Imperial College London

Department of Computing

MEng Individual Project Report

Active Delay Warning Transport Application

Author:
Yangfan Zhang

Supervisor:
Dr. Peter McBrien

June 13, 2015

Submitted in part fulfillment of the requirements for the degree of Master of Engineering in Computing of Imperial College London

Abstract

In London, billions of passengers travel by bus every year. Bus delay is a main concern for passengers when planning their journeys. Currently, there is no data service that provides reliable bus journey time to downstream stops. Transport for London (TfL) offers data to quantify the deviation in bus journey times from the official timetable. We collected the TfL live bus arrival times for each bus at each stop, and computed the average historical and current bus journey time between every pair of neighbouring stops on a route. We built a data service API to provide historical, current, and reference bus journey times for a given hour on a given day of the week. To demonstrate the potential use of our API, we designed a web application to allow passengers to browse these predicted bus jounrney times conveniently.

Acknowledgments

I would like to thank the following people for their support on this project:

Peter McBrien. This project would not have been possible without his exceptional supervision. I appreciate the time and effort he invested in guiding me and helping to remove many of the roadblocks.

My personal tutor, Prof. Susan Eisenbach, for her encouragement and care throughout the year.

My housemates, friends, and family for their support through my ups and downs.

Daniel Wong, for his time to review my code and paper.

Contents

List of Figures

Chapter 1

Introduction

The London bus network carries 2.4 billion passengers a year [1]. While passengers could expect buses to arrive within 10 minutes 83.4% of the time, there was a 15.1% chance of a 10-20 minute wait, a 1.3% chance of a 20-30 minute wait, and a 0.2% chance of a wait exceeding 30 minutes.[6]

Transport for London (TfL) publishes live predictions of bus arrival times. This information is available on digital live bus arrivals signs at more than 2,500 bus stops [2]. Commuters can also access this information via SMS, the TfL website, or its mobile applications.

Passengers use bus arrival times to plan their journey, by factoring in the waiting time when choosing the buses to take. Current London journey planning software takes the journey start time, start location, and destination as input, and recommends routes employing a variety of travel modes, with an estimated duration for each suggested journey. Popular planners include Google Maps [3], Citymapper [4] and the TfL Journey Planner [5].

However, the accuracy of the bus arrival times published is affected by many external factors beyond the distance travelled. For example, when there is heavy traffic, buses are likely to be delayed significantly enough to cause a change in passengers' route picking.

Yet, this delay information is not reflected in the estimated bus journey time early enough for the passengers to make a decision to choose an alternate route. As a result, passengers waste time waiting for buses that come much later than expected, or choose to board a bus that will take much longer than the estimated journey time to reach the destination. Although the average bus delay is 1 minute, there is a 16.6% chance of waiting for more than 10 minutes [16].

This problem can be avoided if passengers are informed of the delays in bus arrival times and estimated travel time in addvance. Currently, the TfL live bus arrivals Application Program Interface (API)feed[2] provides data

on immediate bus arrivals at a given stop. We verified the accuracy of this API in Section 8.1, and found that the accuracy of the arrival predictions vary by the projection time in future.The predictions are largely accurate for the next five minutes, with an average of 45 seconds delay. However, there are no available data services or applications that offer projected bus travel times that incorporate predicted delays.

We collected data from the TfL live bus arrivals API feed to quantify the deviation in estimated bus journey time from the official timetable. We built a data service API to provide historical, current, and reference bus travel times (Chapter 5) and designed a demonstrative web application (Chapter 6). The methodology is presented in Chapter 3.

Chapter 2

Background

2.1 London Bus Network

The bus network in London is one of the largest and most accessible in the world. It carried a staggering number of passengers with more than 2.4 billion journeys in 2013/14, which was more than any year since 1959 [1]. On an average day during 2005 – 2010, London residents made 14% of their trips by bus [9], pending an average of 14 minutes per day on the bus. As at 2015, there are 19,345 bus stops with 680 routes served by 8,765 buses daily in London [10].

2.1.1 Bus Network Performance

Bus service delays and disruptions are commonplace in London. Such delays can significantly affect the estimated duration of a route.

High frequency services. The average scheduled wait was 4.86 minutes, the average excess wait was 0.94 minutes, and the average actual wait was 5.80 minutes. While passengers could expect buses to arrive within 10 minutes 83.4% of the time, there was a 15.1% chance of a 10-20 minute wait, a 1.3% chance of a 20-30 minute wait, and a 0.2% chance of a wait exceeding 30 minutes.

Low frequency services. 87% of buses services were on time, and 11.4% were 5-15 minutes late.

Night buses. 84.5% of services were on time. The average excess wait was 0.68 minutes.

