictor variables. The fitted relationship can then be interpolated or extrapolated to generate forecasts on unseen data.

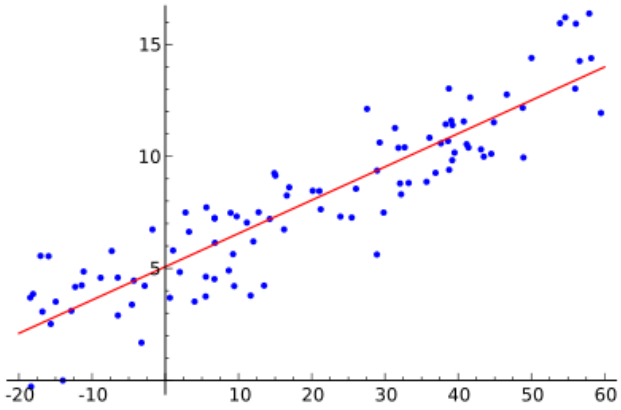


Figure 2.17.: Example of linear regression. Photo courtesy of *https://en.wikipedia.org/wiki/Linear_regression#/media/File:Linear_regression.svg*. Last accessed 9 Feb, 2016.

Patnaik et al.[11] worked on estimation of bus arrival times on an urban bus route in the northeast United States. The generalised linear model was adopted, where the distances, dwell times in between stops, the number of stops, and time period of each day were selected as predictor variables. It was found the RMSE between estimated and actual mean travel times were quite close to each other, and $R^2 = 0.99 \approx 1$, suggesting that the linear model may be a decent approach.

Compared to time series models, the latter is better suited to time series data, as regressions models do not take into account of (auto)correlation of error terms[20].

2.5.5. Time series

Time series models describe relationships between past, present and future observations. An example is the autoregressive integrated moving average (ARIMA) family, in which observations in later times are represented as a linear combination of earlier observations and noise terms. Since time series models are dependent on the similarity between real-time and historical patterns, wide fluctuations in historical data could lead to large differences between predictions and actual observations[7].

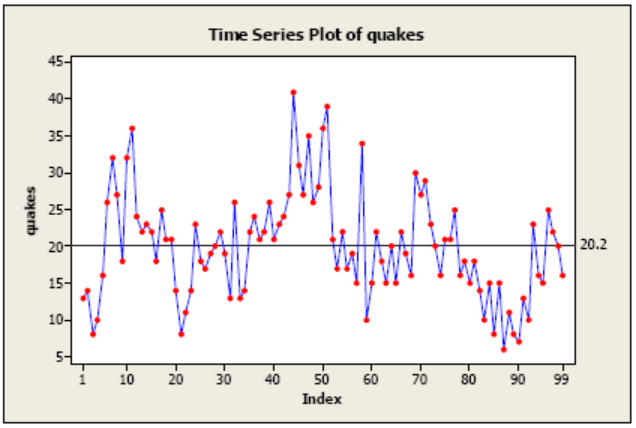


Figure 2.18.: A time series plot of the annual number of earthquakes in the world with seismic magnitude over 7.0, for 99 consecutive years. Photo courtesy of *https://onlinecourses.science.psu.edu/stat510/sites/onlinecourses.science.psu.edu.stat510/files/L01/graph_01.gif*. Last accessed 9 Feb, 2016.

Suwardo el al.[12] studied bus arrival times along the Ipoh-Lumut corridor in Malaysia with the ARIMA model. The forecasts were based solely on historical travel time, without taking into account of other factors such as traffic conditions. It was found that the observed arrival times lay within the 95% confidence interval of predicted arrival times, suggesting time series analysis may be a feasible approach.

We will be pursuing techniques in time series throughout the later chapters.

2.5.6. Kalman filter

The Kalman filter has been another popular statistical method adopted in bus arrivals prediction studies. Its underlying principle is Bayesian inference[14], in which the prior distribution describing a previous state will be updated based on new observations to form the posterior distribution. It is widely adopted in industry for its non-computationally intensive nature, which is advantageous in generating real-time predictions[7].

[14]The Kalman filter is a special case of the Bayesian network where the random variables are Gaussian.

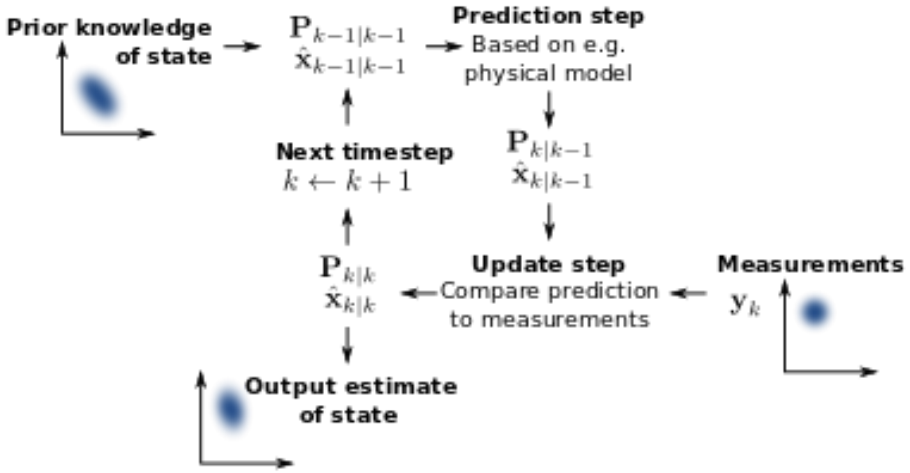


Figure 2.19.: A diagram depicting the internal procedure of the Kalman filter. Photo courtesy of *https://upload.wikimedia.org/wikipedia/commons/thumb/a/a5/Basic_concept_of_Kalman_filtering.svg/400px-Basic_concept_of_Kalman_filtering.svg.png.* Last accessed 10 Feb, 2016.

Chien and Kuchipudi[9] applied the Kalman filter on a section of the New York State Thruway to study vehicle travel times, using the automated tag times when a vehicle passes through a toll plaza. Link-based and path-based variants were investigated, and it was found that the link-based models performed remarkably well, whereas the path-based model fell a bit short in the event of congestion or traffic accident. While we acknowledge that the study was performed on general vehicles, there is reason to believe the results should extend well to buses, as the monitoring of bus arrivals at stops resembles much the tagging of vehicles at toll plazas. Indeed it is revealed that the iBus system (section 2.1.1) applies the Kalman filter on the data collected from GPS in determining the location of buses[14], with which predictions are based upon.

We note that this method is more complex than the time series model, and assumes Gaussian noise terms. We shall therefore take this as a backup plan, should time series analysis fail to give promising results.

2.5.7. Hybrid approaches

Combinations of the above methods are also present in literature. Bai et al.[21] performed a comparison study between the ANN, SVM, Kalman, ANN-Kalman, and SVM-Kalman algorithms on five bus routes running along the Shennan Boulevard in Shenzhen, China. It was found that the ANN-Kalman and SVM-Kalman combination approaches significantly outperformed[15] their pure counterparts, and it was attributed to traditional machine learning algorithms (ANN and SVM)

[15]The pure ANN, SVM and Kalman models recorded RMSEs between 71 and 75, whereas the hybrid ANN-Kalman and SVM-Kalman models recorded a RMSE of only 30.

being able to extract patterns in the historical data better, whereas the Kalman filter adapted well to dynamic traffic conditions in real-time.

3. Design

3.1. Overview

The project can be split into several stages. Since TfL does not provide a historic database that we can query on, we first have to build our own.

The first stage is to gather live bus predictions from TfL, and channel them into our database, so that analysis can then be performed afterwards. The second stage is then the 'trial and error' phase - we shall simulate a series of user journeys, and experiment with different techniques and formulas in generating our predictions to find out which methods will give the most reliable results. The final phase will involve wiring up everything together, presenting and summarising the results of our analysis in a neat, interactive, user-friendly front-end mobile app, so that the ordinary citizen can appreciate our work, and benefit from it by saving valuable time in commuting.

We now go into the design of our project - introducing which technologies have been utilised and why they were chosen, and explain how computing resources are split across various components in our project.

3.2. Technologies

3.2.1. Choice of scripting host - Apache Cloudstack / CloudVM

Given that TfL predictions come in round the clock, and that end users can commute any time, the first thing to consider is a host that is reliably available 24/7. It will also need to be computationally powerful and have fast network connectivity, so that database queries can be processed efficiently, and live feeds from TfL can also be quickly denoised within the 10 second window frame before inserted into the database. Another aspect to be taken into consideration is extensibility, given that we are accumulating massive historic data in the order of Gigabytes, we may want to expand disk capacity in case disk space runs out.

The virtual machines managed by CSG running on Apache Cloudstack[16] address the above needs very well. The machines are highly configurable, such as the number of cores, RAM, and disk space. For our purposes, we have acquired three VMs with 4 cores, 4Ghz and 8 GB of RAM.

[16] Apache CloudStack is an open source cloud computing software for deploying large networks of virtual machines.

3.2.2. Choice of backend scripting language - Go

The language that we write in for our backend has to be highly efficient, in particular modern big data paradigms emphasises importance in parallelism and concurrency. This immediately rules out interpreted and dynamically typed languages such as Python, despite having a wealth of libraries available. In fact, fast programming languages have traditionally been represented by the likes of C, C++, Java, and the call for concurrency pinpoints heavily to Java.

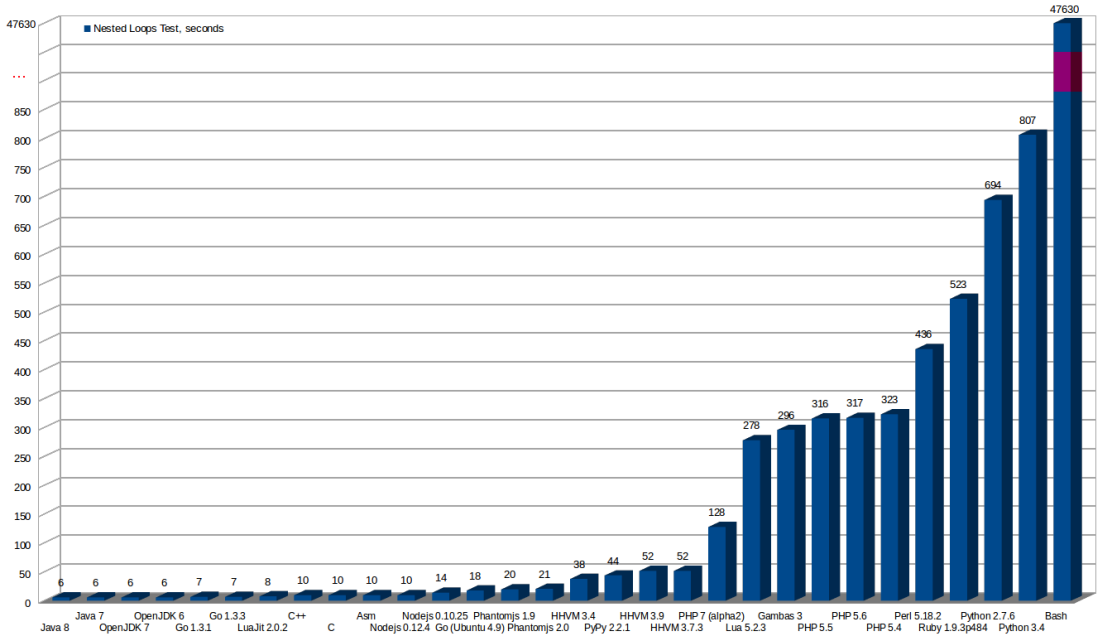


Figure 3.1.: A benchmark of various programming languages on nested loop additions.[22] Photo courtesy of *http://blog.carlesmateo.com/wp-content/uploads/2014/10/blog-carlesmateo-com-performance-several-languages-php7-phantomjs-nodejs-java-bash-go-perl-luajit-hhvm3_9-scale_mod5.png*. Last accessed 1 May, 2016.

Not so until recently that the Go programming language[17] came into the picture. The static, compiled programming language was designed with concurrency at its heart. Its performance is in the class of Java, but with much better memory footprint and scalability when it comes to threading[23][24]. Other advantages include web server support in its standard library, allowing for ease of integration with the rest of code; availability of Postgres binding making it easy to communicate with the database; and less verbose syntax compared to Java, making it favourable in terms of development time.

[17] Developed by Google and released in 2009, the compiled, statically typed language featured a new paradigm for concurrency - “goroutines” and “channels”, which aims at addressing difficulties in sharing the same piece of data in a thread-safe manner across threads in traditional threading models.

3.2.3. Choice of database management system - PostgreSQL

With massive amounts of live feeds continuously received and processed, it is vital that data can be stored and retrieved efficiently. In order to perform valid analysis, data integrity is also of utmost importance.

An SQL[18] database with ACID[19] properties is therefore ideal. PostgreSQL[20] is known to adhere to the ANSI SQL features very well, and is generally acclaimed to be stable and reliable, even over years of heavy use[25]. Handy features include concurrent building of indices without blocking write operations, so that we can defer building of the necessary indices on the fly to save space, without hindering updates on our live production table.

3.2.4. Choice of front end - Android application

We wish to make the final results from our research accessible to a wide audience in a friendly manner. Options of a web app or mobile app have been considered.

Advantages of building a web app include compatibility across a range of devices (desktops, laptops, Blackberry, iOS and Android smartphones), and less costly development time. The downside to this is the lack of user-interactive features, such as location detection, live notification, which could be a game changer in user experience, considering a commuter may only be interested in delays affecting his/her journey between the current location and the destination, and may be put off by the need to actively check a website in order to be informed of delays.

[18] SQL refers to Structured Query Language, which is a special-purpose programming language tailored for data manipulation in a relational database management system.
[19] ACID refers to atomicity, consistency, isolation and durability. These properties are crucial in ensuring data integrity.
[20] PostgreSQL is an open source object-relational database management system implementing the majority of the core SQL:2011 standard.

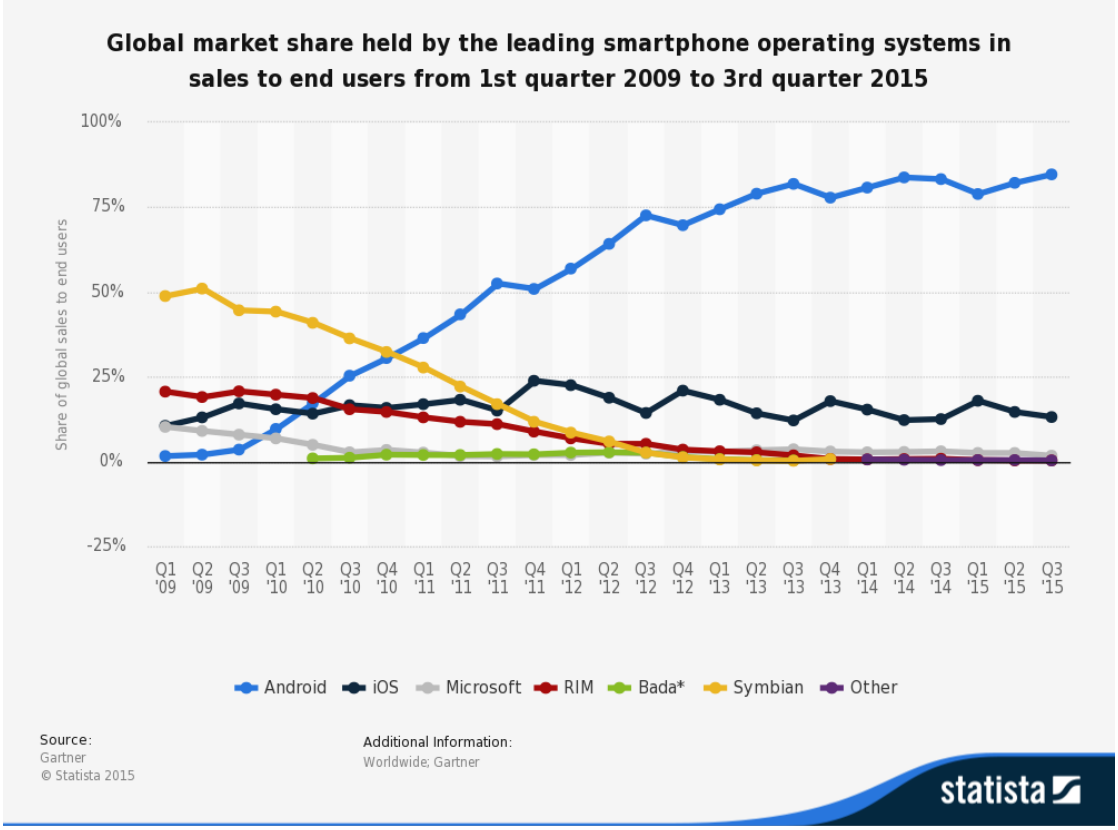


Figure 3.2.: Market share of different smartphone operating systems, as of Quarter 3 of 2015. Photo courtesy of Statista, *http://venturebeat.com/2016/02/05/microsoft-nokia-and-the-burning-platform-a-final-look-at-the-failed-windows-phone-alliance/*. Last accessed 1 May, 2016.

Therefore we decide to stick with a mobile app. For the purpose of our project we will be building an Android app, given that Android is the most popular mobile operating system as of 2015 (figure 3.2). Android also has built in map support, so we could identify the location of users, and give relevant updates on their intended routes based on their current locations.

It is certainly possible to implement another client on other mobile operating systems, should the reader be interested, and can be done easily by querying the same APIs (appendix B) as the Android app does on the web server that we set up.

3.3. Components

We shall now give a high-level overview of the architecture of our backend. As large amounts of data needs to be processed in real time, proper division of work across computing resources is critical. One approach is to visualise the problem as a production chain, where live predictions

from TfL are first gathered into our database, then fed down to the analyser. The results (i.e. predictions in journey times and current delays) can then be made available on a webserver, which provides an endpoint for other clients (e.g. the front-end mobile app) to request from.

3.3.1. Allocation of computing resources

We note that the earlier stages of the chain will be more resource intensive, as live predictions across Central London from TfL come in at 10 second intervals, and they have to be swiftly denoised and inserted into the database before the next batch of data comes in. The process of insertion into the database is also coupled with update of indices, during which the database could be locked up. This may even lead to other `SELECT` queries being blocked when analysis is performed in later stages. Therefore, we decide to allocate more computing power for these actions.

The later stages concerning analysis of historic data and queries from clients are not imposed with strict timing requirements, and are intermittent in nature. To save unnecessary computing resources we allocate only a single VM for all such purposes.

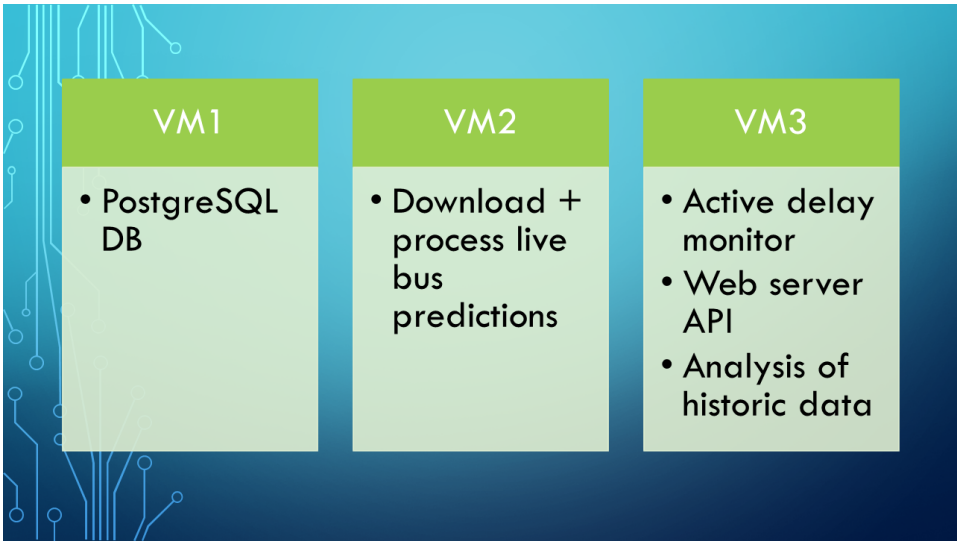


Figure 3.3.: Distribution of work across the 3 virtual machines.

Three virtual machines were acquired from CSG in total, each of 4 cores, 4 GHz and 8 GB RAM. The first virtual machine is solely reserved to host the Postgres database and handle database requests. We also allocated for now 600GB of disk storage, which can be easily extended if need be when the disc fills up with predictions, for the Postgres database. The second virtual machine is assigned the duty of repeated polling of TfL live feeds, and noise removal of insensible predictions before sending them into the Postgres database. The rest of operations are handled by

the remaining virtual machine, including active detection of delays, web server hosting for client connections and evaluation of different prediction methods on historic data.

3.3.2. Database

The database consists of a number of tables, the main of which (table `journey_history`) stores the historic records of bus arrivals, and the rest background information for validation and auxiliary purposes. We will now give a brief overview of important tables, further details to be given in section 4 - Implementation. Figure 3.4 gives a UML representation of some of the key tables in the database.

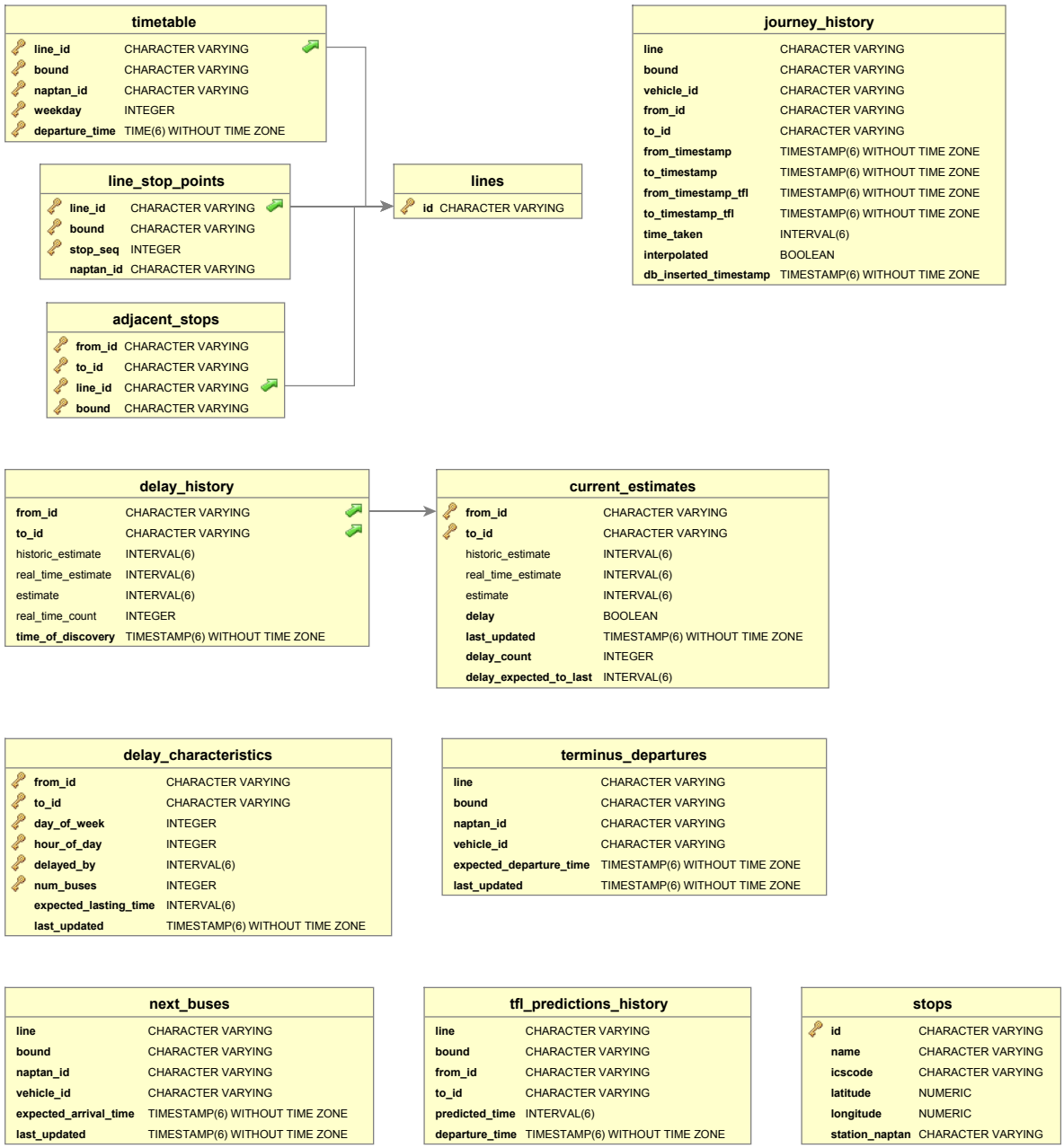


Figure 3.4.: UML diagram of selected tables in database. Primary keys are denoted with a "key icon" next to the table attribute, and foreign keys are represented by green arrows.

The main table `journey_history` is the table that keeps records of past travel of all buses. It includes information of the times (specified by `from_timestamp`, `to_timestamp`) during which individual journeys between each pair of neighbouring stops (represented by `from_id`, `to_id`) along a bus route (`line`) and direction (`bound`) were made, together with the registration number

(vehicle_id) of the bus in question. For example, the record:

```
 line |  bound  | vehicle_id |  from_id   |   to_id    |   from_timestamp    |
  to_timestamp     | from_timestamp_tfl  |  to_timestamp_tfl   | time_taken | in
terpolated | db_inserted_timestamp
------+---------+------------+------------+------------+---------------------+--
------------------+---------------------+---------------------+------------+---
-----------+----------------------
 9    | inbound | LTZ1094    | 490011750E | 490011198E | 2016-05-02 18:38:12 | 2
016-05-02 18:39:38 | 2016-05-02 18:37:33 | 2016-05-02 18:39:01 | 00:01:26   | f
          | 2016-05-02 18:40:42
```

represents that bus LTZ 1094 (serving route 9 in the inbound direction) took 1 minute 26 seconds to travel between Royal Albert Hall and Prince of Wales Gate (corresponding to ids 490011750E and 49001198E respectively), from 18:38:12 to 18:39:38 on 2nd May, 2016.

The tables lines, stops, line_stop_points and adjacent_stops are helper tables that contain all bus route and stop information in Central London. They are used for validation checks, and lookup to convert TfL internal identifiers to human readable form. Their significance will be discussed in the Implementation section. For example:

```
     id     |         name         | icscode | latitude  | longitude
------------+----------------------+---------+-----------+-----------
 490011198E | Prince Of Wales Gate | 1011198 | 51.501991 | -0.171039
 490011750E | Royal Albert Hall    | 1011750 | 51.501886 | -0.176418
```

by looking up the stops table we obtain the correspondence between identifiers 490011198E, 490011750E and the place names “Prince of Wales Gate” and “Royal Albert Hall”.

The remaining tables current_estimates, delay_history, delay_characteristics, tfl_predictions_history, timetable, terminus_departures and next_buses are used for prediction analysis. The current_estimates table caches predictions for current travel times between each pairs of neighbouring bus stops, based on the best prediction method to be found later. We will be going into the details in the *Optimisation* section (section 7). The delay_history table stores information on past delays, the delay_characteristics table on how long delays or unusual traffic are expected to last (section 5.2), and the tfl_predictions_history table contains what TfL gave as predictions and will serve as benchmark for evaluating whether our predictions outperform theirs. In addition to computing travel times on buses, passenger waiting times are also a subject under study, and the tables timetable, terminus_departures and next_buses come into play, which will be discussed also in the *Optimisation* section (section 7) after we develop appropriate prediction techniques.

3.3.3. Backend scripts

Scripts are set up for the actions that needs to be performed periodically, such as the polling and processing of bus predictions from the TfL server, estimation of current travel times, detection of delays, and updating of route information.

In order to safeguard scripts from unexpected termination, Cron jobs are set up to test at one minute intervals whether the scripts are in a running state. Should a script be found not running, it would be restarted.

For debugging purposes output from all scripts are kept as logs. To prevent filling up of disk space, logs older than 3 days are removed at midnight every day.

3.3.4. Web server

In order to pass on our data to our clients, such as predictions and current delay information, we set up a web server endpoint for clients to query on, and return JSON in response. This has the benefit of encapsulating the internal structure of our backend codebase, and allowing the flexibility of extra client implementations. The details can be found in appendix B.

3.3.5. Android app

The Android app is the component that interacts at the forefront user level. It aims at providing a nice user experience, and bridging the gap between the needs for route planning of the user, and the prediction generation process in the backend. It also echos detected delays in the backend to the user as notifications so that the user can pick up the signal in a passive manner.

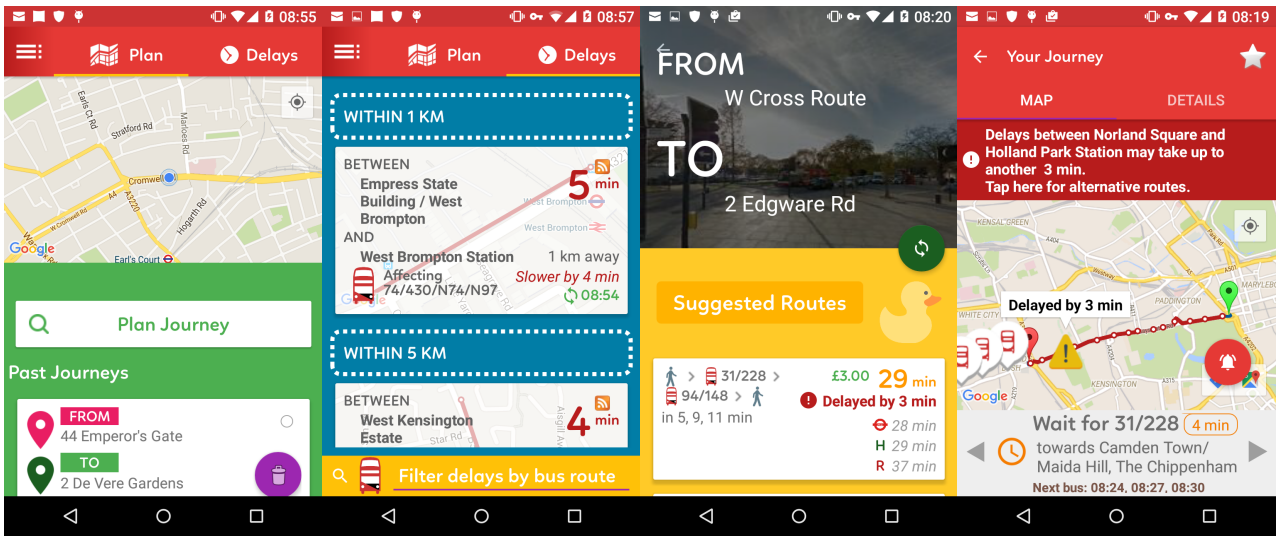


Figure 3.5.: Screenshots of the Android app. (1) The leftmost screenshot is the home page of the app, in which the current location of the user is shown. (2) The second shows all detected delays in Central London, sorted by the distances to the user's current location. (3) The third displays a streetview of the destination, route results, and journey times already taken into consideration of current delays. (4) The last gives an interactive map and detailed instructions of the intended route, and pinpoints where delays are occurring.

We take advantage of the location-aware capabilities of smartphones. For example, a user may only be interested in delays near his / her current position, so we order delays based on their

distances to the user's location. Delays along a route only has effect on the user if it lies between the user's current position and the destination, and so we need not alert the user if we detect that he / she has already travelled past the places affected by the delays.

Details of the app will be discussed in the user guide in appendix C.

4. Implementation

4.1. Stage I - Setting up backbone

4.1.1. Core database

The first stage involved acquiring the necessary background information from the core TfL dataset, including all bus route and stop information. A key motivation being that TfL imposes a quota on the developers' request limit per minute[21], and given the vast number of bus routes in Central London[22] it would be impractical (and inefficient) to query the TfL core dataset each time auxiliary information is required. Another is that by creating our own copy of route and stop information database, we could virtually perform any SQL query, not limited to the ways that the TfL API allows, such as which parameters must be supplied and which would be returned.

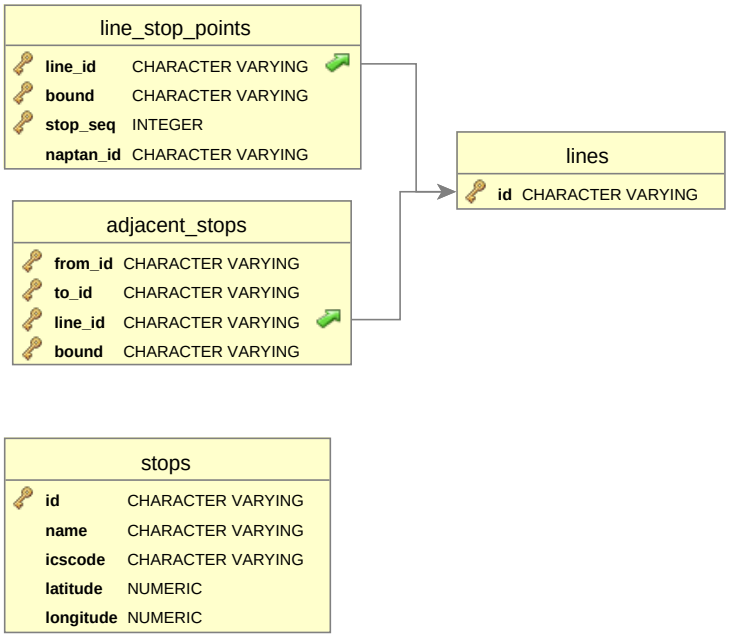


Figure 4.1.: UML diagram of the core tables `lines`, `stops`, `line_stop_points`, `adjacent_stops`

[21] As of 3 May, 2016, TfL sets a maximum request limit of 500 API calls per minute on their core dataset.
[22] 671 bus routes in Central London are managed by TfL as of 4 May, 2016.

Therefore we set up a few tables[23] that replicates the information available on the TfL servers. Figure 4.1 shows the relationships between the core tables that we are going to set up, in which the key icons denote primary keys of the table and arrows represent foreign keys.

The table `lines` contains all the bus routes that are managed by TfL. The `id` attribute is the identifier of each bus route (i.e. the bus route number).

The table `line_stop_points` records the stops information of each bus route. Its attributes consist of `line_id`, the identifier of the bus route; `bound`, the direction the route; `stop_seq`, the sequence number of the stop of the given route and direction[24], and finally `naptan_id` (NaPTAN identifier[25]), which is an internal identifier used by TfL to identify stops across Central London.

The table `adjacent_stops` is a rewrite of the table `line_stop_points`, where each record represents a pair of neighbouring stops of a given line and bound. It therefore contains all individual links in the London bus network, and are essentially the building blocks of all bus journeys, since all valid bus journeys by design can be decomposed into records in the table. The attributes `line` and `bound` have the same meaning as in the table `line_stop_points`, whereas `from_id` and `to_id` represent the NaPTAN identifiers of the pair of neighbouring stops involved.

Finally, the table `stops` contain all stop information across Central London. It records the NaPTAN identifier of each stop `id`, together with the `name` of the stop, `icscode` (another internal identifier that TfL uses to label their stops), and the world coordinates - `latitude` and `longitude`.

4.1.2. Data consistency

To ensure data consistency across the tables, the line identifier `id` in the `lines` table is made a foreign key to the `line_id` attribute of tables `line_stop_points` and `adjacent_stops`:

$$\begin{aligned} \text{line\_stop\_points(line\_id)} &\overset{fk}{\Rightarrow} \text{lines(id)} \\ \text{adjacent\_stops(line\_id)} &\overset{fk}{\Rightarrow} \text{lines(id)} \end{aligned}$$

since we would like to ensure that the details appearing in tables `line_stop_points` and `adjacent_stops` indeed refer to valid routes. By enabling `ON DELETE CASCADE` on the foreign key relations, we also have the added benefit that when TfL withdraws a bus route (i.e. removes an entry from the `lines` table), the corresponding details would be deleted automatically from the tables `line_stop_points` and `adjacent_stops`.

We could have done the same with attributes `naptan_id`, `from_id`, `to_id` in tables `line_stop_points` and `adjacent_stops`, as they all refer to NaPTAN ids, and should appear in the `id` column of

[23] These tables can be set up by running the scripts `createtables.go`, `updatelines.go`, `updatelinestoppoints.go`, `updateadjacentstops.go` and `updatestops.go` in order.

[24] Stop sequences are labelled sequentially from 0 onwards, where 0 represents the starting terminus of the route of a given direction.

[25] NaPTAN is a shorthand for "National Public Transport Access Nodes", which is Britain's national system for unique identification of points of access to public transport.[26]

the table `stops`.

$$\begin{aligned} \text{line\_stop\_points(naptan\_id)} &\overset{fk}{\Rightarrow} \text{stops(id)} \\ \text{adjacent\_stops(from\_id)} &\overset{fk}{\Rightarrow} \text{stops(id)} \\ \text{adjacent\_stops(to\_id)} &\overset{fk}{\Rightarrow} \text{stops(id)} \end{aligned}$$

This was unfortunately not possible execution wise, as the TfL API does not provide a direct call to obtain all stop information, and so we need to aggregate all appearances of stops in the tables `lines_stop_points` and `adjacent_stops` to form the `stops` table. We therefore have it the other way round instead[26].

4.1.3. Flexibility

We must also be aware that bus routes are subject to alteration any time. The TfL API refreshes its source of route information every day. Therefore, we must update our databases accordingly so as to be consistent, particularly for the route planning function that we will be implementing. A Cron job has been set up to synchronise the databases[27] every day at midnight 2am, which is a less busy time of the day to minimise disruptions.

Once the core database is set up we will then have the ability to watch out for invalid bus predictions, or edge cases in the next stage. For example, we might find that a bus prediction actually refers to a stop not on the route (possibly due to diversion), or that a prediction temporarily disappears. All these checks would not have been possible had we not have the exact routing information of each bus route.

Apart from validation of data, the core database helps immensely in providing information that is not straightforwardly available from the TfL API, such as detection of cycles in bus routes, querying intermediate stops and determining the relative order of two stops along a given bus route.

4.2. Stage II - Live feeds data collection

We now describe in detail the process of acquiring live bus predictions from the TfL web server, filtering out invalid predictions and handling of edge cases, before inserting them into the historic database of journey times.

[26] $\pi_{\text{naptan\_id}}\text{line\_stop\_points} = \left(\pi_{\text{from\_id}}\text{adjacent\_stops} \cup \pi_{\text{to\_id}}\text{adjacent\_stops}\right) \supseteq \pi_{\text{id}}\text{stops}$ in relational algebra terms.

[27] In effect the scripts `updatelines.go`, `updatelinestoppoints.go`, `updateadjacentstops.go` and `updatestops.go` are run in order.

4.2.1. Downloading live bus predictions from TfL

There are two main channels through which live bus predictions are broadcast from TfL. The first is the "instant" API, in which static JSON data is returned from the TfL server, and the source is regularly refreshed at regular intervals. The second is the "stream" API, in which JSON is continuously streamed through sockets.

While the latter has the advantage that bus predictions are released immediately once ready from the TfL server, and hence should lead to lower latency, it doesn't address our needs very well. For example, as predictions for each stop come in one by one, and that no particular ordering relative to the sequence of bus stops down a route is guaranteed, it is difficult to gather a complete snapshot of all arrival predictions, so as to determine the location of a bus based on the expected arrival times at each stop. TfL has also declared that owing to large amounts of data potentially transferred, the usage of each developer is actively monitored, and actions will be taken should there be excessive use. The term "excessive use" is not quantified clearly, and it is not worth the risk since we require live bus predictions 24/7 uninterrupted.

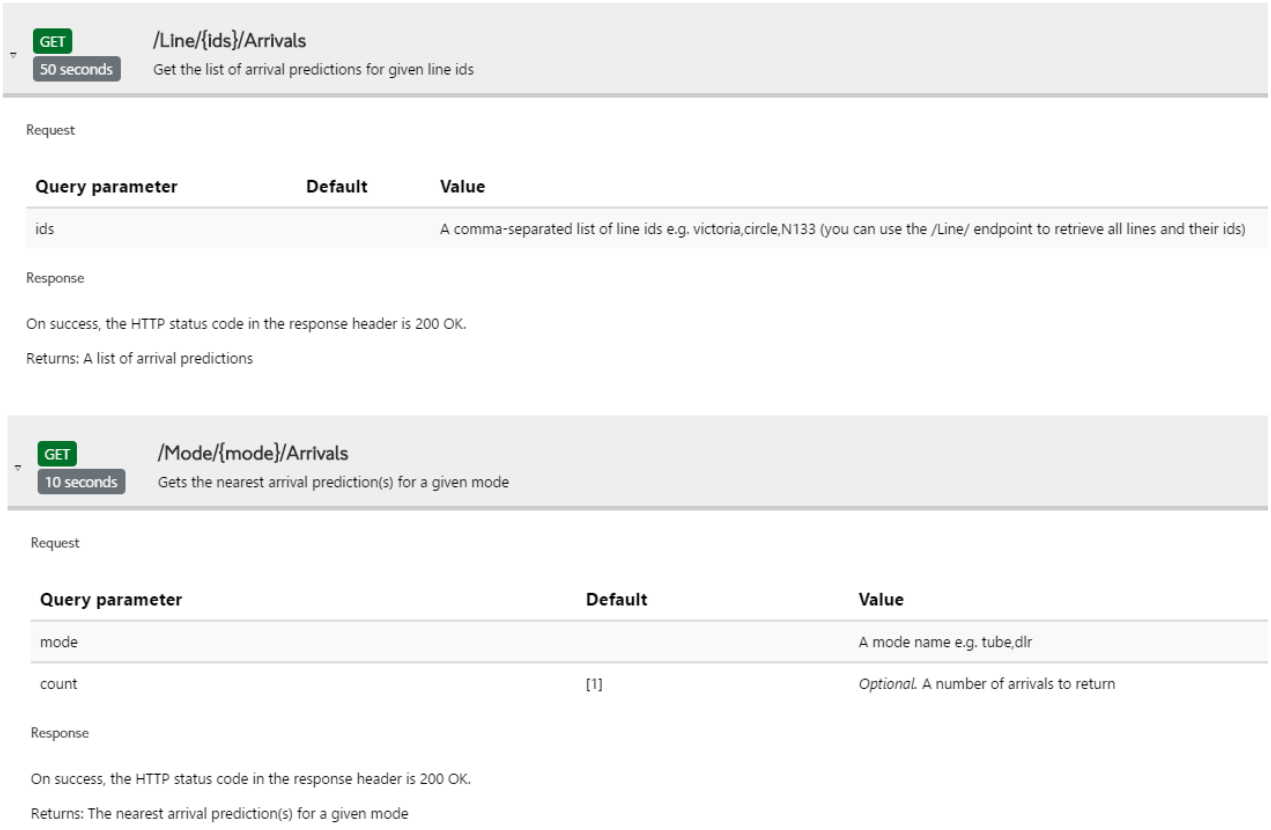


Figure 4.2.: A screenshot of the available "instant" TfL API calls that return live bus predictions. (Top) Returns arrival predictions by bus route, whose source is updated every 50 seconds. (Bottom) Returns the top N predictions of each bus stop for all stops in Central London, whose source is updated every 10 seconds.

Therefore, it is decided that we stick to the "instant" API. The "instant" API provides two main entry points for obtaining live bus predictions, one is querying by bus route, so that all predictions associated with the desired bus route are returned. The other returns the top N predictions of each bus stop for all stops across Central London (figure 4.2).

We would very much prefer the first approach, as we essentially would like to track the buses that are assigned to each bus route, in order to determine their locations within the route. Unfortunately, TfL imposes a limit of 500 requests per minute for each developer across all "instant" API calls, and given that there are 671 bus routes[28] in Central London, at best we would only be able to acquire all bus predictions in 2 minute cycles, which is not practical given the dynamic nature of bus predictions.

We therefore have to resort to the second approach. We would eventually need to create our own adapter to classify bus predictions by bus route, so as to monitor their relative positions along respective routes.

Since only the top N predictions are returned for each bus stop, it now boils down to determining an appropriate choice of N. Although TfL claims that the source is refreshed at 10 second intervals, after some observations it was found that predictions for each individual bus route are updated per 5 refreshes, and hence ≈ 50 seconds. Coincidentally this is exactly that of the first approach. We therefore need to ensure that N is large enough, to cover all bus arrivals that are about to happen in the next 50 seconds. What hinders N from being too large is the amount of data transferred (and internet speed), since with $N = 1$ it was found that approximately 1MB of data was transferred, so we would expect roughly NMB of data usage per query. Given that the source is updated every 10 seconds, we would ultimately want our handler to be able to complete each cycle within 10 seconds. Therefore, by trial and error we picked $N = 10$, assuming that less than 10 buses are arriving at each stop in the next 50 seconds, and checking that we are indeed collecting all predictions by examining the busiest stops in Central London.

4.2.2. Processing live bus predictions

Assumptions Now that we have got a bunch of bus predictions at hand, we need to interpret them to find out where the buses actually are. To this end, we need some assumptions:

- *(Assumption 1) A bus has departed a stop if its prediction at that stop disappears.*
- *(Assumption 2) The dwelling time at each stop is negligible, i.e. a bus is in between two stops almost surely.*
- *(Assumption 3) The time of departure (= time of arrival by the previous assumption) at a stop is taken to be the* `expectedArrival` *field claimed by the TfL prediction.*

[28] As of 4 May, 2016.

While at first sight assumption 2 seems unreasonable, it is regretfully difficult to obtain estimates for dwelling times, since such data is not provided by TfL. As we gather a large sample of bus predictions, each individual bus predictions would either overestimate or underestimate the actual arrival / departure times at stops, as hence the dwelling times should on average cancel out, and the averaged journey times should tend to the true mean[29].

Procedure With these assumptions settled, we could start figuring out when the buses arrived at each stop. We first partition all incoming predictions into their corresponding bus routes, so that we will monitor each vehicle continuously along the route the prediction claims it is on. This is done by setting up a cache for each bus route[30], in which we hold as keys the registration number of the vehicle to identify the bus currently serving that route, and as values all the predictions that refer to the bus. We can then determine which stop a bus is going to approach next, by extracting the prediction that yields the closest `expectedArrival` time from now.

We expect TfL to continuously give updates to each prediction, until the prediction disappears as the bus leaves the station. All we have to do is then update the corresponding cache entry, when the next batch of predictions come in. In case TfL withdraws a bus from the route (say at the end of the day or alteration of bus assignments), the corresponding entry in the cache would be deleted.

Noise removal As innocent as it seems, the above procedure doesn't work well in reality, due to unexpected behaviour[31] of the predictions. We list below the pitfalls that we have fallen into, and the corresponding workarounds.

1. Temporary disappearance of prediction

   It was found that predictions could temporarily disappear then reappear, thereby undermining assumption 1. A particular difficulty with this is that there is no guarantee on the maximum gap in between the disappearance and reappearance, and hence we cannot be sure that a disappearance is final and for real. To mitigate this, we need another assumption:

   - *(Assumption 4) The disappearance of a prediction after its claimed* `expectedArrival` *time is treated for real.*

[29] The central limit theorem says that if X_i are identical and independent random variables, then

$$\frac{\sum_{i=1}^{n} X_i}{n} \xrightarrow{d} N\left(E\left(X_i\right), \frac{\mathrm{Var}\left(X_i\right)}{n}\right)$$

i.e. the arithmetic mean of the X_i's would be asymptotically distributed as a normal distribution with the same mean as each random variable, and $\frac{1}{n}$ of the original variance.

If X_i represents the journey times inferred by bus predictions, then as we average over a large sample size (large n), we should obtain predictions very close to the true journey time, since the variance tends to 0 as $n \to \infty$.

[30] The caches are backed by the hashmap data structure. Operations on each cache run concurrently in different goroutines, for efficiency and isolation, so that in case an operation doesn't work for a bus route due to an edge case, the others would not be impacted.

[31] Unexpected in the sense that they are undocumented in the TfL API documentation[16].

Therefore, if a bus claimed that it would arrive at stop XYZ at 15:04:05, and the prediction disappeared at 15:04:26, we would see it as the bus departed stop XYZ at 15:04:05, regardless of whether the prediction reappears some time later.

It is debatable whether this is the best approach, as we might as well wait for another minute or so, by which time the probability of reappearance should have been much lower[32], before taking the disappearance of a prediction for real. Nevertheless we prefer to keep things simple and adopt the former approach.

2. Batch update of predictions

   Another anomaly is that rather than pushing out incremental updates at 10 second intervals for all predictions, i.e. if a bus departed a stop we would expect its prediction to disappear by the feed within the next 10 seconds, it was found that TfL releases all updates of a given bus route at the same time in batches approximately every 50 seconds[33]. There are even occasions where the source fails to refresh for some minutes, therefore predictions may not catch up with the live status of the bus very well.

   In particular we would often notice that by the next batch of updates, a bus would have been a few stops down the route from its last reported location. In such case, we could only rely on the same batch of predictions when it was last reported for the arrival times of the stops in between. This will be further discussed in the "Interpolation" section below.

3. Missing predictions

   There are cases where predictions are never supplied for some stops in between, possibly owing to diversion or other causes[34].

   It is not certain whether the bus has actually visited those stops, and therefore we discard the affected portions of the journey, without logging them in our database.[35]

4. Inconsistent predictions

   Since a prediction describes the expected arrival time to a stop, it does not automatically imply relationships across stops of the same bus route. It was discovered that, occasionally the expected arrival time of a stop comes temporally after that of another stop further down the route. This is clearly not possible in real life.

[32] We can model the temporary disappearance of a prediction as a two state Markov model. Let state 1 denote that a prediction is present, and state 2 denote its absence. Assuming constant hazards μ_{12} in a prediction disappearing and μ_{21} in reappearing, the probability that a prediction disappearing at time 0 and reappearing at time t is given by

$$p_{21}(t) = \frac{\mu_{21}}{\mu_{12}+\mu_{21}} - \frac{\mu_{21}}{\mu_{12}+\mu_{21}} e^{-(\mu_{12}+\mu_{21})t}$$

which follows an exponential decay.

[33] In essence each update comes every 50 seconds for each bus route, and there is a phase difference in multiples of 10 seconds between different routes.

[34] Another associated anomaly is when predictions do appear for a route, but not on stops of the route

[35] There is a saying of "*garbage in, garbage out*", that if reconstructing predictions do not represent the true arrival times well, then it may become noise and worsen our results rather than help.

As this anomaly is not observed to occur frequently, we decide to discard the affected portions.

5. Outdated predictions

   This appears to be a bug of the TfL server. At random times, the server returns a historic image of predictions (i.e. all predictions refer to some time in the past, as far as a day ago). By trial and error it was discovered that the source is dependent on the *count* parameter (the variable N referred to earlier in this section) supplied in the API request, and so by alternating between different numbers we seem to be able to bypass the problem, provided that not all sources fail at the same time.

   As a hack we decide to make N variable, rather than fixed at $N = 10$. As before we need to ensure that N is large enough to cover all predictions in Central London, but sufficiently small so that data can be transferred quickly for all data to be processed within the 10 second window frame. A little experiment suggested that

   $$N = \{10, 11, 12, 13, 14, 15\}$$

   yielded good results, since we did not notice any extra problems brought out compared to the case when $N = 10$. Therefore, whenever outdated predictions are detected, the program would try switching to another value of N, until the predictions resume normal.

Interpolation In order to deal with the edge cases described above, we apply interpolation[36] techniques in order to reconstruct or rectify invalid predictions in between stops.

We first collect all TfL predictions into the caches of each bus route described above. We note that TfL usually starts issuing predictions a few stops ahead, and hence in most cases we would have got the predictions for the few stops that are upcoming of each bus in our cache.

We are now ready to handle anomaly 2 - "Batch update of predictions". In the case when a bus has got to a few stops down the route by the time the next batch of predictions arrive, we would first find out the stops that were in between, and look at whether we have got those predictions in our cache. We note that these predictions may not be very accurate since they have not been subject to updates the whole time during this leg of the journey. Nevertheless, we adopt them whenever they are available and does not violate temporal constraints (*see* anomaly 4 - "Temporally Inconsistent predictions").

[36] Interpolation refers to a method of constructing new data points within the range of a discrete set of known data points.

4.2.3. Insertion into database

The table `journey_history`, which has been briefly introduced in the Design section (section 3.3.2), is the one that we keep past travel records of all buses. Each row consists of the fields:

- `line` - The bus route that a bus is serving
- `bound` - The direction of the bus on the bus route (either *inbound* or *outbound*)
- `vehicle_id` - The registration number of the bus
- `from_id` - The NaPTAN identifier of the stop that the bus has departed from
- `to_id` - The NaPTAN identifier of the stop that the bus is arriving at, or the next stop of `from_id` along the bus route and direction
- `from_timestamp` - The time at which the bus is acknowledged to have departed `from_id`
- `to_timestamp` - The time at which the bus is acknowledged to have arrived `to_id`
- `from_timestamp_tfl` - The time at which TfL generated the prediction `from_timestamp`
- `to_timestamp_tfl` - The time at which TfL generated the prediction `to_timestamp`
- `time_taken` - The time taken for this journey, defined as `to_timestamp - from_timestamp`.
- `interpolated` - A boolean indicating whether this record is a result of interpolation.
- `db_inserted_timestamp` - The time at which the record was inserted into database.

We note that each record refers to adjacent stops along a bus route, and hence we should be able to decompose any bus journey into the sum of these records. In effect we are pivoting the predictions obtained earlier to form this database, after applying the noise removal techniques discussed in the previous subsection.

To allow for fast searches for later analysis we have built multi-column indices on the frequently used columns, such as `from_id`, `to_id`, `from_timestamp`, `to_timestamp`, `line`, `bound` and `vehicle_id`.

We deliberately do not enforce foreign key relationships between this table and the core database, although in theory some columns should match the core database[37]. This is just in case TfL removes existing bus routes, in which case the corresponding entries in the core database will also be deleted on cascade, but we do not wish to lose the corresponding historic records in this table.

[37] For example the attribute `line` should match `line_id` in table `lines`, and `from_id`, `to_id` should match `id` in table `stops`.

4.2.4. Safety checks

Timing of each processing cycle To ensure that live feeds are processed fast enough within the 10 second window frame, so that each batch of incoming predictions won't lag further and further behind schedule cumulatively, we timed each cycle and logged them into our database.

Over the course of 100,000 cycles, it was found that on average 9.864 seconds were taken for each cycle. To be conservative, taking into account of internet speeds[38], the total cycle time might just exceed 10 seconds a little bit. Just to be sure of the effects, we also monitored the database of historic predictions from time to time, and did not notice any significant traces of delayed insertion[39].

Detection of loops in bus routes Loops in bus routes may also pose problems, in particular if a bus visits the same stop twice by design, we will need to take the effort to find out which visit does a bus prediction refer to.

By querying the core database, it turns out that only one route[40] is affected, it being a circular route (with the same departure and arrival terminus). Since there is insufficient information to tell whether a prediction refers to the time of arrival at the terminus or the time to wait until for departure, we simply do not log them into our database of historic records.

[38] Though this should be minor since the virtual machines which hosted the script had a measured internet connection speed of ≈500Mb/s.

[39] This is done by spotting the difference between the fields `db_inserted_timestamp` and `to_timestamp`, which in theory should be minimal. Indeed we observed insignificant deviation throughout the project (at most a few seconds off).

[40] Route H2 serving the Golders Green area, as of 6 May, 2016.
Search performed with the following SQL query:

```
SELECT   DISTINCT line_id, naptan_id, name
FROM     line_stop_points AS t
         JOIN    stops
         ON      line_stop_points.naptan_id=stops.id
         WHERE   EXISTS (   SELECT *
                            FROM    line_stop_points
                            WHERE   t.line_id=line_id
                            AND     t.bound=bound
                            AND     t.naptan_id=naptan_id
                            AND     t.stop_seq<>stop_seq);
```

returning results

line_id	naptan_id	name
h2	490015496GW	Golders Green Station

5. Experimentation

With the historic database of bus travel times set up, we can start to utilise information stored therein to generate better predictions. The question is how - is there a formula or theory that tells us how best to use them?

5.1. Periodicity

A first attempt involves figuring out patterns between historic data and real time data, since if there is a high correlation between the past and future, we might be able to get very good results just by reusing historic data.

A natural potential to explore is the periodicity of data. Commuters may have regular travel patterns, such as daily commutes to work. Buses may adhere to the same schedule on different days of the week. Road conditions may exhibit certain patterns over time, such as congestions during peak hours of the day, and quiet in the middle of the night. We therefore try to look into any trends that are prevalent in the set of historic data.

5.1.1. Choice of bus stop pairs under study

To find out whether bus journey times are periodic, an experiment has been set up on a number of route segments in Central London. In order to be fair, and to find out any interesting variations across the city, we have chosen a few locations that are at the heart of the city centre, and some near the outskirts. The specific route segments that have been studied are:

- Savoy Street - Bedford Street[41]
- Waterloo - Lancaster Place[42]
- Marble Arch Underground Station - Dorchester Hotel[43]

[41] Served by 24 routes, namely 11,13,139,15,176,23,6,87,9,91,N11,N13,N15,N155,N199,N21,N26,N343,N44,N551,N87,N89,N9,N91, as of 8 May, 2016.
[42] Served by 20 routes, namely 1,139,168,171,172,176,188,243,26,341,4,59,68,76,N1,N171,N343,N68,RV1,X68, as of 8 May, 2016.
[43] Served by 15 routes, namely 10,137,148,16,2,36,414,436,73,74,82,N137,N16,N73,N74, as of 8 May, 2016.