Causes of bus delays included traffic congestion, staff availability, engineering problems, or mechanical breakdown [16]. The above statistics were published in TfL's 2014/2015 Q2 bus performance data [6].

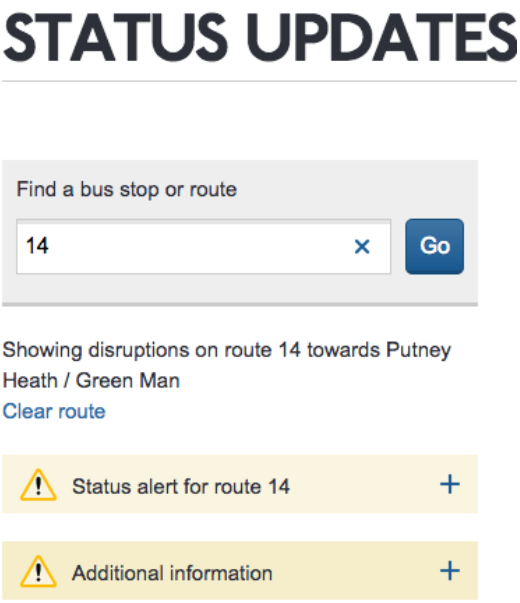


Figure 2.1: Tfl Buses Status Update Service

2.2 Buses Status Updates

The frequency of bus delays makes it essential for an estimate of the length of a bus journey to incorporate estimated delays based on live bus tracking data. To inform passengers of bus service disruptions or diversions, TfL provides a bus status updates service online [7]. Passengers can check the service status of a given bus stop or route (Figure 2.1). The status update consists of textual descriptions of service disruptions, diversion, suspensions, and delays due to heavy traffic. While this service allows passengers to discover disruptions on their route, it requires passengers to actively monitor the site, and does not quantify the effect of the disruptions in extending the passenger's travel time.

TfL also publishes live status news and updates on Twitter [8]. This has the advantage of making the information more easily accessible to passengers. However, the general broadcast does not solve the problem of giving passengers information that is specific to their planned journey.

Many popular journey planners such as Google Maps [3], Citymapper London [4], and TfL Journey Planner incorporate the bus status information in the suggested journeys as a textual alert. However, these applications do not calculate the additional time that the service disruption will add to the journey, leaving passengers to estimate this for themselves.

Thus, it can be clearly seen that no solution currently exists which is able to quantify the additional delay that a disruption adds to a journey, and incorporate those predicted delays into time estimates of different routes presented to a commuter.

2.3 Transport for London Open Data

TfL provides data to quantify the delays in bus journey time. We collected the necessary data from the Live Bus Arrivals API, Bus Stop Locations and Routes, and Journey Planner Bus Timetables. The Live Bus Arrivals API enables a query to receive a bespoke response, depending on the parameters supplied. Bus Stop Locations and Routes come as static data files which rarely change. Journey Planner Bus Timetables are feeds that refresh at regular intervals.

2.3.1 Live Bus Arrivals API

The Live Bus Arrivals API provides the predicted time until a bus is expected to arrive at a stop. This API is designed to enable application developers to subscribe to live bus information and use this data to develop innovative services[15].

The predictions are generated from London buses locations, tracked using a combination of Global Positioning System (GPS), roadside transponders, gyrometers to recognise orientation and turns, and software to match all of the information accurately to a point on a street. Next, the bus location information is transmitted back to a control centre which then works out the predicted bus journey time to reach each downstream stop, given typical journey times at that time of the day [14]. These arrival predictions are available for the next 30 minutes at any point in time. For example, at 9am, the API will provide predicted bus arrivals up to 9.30am on the same day. This data is refreshed every 30 seconds [15].

This API is controlled via a number of different HTTP requests and parameters. A request is structured as follows:
`http://server/virtualDirectory/type/version?HTTPparameters`

Data Types

This API service provides two types of request to users:

- **instant**. The instant requests are made by the client and the server will respond with a single message.

- **stream**. The streaming requests are made by the client, in response the server will continually serve data to satisfy the request until the connection is terminated.

The data source for both instant and streaming requests is consistent, ensuring that the data provided to the public remains the same. We used this API as a source for bus arrival times. See details in Section 4.4.1.

2.3.2 Bus Stop Locations and Routes

The TfL Open Data provides network information on the location of all bus stops in London, and the sequence of bus stops in every bus route. This data is in the Comma Separated Values (CSV) format. Every entry in the bus sequence file consists of information on the route number, the route direction, the sequence at which the given bus stop is positioned in the route. There is also information on the bus stop name, and bus stop codes for identification purposes. We used the bus sequences as a reference when calculating the bus journey tiem. See Section 4.4.2 for details.