- Regent Street / St Jame's - Piccadilly Circus Underground Station[44]
- Exhibition Road - Royal Albert Hall[45]
- Nutfield Close - White Hart Lane Rail Station[46]
- East Croydon - Lunar House[47]
- Heathrow Park Thistle Hotel - Pinglestone Close[48]
- Rochester Way / Falconwood Station - Eltham Cemetery[49]

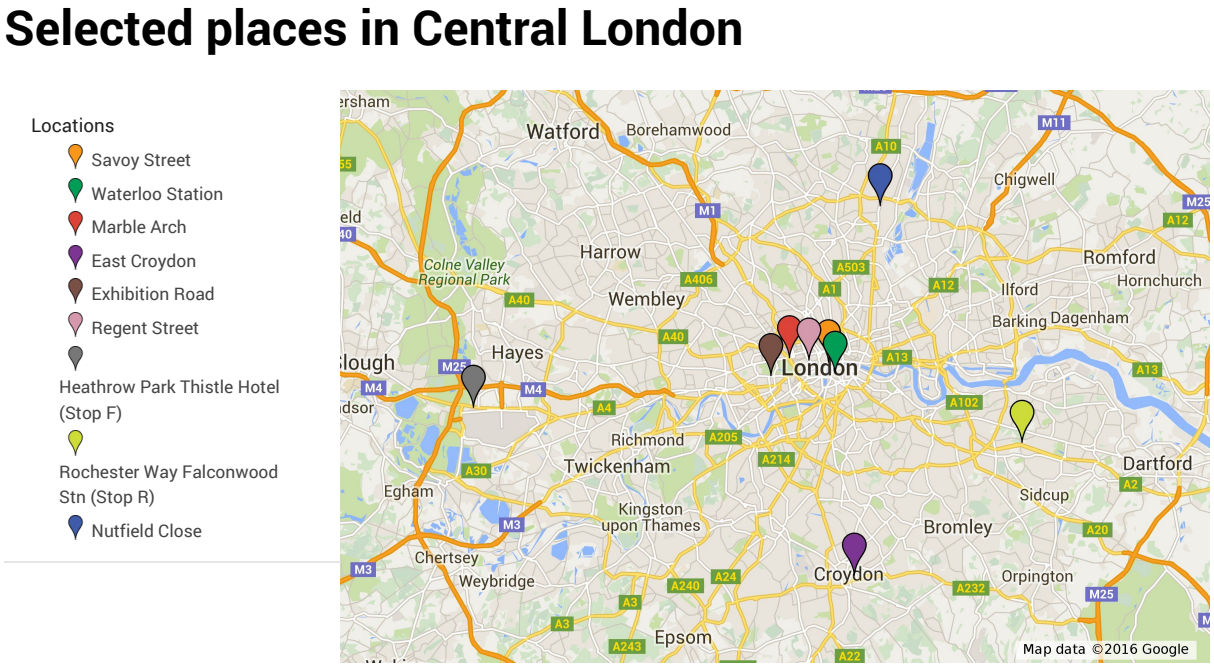


Figure 5.1.: A map of the route segments that we are examining for periodicity in bus journey times.

[44] Served by 17 routes, namely 12,13,139,159,23,3,453,6,88,94,N109,N113,N13,N136,N18,N3,N97, as of 8 May, 2016.
[45] Served by 5 routes, namely 10,452,52,9,N9, as of 8 May, 2016.
[46] Served by 5 routes, namely 149,259,279,349,N279, as of 8 May, 2016.
[47] Served by 2 routes, namely 50,N68, as of 8 May, 2016.
[48] Served by 2 routes, namely 423,81, as of 8 May, 2016.
[49] Served by 3 routes, namely 621,B15,B16, as of 8 May, 2016.

Figure 5.1 shows the locations of the selected route segments on a map. Four have been chosen to spread out a little bit in the north, east, south and west directions, and the remaining five in the city centre. We hope that these locations will be representative of the general situation of Central London.

5.1.2. Autocorrelation Plot

A popular method in time series analysis is examining the autocorrelation of a set of data. Autocorrelation refers to the correlation of a time series with itself a specific time apart (called the time lag). For example, if a series has a period of 12 months, we should expect a high correlation between the series and itself shifted by 12 months. Formally, the autocorrelation sequence ρ_τ representing the autocorrelation at different lags τ of a time series X_t is defined by

$$\rho_\tau := \frac{\mathrm{Cov}\,(X_t, X_{t+\tau})}{\mathrm{Var}\,(X_t)} \tag{5.1}$$

As with correlation coefficients, a value near 1 indicates X_t and $X_{t+\tau}$ are highly positively correlated, and hence strongly suggests a period of τ for the time series X_t. On the contrary, a value near -1 indicates negative correlation, and suggests that the series exhibits opposite effects when the time is τ apart. A value near 0 shows weak / no correlation, so the series may not exhibit periodicity.

The autocorrelation series are then computed[50] for each of these sections, with historic bus journey times collected in our database ranging over a month[51]. The series can be visualised as plots, with time lag on the x - axis and the autocorrelation at each lag on the y - axis.

[50] We first partition each day into 24-hour blocks, and classify the historic bus journey times into their corresponding blocks. We can then represent each block with a single value on the average journey time within that hour. We treat each block as realisations of the random variable X_t, where X_t represents the average journey times within the t^{th} block. Finally, the estimator (corresponding to equation 5.1)

$$\hat{\rho_\tau} = \frac{\frac{1}{N}\sum_{t=1}^{N-\tau}(X_t - \bar{X})(X_{t+\tau} - \bar{X})}{\frac{1}{N}\sum_{t=1}^{N}(X_t - \bar{X})^2}$$

is used to compute the autocorrelation coefficients of the corresponding time lags τ (the difference in indices of the blocks, which is exactly how many hours apart are the two blocks of data).

[51] Data taken from the period 23 February - 31 March, 2016.

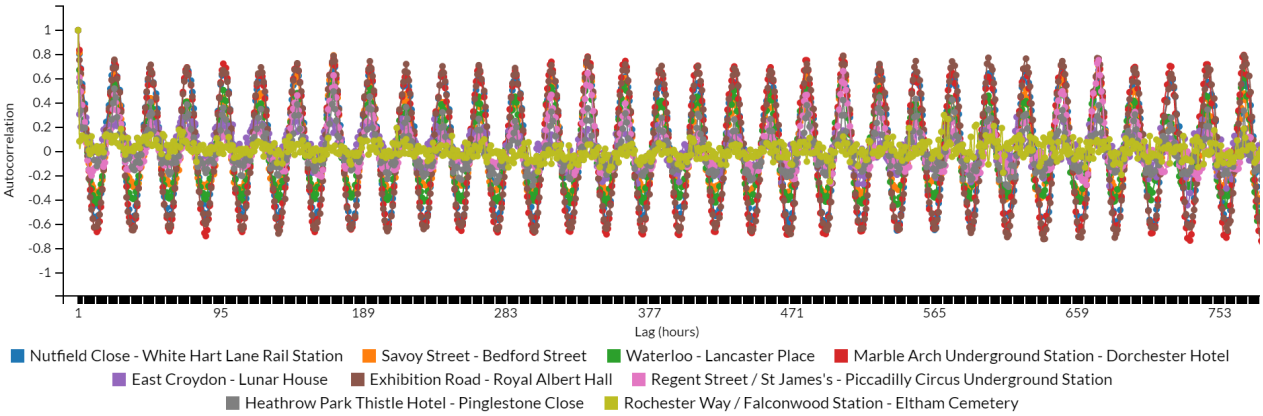
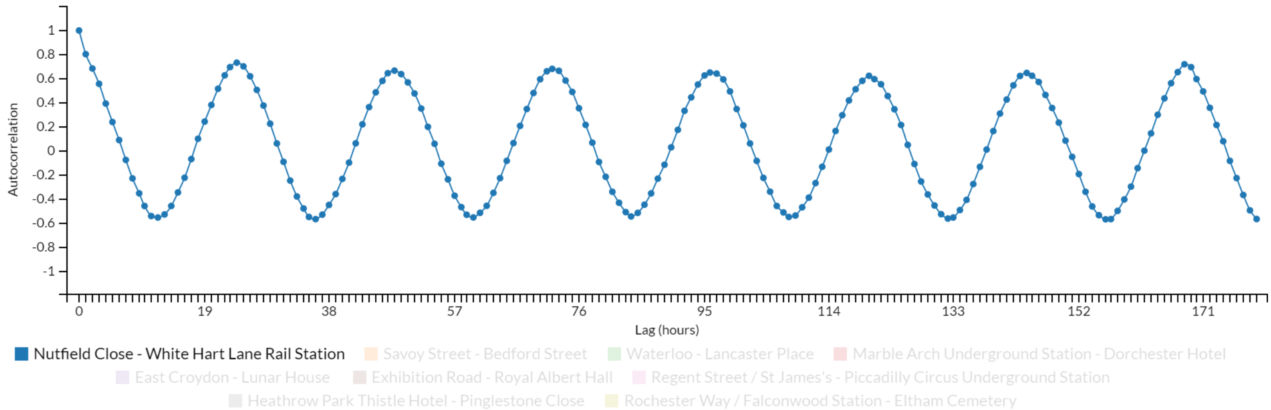
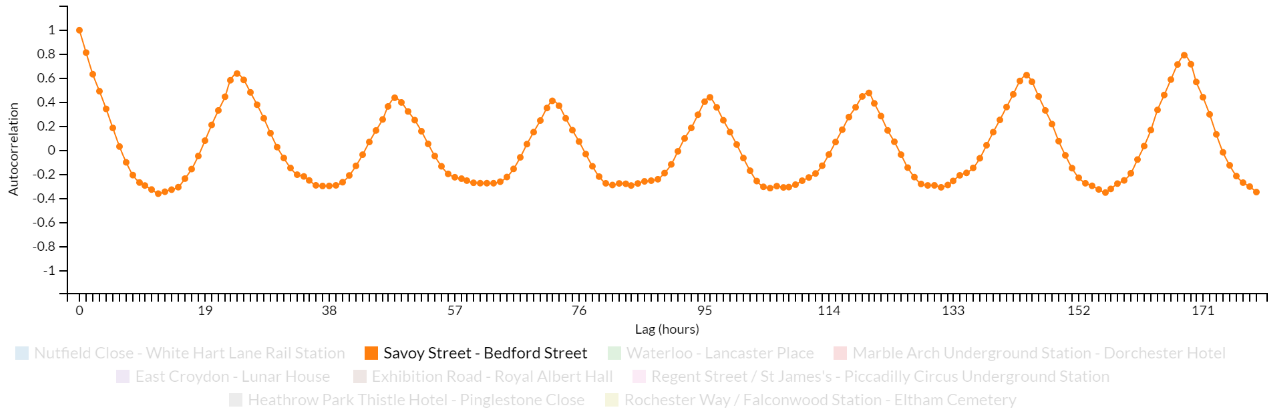


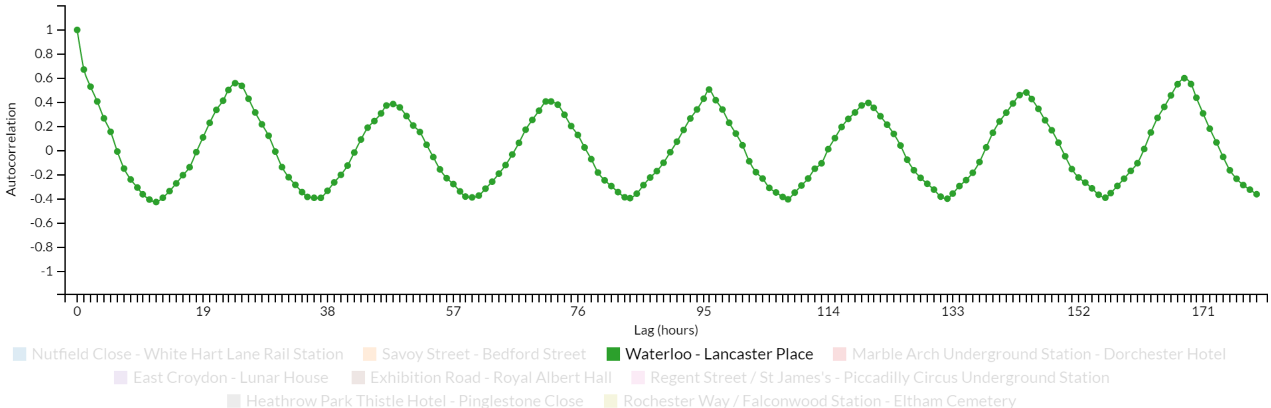
Figure 5.2.: Autocorrelation plot of bus journey times for the selected route segments, with lags shown until 31 days = 744 hours. We see different shapes (and therefore periods) depending on the location.

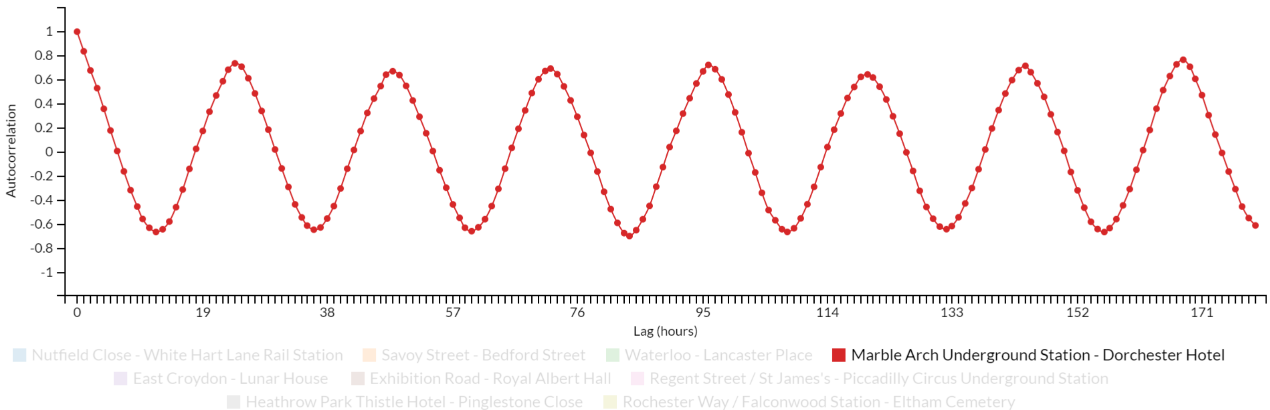
A plot of the autocorrelation sequences of bus journey times for lags between 0 and 31 days is shown in figure 5.2. On first sight we see that the sequences vary in amplitude and shape, but are rather consistent in terms of periodicity. In fact, all of them seem to have local peaks at 24 hour intervals. This strongly suggests that journey times have something to do with the time of the day.

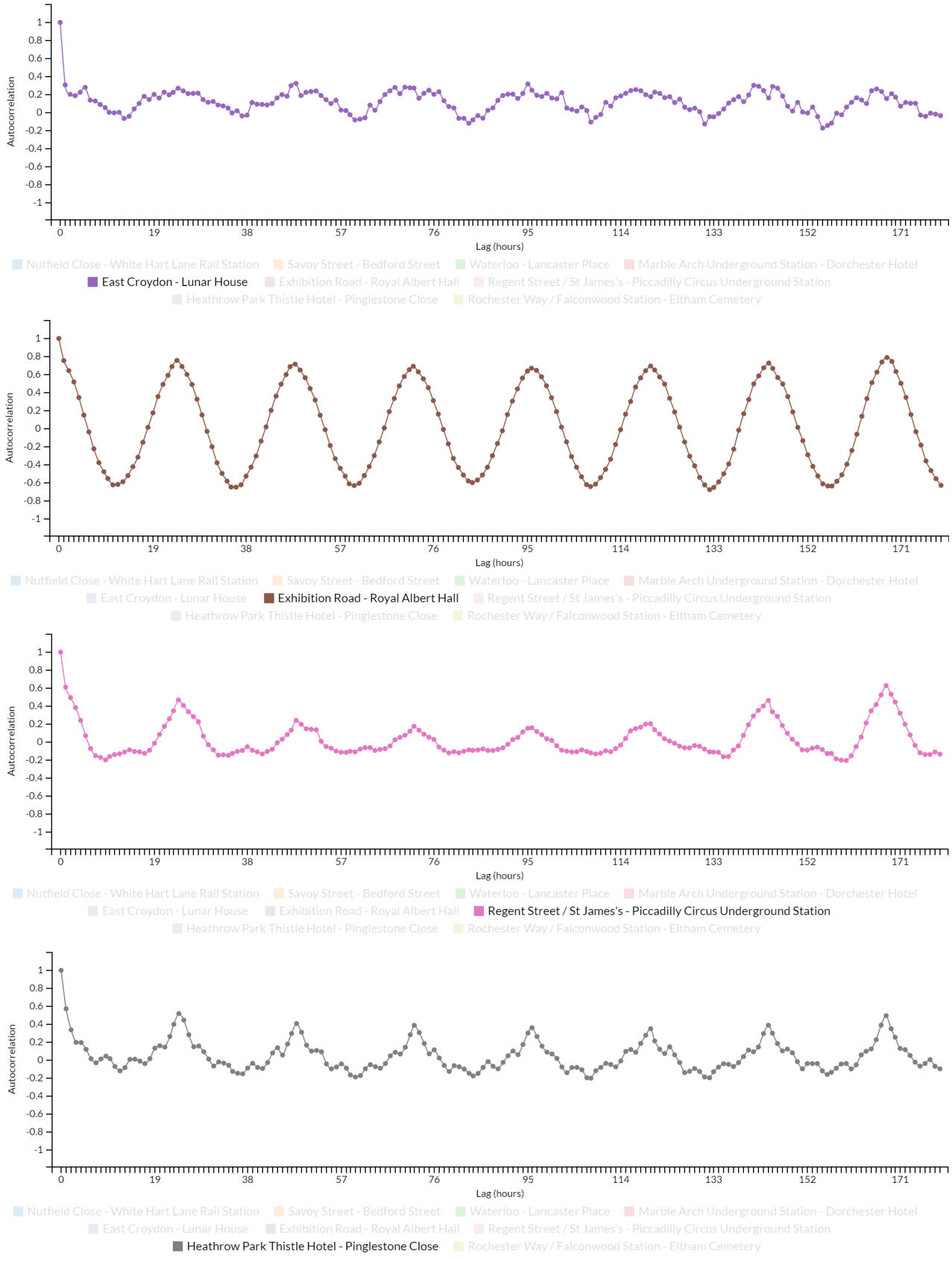
We also observe that the graph almost certainly repeats itself every 7 peaks (i.e. 7 days). To look into the details, we present each of the plots separately, and truncated to show only the first 7 peaks (i.e. time lag $\leq 7 \times 24 = 168$ hours).


Autocorrelation
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
19
38
57
76
95
114
133
152
171
Lag (hours)
Nutfield Close - White Hart Lane Rail Station
Savoy Street - Bedford Street
Waterloo - Lancaster Place
Marble Arch Underground Station - Dorchester Hotel
East Croydon - Lunar House
Exhibition Road - Royal Albert Hall
Regent Street / St James's - Piccadilly Circus Underground Station
Heathrow Park Thistle Hotel - Pinglestone Close
Rochester Way / Falconwood Station - Eltham Cemetery


Autocorrelation
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
19
38
57
76
95
114
133
152
171
Lag (hours)
Nutfield Close - White Hart Lane Rail Station
Savoy Street - Bedford Street
Waterloo - Lancaster Place
Marble Arch Underground Station - Dorchester Hotel
East Croydon - Lunar House
Exhibition Road - Royal Albert Hall
Regent Street / St James's - Piccadilly Circus Underground Station
Heathrow Park Thistle Hotel - Pinglestone Close
Rochester Way / Falconwood Station - Eltham Cemetery


Autocorrelation
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
19
38
57
76
95
114
133
152
171
Lag (hours)
Nutfield Close - White Hart Lane Rail Station
Savoy Street - Bedford Street
Waterloo - Lancaster Place
Marble Arch Underground Station - Dorchester Hotel
East Croydon - Lunar House
Exhibition Road - Royal Albert Hall
Regent Street / St James's - Piccadilly Circus Underground Station
Heathrow Park Thistle Hotel - Pinglestone Close
Rochester Way / Falconwood Station - Eltham Cemetery


Autocorrelation
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
19
38
57
76
95
114
133
152
171
Lag (hours)
Nutfield Close - White Hart Lane Rail Station
Savoy Street - Bedford Street
Waterloo - Lancaster Place
Marble Arch Underground Station - Dorchester Hotel
East Croydon - Lunar House
Exhibition Road - Royal Albert Hall
Regent Street / St James's - Piccadilly Circus Underground Station
Heathrow Park Thistle Hotel - Pinglestone Close
Rochester Way / Falconwood Station - Eltham Cemetery


Autocorrelation
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
19
38
57
76
95
114
133
152
171
Lag (hours)
Nutfield Close - White Hart Lane Rail Station
Savoy Street - Bedford Street
Waterloo - Lancaster Place
Marble Arch Underground Station - Dorchester Hotel
East Croydon - Lunar House
Exhibition Road - Royal Albert Hall
Regent Street / St James's - Piccadilly Circus Underground Station
Heathrow Park Thistle Hotel - Pinglestone Close
Rochester Way / Falconwood Station - Eltham Cemetery

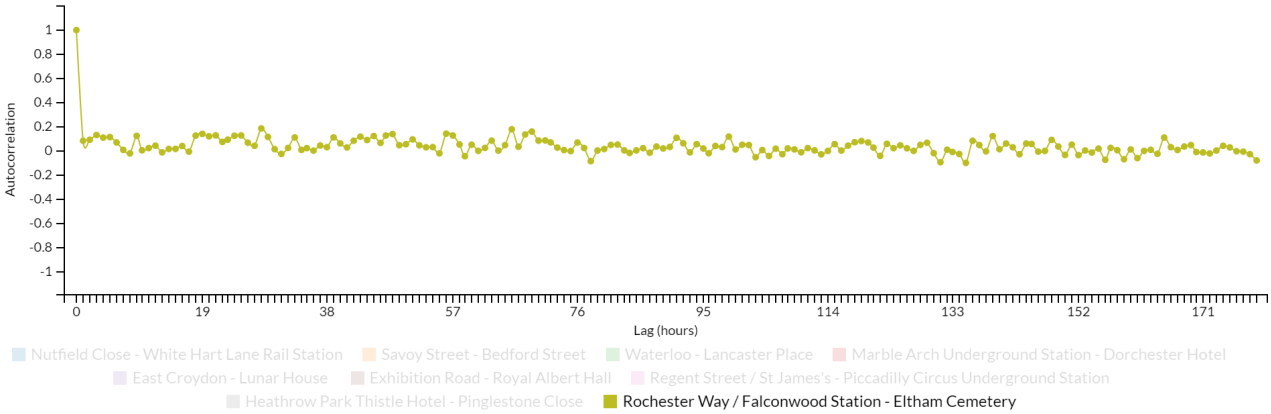


Figure 5.3.: Closer looks into the autocorrelation plots of each individual route segment, with lags shown for the first 7 days ($\approx$ 168 hours).

We note that all of them has perfect autocorrelation at time lag 0. This is not at all surprising, as a time lag of 0 simply means that we are correlating the series with itself, which should yield perfect match. This is not to be taken as the series has a period of 0.

An interesting finding is the variety in smoothness of the shapes of the autocorrelation plots. We see that most plots are sinusoidal like, whereas some are a bit rough and noisy. This may be due to the nature of the route section being examined, some such as *Heathrow Park Thistle Hotel - Pingleton Close* and *Rochester Way / Falconwood Station - Eltham Cemetery* are not served by night buses, and hence the lack of data in late night and early mornings. We therefore do not see that much of a negative correlation at time lags of $12, 36, \ldots, 12 + 24k$ hours.

It may also be the case that the road conditions are not sensitive to the time of the day, particularly in the suburbs (*Rochester Way / Falconwood Station - Eltham Cemetery*), and hence we observe a relatively flat autocorrelation series, mostly composed of random white noise.

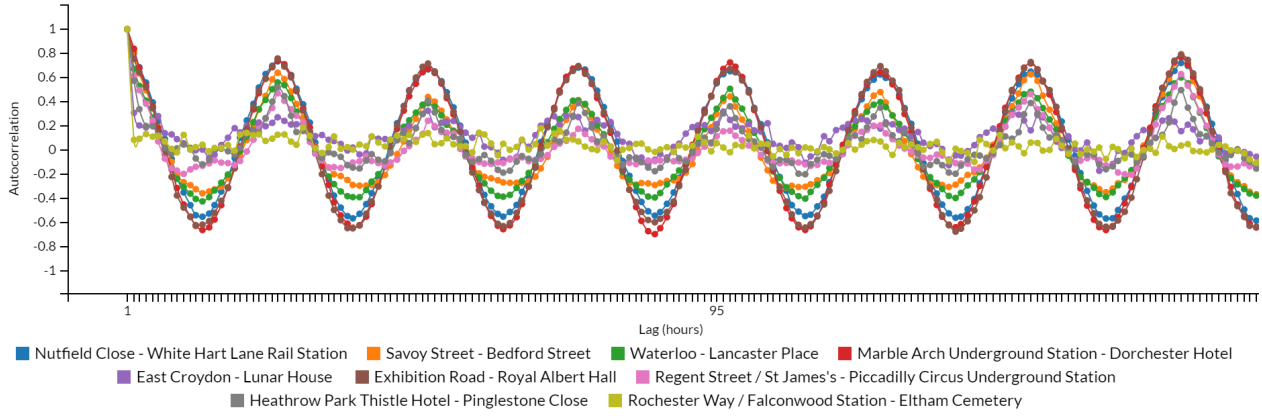


Figure 5.4.: Autocorrelation plots of all route segments superposed together, with lags shown for the first 7 days ($\approx$ 168 hours).

Another observation worth mentioning is the relative peak heights of the autocorrelation sequences. Figure 5.4 shows the autocorrelation plots superposed together, covering time lags up to seven days. The route sections

- *Exhibition Road - Royal Albert Hall*,
- *Marble Arch Underground Station - Dorchester Hotel*,
- *Nutfield Close - White Hart Lane Rail Station*

consistently reach high peaks every 24 hours, with an autocorrelation as much as $\pm - 0.8$, whereas some others, such as

- *Savoy Street - Bedford Street*,
- *Regent Street / St Jame's - Piccadilly Circus Underground Station*

have considerably lower peaks for time lags of 2 - 5 days. This may mean that traffic conditions of the former route sections are almost the same every day, and therefore less sensitive to the day of the week, whereas the latter ones exhibit very different behaviour depending on whether the day is a weekday or weekend.

Having looked at the autocorrelation plots, there is reason to believe that bus journey times of most routes exhibit periodicity. The problem rather lies with the different periods that each route has. While it would be great if we could determine the optimal periods of all individual route segments in Central London, it is far too costly. We seek a trade-off - we assume that all route segments have the same period.

The autocorrelation plots above suggest 24 hours and 7 days as potential candidates. While almost all plots achieve their highest peaks at 7 day intervals, and hence we should expect better results, reusing data every 24 hours means that we could also take into account of recent disruptions, such as road works. We therefore need to investigate the effectiveness of both methods. This will be left to the Evaluation section (section 6).

5.2. Delay classification

A main aim of the project is to successfully detect real-time delays, and alerting the user for alternative routes. We therefore need a mechanism to determine whether a delay has occurred, and ideally whether we expect it to persist until the user passes by that section.

An approach that we will be taking involves investigating historic delays, in order to find out any characteristics that we can associate with the delays at different locations. For example, we might look at the "shapes" of the delays, that is, how long a delay typically lasts before traffic resumes

normal. We could also look at the certainty of delay inference - as we observer more and more buses to take longer than usual, should we be more or less optimistic that a delay will persist for the next 10 minutes?

5.2.1. Choice of bus stop pairs under study

In contrast to section 5.1.1, in which we have sampled a wide variety of locations throughout London to examine the periodicity in bus journey times, we focus our attention on areas of high traffic flow where delays occur frequently. By searching through the historic database of delay records, the top 10 locations where bus journey times take longer or shorter than usual[52] are:

1. Oxford Circus - Tottenham Court Road
2. Museum Street - Cambridge Circus
3. Monument - Great Tower Street
4. Grosvenor Gardens - Victoria Coach Station
5. Gerrard Place - Denmark Street
6. Edith Grove - Worlds End Health Centre
7. Blackbird Cross - Kings Drive
8. Park House - Hampton Court Gardens
9. Beechfield Road - Catford & Catford Bridge Stations
10. Cephas Street - Darling Row

[52] Based on records between 24 February, 2016 and 12 May, 2016.

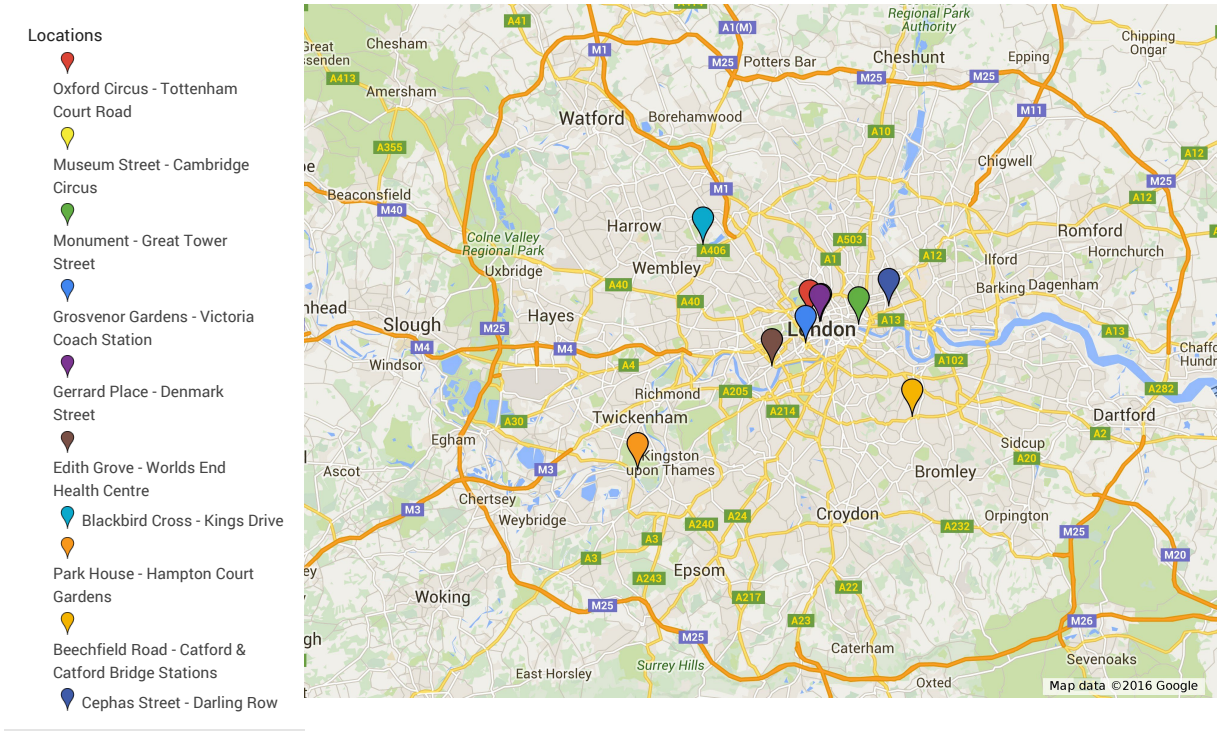


Figure 5.5.: A map of the route segments that we are examining for delay in bus journey times.

Figure 5.5 shows the locations of the above selected route segments. We see that most lie within the city centre, with only a few exceptions.

5.2.2. Choice of Measures

In order to identify delays, we must choose a baseline that we assume are normal traffic times, so that we can then claim a delay if recent travel times are much in excess of the baseline. To this end, we pick the *Historic 5 days weekdays; 7/14/21 days weekends (half hour offset with 1 hour window)* method, one of the best historic based prediction methods (to be discussed in section 6.2 - *Prediction Methods*), which utilises past data to generate current estimates of travel times, as a basis for the usual traffic times when delays are not occurring.

We also need to define the thresholds that we will be using, which when exceeded can we then claim a delay.

The first is the "delay threshold", that is, the length of period (or duration) by which recent travel times exceed historic counterparts. Say if we set a delay threshold of 2 minutes, then if a bus typically takes 5 minutes to travel along a road section, and today we find that it takes 7 minutes, we can conclude that a delay is occurring.

The second is the "bus count threshold", that is, the consecutive number of buses which we observe have exceeded the delay threshold. If we set it to three, then if we detected three consecutive buses having taken longer than usual by 2 minutes (assuming we are using the same delay threshold of 2 minutes as above), then we claim that a delay is occurring. Note that the "bus count threshold" is independent of the "delay threshold".

With these thresholds in mind, we can derive useful heuristics that helps us understand the underlying structure of delays.

5.2.3. Lasting time of delays

We can construct a table of the combined effect of knowing the delay threshold and bus count threshold.

Average lasting time of delays

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes	4m43s	5m7s	6m10s	6m56s	6m17s	7m41s	8m11s	9m34s	9m25s	8m17s
3 minutes	4m43s	4m20s	5m1s	6m6s	4m56s	5m26s	5m5s	5m49s	8m59s	6m41s
4 minutes	3m45s	3m34s	3m43s	3m45s	3m55s	3m51s	4m18s	3m47s	2m34s	
5 minutes	4m0s	4m5s	3m13s	4m21s	3m34s	3m7s	1m50s	1m54s	48s	
6 minutes	3m12s	3m1s	1m54s	1m42s	1m27s					
7 minutes	2m7s	1m58s	2m51s	2m18s						
8 minutes	42s									
9 minutes										
10 minutes										

Average lasting time of early trips

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes	10m45s	8m47s	9m51s	10m31s	10m30s	10m53s	10m8s	9m59s	10m48s	9m41s
3 minutes	7m50s	7m52s	7m57s	7m34s	6m54s	11m15s	7m10s	10m51s	10m1s	7m34s
4 minutes	8m3s	7m8s	6m52s	11m58s	7m44s					
5 minutes	8m21s	7m33s	7m24s							
6 minutes	7m39s	6m38s								
7 minutes	8m33s									
8 minutes	9m15s									
9 minutes	13m27s									
10 minutes										

Figure 5.6.: A table of the average persisting delay and "early" durations of buses travelling between Oxford Circus and Tottenham Court Road Station. Cells of deeper green represent delays can be expected to last longer, given that we have continuously observed that number of buses exceeding a particular delay threshold. Cells in red refer to lack of information, i.e. we have not observed any occurrences of that number of consecutive buses being delayed by a minimum of the corresponding delay threshold.

Figure 5.6 shows the average persisting duration of delays and "early" bus journeys (bus journeys that are faster than usual)[53] between Oxford Circus sand Tottenham Court Road, given that we have observed a consecutive number of buses that have exceeded the delay threshold (for "early" journeys that would refer to journeys that are faster than the usual travel time by the duration specified by the delay threshold). We observe that on the whole (subject to small perturbations), if we look at columns, extreme delays actually occur less frequent than minor delays, conditioning on observing the same number of buses that have exceeded a particular delay threshold. If we look at rows, we see that certainty in delays lasting longer first increases, then decreases, as we observe more and more buses that exceed a particular delay threshold.

The details for the other pairs of stops under study are shown below:

Average lasting time of delays

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes	13m49s	18m11s	17m36s	16m50s	20m15s	26m21s	29m6s	28m32s	26m20s	37m22s
3 minutes	12m25s	23m45s	23m6s	23m16s	23m56s	31m1s	27m26s	31m32s	19m0s	55m7s
4 minutes	23m27s	32m27s	45m11s	36m11s	40m51s	36m5s	33m5s	39m50s	36m5s	55m7s
5 minutes	38m20s	57m52s	34m8s	40m56s	28m32s	45m35s	19m23s	17m2s	15m39s	13m0s
6 minutes	11m11s	17m2s	15m39s	13m0s	11m59s	2m58s	1m22s			
7 minutes	15m40s	14m17s	11m38s	10m37s	1m36s					
8 minutes	11m38s	10m37s	1m36s							
9 minutes										
10 minutes										

Average lasting time of early trips

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes	20m48s	28m0s	32m58s	41m37s	46m6s	53m12s	51m36s	49m49s	50m54s	51m5s
3 minutes	16m11s	22m37s	28m16s	38m19s	45m43s	42m5s	48m35s	50m29s	1h5m58s	1h11m27s
4 minutes	15m26s	22m4s	27m42s	30m44s	29m46s	30m26s	36m37s	39m10s	36m38s	32m30s
5 minutes	14m45s	23m40s	45m50s	55m11s	52m49s	49m58s	1h8m8s	1h4m32s	1h2m34s	57m0s
6 minutes	13m59s	18m58s	20m37s	17m12s	28m1s	26m19s	20m14s	18m17s	15m41s	22m26s
7 minutes	14m0s	20m16s	26m48s	22m36s	19m19s	17m28s	22m55s	21m23s	18m9s	17m7s
8 minutes	5m12s	2m12s								
9 minutes	4m42s	3m14s								
10 minutes										

Figure 5.7.: Museum Street - Cambridge Circus

Average lasting time of delays

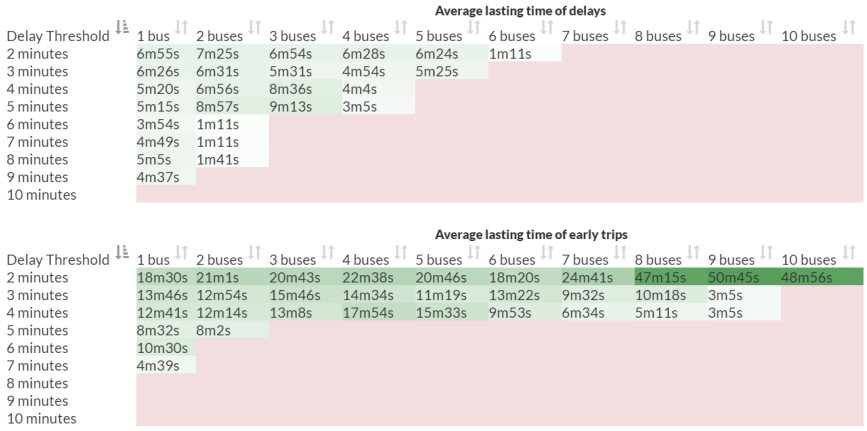
Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes	20m9s	24m1s	24m32s	22m28s	24m5s	22m18s	20m34s	23m27s	34m48s	34m13s
3 minutes	17m13s	24m55s	19m30s	22m56s	56m8s	39m46s	1h11m43s	1h3m20s	58m19s	58m41s
4 minutes	15m59s	20m55s	18m31s	14m16s	17m27s	9m4s	4m3s	4m25s		
5 minutes	13m37s	20m52s	20m21s	4m43s						
6 minutes	14m38s	24m17s	28m14s							
7 minutes										
8 minutes										
9 minutes										
10 minutes										

Average lasting time of early trips

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes	40m38s	49m10s	1h0m28s	59m39s	1h4m58s	1h6m38s	1h11m49s	1h9m58s	1h11m53s	1h12m30s
3 minutes	31m9s	36m27s	44m5s	44m40s	45m48s	45m27s	48m46s	52m30s	50m13s	57m0s
4 minutes	25m49s	36m20s	36m44s	41m10s	45m27s	45m43s	1h13m6s	1h20m45s	1h13m59s	1h45m39s
5 minutes	29m18s	51m11s	1h53m36s	1h49m58s	1h43m49s	1h39m12s	3h3m54s	2h46m1s	2h40m23s	2h40m58s
6 minutes	19m29s	26m26s	1h7m13s	1h3m10s	53m46s	42m58s	42m45s	35m49s	23m57s	19m59s
7 minutes	16m15s	29m49s	19m1s	18m48s	11m52s					
8 minutes	18m48s	11m52s								
9 minutes	6m56s									
10 minutes										

Figure 5.8.: Monument - Great Tower Street

[53] This case is prominent especially during bank holidays, when traffic tends to be much quieter than usual.



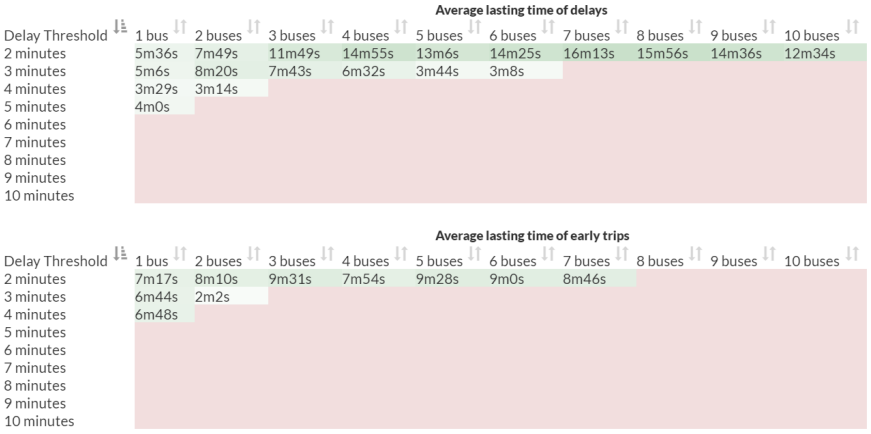
Average lasting time of delays

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes	6m55s	7m25s	6m54s	6m28s	6m24s	1m11s				
3 minutes	6m26s	6m31s	5m31s	4m54s	5m25s					
4 minutes	5m20s	6m56s	8m36s	4m4s						
5 minutes	5m15s	8m57s	9m13s	3m5s						
6 minutes	3m54s	1m11s								
7 minutes	4m49s	1m11s								
8 minutes	5m5s	1m41s								
9 minutes	4m37s									
10 minutes										

Average lasting time of early trips

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes	18m30s	21m1s	20m43s	22m38s	20m46s	18m20s	24m41s	47m15s	50m45s	48m56s
3 minutes	13m46s	12m54s	15m46s	14m34s	11m19s	13m22s	9m32s	10m18s	3m5s	
4 minutes	12m41s	12m14s	13m8s	17m54s	15m33s	9m53s	6m34s	5m11s	3m5s	
5 minutes	8m32s	8m2s								
6 minutes	10m30s									
7 minutes	4m39s									
8 minutes										
9 minutes										
10 minutes										

Figure 5.9.: Grosvenor Gardens - Victoria Coach Station



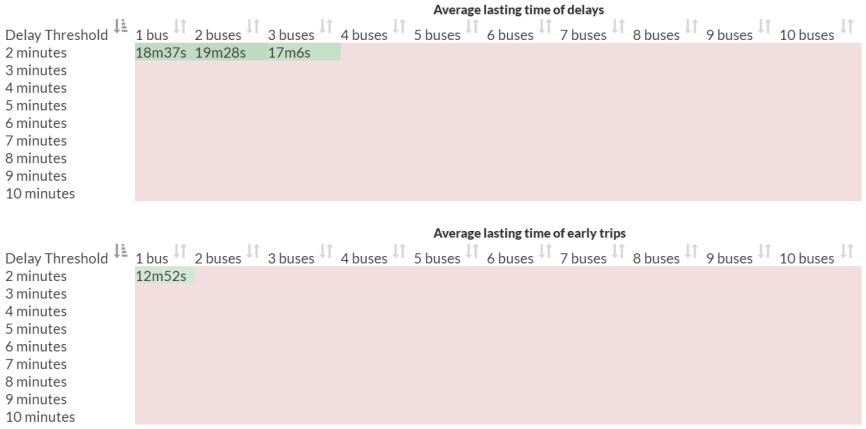
Average lasting time of delays

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes	5m36s	7m49s	11m49s	14m55s	13m6s	14m25s	16m13s	15m56s	14m36s	12m34s
3 minutes	5m6s	8m20s	7m43s	6m32s	3m44s	3m8s				
4 minutes	3m29s	3m14s								
5 minutes	4m0s									
6 minutes										
7 minutes										
8 minutes										
9 minutes										
10 minutes										

Average lasting time of early trips

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes	7m17s	8m10s	9m31s	7m54s	9m28s	9m0s	8m46s			
3 minutes	6m44s	2m2s								
4 minutes	6m48s									
5 minutes										
6 minutes										
7 minutes										
8 minutes										
9 minutes										
10 minutes										

Figure 5.10.: Gerrard Place - Denmark Street



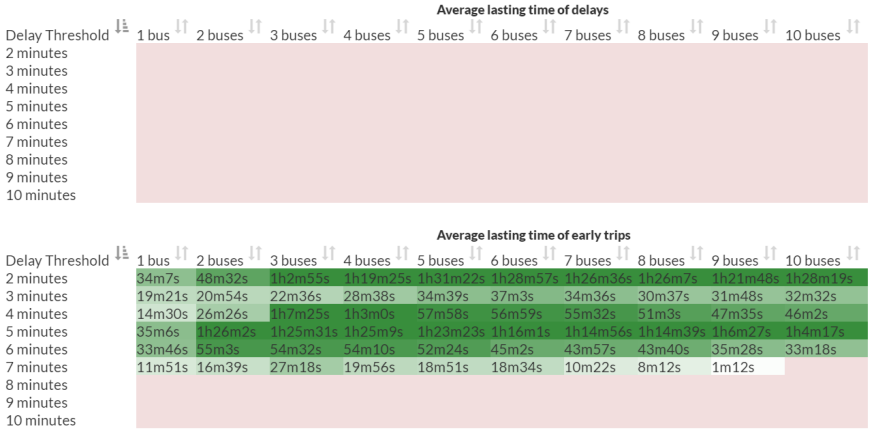
Average lasting time of delays

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes	18m37s	19m28s	17m6s							
3 minutes										
4 minutes										
5 minutes										
6 minutes										
7 minutes										
8 minutes										
9 minutes										
10 minutes										

Average lasting time of early trips

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes	12m52s									
3 minutes										
4 minutes										
5 minutes										
6 minutes										
7 minutes										
8 minutes										
9 minutes										
10 minutes										

Figure 5.11.: Edith Grove - Worlds End Health Centre



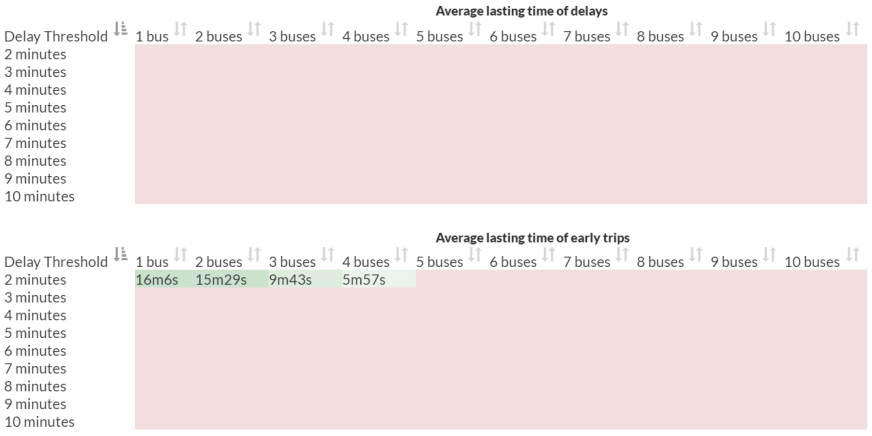
Average lasting time of delays

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes										
3 minutes										
4 minutes										
5 minutes										
6 minutes										
7 minutes										
8 minutes										
9 minutes										
10 minutes										

Average lasting time of early trips

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes	34m7s	48m32s	1h2m55s	1h19m25s	1h31m22s	1h28m57s	1h26m36s	1h26m7s	1h21m48s	1h28m19s
3 minutes	19m21s	20m54s	22m36s	28m38s	34m39s	37m3s	34m36s	30m37s	31m48s	32m32s
4 minutes	14m30s	26m26s	1h7m25s	1h3m0s	57m58s	56m59s	55m32s	51m3s	47m35s	46m2s
5 minutes	35m6s	1h26m2s	1h25m31s	1h25m9s	1h23m23s	1h16m1s	1h14m56s	1h14m39s	1h6m27s	1h4m17s
6 minutes	33m46s	55m3s	54m32s	54m10s	52m24s	45m2s	43m57s	43m40s	35m28s	33m18s
7 minutes	11m51s	16m39s	27m18s	19m56s	18m51s	18m34s	10m22s	8m12s	1m12s	
8 minutes										
9 minutes										
10 minutes										

Figure 5.12.: Blackbird Cross - Kings Drive



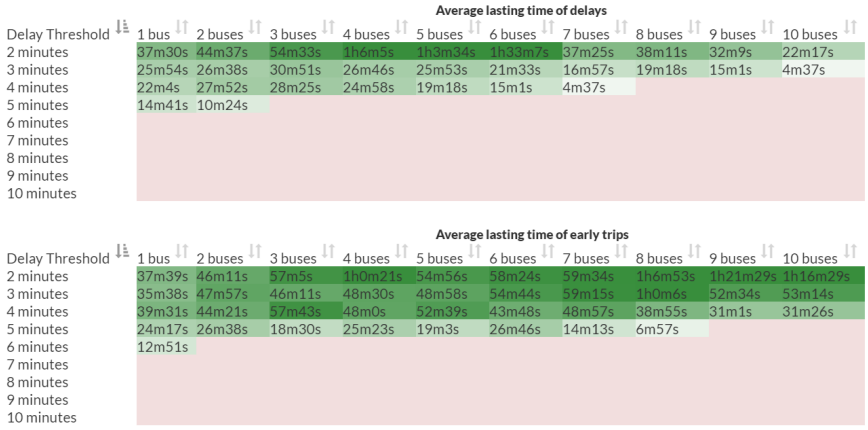
Average lasting time of delays

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes										
3 minutes										
4 minutes										
5 minutes										
6 minutes										
7 minutes										
8 minutes										
9 minutes										
10 minutes										

Average lasting time of early trips

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes	16m6s	15m29s	9m43s	5m57s						
3 minutes										
4 minutes										
5 minutes										
6 minutes										
7 minutes										
8 minutes										
9 minutes										
10 minutes										

Figure 5.13.: Park House - Hampton Court Gardens



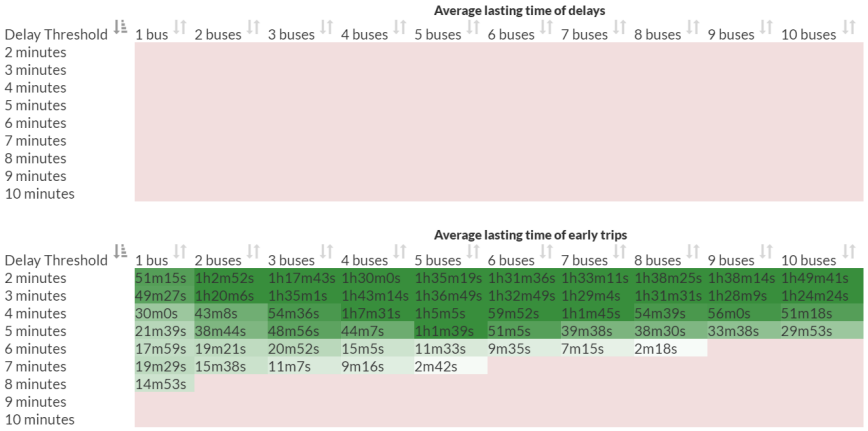
Average lasting time of delays

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes	37m30s	44m37s	54m33s	1h6m5s	1h3m34s	1h33m7s	37m25s	38m11s	32m9s	22m17s
3 minutes	25m54s	26m38s	30m51s	26m46s	25m53s	21m33s	16m57s	19m18s	15m1s	4m37s
4 minutes	22m4s	27m52s	28m25s	24m58s	19m18s	15m1s	4m37s			
5 minutes	14m41s	10m24s								
6 minutes										
7 minutes										
8 minutes										
9 minutes										
10 minutes										

Average lasting time of early trips

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes	37m39s	46m11s	57m5s	1h0m21s	54m56s	58m24s	59m34s	1h6m53s	1h21m29s	1h16m29s
3 minutes	35m38s	47m57s	46m11s	48m30s	48m58s	54m44s	59m15s	1h0m6s	52m34s	53m14s
4 minutes	39m31s	44m21s	57m43s	48m0s	52m39s	43m48s	48m57s	38m55s	31m1s	31m26s
5 minutes	24m17s	26m38s	18m30s	25m23s	19m3s	26m46s	14m13s	6m57s		
6 minutes	12m51s									
7 minutes										
8 minutes										
9 minutes										
10 minutes										

Figure 5.14.: Beechfield Road - Catford & Catford Bridge Stations



Average lasting time of delays

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes										
3 minutes										
4 minutes										
5 minutes										
6 minutes										
7 minutes										
8 minutes										
9 minutes										
10 minutes										

Average lasting time of early trips

Delay Threshold	1 bus	2 buses	3 buses	4 buses	5 buses	6 buses	7 buses	8 buses	9 buses	10 buses
2 minutes	51m15s	1h2m52s	1h17m43s	1h30m0s	1h35m19s	1h31m36s	1h33m11s	1h38m25s	1h38m14s	1h49m41s
3 minutes	49m27s	1h20m6s	1h35m1s	1h43m14s	1h36m49s	1h32m49s	1h29m4s	1h31m31s	1h28m9s	1h24m24s
4 minutes	30m0s	43m8s	54m36s	1h7m31s	1h5m5s	59m52s	1h1m45s	54m39s	56m0s	51m18s
5 minutes	21m39s	38m44s	48m56s	44m7s	1h1m39s	51m5s	39m38s	38m30s	33m38s	29m53s
6 minutes	17m59s	19m21s	20m52s	15m5s	11m33s	9m35s	7m15s	2m18s		
7 minutes	19m29s	15m38s	11m7s	9m16s	2m42s					
8 minutes	14m53s									
9 minutes										
10 minutes										

Figure 5.15.: Cephas Street - Darling Row

We see that the magnitudes of average delays differ quite substantially depending on the location. Some even hardly had major delays, such as *Edith Grove - Worlds End Health Centre*, *Park House - Hampton Court Gardens.* It is therefore not straightforward to apply a common absolute measure to all locations in detecting delays, such as fixing a common delay or bus count threshold.

A resolution is to consider delays individually for each pair of neighbouring stops. We may want to profile each of them separately to devise different heuristics. In fact we can build tables as above for all such pairs of stops, and cache them in our database for reverse lookup. By examining recent arrivals, we can look up the corresponding cells that apply for the estimated duration that the “delay” shall last.

One drawback is the computationally intensive nature, as there are more than 23,000 such pairs of stops[54]. It typically takes hours to complete an iteration. We will look into implementation for the general case when we study prediction methods (section 6.7), and optimising the speed in the "Optimisation" section (section 7.3 on page 100) .

[54] As of 21 May, 2016, there are 23,424 pairs of neighbouring bus stops in the TfL Central London bus network.

6. Evaluation

In section 5.1 we explored the periodicity of bus journey times between pairs of neighbouring stops. It has been found that most journeys under examination exhibit periodic variations every 24 hours / 7 days.

In practice few journeys would consist only of two stops, so we would also like to see how the above results extend to longer journeys. We therefore have selected a few bus routes for investigation across Central London to compare the effectiveness of our predictions versus the baseline ones given by the TfL Journey Planner (section 2.4.3).

In order to predict the travel time across the entire journey, we first decompose it into pairs of neighbouring stops. For each pair we apply a method (section 6.2) to generate a current estimate of travel times, and finally aggregate them to form the prediction for the entire journey.

6.1. Choice of routes under study

To be fair, we have sampled a variety of routes with contrasting characteristics. Some are the most crowded[27] and pass through the busiest districts in the city centre, while others serve the countryside. They also vary considerably in length, the longest taking 1.5 hours to complete a single trip[55] while the shortest within a mere 20 minutes[56]. Some of the selected routes also overlap with each other[57]. The bus routes that we will be investigating are:

- Route 5: Romford ↔ Canning Town
- Route 9: Hammersmith ↔ Aldwych
- Route 10: Hammersmith ↔ King's Cross
- Route 18: Euston ↔ Harrow
- Route 25: Oxford Circus ↔ Ilford
- Route 29: Trafalgar Square ↔ Wood Green

[55] Route 436 takes on average 70 - 105 minutes during day time to complete a single trip.
[56] Route 507 takes on average 15 - 20 minutes on a single trip.
[57] Routes 36 and 436 share the section between Paddington and New Cross.
Routes 9 and 10 share the section between Hammersmith and Hyde Park Corner.
Routes 279 and 349 share the section between South Tottenham and Ponders End.

- Route 36: Queen's Park ↔ New Cross
- Route 38: Victoria ↔ Clapton Road
- Route 73: Victoria ↔ Stoke Newington
- Route 86: Stratford ↔ Romford
- Route 149: London Bridge ↔ Edmonton Green
- Route 162: Eltham ↔ Beckenham Junction
- Route 208: Lewisham ↔ Orpington
- Route 243: Waterloo ↔ Wood Green
- Route 279: Manor House ↔ Waltham Cross
- Route 336: Locksbottom ↔Catford Bridge
- Route 349: Stamford Hill ↔ Glyn Road
- Route 436: Paddington ↔ Lewisham
- Route 484: Lewisham ↔ Camberwell Green
- Route 507: Victoria ↔ Waterloo

A map marked with the 20 bus routes above are shown in figure 6.1. The routes stretch out from the city centre in most directions, possibly a little short in the south west direction. Selecting these routes with overlapping sections allows us to investigate whether journey times between the same pairs of stops are route dependent.