2.3.3 Journey Planner Bus Timetables

The Journey Planner Bus Timetables [22] contains information on official bus schedules including stops, routes, departures times, departure frequencies, operational notes, as well as the days on which the services run.

The timetables uses the Extensible Markup Language (XML) [18] format, with the schema defined in TransXChange [17], the UK nationwide standard for exchanging bus schedules and related data. For this project, we used the General schema version 2.1[19][20], the latest available version for download. Each XML file contains the bus timetables for one route.

Data Structure

The TransXChange timetable model has the following seven basic concepts[21]:

1. `Service` contains one or more `JourneyPattern` elements and one or more `VehicleJourney` elements. This is the basic concept that brings together the information about a registered bus service.

2. `Registration` specifies the registration details for a service.

3. `Operator` indicates the entity who runs the service.

4. `Route` describes the physical path taken by buses on the service as an ordered list of `StopPoints`.

5. `StopPoint` contains reusable declarations of the stops used by the routes and journey patterns of the schedule. All StopPointRef instances elsewhere in a document are resolved against the contents of the StopPoints element. All stops are defined as being NaPTAN points.

6. `JourneyPattern` specifies an ordered list of links between the `StopPoints`, giving the *relative travel times* between each pair of neighbouring stops.

7. `VehicleJourney` specifies the individual scheduled journey *at a specific absolute time.*

These elements give a complete official bus schedule for each route with the departure time for each bus journey, the relative bus travel time between stops, and the days on which the service operates on. We extracted the offical bus timetables from these XML files, as discussed in Section 4.3.

2.3.4 Summary on Available Data

While there is sufficient data on London bus network, official bus timetables, and live arrival times, there is no data service offering real-time predictions on bus delays. There is no reliable predictions estimating the travel times between any two downstream stops and the arrival times at each downstream stop from the point of real-time observation. We used these available data to create historical and current bus travel timetables, as discussed in Chapter 4.

2.4 Current Travel Applications

Over 5,000 developers have registered for the open data[13], and around 200 travel apps are powered by it [1]. The most popular travel applications include TfL Journey Planner, Google Maps (Figure 2.2), and Citymapper London. Given the departing location, destination, as well as departure time, these applications can provide suggested routes and travel times. Users can further customise their desired journey by specifying the desired walking distance, and accessibility requirements. Moreover, the TfL Journey Planner and homepage status board were redesigned and integrated so customers planning their trip can see immediately if their route is likely to be affected by upgrade work or other disruptions [1], with a textual warning message shown.

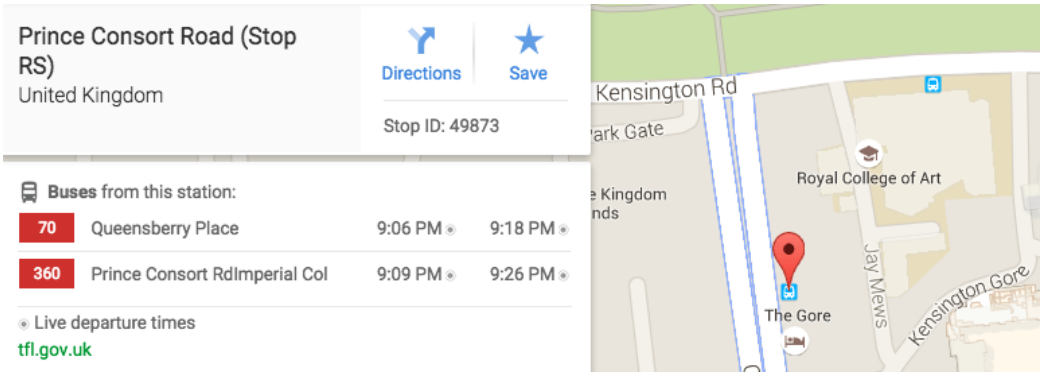


Figure 2.2: Google Maps Using TfL Bus Arrival Data

However, currently, there are no applications that give predictions on travel times and warnings on potential delays. The estimated travel time shown in apps is extracted from the TfL Journey Planner Bus Timetables. This information does not capture the real time delays according to instant traffic conditions.

2.5 Literature

2.5.1 Overview

We did some literature review on the conventional methods used to predict bus arrival times. They include regression models, Kalman filters models, Artificial Neural Network (ANN) models, K-nearest neighbours models, and analytical approaches.

2.5.2 Regression

Regression estimates the relationship between a dependent variable and one or more independent variables. It was used by Patnaik et al. (2004) to predict bus arrival times to downstream stops with data collected by the Automatic Passenger Counting System (APC System) [23].