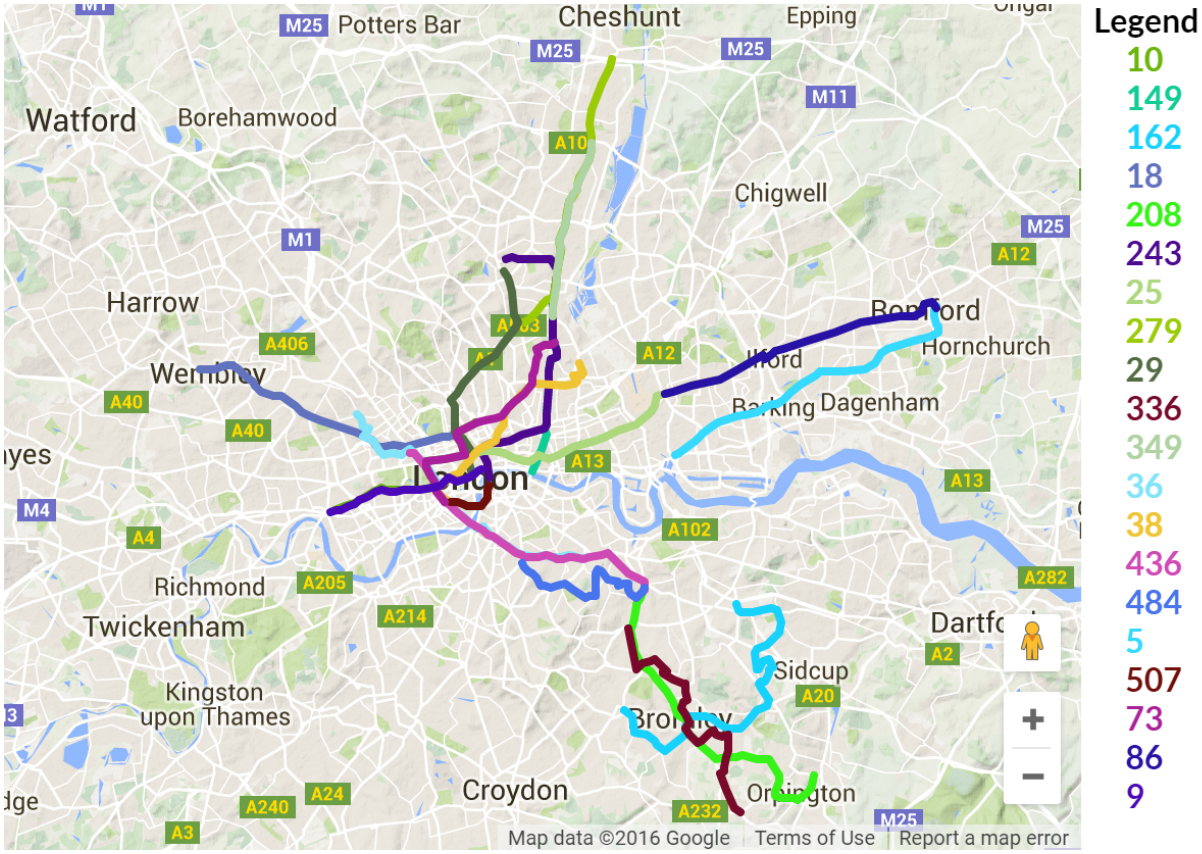


Figure 6.1.: A map of the bus routes that we are examining for evaluation.

We have taken all bus routes in their entirety, except the two end points (the first stop and the last). One reason being the problem of circular routes that we detected in section 4.2.4, where a bus route uses the same stop as the starting and ending terminus of the route. We would be unable to tell whether a prediction refers to whether the bus is going to depart the starting terminus, or arriving at the ending terminus. Another being that predictions are sometimes not stable at the endpoints, say a bus might terminate early or enter service a few stops down the route from the starting terminus, and erratic behaviour of predictions may be observed at the termini.

6.2. Prediction methods

It is time to put theory into practice. While we found in section 5.1 that good results should be obtained by reusing historic data in intervals of 24 hours / 7 days, we need to test our theory in the

real world. We propose a number of methods in generating predictions[58], and perform a shoot-out among them.

6.2.1. Historic based methods

1. Historic 1/2/3 days

   This method predicts the current journey times by taking the arithmetic mean of historic travel times 1, 2 and 3 days earlier at the same time within $\pm$15 minutes. For example, if the time now is 9:30AM, then bus travel records lying within 9:15-9:45AM of yesterday, the day before, and 2 days before will be taken into consideration. Since arithmetic means are susceptible to outliers, extreme data points, which we assume being the top 2 and bottom 2 percentile of the travel times in consideration, are taken away before the mean is computed[59].

   One may challenge the suitability of the $\pm$15 minute window being too narrow, and hence leading to a small sample size when the times are averaged. To this end, we will also consider variations of $\pm$30 minute window later.

2. Historic 7/14/21 days

   This method predicts the current journey times by taking the arithmetic mean of historic travel times 7, 14 and 21 days earlier at the same time within $\pm$15 minutes. The top 2 and bottom 2 percentile are cut off where possible in the same fashion before computing the mean.

   The main aim of this method is to rival against the *Historic 1/2/3 days* method. They both contain 3 days' worth of predictions, and should be of similar sample size when the travel times are averaged. The success of the *Historic 1/2/3 days* method provides evidence for choosing 24 hours as the period, whereas that of *Historic 7/14/21 days* would lead us to lean towards 7 days.

3. Historic 7/14/21 days (discarding interpolation)

   This method is based on *Historic 7/14/21 days*, except that we do not take into account of historic records that are a result of the interpolation techniques we applied.

[58] Essentially we will be considering a class of models in time series analysis - the autoregressive (AR) model. An AR(p) model represents the current value X_t based on the last p values $X_{t-1}, X_{t-2}, \ldots, X_{t-p}$:

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t$$

where c denotes the constant term and ε_t the error term. The historic based methods that we will be exploring are essentially variants on the combinations of the φ_i's.

[59] Truncation is performed only if we have more than two data points, or else no data will be left.

The purpose is to evaluate whether the interpolation methods that we have adopted are reliable. If so, this method should give non-inferior results to the *Historic 7/14/21 days* method.

4. Historic 7/14/21 days (same route only)

   This method is based on *Historic 7/14/21 days*, except that we only take into account of historic records that refer to the same bus route only, in case there are multiple routes serving the same pair of neighbouring bus stops.

   This is to test whether bus journey times along the same road exhibit route dependent characteristics, such as different passenger demands for each bus route leading to deviations in loading and unloading times.

5. Historic 5 days weekdays; 7/14/21 days weekends

   This method takes a hybrid approach; it adopts different strategies depending on whether the day is a weekday or a weekend. Should it be a weekday, the current journey times are predicted by taking the arithmetic mean of historic travel times of the last 5 weekdays. For example, if today is Thursday, then historic records of last Thursday, Friday and this Monday, Tuesday, Wednesday are considered. On weekends, it falls back on the *Historic 7/14/21 days* method. As with the previous methods, we discard the top and bottom 2 percentile of the travel times before computing the mean.

   This has the benefit of being able to reuse more recent travel records on weekdays (within 5 days instead of 21 as with the *Historic 7/14/21* days method), and therefore taking into consideration of recent changes on road conditions such as temporary roadworks which might last only for a few days. As for weekends, it is unlikely that Saturday and Sunday exhibit similar traffic flows[60], and hence it was decided not to group them together. We shall see whether treating weekdays as the same category will give improved or worse results compared to the *Historic 7/14/21 days* method.

6. Historic 5 days weekdays; 7/14/21 days weekends (discarding interpolation)

   Similar to *Historic 7/14/21 days (discarding interpolation)*, but based on *Historic 5 days weekdays; 7/14/21 days weekends*.

7. Historic 5 days weekdays; 7/14/21 days weekends (same route only)

   Similar to *Historic 7/14/21 days (same route only)*, but based on *Historic 5 days weekdays; 7/14/21 days weekends*.

Moving offset The methods that we have been exploring have so far been "static", in the sense that they all look up historic records with window centred about the time of the day when the

[60] Bus schedules are mostly different for Saturdays and Sundays.

trip begins, rather than actively tracking the times as the journey progresses and compensating for fluctuations in road conditions.

An example should clarify the problem statement. Suppose a commuter needs to travel on a long distance trip that takes an hour. He sets off at 7am, when traffic is relatively clear. The above methods would take into account historic records that are centred about the 7am mark. But by the time he reaches near his destination it would have been 8am, when traffic starts to build up and journey times could take considerably longer. Thus the above methods could potentially lead to an underestimate of travel times by design. We aim to address this issue by introducing "active tracking" of the user's journey - we incrementally predict journey times between neighbouring stops based on the times that the user is expected to reach the corresponding sections.

To illustrate the idea, consider a simple journey of 3 stops, from bus stop A, stopping at B, before reaching C. Suppose it is now 7am. For the first section of the journey (between stops A and B) we can obtain an estimate of travel times by considering historic journey times (between stops A and B) as above. Say an estimate of 3 minutes is returned. We then proceed to generate the prediction between stops B and C, taking into account of the fact that the user is expected to arrive at stop B only by 7:03am, and therefore any historic records used to generate the prediction between stops B and C will be centred around 7:03am, rather than 7am.

8. Historic 7/14/21 days (moving offset)

   This method is based on *Historic 7/14/21 days*, where the historic records used to generate predictions for each pair of neighbouring stops are centred differently, based on when the user is expected to be at that part of the journey, as described in the paragraph above.

   By comparing this method with the baseline (*Historic 7/14/21 days*), we can find out whether slight changes in the time of day has a significant effect on bus journey times.

9. Historic 7/14/21 days (moving offset discarding interpolation)

   This method combines *Historic 7/14/21 days (moving offset)* and *Historic 7/14/21 days (discarding interpolation)*.

10. Historic 7/14/21 days (moving offset same route only)

    This method combines *Historic 7/14/21 days (moving offset)* and *Historic 7/14/21 days (same route only)*.

11. Historic 5 days weekdays; 7/14/21 days weekends (moving offset)

    Similar to *Historic 7/14/21 days (moving offset)*, but based on *Historic 5 days weekdays; 7/14/21 days weekends*.

12. Historic 5 days weekdays; 7/14/21 days weekends (moving offset discarding interpolation)

    This method combines *Historic 5 days weekdays; 7/14/21 days weekends (moving offset)*

and *Historic 5 days weekdays; 7/14/21 days weekends (discarding interpolation)*.

13. Historic 5 days weekdays; 7/14/21 days weekends (moving offset same route only)

    This method combines *Historic 5 days weekdays; 7/14/21 days weekends (moving offset)* and *Historic 5 days weekdays; 7/14/21 days weekends (same route only)*.

Sliding window So far we have been taking predictions falling within a sliding window of 30 minutes ($\pm$15 minutes) around the times of travel. While choosing a narrow window has the advantage that the results are less likely to be blurred by neighbouring regions of times, having a small sample falling within the window means the any adverse effects exerted by outliers will be more prominent. We seek to find out the effects in reality by also considering a sliding window of 1 hour ($\pm$30 minutes) to see whether there are statistically significant differences.

14. Historic 7/14/21 days (moving offset with 1 hour window)

    This method is the same as *Historic 7/14/21 days (moving offset)*, except we consider all historic records lying within $\pm$30 minutes, rather than $\pm$15 minutes, of the desired travel times.

15. Historic 7/14/21 days (moving offset discarding interpolation with 1 hour window)

    Same as *Historic 7/14/21 days (moving offset discarding interpolation)*, except we consider all historic records lying within $\pm$30 minutes of the desired travel times.

16. Historic 5 days weekdays; 7/14/21 days weekends (moving offset with 1 hour window)

    Same as *Historic 5 days weekdays; 7/14/21 days weekends (moving offset)*, except we consider all historic records lying within $\pm$30 minutes of the desired travel times.

17. Historic 5 days weekdays; 7/14/21 days weekends (moving offset discarding interpolation with 1 hour window)

    Same as *Historic 5 days weekdays; 7/14/21 days weekends (moving offset discarding interpolation)*, except we consider all historic records lying within $\pm$30 minutes of the desired travel times.

6.2.2. Real-time based methods

Apart from estimating current travel times with historic data, we would also like to attempt the same with real time information, and compare their performances.

One major limitation of using real time data only is the sample size available. Unlike the case with historic data, where we could easily increase the sample size just by including data from more days in the past, real time data is restricted to the recent and immediately available data. It could

be the case that within the last 30 minutes, only 1 or 2 records exist, and we must be cautious of whether they are truly representative of the actual traffic conditions.

18. Real time median

    This method predicts the current journey times by taking the median of the travel times within the last 30 minutes.

    One may notice that median, rather than the arithmetic mean, is adopted here. This is because we expect a small sample of real time data available, and taking the median is a simple and straightforward way of suppressing the effect of outliers.

    Note that unlike historic based methods, we do not remove the top and bottom 2 percentile of travel times.

19. Real time last two

    This method predicts the current journey times by taking the arithmetic mean of the last two records, or not at all if fewer than two records exist the past 30 minutes.

    The purpose is to test whether data closer to the time of travel is more indicative of the real time situation. For instance, we may be able to differentiate the peak from the demise of a delay, as we should obtain high values at peaks and low values when a delay is near its end, whereas taking the median as in *Real time median* could give the same number regardless of whether a delay has just started (on an increasing trend) or is about to end (on a decreasing trend).

Similar to the historic methods, we consider variations of 1 hour windows, rather than 30 minutes, in order to investigate whether a larger sample size would outweigh the effects of potentially outdated data.

20. Real time mean (1 hour window)

    This method predicts the current journey times by taking the arithmetic mean of travel times falling within the past hour. In order to reduce the effect of outliers, the top and bottom two percentiles of data are truncated before computing the arithmetic mean.

    Thanks to the one hour window, a reasonable sample size remains after truncation. A sample of ≈ 10 is observed for stops in the city centres and ≈ 4 in the country side after truncation, so we should be able to obtain representative results.

21. Real time median (1 hour window)

    Same as *Real time median*, except that we consider all records lying in the past hour rather than 30 minutes.

22. Real time last three (1 hour window)

This method is an extension of the *Real time last two* method, where we consider the arithmetic mean of the last three records rather than two, over an observation period of one hour rather than 30 minutes.

The point of adopting a one hour window is simply to allow for more data to be captured, and therefore increasing our likelihoods of obtaining three or more records in order to generate our prediction. If we do not observe three records within the past hour, we would rather not generate a prediction than to rely on data further in the past, which may not be reflective of the current traffic conditions.

6.3. Qualitative Evaluation: Visualisation

In order to qualitatively visualise how our predictions perform over a period of time, we have also plotted the sequence of journey times falling within a time frame as time series. For each route under examination we produce line charts of journey times against the times at which the journeys took place. To simplify our discussion we refer to only bus route 9 for now. The features that we discuss below hold in general for the other routes under examination.

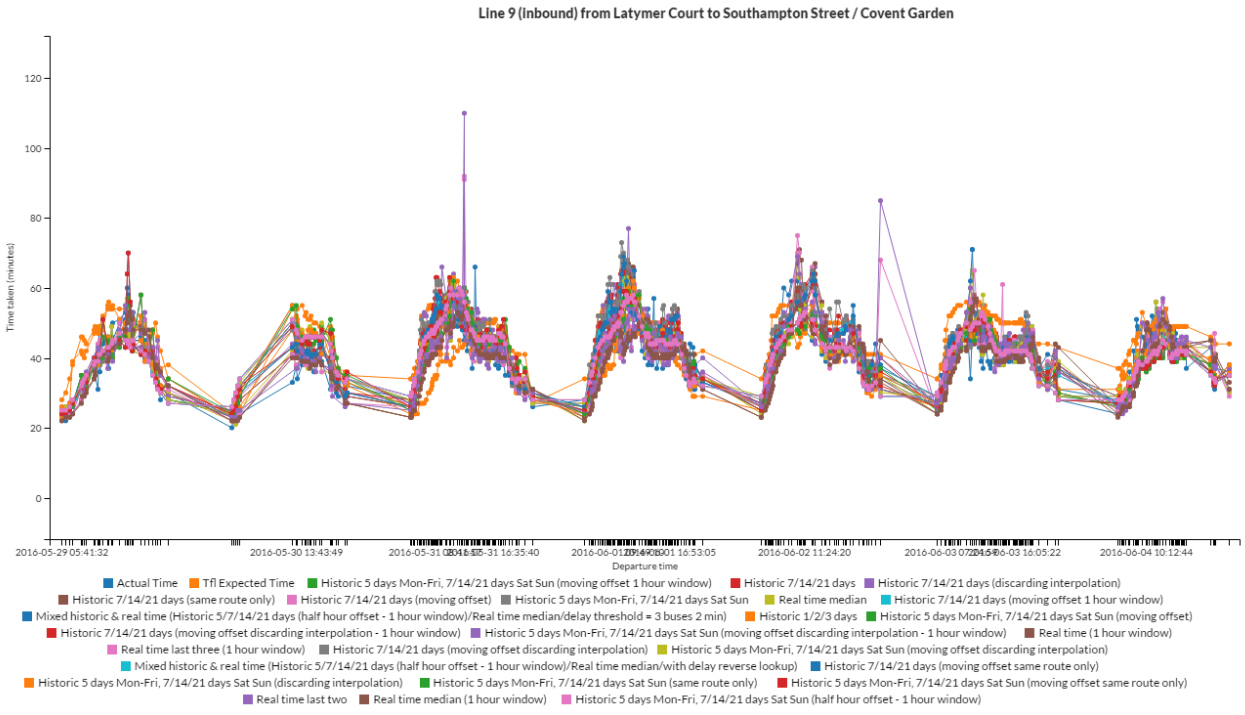


Figure 6.2.: Time series chart of journey times in minutes (y-axis) against the departure times of the corresponding journeys (x-axis), of bus route 9 in the inbound direction over a one week period between 29th May, 2016 and 4th June, 2016.

Figure 6.2 shows the journey times of bus route 9 over a one week interval. We see that travel times are in general much higher in the midst of the day than in the early mornings or late nights. This appears to justify our need for moving offset predictions. There are also periods of inactivity as this is not a 24-hour route.

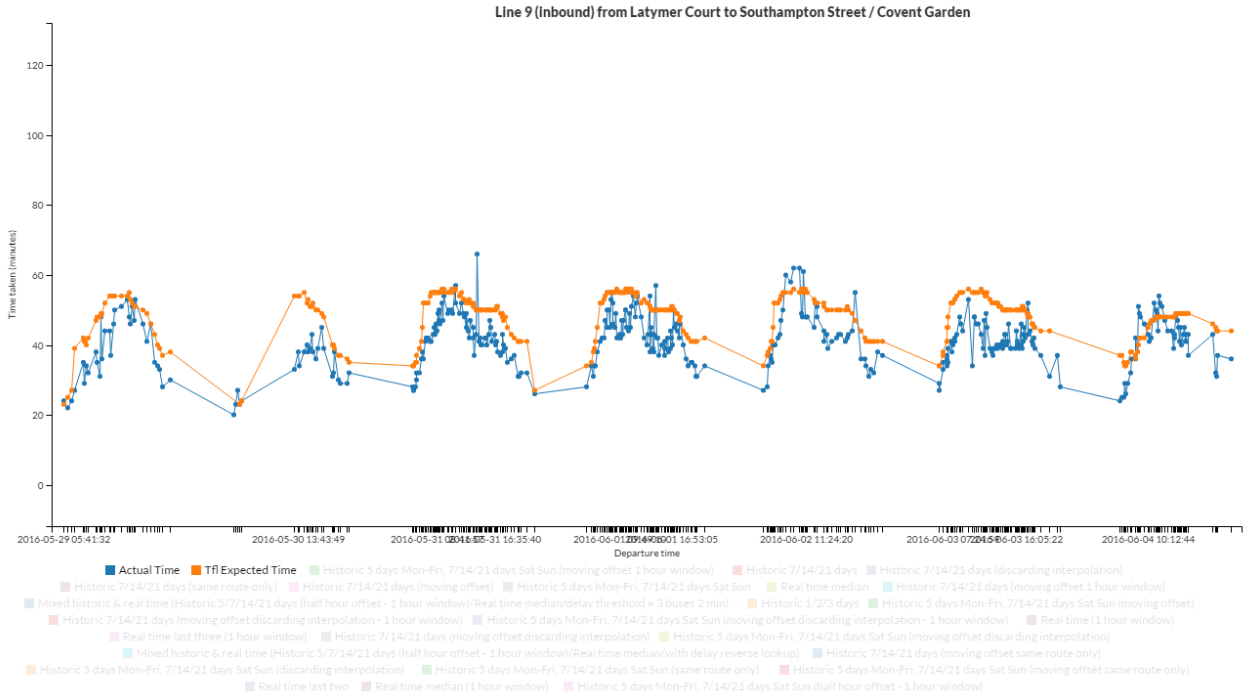


Figure 6.3.: Time series chart of journey times in minutes (y-axis) against the departure times of the corresponding journeys (x-axis), of bus route 9 in the inbound direction between 29th May, 2016 and 4th June, 2016, showing only the actual journey times and official predictions given by the TfL journey planner.

To dig deeper we break down the chart into individual components. Figure 6.3 shows only the actual journey times recorded, and the official expected journey times returned from the TfL Journey Planner[61]. The blue line corresponds to the actual journey times, whereas the orange line represents the official predictions. We see that the actual journey times fluctuate erratically over time, possibly owing to traffic lights or customer incidents. The TfL predictions on the other hand exhibit much less variability, and are more or less piecewise constant during the day except in the early mornings and late nights. In general both tend to rise and fall at the same time, suggesting that TfL may also have performed similar historic analysis in generating their predictions.

[61] More precisely, we have been polling the TfL server for expected journey times at 10 minute intervals, and logging them into our database. We can therefore select the record closest to the departure time of the journey as the official TfL prediction (i.e. an error of at most ± 5 minutes from the actual time of departure), assuming that TfL predictions remain more or less the same within a 10 minute interval.

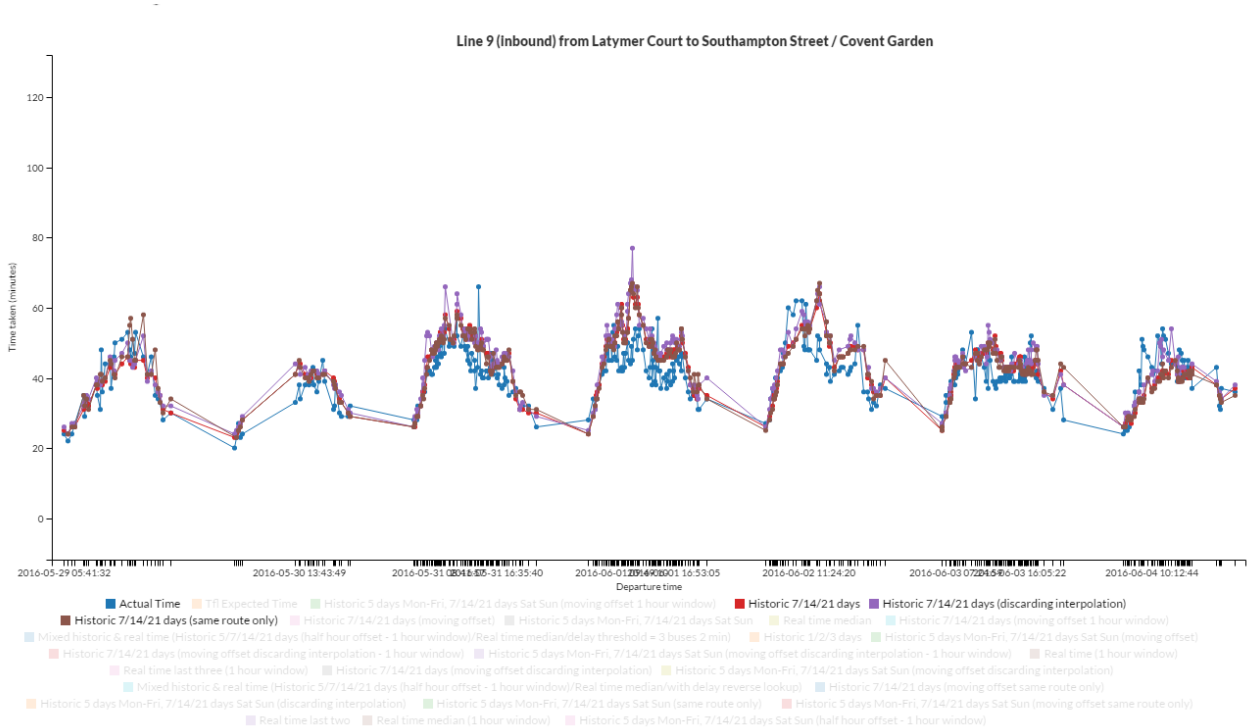


Figure 6.4.: Time series chart of journey times in minutes (y-axis) against the departure times of the corresponding journeys (x-axis), of bus route 9 in the inbound direction between 29th May, 2016 and 4th June, 2016, showing only the actual journey times, and prediction methods of the *historic 7/14/21 days* family.

We now investigate prediction methods from the same family. Figure 6.4 shows the actual journey times, together with variations of the *Historic 7/14/21 days* method, namely

- Historic 7/14/21 days
- Historic 7/14/21 days (discarding interpolation)
- Historic 7/14/21 days (same route only)

Note that we have not included the moving offsets counterparts, for simplicity. We can observe that the line charts representing the three methods (non-blue) are very close together, nearly overlapping with each other, compared to the one representing actual journey times (in blue). We therefore proceed below to show only a representative from each family, in order to clearly visualise the differences between families in the graphs.

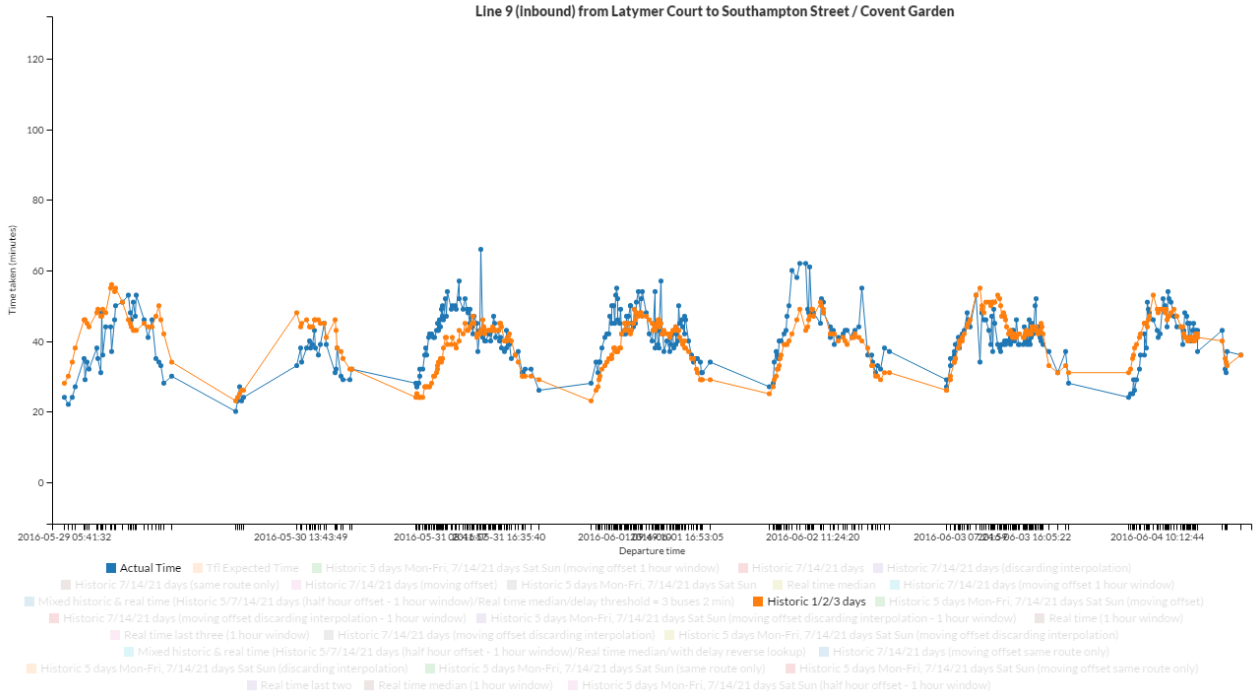


Figure 6.5.: Time series chart of journey times in minutes (y-axis) against the departure times of the corresponding journeys (x-axis), of bus route 9 in the inbound direction between 29th May, 2016 and 4th June, 2016, showing only the actual journey times and the *historic 1/2/3 days* prediction method.

We start off with the *Historic 1/2/3 days* family. Figure 6.5 shows the charts of the actual journey times and the *Historic 1/2/3 days* method. The *Historic 1/2/3 days* method is really a hit or a miss, doing extremely well occasionally (orange line being close to the blue) but could also be very off (an error of 20 minutes accounting for $\frac{1}{3}$ of the journey time) for a significant portion of time. It also is not consistent as to which times of the day does this prediction method perform well or worse. This method seems to give a better fit for the last 4 days than the first 3 days, the last 4 days being Wednesday, Thursday, Friday and Saturday. This is not surprising, as traffic conditions are likely to be different on weekdays and weekend, so prediction for a Monday using historic travel times on Sunday, Saturday and Friday is unlikely a good idea.

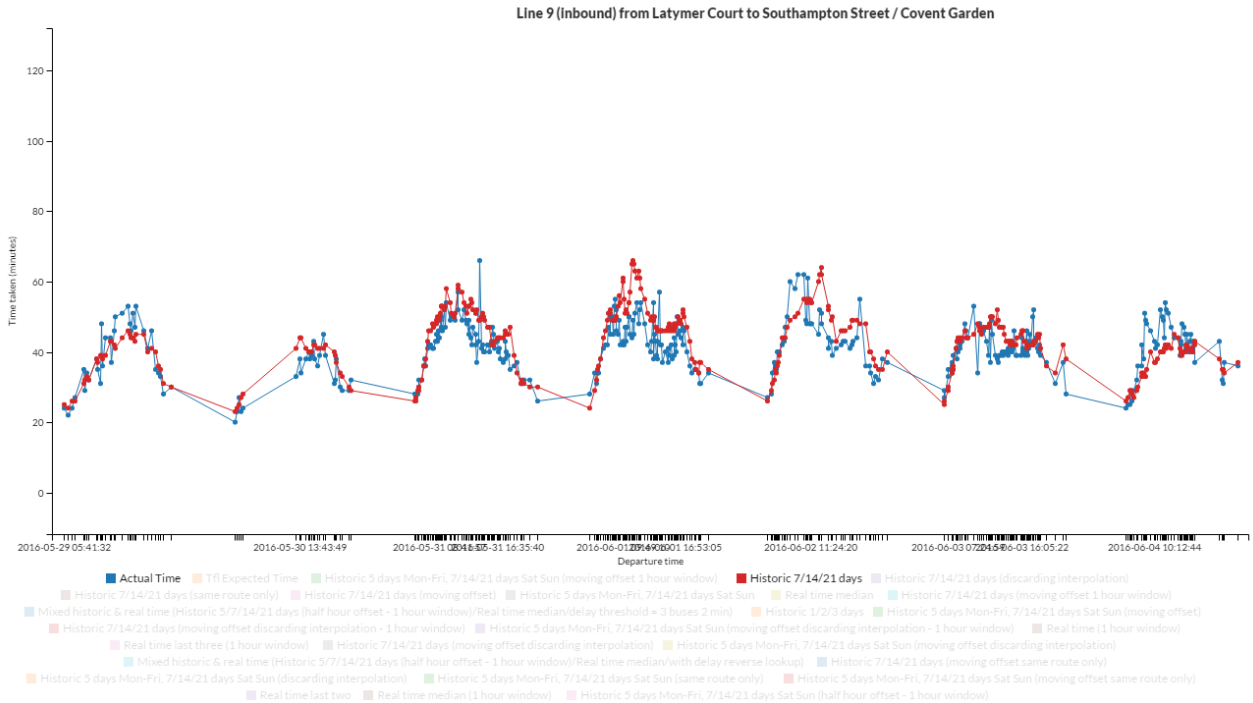


Figure 6.6.: Time series chart of journey times in minutes (y-axis) against the departure times of the corresponding journeys (x-axis), of bus route 9 in the inbound direction between 29th May, 2016 and 4th June, 2016, showing only the actual journey times and the *historic 7/14/21 days* prediction method.

We now look at the *Historic 7/14/21 days family* in a similar fashion. The results from the *Historic 7/14/21 days* method (figure 6.6) are clearly better than the *Historic 1/2/3 days* method, with the shape mostly coinciding with the actual journey times, though at times the peaks do not match in height. We can however see that the predictions are at least consistently well at the start of each day, between 5 and 7 AM. This shows that sampling from historic data in intervals of 7 days does extract some useful information, as long as trends do not change by much, such as the early mornings in this case.

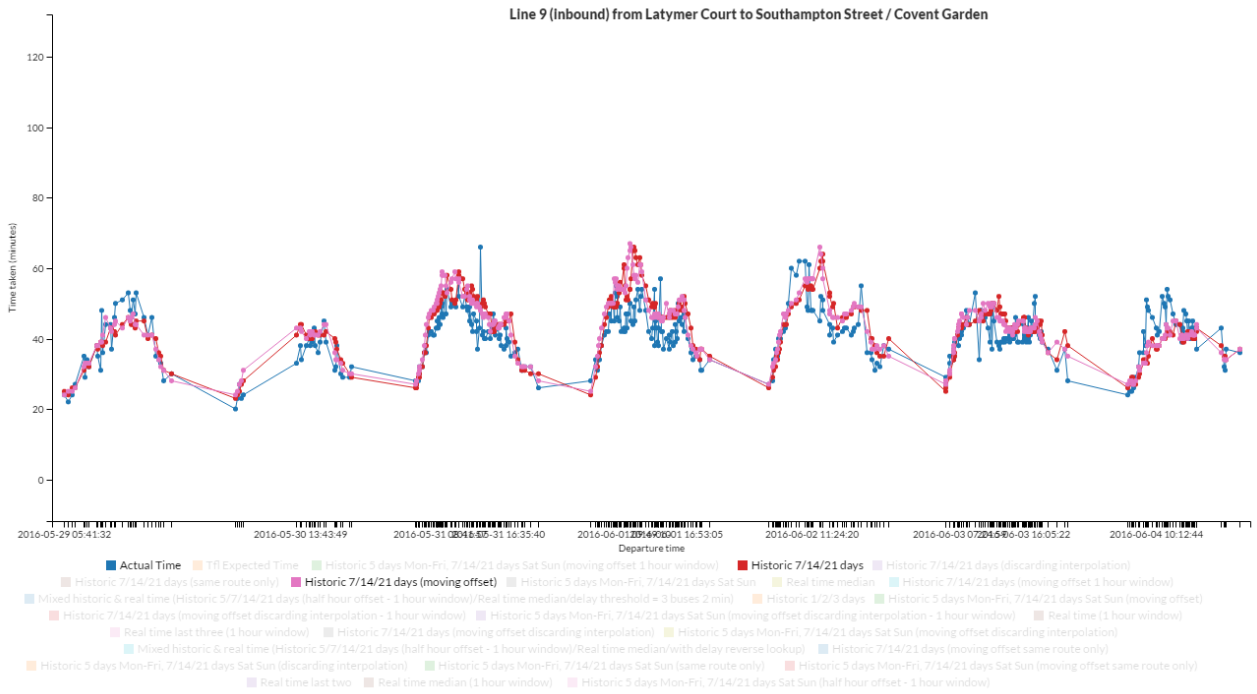


Figure 6.7.: Time series chart of journey times in minutes (y-axis) against the departure times of the corresponding journeys (x-axis), of bus route 9 in the inbound direction between 29th May, 2016 and 4th June, 2016, showing only the actual journey times, *historic 7/14/21 days* and *historic 7/14/21 days with moving offset* prediction method.

Figure 6.7 overlays also the *Historic 7/14/21 days (moving offset)* method, indicated by the pink line. It pretty much has the same shape as the base *Historic 7/14/21 days* method, but reacting to changes earlier, and results in translation-like effect to the left. With the moving offset technique, we seem to get better predictions at the end of the day, in addition to the early mornings.

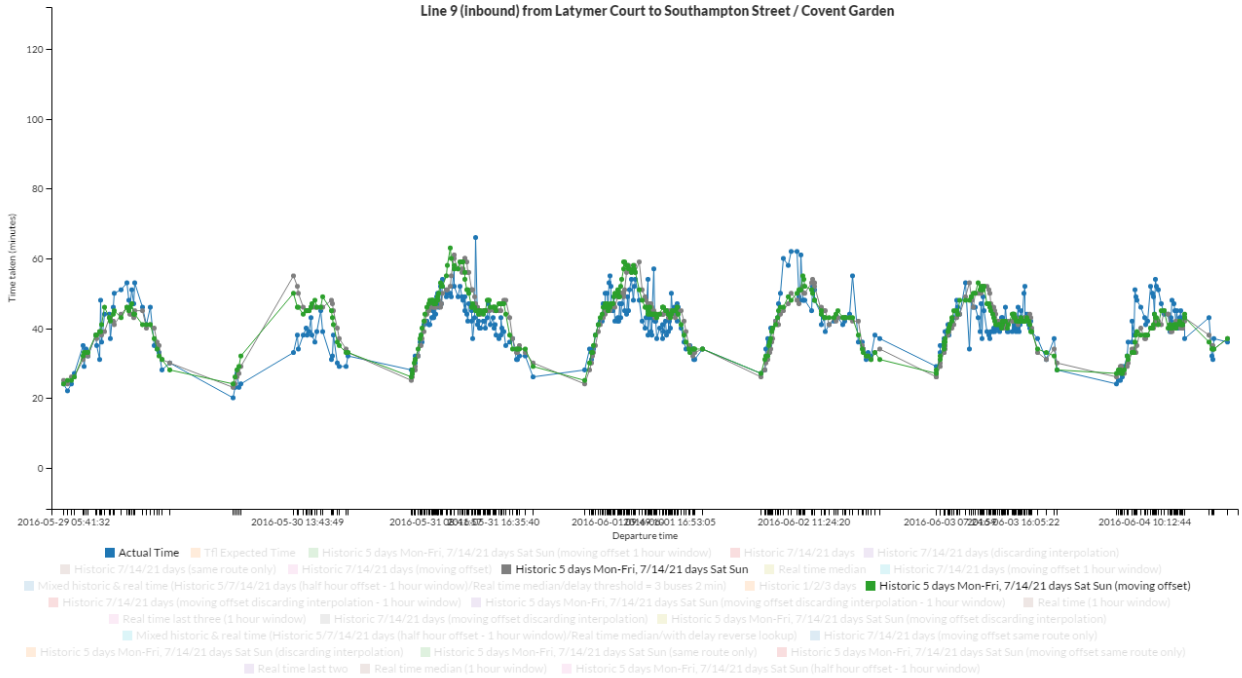


Figure 6.8.: Time series chart of journey times in minutes (y-axis) against the departure times of the corresponding journeys (x-axis), of bus route 9 in the inbound direction between 29th May, 2016 and 4th June, 2016, showing only the actual journey times, *historic 5 days weekdays; 7/14/21 days weekends* and *historic 5 days weekdays; 7/14/21 days* weekends with moving offset prediction method.

Moving on to the *Historic 5 days weekdays; 7/14/21 days weekends* family, we immediately notice a huge improvement (figure 6.8). The grey line represents the base *Historic 5 days weekdays; 7/14/21 days weekends* method, whereas the green line represents the moving offset counterpart. Both of them are fairly close to the actual journey times (blue line) most of the time, but the moving offset version performs better particularly in early mornings and evenings where there are continuous sharp increases and decreases in journey times. This suggests that the moving offset technique does allow for unidirectional changes in road traffic better.

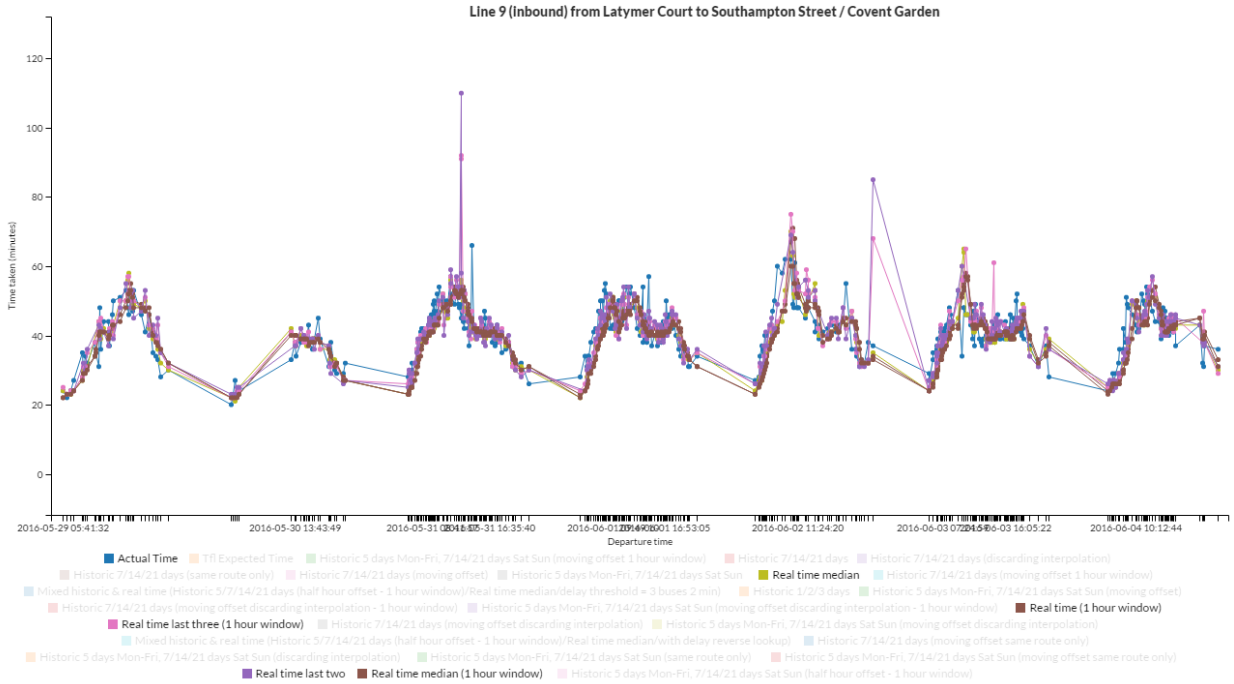


Figure 6.9.: Time series chart of journey times in minutes (y-axis) against the departure times of the corresponding journeys (x-axis), of bus route 9 in the inbound direction between 29th May, 2016 and 4th June, 2016, showing only the actual journey times, and all real time based prediction methods.

Finally we move onto real time based methods. Figure 6.9 shows the actual journey times as well as all real time prediction methods for comparison. A first observation is that all of them are very close to each other, with shape almost the same as the actual journey times, but being shifted to the right. This is within expectation, as predictions are generated based on actual travel times in the immediate past, and therefore should be more or less identical to the recent travel times within a 30 minute or one hour interval. We also notice that the time series are less smooth and have greater fluctuations than those of the historic based methods, potentially owing to a lower available sample size when evaluating predictions and therefore larger variances.

The above results extend in general to the other routes under examination. The corresponding charts for the remaining routes can be found in the source code repository (*see* Appendix A).

6.4. Quantitative Evaluation: Performance measures

With the above historic and real time prediction methods introduced, we can perform quantitative evaluations on selected bus routes in section 6.1. This analysis could be extended to all routes in

Central London in theory, but owing to complexity constraints[62], we have decided to explore on a small sample of routes that should be representative.

The idea is that good predictions should centre around the actual journey times, and therefore any deviations (or error terms) when aggregated across stops along a route should on average cancel out. As an example, consider a short route consisting of three stops W, X, Y, Z :

Journey segment	$W \to X$	$X \to Y$	$Y \to Z$	Entire route
Prediction (in seconds)	43	30	57	130
Actual travel time (in seconds)	42	32	55	129
Deviation / error	-1	+2	-2	-1

Table 6.1.: Example of a good prediction method. Predictions centre around actual journey times and therefore errors on average cancel out along the route.

Journey segment	$W \to X$	$X \to Y$	$Y \to Z$	Entire route
Prediction (in seconds)	43	30	57	130
Actual travel time (in seconds)	40	28	55	123
Deviation / error	-3	-2	-2	-7

Table 6.2.: Example of a poor prediction method. Predictions are skewed to one side of the actual times, and therefore errors accumulate, leading to a wide margin when summed over the entire route.

Tables 6.1 and 6.2 shows the effects of good and poor prediction methods. We can see that when the errors terms are summed up along the route, the combined error given by the good prediction method remains small, whereas that of the poor becomes worse.

We therefore can use the error term aggregated across entire bus routes as an indicator of how a prediction method performs. To this end, we pick a time frame in the past and calculate deviations of our predictions of the total journey time on the selected routes from the actual journey times. We consider three performance measures:

- Root mean square error (RMSE) - This measure is defined by

$$\sqrt{\frac{\sum_{t=1}^{n} (\hat{x}_t - x_t)^2}{n}}$$

  where x_t refers to the actual journey time taken on a bus trip, $\hat{x}_t$ refers to the corresponding prediction generated by one of our methods, and n the total number of such journeys that we are including. For example, we might be able to observe six journeys on route 9 over a two hour period and obtain the following dataset:

[62] Since we are exploring 20+ prediction methods, each covering 20 bus routes and a time frame of a few days, it typically takes a number of hours to complete an iteration on the VMs that we are using.

t	1	2	3	4	5	6
x_t (Actual journey time in minutes)	40	60	42	45	43	46
$\hat{x}_t$ (Predicted journey time in minutes)	42	43	43	44	45	45

in such case the root mean square error will be given by

$$\sqrt{\frac{(40-42)^2+(60-43)^2+(42-43)^2+(45-44)^2+(43-45)^2+(46-45)^2}{6}} = 7.07 \text{ minutes}$$

meaning our predictions are on average 7.07 minutes off from the actual journey times.

We note that this measure considers the sum of squares of each error term, before taking the square root, and therefore the effect of extreme errors (when predictions give poor results) will be given more weight than smaller errors (when predictions give relatively accurate results), and therefore may give a large deviation even if only one or two predictions are poor. We mitigate this by introducing alternative performance measures.

- Mean absolute error (Mean AE) - This measure is defined to be

$$\frac{\sum_{t=1}^{n} |\hat{x}_t - x_t|}{n}$$

where we use the same notation as before. Taking the same data set as above we obtain a mean absolute error of

$$\frac{|40-42|+|60-43|+|42-43|+|45-44|+|43-45|+|46-45|}{6} = 4 \text{ minutes}$$

We see that this is significantly smaller than that given by the RMSE measure. This is because we have mostly accurate predictions (within ± 2 minutes), with the exception of the 2nd journey in which the actual journey time was underestimated by 17 minutes. Thus a smaller error given by the Mean AE than the RMSE may mean that on the whole our predictions are mostly reliable, with the occasional exceptions.

- Median absolute error (Median AE) - This measure is defined to be

$$\text{median} \, |\hat{x}_t - x_t|$$

over the n samples. With the same data set as above we obtain a median absolute error of

$$\text{median} \ (|40-42|, |60-43|, |42-43|, |45-44|, |43-45|, |46-45|) = 1.5 \text{ minutes}$$

We see that this gives an even smaller number than the Mean AE. We have completely excluded the effect of the 2nd prediction, as the median turns a blind eye on extreme data points. Despite suggesting very good results, we may need to be cautious in giving our complete trust as the existence of the poor predictions may mean something can be done to improve our model, or reveal bugs.

Prediction method	Root mean squared error	Mean absolute error	Median absolute error	# Data points
Actual Time	NA	NA	NA	385
Historic 1/2/3 days	6m52s	5m13s	3m57s	385
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun	5m55s	4m25s	3m21s	385
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (discarding interpolation)	5m53s	4m20s	3m3s	384
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (half hour offset - 1 hour window)	5m25s	4m8s	3m12s	385
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset 1 hour window)	5m21s	4m3s	2m57s	385
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset discarding interpolation - 1 hour window)	5m10s	3m50s	2m42s	385
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset discarding interpolation)	5m33s	4m6s	3m1s	383
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset same route only)	5m37s	4m10s	2m58s	385
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset)	5m25s	4m5s	3m0s	385
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (same route only)	6m3s	4m33s	3m24s	383
Historic 7/14/21 days	6m12s	4m52s	3m59s	385
Historic 7/14/21 days (discarding interpolation)	6m52s	5m21s	4m13s	372
Historic 7/14/21 days (moving offset 1 hour window)	5m57s	4m43s	3m47s	385
Historic 7/14/21 days (moving offset discarding interpolation - 1 hour window)	6m28s	5m5s	4m4s	385
Historic 7/14/21 days (moving offset discarding interpolation)	7m12s	5m36s	4m33s	377
Historic 7/14/21 days (moving offset same route only)	6m7s	4m46s	4m0s	385
Historic 7/14/21 days (moving offset)	5m59s	4m44s	3m54s	385
Historic 7/14/21 days (same route only)	6m25s	4m58s	3m59s	383
Mixed historic & real time (Historic 5/7/14/21 days (half hour offset - 1 hour window)/Real time median/delay threshold = 3 buses 2 min)	5m36s	4m10s	3m9s	381
Mixed historic & real time (Historic 5/7/14/21 days (half hour offset - 1 hour window)/Real time median/with delay reverse lookup)	5m26s	4m10s	3m12s	381
Real time (1 hour window)	5m41s	4m25s	3m41s	385
Real time last three (1 hour window)	6m24s	4m14s	3m5s	382
Real time last two	6m42s	4m10s	2m51s	354
Real time median	5m26s	4m3s	3m5s	381
Real time median (1 hour window)	5m42s	4m31s	3m55s	385
Tfl Expected Time	8m10s	7m15s	6m59s	380

Figure 6.10.: Table of performance measures evaluated on the different prediction methods introduced in section 6.2, for route 9 in the inbound direction between 29th May, 2016 and 4th June, 2016.

Figure 6.10 shows the performance measures applied on bus route 9 over the same two day period as we did in *Qualitative Evaluation: Visualisation* (section 6.3). Note that the actual journey times serve as the baseline, and therefore all errors are computed with respect to the actual journey times.

The rightmost column "# Data points" indicates the number of data points (n in the formulae introduced above) that we took into consideration in error computation. We see that out of the 385 actual journeys recorded (as depicted by the row corresponding to *Actual Time*), we have been able to generate the corresponding predictions, with the exception of those prediction methods with less than 385 data points[63]. These methods are usually more susceptible to loss of data points, as they discard individual predictions for each pair of adjacent stops along the route whenever they are a result of interpolation, not belonging to the same bus route, or fail in real time when there are lack of buses in the past 30 minutes or so. Individual predictions failing for any pairs of adjacent stops would invalidate the prediction for the entire journey, so what we have is really a lower bound of the available data points (or upper bound in the loss of data points), for passengers that do not travel on the entire bus route.

Cells in green indicate that our prediction method, as suggested by the row, outperforms the official TfL prediction under the performance measure suggested by the column. On this occasion we see that all prediction methods do better than the TfL prediction. We note that different prediction methods outperform the TfL prediction to different extents. Most notably is the *Historic 5 days weekdays; 7/14/21 days weekends* family, with a mere average median absolute error of approximately 3 minutes, compared to 4 minutes given by the *Historic 7/14/21 days* family, 3 minutes

[63] These are *Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (discarding interpolation)*, *Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset discarding interpolation)*, *Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (same route only)*, *Historic 7/14/21 days (same route only)*, *Historic 7/14/21 days (moving offset discarding interpolation)*, *Historic 7/14/21 days (discarding interpolation)*, *Real time last three (1 hour window)*, *Real time median*, and *Real time last two* on this occasion.

30 seconds given by the real time family, and 7 minutes by the official TfL prediction. This shows that historic analysis is indeed useful.

In order to piece together a complete picture, we need to aggregate performance measures over the other routes under examination to find out which prediction method(s) consistently do well across a diversity of routes. A difficulty is that the errors are not directly comparable across routes, since certain routes take much longer to complete and therefore larger errors are within expectation. We therefore attempt two approaches - the first being a crude measure, hoping to eliminate route dependent effects on the performance measures; the second being evaluation of percentage errors, so that the error terms can be scaled according to the length of the route, in order to give a fair comparison between routes.

Aggregation of performance measures

1. Crude scoring method

   We devise a simple scoring mechanism below. We award one point to prediction methods that outperform the TfL predictions on a particular route under a performance measure, zero if on average equal, and minus one if worse.

$$\text{score} = \begin{cases} +1 & \text{if prediction method is better than TfL} \\ 0 & \text{if prediction method is equal to TfL} \\ -1 & \text{if prediction method is worse than TfL} \end{cases}$$

   In figure 6.10 all our prediction methods would get one point under all performance measures. Aggregating the scores over all 20 routes, and we count each direction (either outbound or inbound) as a separate entity, we obtain results in figure 6.11, where we have ordered prediction methods by descending score with respect to the median absolute error measure. We note that 40 is the maximal achievable score[64] and -40 the minimal.

[64] 20 routes $\times$ 2 directions per route $=$ 40 combinations.

	Total count of routes with superior predictions to TfL		
Method	Root mean squared error	Mean absolute error	Median absolute error
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (half hour offset - 1 hour window)	25	33	35
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset)	24	32	34
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset 1 hour window)	25	31	33
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset same route only)	17	30	32
Mixed historic & real time (Historic 5/7/14/21 days (half hour offset - 1 hour window)/Real time median/delay threshold = 3 buses 2 min)	15	25	31
Mixed historic & real time (Historic 5/7/14/21 days (half hour offset - 1 hour window)/Real time median/with delay reverse lookup)	13	25	31
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset discarding interpolation - 1 hour window)	10	20	30
Historic 7/14/21 days (moving offset)	20	24	28
Real time last three (1 hour window)	21	27	27
Real time median	21	25	27
Real time last two	19	25	27
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun	16	26	26
Historic 7/14/21 days (moving offset 1 hour window)	22	24	26
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (same route only)	7	16	26
Real time (1 hour window)	23	25	25
Real time median (1 hour window)	15	25	25
Historic 7/14/21 days	19	23	25
Historic 7/14/21 days (moving offset same route only)	17	17	25
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset discarding interpolation)	0	10	24
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (discarding interpolation)	-2	4	24
Historic 7/14/21 days (same route only)	7	11	19
Historic 7/14/21 days (moving offset discarding interpolation - 1 hour window)	0	10	18
Historic 1/2/3 days	3	11	17
Historic 7/14/21 days (moving offset discarding interpolation)	-7	4	16
Historic 7/14/21 days (discarding interpolation)	-12	0	16

Figure 6.11.: Table of scores generated by the *crude scoring* method aggregating over all routes under examination evaluated on the different prediction methods introduced in section 6.2, between 29th May, 2016 and 4th June, 2016, ordered in descending order by the median absolute error measure.

In fact, the rankings are fairly consistent across all three performance measures, with the *Historic 5 days weekdays; 7/14/21 days weekends* family clearly topping the other families.

2. Percentage errors

One may challenge that the crude scoring method does not really reflect to what extent a prediction method outperforms TfL. This can be seen when a prediction scores one point, regardless of whether it significantly or marginally outperforms the TfL prediction method, for a given bus route. This may lead to prediction methods that are just slightly better than TfL but consistent across all routes to come on top of the scoring table, whereas those that performs really well in some of the cases may be drowned and neglected.

To this end, we consider the percentage improvements over the offical TfL prediction in terms of the errors (i.e. the performance measures defined above - RMSE, mean absolute error and median absolute error) for each prediction method. More formally, for each prediction method on a particular route, we evaluate

$$\text{score} \quad = \quad \frac{\text{error given by TfL prediction} \ - \text{error given by prediction method}}{\text{error given TfL prediction}} \times 100\%$$

as a measure of how "superior" our prediction method is compared to the errors given by the TfL baseline (positive if better and negative if worse than TfL). Taking the arithmetic mean across all routes under study, we obtain a score for each prediction method.

	Relative superiority to TfL (positive is better)		
Method	Root mean squared error	Mean absolute error	Median absolute error
Mixed historic & real time (Historic 5/7/14/21 days (half hour offset - 1 hour window)/Real time median/delay threshold = 3 buses 2 min)	19.07%	31.49%	47.34%
Mixed historic & real time (Historic 5/7/14/21 days (half hour offset - 1 hour window)/Real time median/with delay reverse lookup)	16.37%	29.36%	46.73%
Real time last three (1 hour window)	24.21%	35.87%	46.60%
Real time median	26.02%	35.72%	45.36%
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset)	23.63%	33.81%	45.31%
Real time last two	25.35%	35.46%	45.24%
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset same route only)	18.75%	30.43%	45.19%
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (half hour offset - 1 hour window)	23.68%	32.83%	43.84%
Real time (1 hour window)	24.84%	33.36%	42.72%
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset 1 hour window)	22.97%	31.94%	42.50%
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset discarding interpolation - 1 hour window)	11.69%	24.50%	41.17%
Real time median (1 hour window)	21.61%	29.09%	38.26%
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset discarding interpolation)	8.01%	19.11%	37.36%
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun	18.59%	25.87%	35.38%
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (same route only)	12.43%	20.93%	33.10%
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (discarding interpolation)	4.45%	14.60%	31.41%
Historic 7/14/21 days (moving offset discarding interpolation - 1 hour window)	1.23%	11.80%	28.49%
Historic 7/14/21 days	5.37%	13.14%	24.94%
Historic 7/14/21 days (moving offset discarding interpolation)	-4.20%	4.77%	23.09%
Historic 7/14/21 days (discarding interpolation)	-8.87%	1.35%	21.26%
Historic 7/14/21 days (moving offset 1 hour window)	5.62%	12.62%	21.14%
Historic 1/2/3 days	6.21%	10.30%	19.81%
Historic 7/14/21 days (moving offset)	3.00%	8.87%	18.25%
Historic 7/14/21 days (same route only)	-1.94%	5.59%	17.98%
Historic 7/14/21 days (moving offset same route only)	-1.11%	7.42%	15.51%

Figure 6.12.: Table of scores generated by the *percentage errors* method aggregating over all routes under examination evaluated on the different prediction methods introduced in section 6.2, between 29th May, 2016 and 4th June, 2016, ordered in descending order by the median absolute error measure.

Figure 6.12 shows the scores evaluted under this scoring mechanism. The prediction methods have been sorted by descending score with respect to the median absolute error measure. One the whole we see similar results, where the *Historic 5 days weekdays; 7/14/21 days weekends* and *Real time* families are neck and neck depending on the performance measure, but they both clearly beat the *Historic 7/14/21 days* family.

Bank holidays We obtain interesting (and surprising) results if we look specifically at days on which people have special commuting patterns, such as bank holidays. Figures 6.13 and 6.14 show the score tables under the same bus routes, where we have restricted the time frame to a 24 hour period on 29th May, 2016, which happens to be a spring bank holiday. Scores have been ordered in descending order according to the RMSE measure. We see that the predictions methods that performed consistently well across all three error measures are the real-time based methods, achieving 20+ points and relative superiority of $\approx 24-30\%$ to the TfL method, whereas all other methods achieved a score near 0 or negative in the RMSE measure. This is because journey times are in general faster than usual, and therefore leading to historic based prediction methods overestimating travel times.

Total count of journeys with superior predictions to TfL			
Method	Root mean squared error	Mean absolute error	Median absolute error
Real time last two	25	25	23
Real time median	23	27	23
Real time last three (1 hour window)	23	23	21
Real time (1 hour window)	21	27	23
Real time median (1 hour window)	21	23	21
Mixed historic & real time (Historic 7/14/21 days (half hour offset - 1 hour window)/Real time last three (1 hour window)/delay threshold = 3 buses 2 min)	1	17	23
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun	1	15	21
Historic 7/14/21 days	1	15	21
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (same route only)	1	12	20
Historic 7/14/21 days (half hour offset - 1 hour window)	1	14	20
Historic 7/14/21 days (same route only)	1	9	17
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset)	0	17	25
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset 1 hour window)	0	16	22
Historic 7/14/21 days (moving offset 1 hour window)	0	16	18
Mixed historic & real time (Historic 7/14/21 days (half hour offset - 1 hour window)/Real time last three (1 hour window)/with delay reverse lookup)	-1	13	21
Historic 7/14/21 days (moving offset)	-1	13	19
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset same route only)	-2	10	18
Historic 7/14/21 days (moving offset same route only)	-3	11	17
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (discarding interpolation)	-4	8	20
Historic 1/2/3 days	-5	-5	-1
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset discarding interpolation - 1 hour window)	-10	10	18
Historic 7/14/21 days (discarding interpolation)	-10	10	18
Historic 7/14/21 days (moving offset discarding interpolation)	-10	10	16
Historic 7/14/21 days (moving offset discarding interpolation - 1 hour window)	-12	12	18
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset discarding interpolation)	-12	10	16

Figure 6.13.: Table of scores under the *crude scoring* method aggregating over all routes under examination evaluated on the different prediction methods introduced in section 6.2 on the spring bank holiday, 29th May, 2016, ordered in descending order by the RMSE measure.

Relative superiority to TfL (positive is better)			
Prediction method	Root mean squared error	Mean absolute error	Median absolute error
Real time last two	30.85%	33.93%	39.58%
Real time median	29.80%	34.53%	42.69%
Real time last three (1 hour window)	29.50%	33.90%	41.15%
Real time (1 hour window)	28.56%	33.12%	39.35%
Real time median (1 hour window)	24.97%	29.93%	36.72%
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun	3.46%	14.27%	30.71%
Mixed historic & real time (Historic 7/14/21 days (half hour offset - 1 hour window)/Real time last three (1 hour window)/delay threshold = 3 buses 2 min)	3.23%	17.13%	37.28%
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset)	1.54%	14.57%	32.85%
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset 1 hour window)	0.83%	13.42%	31.51%
Mixed historic & real time (Historic 7/14/21 days (half hour offset - 1 hour window)/Real time last three (1 hour window)/with delay reverse lookup)	-0.93%	13.37%	34.62%
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (same route only)	-3.35%	10.17%	30.16%
Historic 7/14/21 days	-6.96%	7.43%	29.64%
Historic 7/14/21 days (moving offset 1 hour window)	-9.52%	5.50%	27.03%
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset same route only)	-10.84%	7.68%	28.34%
Historic 7/14/21 days (half hour offset - 1 hour window)	-11.02%	3.91%	26.88%
Historic 7/14/21 days (moving offset)	-13.16%	1.86%	22.71%
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset discarding interpolation - 1 hour window)	-13.22%	6.61%	30.33%
Historic 7/14/21 days (moving offset discarding interpolation - 1 hour window)	-13.23%	7.18%	31.28%
Historic 7/14/21 days (discarding interpolation)	-14.10%	6.13%	30.25%
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (discarding interpolation)	-14.37%	4.67%	28.10%
Historic 1/2/3 days	-16.46%	-17.97%	-18.71%
Historic 7/14/21 days (moving offset discarding interpolation)	-20.02%	0.92%	28.95%
Historic 5 days Mon-Fri, 7/14/21 days Sat Sun (moving offset discarding interpolation)	-22.66%	-1.92%	27.18%
Historic 7/14/21 days (same route only)	-32.22%	-22.03%	-2.59%
Historic 7/14/21 days (moving offset same route only)	-38.95%	-23.79%	-2.73%

Figure 6.14.: Table of scores under the *percentage errors* method aggregating over all routes under examination evaluated on the different prediction methods introduced in section 6.2 on the spring bank holiday, 29th May, 2016, ordered in descending order by the RMSE measure.

Unfortunately there has not been libraries available for GoLang to determine UK bank holidays as of writing[65], and there is particular difficulty in screening out such cases in production. We therefore regretfully have to neglect it on this occasion, but in theory we should obtain better results when implementations are feasible in the future.

[65] As of 8 June, 2016.

6.5. Results

With the score table in figure 6.11, we are now in a position to answer the questions that have been posed as we came up with the prediction methods.

6.5.1. Reliability of interpolation

In section 4.2.2 we discussed the use of interpolation to deduce the arrival times of buses where necessary. Looking at the scores of the baseline methods with their *discarding interpolation* counterparts, we do not observe consistently better results by discarding historic records as a result of interpolation. We therefore trust that our interpolation method does not give inferior results, and that our interpolation method is reliable.

6.5.2. Effect of bus route on journey times along shared road sections

By comparing the scores of the baseline methods with their *same route only* counterparts, we do not observe improved results by considering only historic records that belong to the same bus route on road sections that are shared by multiple bus routes. The effects of individual bus routes, such as differences in passenger demands and dwelling times, are therefore negligible compared to road traffic.

6.5.3. Periodicity vs Recent Data

The superiority of the *Historic 5 days weekdays; 7/14/21 days weekends* family over the *Historic 7/14/21 days* family suggests that it is more important to adapt to changes in recent road conditions, than to strictly enforce the periodicity of data of 7 days when considering historic records, leading to the use of records in the distant past as far as 21 days.

6.5.4. Moving offset

The *moving offset* versions for both the *Historic 5 days weekdays; 7/14/21 days weekends* and *Historic 7/14/21 days* family almost always give a better score than the baseline methods. This suggests that compensating for changing road traffic as a user travels is indeed preferable where possible.

6.5.5. Width of sliding window

So far we have considered two types of sliding windows - half hour and one hour windows centred around the desired travel times. The scoring table suggests the latter as a better candidate. This

means that a larger sample size outweighs any potentially negative effects due to changing road conditions.

6.5.6. Historic vs real-time based methods

We see that the *Historic 5 days weekdays; 7/14/21 days weekends* gains an edge over the *real time* family on the *crude scoring* evaluation method, whereas the *percentage error* method suggests the other way round. This suggests that they have strengths in different areas - the historic based prediction methods are able to outperform the TfL prediction in a wider variety of situations, while the real-time based methods on average gives superior results than the historic methods when they both outdo the TfL prediction method.

6.6. Winners

6.6.1. Historic based methods

Based on the above discussion, the hybrid method *Historic 5 days weekdays; 7/14/21 days weekends (moving offset 1 hour window)* establishes itself as the king of historic based prediction methods, having achieved one of the best scores both within its family and other prediction methods combined.

Realistically, it is difficult to implement moving offset methods in production, owing to computational costs. This is because the historic records to be retrieved for computation of predictions are highly dependent on the user's choice of route, based on the time which the user is expected to reach each individual chunk along the intended route. This will mean special servicing (looking up of the live database of historic journey times) is required for each user request, and scalability will be disastrous as the number of simultaneous user request grows.

We would therefore prefer methods that are static, so that a cache of current journey times, independent of the users' choice of route can be built, ensuring all user requests can be serviced by fast lookup on the cached table. This immediately suggests the simple *Historic 5 days weekdays; 7/14/21 days weekends* method, which performed reasonably well in our score table, having put all prediction methods from the *Historic 7/14/21 days* family to shame.

However, we may be able to do better from the results just learnt. We can tweak the sliding window to an hour instead of 30 minutes, and attempt to compensate for changes in road traffic by centring predictions at some predefined time beyond the time of departure, with the same rational as that of moving offset.

As most journeys under examination are found take on average an hour to complete, a first attempt would be setting the predefined time to be half of that, so that predictions get centred around the

30 minute mark, to hopefully balance off any increasing or decreasing trends in road traffic.

We therefore introduce the historic based prediction method:

3. Historic 5 days weekdays; 7/14/21 days weekends (half hour offset with 1 hour window)

   This method is based on the *Historic 5 days weekdays; 7/14/21 days weekends* method, but we consider historic records lying between ± 30 minutes of the time 30 minutes after departure. For example, if we are departing at 9 AM, then predictions are generated based on records lying within ± 30 minutes of 9:30 AM of 7, 14 and 21 days ago, effectively records between 9 AM and 10 AM on those days.

The results are shown in figures 6.10 and 6.11 above. We see that this method surprisingly performs at least as good as *Historic 5 days weekdays; 7/14/21 days weekends (moving offset 1 hour window)*, the best performing historic based method that we have found thus far.

With all the advantages we have listed out, we will be using the method *Historic 5 days weekdays; 7/14/21 days weekends (half hour offset with 1 hour window)* as our desired historic-based prediction method in later discussion.

6.6.2. Real-time based methods

All prediction methods from the real time family are pretty much comparable to each other, with only the *Real time median (1 hour window)* method falling short of its peers a bit. Nevertheless we see a slight edge of the *Real time median* method over the others, winning by a slight margin based on the three performance measures. We therefore pick *Real time median* as the ultimate real time based prediction method for the purpose of prediction generation.

6.7. Mixed methods

Having selected the best historic and real-time based prediction methods, it is time to piece both together, to take advantage of known periodicities of journey times and also quickly adapt to temporary changes, such as delays or bank holidays.

In section 5.2 we looked at a couple of pairs of adjacent stops for the average lasting time of delays (or journeys faster than usual). We built individual tables for each pair of stops, so that the expected lasting time of such delays or early trips can be looked up. We can apply the same idea on historic and real-time based prediction methods. The idea is to use journey times given by the historic based prediction method by default, unless looking up of the real time situation on the tables suggest any delays / early trips to last beyond the time when the user is expected to go past the section.

To this end, we introduce two additional prediction methods, based on the best historic and real-time prediction methods we have found:

4. Mixed historic & real time (with delay threshold of 3 buses 2 minutes)

   This method is a combination of the *Historic 5 days weekdays; 7/14/21 days weekends (half hour offset with 1 hour window)* and *Real time median* method. It falls back on the historic based *Historic 5 days weekdays; 7/14/21 days weekends (half hour offset with 1 hour window)* method, unless it is detected that the last 3 buses in the recent past all exceed the historic journey times by more than ± 2 minutes, with which we then switch to the prediction generated by the *Real Time median* method. To prevent looking at records which are too outdated back in time, we set up an artificial time limit of 120 minutes, so that if we do not observe 3 buses within the last 2 hours, we would simply stick with the historic based prediction.

5. Mixed historic & real time (with delay reverse lookup)

   This method is also a mixture of the *Historic 5 days weekdays; 7/14/21 days weekends (half hour offset with 1 hour window)* and *Real time median* method, where we only adopt real time estimates if the expected lasting time of the current situation indicated by the cell entries[66] (figure 5.6), is longer than the time that the user is expected to reach the section. Should there be multiple cell entries applicable[67], we pick the entry with the greatest expected lasting time.

These two methods are different in that the former is a simplified model - it considers a fixed threshold in delays for all pairs of stops in Central London, i.e. assuming the real-time based method is more reliable whenever there are at least three buses consecutively being delayed or faster than usual by 2 minutes, and that any delays happening now are for sure affecting the user so long as his/her route will be passing through the sections being delayed, regardless of how long after now will that be.

We can therefore be able to find out whether the simpler model delivers results just as good.

6.7.1. Building of reverse lookup tables

The latter mixed method involves looking up of the current situation on the table of historic lasting times of delays to determine whether the real time estimates should be used rather than historic estimates. We now describe the implementation of these tables.

We partition each week into individual days of the week, and each day into 24 hour slots. This is based on the assumption that delays happening on the same day of the week (e.g. Mondays)

[66] The number of buses in the recent past that has consistently exceeded historic journey times by more than $\pm x$ minutes, $x \geq 2$.

[67] For example having detected 4 buses late by 5 minutes also implies that 3 buses were late by 4 minutes.

and the same time (e.g. 9 AM) last more or less the same. So, our table will consist of entries for each day of the week at a certain hour, for each pair of adjacent stops in Central London. Lookups will be performed on the rows that match in terms of the the day of the week and hour slot. Therefore, we will not be looking up historic delays on Mondays 9 AM if it is now Tuesday 9 AM for instance.

6.7.2. Qualitative Evaluation: Visualisation

Repeating the above evaluation procedures (sections 6.3 and 6.4) we obtain figures 6.15, 6.16, 6.10 and 6.11. Figures 6.15 and 6.16 display the time series charts of the two mixed based methods on route 9 between 29th May and 4th June, accompanied by the actual journey times for comparison.

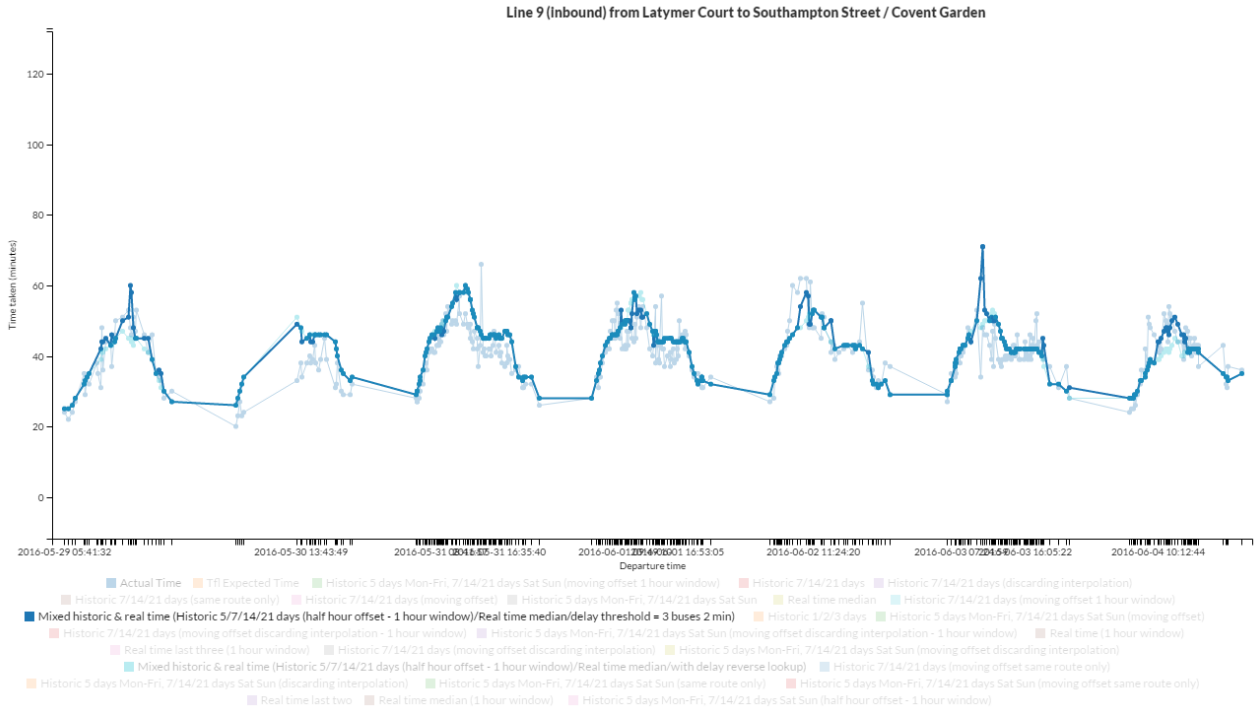


Figure 6.15.: Time series chart of journey times in minutes (y-axis) against the departure times of the corresponding journeys (x-axis), of bus route 9 in the inbound direction between 29th May, 2016 and 4th June, 2016, showing the actual journey times and the two mixed methods. Highlighted in blue is the simpler mixed method *Mixed historic & real time (with delay threshold of 3 buses 2 minutes)*.

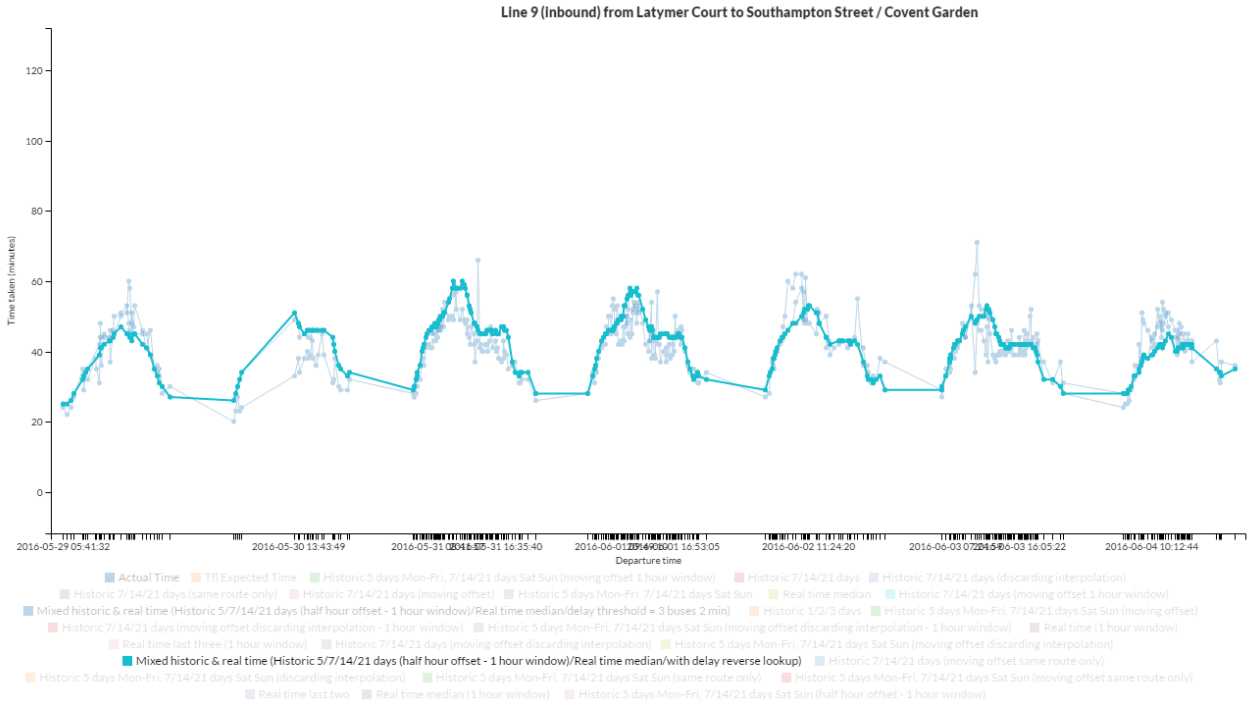


Figure 6.16.: Time series chart of journey times in minutes (y-axis) against the departure times of the corresponding journeys (x-axis), of bus route 9 in the inbound direction between 29th May, 2016 and 4th June, 2016, showing the actual journey times and the two mixed methods. Highlighted in light blue is the reverse lookup mixed method *Mixed historic & real time (with delay reverse lookup)*.

We see that the chart for the *Mixed historic & real time (with delay threshold of 3 buses 2 minutes)* method (figure 6.15) appears more spiky, due to the effects from the real time components kicking in - note that this simpler method has successfully detected delays on 2 PM of 29th May and 9 AM of 2nd June, compared to the other mixed method. Unfortunately, false positives can also be seen near 9 AM on the 3rd June, where it spiked upwards whereas the actual journey times actually went down in the other direction.

On the other hand, the more complex *Mixed historic & real time (with delay reverse lookup)* method did not exhibit irregularities. In fact, on this occasion it overlaps completely with the underlying historic method *Historic 5 days weekdays; 7/14/21 days weekends (half hour offset with 1 hour window)* (figure 6.17), suggesting that real time estimates were not switched to at all. Digging down it was found to be a lack of training examples on the reverse lookup table, since we partitioned time by weekday and hour, and therefore any delays have mostly been new unseen examples, and hence we fail to learn from past experience and switch to real time estimates where necessary.

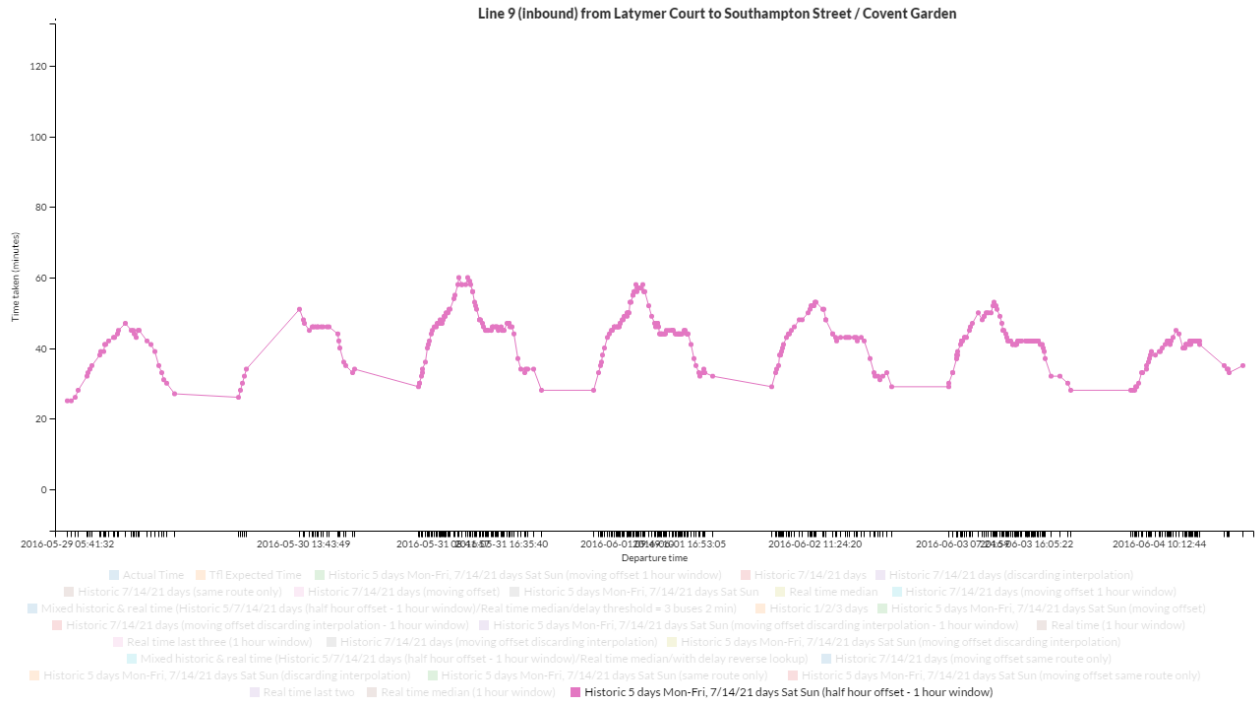


Figure 6.17.: Time series chart of journey times in minutes (y-axis) against the departure times of the corresponding journeys (x-axis), of bus route 9 in the inbound direction between 29th May, 2016 and 4th June, 2016, showing the *Historic 5 days weekdays; 7/14/21 days weekends (half hour offset with 1 hour window)* method only.

6.7.3. Quantitative Evaluation: Performance measures

Looking back at figures 6.10, 6.11 and 6.12 we obtain a quantitative estimate of how our mixed based methods perform relative to the pure historic and real-time based methods.

They both gained a top of the range score in the Mean and Median absolute error measures overall, despite falling a bit short in the RMSE measure in figures 6.11 and 6.12. Digging into the individual journeys (figure 6.10), we see that the reverse lookup method *Mixed historic & real time (with delay reverse lookup)* displayed a slight edge over the simpler *Mixed historic & real time (with delay threshold of 3 buses 2 minutes)*. This may mean that it is better to be conservative in using the real time information, unless it is certain that the delays are going to last long.

6.7.4. Winner

With the above discussion in mind we will be giving the award to *Mixed historic & real time (with delay reverse lookup)*, and utilising it hereafter and for prediction generation for the final app.

7. Optimisation

Now that we have established a desirable prediction method, it remains to wire up our backend and frontend, so that our users can be easily informed of journey times and any delays affecting them. An important factor to consider is the responsiveness of our frontend client, as users would not want to have to wait for a substantial amount of time before predictions are made available to them.

So far to generate predictions we have been looking up historic records from the live production table `journey_history` directly. This is undesirable as reads from this table may block updates to the table, and therefore hindering incoming predictions to be stored swiftly. As the number of user grows, we will have difficulty in serving requests concurrently.

7.1. Caching current predictions

One way out is to create another temporary table, mainly for fast lookup in order to generate estimated journey times of users. We could precompute the current estimates of travel times between all pairs of adjacent stops within the Central London network, which takes only one pass, and store them in a table for universal lookup. Note that the precomputation is over all pairs of stops at once, and therefore saving any network traffic overhead otherwise if they were to be computed lazily when each user makes a request. This has the effect of diverting traffic away from the `journey_history` table, and preventing live updates from being held up by locks.

Figure 3.4 on page 38 shows the internal structure of the `current_estimates` table, where we record for each pair of adjacent stops (identified by `from_id`, `to_id` - NaPTAN identifiers of the stops), the current historic based estimate, real-time based estimate and estimate (i.e. mixed-based estimate), with the winner methods that we found in the *Evaluation* section (section 6). The fields `delay`, `delay_count` and `delay_expected_to_last` suggests whether a delay is ongoing, and if so, tells us also how many recent buses have been consecutively delayed, and how long the delay is expected to last based on past experience (section 5.2). We also record the time when the record was generated, as a timestamp in the field `last_updated`, so that we only fetch records that are computed recently when user requests come in[68].

[68] This is necessary as some pairs of stops are not served 24 hours (i.e. those with only day buses or night buses), and occasionally we may not get updates if buses are on diversion. In such cases the entries in the table `current_estimates` would be outdated, and not reflective of the current traffic and travel times.

When we need to generate predictions for a user defined route departing now, we first decompose the route into pairs of adjacent bus stops, and look up the corresponding entries in the `current_estimates` table. Should all entries be recently updated enough, we can proceed to sum them up as the prediction for the intended route. One catch is that we prefer to use real-time based estimates only when delays are expected to last till the time when the user travels past the route section, whereas the `estimate` field refers to the mixed-based estimates based on the "current" road conditions. We therefore having calculated as the first pass of when the user is expected to arrive at each stop pair, can then consult the `delay_expected_to_last` field, so that we really use the `real_time_estimate` only if we find that delays should persist long enough to affect the user's journey.

7.2. Waiting times

An important component in commuting is the time spent at bus stops waiting for the next bus. It would be misleading to the user if we only take into account of the travel times of buses on the road, in particular when the waiting times could make up 10 or 20 minutes of the total journey time. Therefore, we need to build facilities that allow us to predict times that a user can expect to wait for the next bus to arrive. Fortunately, we have actually got most bits done, the remaining is to infer the times that buses would arrive at each stop based on their current locations. In other words, we simply have to apply our prediction method on all current buses to find out the times required to travel to each stop down their respective routes.

The `next_buses` table (figure 3.4 on page 38) caches all the times that buses currently in operation are expected to arrive at the stops along their routes. Each entry consists of the `line` and `bound` (i.e. direction - either outbound or inbound) of the bus, the `naptan_id` of the stop that the bus is expected to arrive, the registration plate of the bus as the `vehicle_id`, and the time that the bus is expected to arrive, `expected_arrival_time`. We also store the timestamp of when the record was generated, `last_updated`, for debugging purposes.

While this method should generate arrival times for most stops and bus route combinations, we still need to handle the remaining edge cases. One such case is at stops near the departing terminus, when no buses are found to have been on its way to pick up passengers yet. In such cases, we could only rely on the times that TfL give on when buses are expected to depart the terminus, and from then we calculate when the buses should arrive at the stops of interest. The table `terminus_departures` is used to store such predictions. We store the same content as in `next_buses`, only that entries in this table contain the departing terminus of each bus route.

For the minority of cases where the user requests to depart at some future point in time, predictions may not be available and we need to infer arrival times based on bus schedules. The table `timetable` records all scheduled departures of all bus routes at their termini, based on the day of the week. Assuming that buses depart on schedule, we can apply the prediction method *Mixed*

historic & real time (with delay reverse lookup) as above to infer the arrival times at stops, and therefore obtain corresponding predictions of how long users can expect to wait at bus stops.

7.3. Delay characteristics

In section 5.2 we looked at classification of delays based on the number of consecutive buses exceeding a certain threshold in time longer or shorter than usual, and came up with the table `delay_characteristics` to cache the expected lasting times of such anomalies. A problem we encountered was the vast number of pairs of neighbouring bus stops in Central London, and partitioning time into weekdays and one hour slots resulted in large number of combinations to handle, therefore causing each iteration of database update to take immensely long.

A quick observation at figures 5.6 - 5.15 suggests that more than half of the entries are actually 0 for most of the time. By leaving omitting such entries we can reduce the size of update by 50%+. We therefore apply updates only for entries whose `expected_lasting_time` are greater than zero, and we also attach information on when this update is performed in the `last_updated` field. This has drastically reduced the time of each iteration to approximately 4 hours, which is much more manageable.

8. Conclusion

8.1. Regularities in bus journey times

Having examined multiple prediction methods on bus journey times across a variety of routes in Central London, we find that the best performing historic methods are from the historic based *Historic 5 days weekdays; 7/14/21 days weekends* family. This suggests that bus journey times subject to usual traffic are periodic to a certain extent, where journey times are pretty much similar between Mondays and Fridays, and standalone on Saturdays and Sundays.

Historic journey times are therefore in general indicative of the current journey times, with the exception of the occasional delay and abrupt changes in commuting patterns, such as bank holidays. In such cases the real time based prediction methods are far more reliable.

In delay handling, it is better to be conservative when trying to adopt real time estimates, to avoid being tricked by short-lived delays which may not have much of an impact on actual journeys.

The superiority of the hybrid *Historic 5 days weekdays; 7/14/21 days weekends* family over the *7/14/21 days weekends* family suggests importance in adapting to recent changes in road conditions, and not relying on historic data too distant in the past.

8.2. Working with third party components

Working with external APIs has always presented certain challenges, as limited access and control over them means that any anomalies therein cannot be solved from the root, but rather workarounds have to be developed. It also takes continuous effort to discover any edge cases and develop corresponding patches.

8.3. Gains on concurrency

The use of concurrency has proven to greatly enhance efficiency by reducing the effects of I/O bottlenecks from network or database communication. Proper division of work into threads (such as subdividing all bus routes into their own goroutines in our case) has also allowed for isolation, so that an edge case for one would not take down the others.

9. Future Work

9.1. Variations on parameters of existing prediction methods

The prediction methods studied were a coarse illustration of the main directions, where we considered only one or two representatives in the aspects of *discarding interpolation*, *same route only*, averaging with *means, medians*, and families with different periods. There is still potential to fine tune the parameters, such as

- Swapping *Historic 7/14/21 days* for *Historic 7 days* or *Historic 7/14 days*, in order to investigate the trade-off between the extent to which historic data should be considered verses sample size,
- Trying out other sliding windows, such as 45 minutes, 75 minutes or 90 minutes etc.

9.2. Aggregating results over all bus routes

While the structure and code of the project has been designed to be able to evaluate on all bus routes in Central London, owing to computational complexity and running time we have limited ourselves to approximately 20 bus routes in the analysis. It will be straightforward to include all bus routes in the numerical analysis, provided we have more powerful machines.

9.3. Other prediction methods in literature

While other varieties of bus journey times prediction methods are present in literature, such as ANNs, SVMs, regression models and Kalman filters introduced in the Background section (section 2.5), we haven't had the opportunity to explore them in our project and compare with the time series based approaches that we have used. It will be interesting to find out how the above methods generalise to the case of large cities, rather than focusing on only a small number of routes.

Bibliography

[1] Transport for London. (2016, April) Vehicles entering in the congestion charge zone by month. Last accessed 21 April, 2016. [Online]. Available: http://data.london.gov.uk/dataset/vehicles-entering-c-charge-zone-month/resource/867ccde0-e52b-45d0-b2f4-9cf58b6c8085

[2] (2015, August) London 'most congested city in europe'. Last accessed 21 April, 2016. [Online]. Available: http://www.bbc.co.uk/news/uk-england-london-34044423

[3] Department for Transport, Ed., *Annual Bus Statistics: 2013/2014*, 2014.

[4] J. Peckham. (2015, September) Tfl will never make the ultimate travel app - but there is a very good reason. Last accessed 21 April, 2016. [Online]. Available: http://www.techradar.com/news/phone-and-communications/mobile-phones/tfl-will-never-make-the-ultimate-travel-app-but-there-is-a-very-good-reason-1304247

[5] Y. Bin, Y. ZhongZhen, and Y. Baozhen, "Bus arrival time prediction using support vector machines," *Journal of Intelligent Transport Systems*, vol. 10, no. 4, pp. 151–158, 2006.

[6] B. Yu, W. H. Lam, and M. L. Tam, "Bus arrival time prediction at bus stop with multiple routes," *Elsevier*, August 2011.

[7] H. Chang, D. Park, S. Lee, H. Lee, and S. Baek, "Dynamic multi-interval bus travel time prediction using bus transit data," *Transportmetrica*, vol. 6, no. 1, pp. 19–38, January 2010.

[8] S. I.-J. Chien, Y. Ding, and C. Wei, "Dynamic bus arrival time prediction with artificial neural networks," *Journal of Transportation Engineering*, vol. 128, no. 5, pp. 429–438, September/October 2002.

[9] S. I.-J. Chien and C. M. Kuchipudi, "Dynamic travel time prediction with real-time and historic data," *Journal of Transport Engineering*, vol. 129, no. 6, pp. 608–616, November/December 2003.

[10] L. Deng, Z. He, and R. Zhong, "The bus travel time prediction based on bayesian networks," in *International Conference on Information Technology and Applications*, 2013.

[11] J. Patnaik, S. Chien, and A. Bladikas, "Estimation of bus arrival time using apc data," *Journal of Public Transportation*, vol. 7, no. 1, 2004.

[12] Suwardo, M. Napiah, and I. Kamaruddin, "Arima models for bus travel time prediction," *The Institution of Engineers, Malaysia*, vol. 71, no. 2, June 2010.

[13] Transport for London. Buses. [Online]. Available: https://tfl.gov.uk/corporate/about-tfl/what-we-do/buses

[14] Wikipedia. ibus (london). [Online]. Available: https://en.wikipedia.org/wiki/IBus_(London)

[15] Transport for London. Data feeds. Last accessed 21 April, 2016. [Online]. Available: https://tfl.gov.uk/info-for/open-data-users/data-feeds?intcmp=29422#on-this-page-1

[16] ——, *TfL Live Bus & River Bus Arrivals API Interface Documentation*, 2016.

[17] Citymapper. Our cities. [Online]. Available: https://citymapper.com/cities

[18] Citymapper - the transport app with live times for bus, train and tube. [Online]. Available: https://itunes.apple.com/gb/app/citymapper-london-hong-kong/id469463298?mt=8

[19] T. Singh. (2014, 4) How google maps traffic works? Website. Last accessed 21 April, 2016. [Online]. Available: http://geeknizer.com/how-google-maps-traffic-works/

[20] liangjy. (2014, June) Time series forecasting vs linear regression extrapolation. [Online]. Available: http://stats.stackexchange.com/questions/103209/time-series-forecasting-vs-linear-regression-extrapolation

[21] C. Bai, Z.-R. Pang, Q.-C. Lu, and J. Sun, "Dynamic bus travel time prediction models on road with multiple bus routes," *Computational Intelligence and Neuroscience*, 2015.

[22] C. Mateo. (2015, Aug) Performance of several languages. [Online]. Available: http://blog.carlesmateo.com/2014/10/13/performance-of-several-languages/

[23] K. Kowalczyk. (2012, Sep) How i sped up go by 20java?). [Online]. Available: https://blog.kowalczyk.info/article/u3d4/How-I-sped-up-Go-by-20-or-is-Go-really-slower-th.html

[24] C. Oblikov. (2011, Dec) Golang: goroutines performance. [Online]. Available: http://en.munknex.net/2011/12/golang-goroutines-performance.html

[25] PostgreSQL. (2016, Apr) Advantages. [Online]. Available: http://www.postgresql.org/about/advantages/

[26] Transport for London. National public transport access nodes (naptan). [Online]. Available: https://data.gov.uk/dataset/naptan

[27] London Assembly Transport Committee, "Bus services in london," October 2013.

[28] Designmodo. Like icon. Last accessed 21 April, 2016. [Online]. Available: https://www.iconfinder.com/icons/115720/like_thumbs_thumbs_up_up_vote_icon#size=128

[29] Freepik, "Tick inside a circle free icon," Flaticon Basic License. [Online]. Available: http://www.flaticon.com/free-icon/tick-inside-a-circle_19973

[30] S. de Jonge. Pedestrian walking icon. Creative Commons (Attribution 3.0 Unported). Last accessed 21 April, 2016. [Online]. Available: http://www.flaticon.com/free-icon/pedestrian-walking_8818#term=walk&page=1&position=1

[31] Paomedia. Alert icon. Last accessed 26 April, 2016. [Online]. Available: http://www.iconarchive.com/show/small-n-flat-icons-by-paomedia/sign-warning-icon.html

[32] Squid.ink. Rubber duck icon. Creative Commons (Attribution 3.0 Unported). Last accessed 21 April, 2016. [Online]. Available: https://www.iconfinder.com/icons/416395/bath_bathroom_clean_duck_kids_rubber_water_icon

[33] Transport for London. Bus icon. Brand Iconography Standard. Last accessed 21 April, 2016. [Online]. Available: content.tfl.gov.uk/onl-std-081-brand-iconography-standard.pdf

[34] Webalys. Bus icon. Free for commercial use. Last accessed 21 April, 2016. [Online]. Available: https://www.iconfinder.com/icons/379528/bus_icon#size=128

A. Source code

A.1. GitHub repository

A copy of the source code is available on

http://github.com/karl-chan/Active_Delay_Warning_Transport_App.

All plots in the report can be located under the */charts* directory.

B. Web server API

We introduce here the server that we have set up, from which front end clients (as in the case of our Android app) can query for route planning and live delay data.

B.1. Journey planner

`GET`	http://146.169.46.107/journeyplanner	
Request parameters	`fromLat:`	the latitude of the origin
	`fromLng:`	the longitude of the origin
	`toLat:`	the latitude of the destination
	`toLng:`	the longitude of the destination
	`isDeparting:`	'true' if journey needs to be based on fixed departure time, 'false' if to be based on fixed arrival time.
	`departureOrArrivalTime:`	timestamp in `RFC3339Nano` format, signifying the departure or arrival time depending on the `isDeparting` field

This API call acts as a proxy to the TfL Journey Planner for routing, and replaces travel times with our estimates. It returns routing information, historic and real time estimates of travel times, and the next buses to arrive.

B.2. Delay status

`GET` http://146.169.46.107/delaystatus
No Request parameters required

This API call returns all pairs of neighbouring bus stops that are currently affected by delays, together with their historic, real time estimates, and the number of buses in the last two hours that were delayed.

B.3. Next bus

`GET`	http://146.169.46.107/nextbus	
Request parameters	`stops:`	the NaPTAN ids of stops in interest, delimited by comma
	`numRouteOptionsPerStop:`	the number of bus routes to observe for each stop in the `stops` list, delimited by comma (Note: they need to be in same order as in the `stops` list)
	`lineBoundCombinations:`	the interested bus routes and directions for each stop, in the format `line,bound` for each stop in the `stops` list, delimited by comma (Note: they need to be in same order as in the `stops` list)

This API call gives updates on the next buses that are about to arrive at a given stop, including bus route, registration number of the bus and expected arrival time.

B.4. Delayed routes

`GET` http://146.169.46.107/delayedroutes

No Request parameters required

This API call returns all bus routes affected by delays, and their end-to-end estimated historic and real-time based travel times.

C. User Guide

C.1. Home page

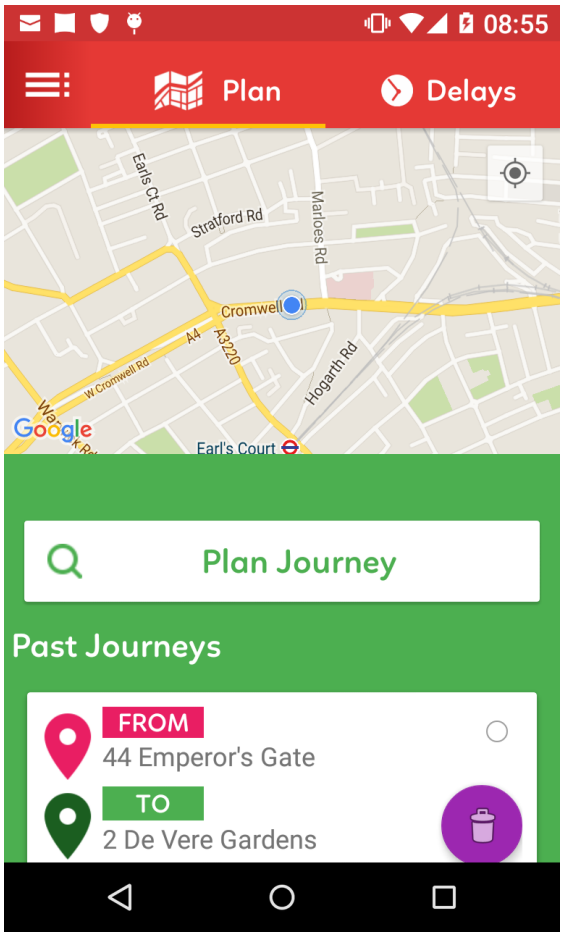


Figure C.1.: Home page of the app, in which the current location of the user is shown

The home page (figure C.1) of the app shows a handy map of the current location of the user, and is the entry point of the route planning feature. For convenience the most recent 5 journeys are also listed, and the user has the option of pinning them, so that for daily commutes the user can

simply tap onto them to get to the route results page.

C.2. Current delays

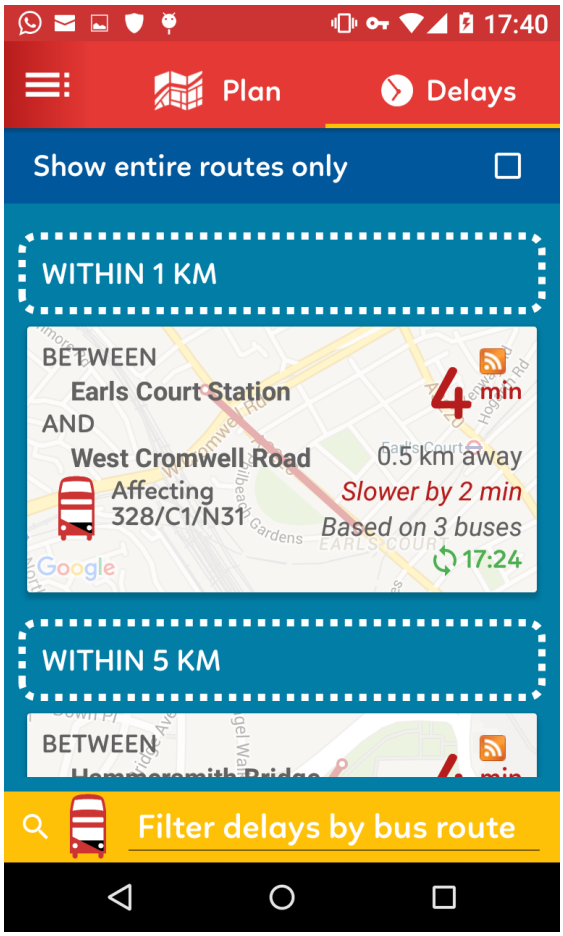


Figure C.2.: Detected delays in Central London, sorted by the distances to the user's current location.

Swiping the home page to the right gives the page of all delays in Central London (figure C.2). It includes information on between which pairs of stops do delays occur, the bus routes that are affected, their distances to the user's current location, and current estimated travel time compared to the historic average. At the bottom is the filter bar that enables users to filter out delays that affect the bus route(s) of interest.

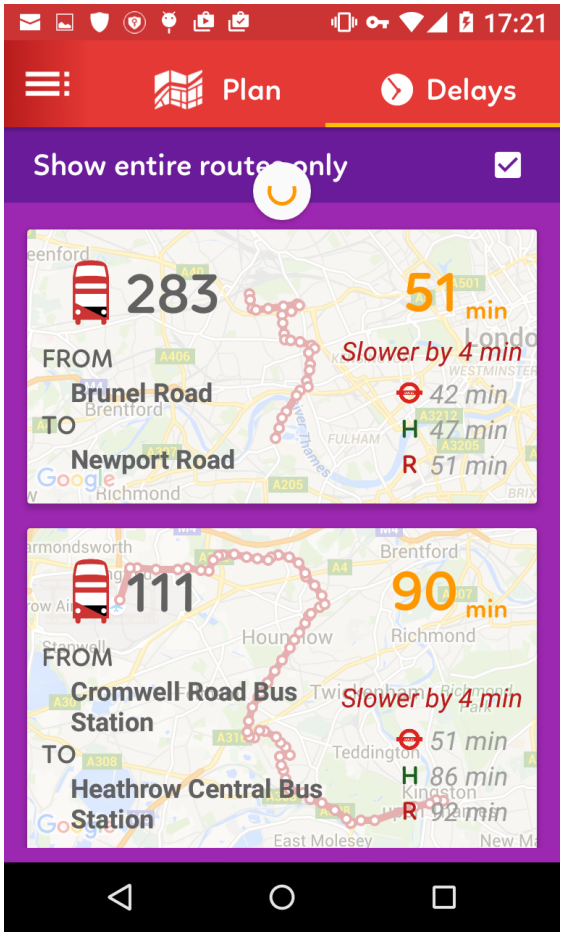


Figure C.3.: Routes affected by delays, together with their historic and real-time journey estimates.

If we are interested in the effects of delays on bus routes end-to-end, tapping the checkbox switches us to another view. Instead of individual pairs of stops, we now obtain information on entire bus routes (figure C.3), sorted in descending order by the amount of delay incurred. We can therefore easily find out which bus routes are currently most severely delayed.

C.3. Route planner

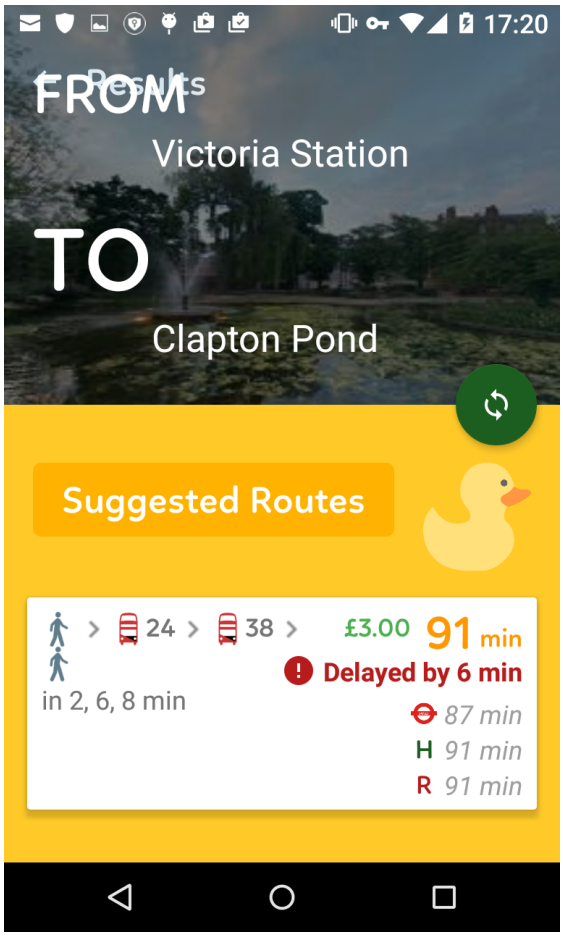


Figure C.4.: Street view of the destination, route results, and journey times taking into consideration of current delays.

The journey results page (figure C.4) provides a number of available routes on the Central London bus network between the user's intended origin and destination. The estimated travel time has already taken into account of any current delays[69] affecting the route, and should it be the case a helpful message is also displayed regarding how much longer the route may take than usual. At the top is a street view of the destination, which hopefully helps users who is new to the route to locate their destination when they get near.

[69] One may notice that the estimate is the same as the historic travel time here. This is because delay for the section is expected to be short-lived based on past data. The real time estimate reflects the travel time with the current road conditions, and assumes all current delays will definitely affect the user. This will be further discussed in the *performance evaluation* section - section 6.2.

C.4. Navigation

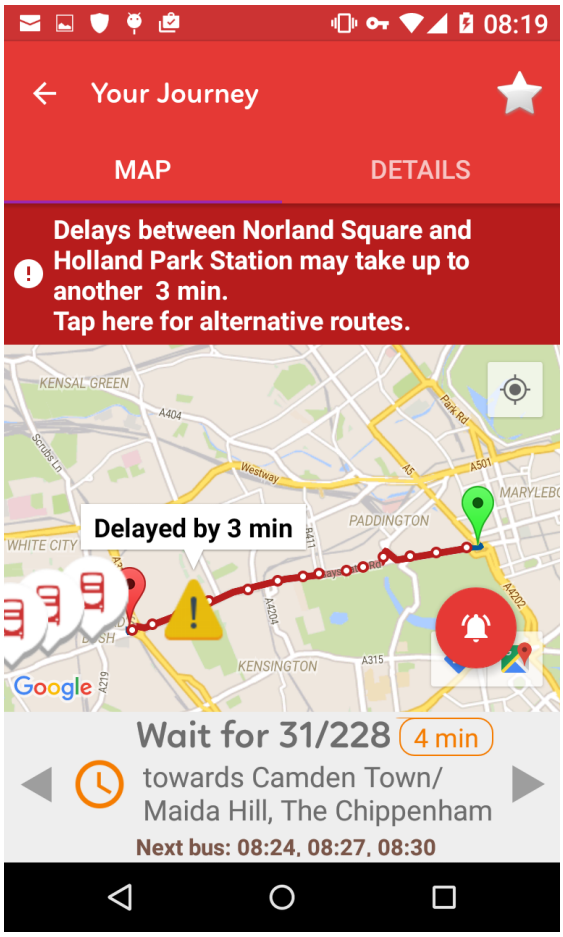


Figure C.5.: Interactive map with instructions of the intended route, and pinpoints where the next buses and delays are.

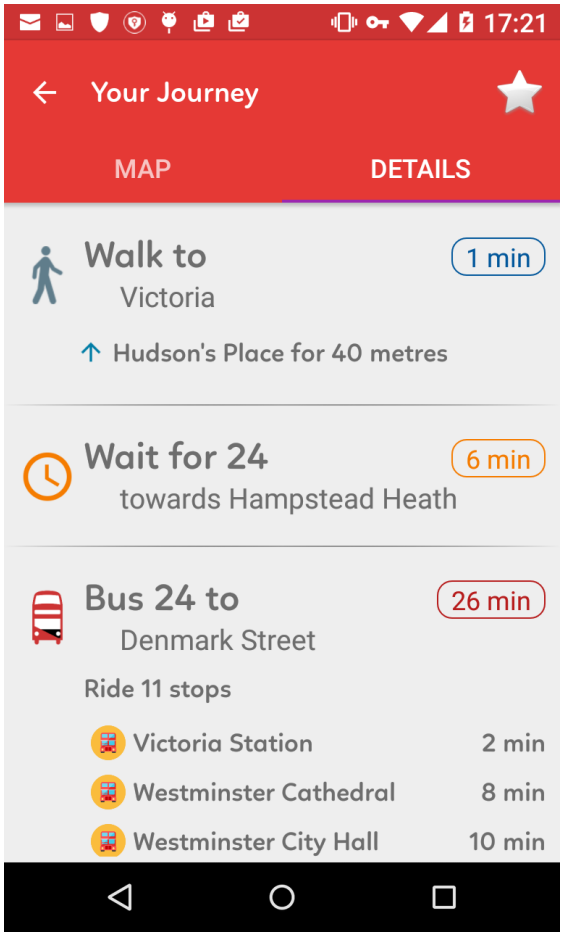


Figure C.6.: Detailed instructions of the intended route, including navigation by foot and expected times to each stop.

Tapping on a route in the journey results page will bring you to the details page (figure C.5), which displays the entire trip on a map, with brief step-by-step navigation instructions at the bottom. Detailed navigation instructions (such as turning at which road, walking for how many metres on each road) can be obtained by swiping to the right to the *Details* tab (figure C.6).

C.5. Delay notification

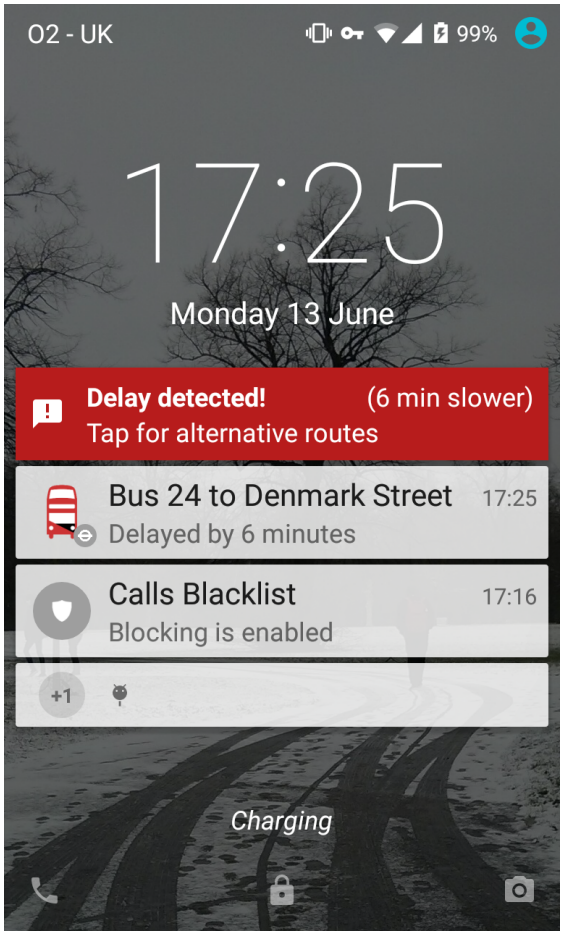


Figure C.7.: Background monitoring of user's route, so that navigation instructions are continuously updated and issued as notifications. Delays notifications are issued in red, and by tapping users will be presented with alternative routes.

On the lower right corner of the map is the notification toggle, which the user should enable when he starts travelling on the intended route, so that his / her current location will be continuously monitored and notifications issued should delays be found on the way (figure C.7). Delays that affect the route are also shown on the map, and by clicking on the appropriate markers, the expected additional travel time will be displayed as a tooltip. A short paragraph regarding the sections that are delayed are shown in red, and the user could simply tap onto it to search for alternative routes from his / her current location to the destination.