The regression models require the independent variables to be uncorrelated to each other. It is difficult to provide such a set of variables in the context of bus arrivals predictions. This is because most of the independent variables are correlated (e.g. the number of intermediate bus stops and the number of signalised intersubsections). Therefore, defining a set of uncorrelated independent variables is the main challenge of building regression models.

2.5.3 Artificial Neural Network

ANN models are used to estimate functions that can depend of a large number of inputs by adjusting their parameters through message passing between neuron layers. Building an ANN model involve a training process where dependent variables and prediction results are fed in. The advantage of ANN is that the variables can be correlated, whereas the main disadvantage is that the model training process can take very long (more than 10 hours). Mazloumi (2009) build an artificial neural network with data collected by the Sydney Coordinated Adaptive Traffic System at intermediate signalised intersubsections and schedule adherence to predict bus travel time[24].

2.5.4 Kalman Filters

The Kalman filter, also known as linear quadratic estimation(LQE), is a linear recursive predictive algorithm. It starts with a primary estimate and allows parameters to be tuned with each new measurement, in order to find the optimal estimates of unknown variables [25]. It can respond to dynamic conditions of a modelled process, and has been used for dynamic travel time prediction models. Chen et al. (2005) used Kalman filters to predict arrival time with taking into account the effect of schedule recovery impact [26].

2.5.5 K-Nearest Neighbours Regression

K-Neareast Neighbours (KNN) regression algorithm takes in the k closest training exmaples and produces an output of the property value based of the average of the values of its neighbours. Baker C. M. and Nied A. C. (2013) created models to predict arrival times using KNN, Kernel Regression and seven sets of features[27].

2.5.6 Analytical Approaches

Analytical approaches were usually developed based on specific available data sets or special conditions. For example, Sun et al.(2007) proposed an algorithm that firsly tracks the bus to obtain the distance to each bus stop, and then predicts bus arrival time using the average speed in various temporal and spatial segmentations [28].

2.5.7 Summary on Literature Review

While there are many available methods to estimate bus delays, we have not seen any implementations specificaly for London. To make use of the TfL

open data, we decided to use an analytical approach to predict London bus delays.

Chapter 3

Methodology

3.1 Objectives

We aim to quantify the deviation in bus journey times from the official timetable, so as to improve the prediction of bus travel time downstream from location of last observation in mixed traffic operations. We achieved this by providing 1) a data service API of bus travel time that provides historical, current, and reference bus timetables, and 2) a demonstrative web application to show case the use of the API.

3.2 Bus Journey Times

The bus journey time for a given route depends on many unpredictable external factors. These include weather conditions, passenger flow, temporary lane closures, as well as the time of the bus trip. Predicting bus journey times by discovering and analysing these contributing factors is complicated.

We decided to bypass looking at these factors, and examined the historical and current bus travel times instead. We assumed that for a specific short time frame, the external factors remain largely unchanged. Also, the bus journey times between any two stops on a route is the sum of the travel times between any pair of neighbouring stops. In this case, the bus travel time between a given pair of neighbouring stops is similar to the previous trips performed in the same time frame.

For bus journey time between every pair of neighbouring stop at each hour of the day, we provide estimations for the following:

- **Reference Timtable** How long does TfL says it take?
- **Current Timetable** How long does it currently take?

- **Historical Timetable** How long does it usually take?

Since the reference timetable shows the typical bus journey time, the historical timetable should converge to the reference timetable over time. The current timetable shows the most relevant bus travel time at the observation point, a significant increase in travel time compared to the historical or refernece timetables would indicate a bus delay.

3.2.1 Reference Timetable

We extracted the average bus travel time between every pair of neighbouring stops for every route during every hour of the day for every day of the week from the TfL Journey Planner Bus Timetables. This is discussed in Section 4.3.

3.2.2 Current Timetable

We collected the live bus arrival times for the past 1 hour, and stored the final bus arrival times for each bus at each stop. We then found out the travel time of each bus between every pair of neighbouring stops for the given hour. Next, we calculated the average travel time between each pair of neighbouring bus stops. This serves as a prediction for how long the bus currently takes to travel between two neighbouring stops. See implementation details in Section 4.4.

3.2.3 Historical Timetable

We stored the current timetable generated at each hour, and grouped them by the hour of the day for the same day of the week. We then calculated the average bus travel time between each pair of neighbouring stops for each hour of the day in each day of the week. For example, the average bus travel time between stop A and stop B for 3pm on Wednesday is the average travel time for all the bus trips between these two stops between 2pm to 3pm in the past Wednesdays. More details cound be found in Section 4.5.

3.3 Contributions

We provided the above mentioned three timetables as a data service API (Chapter 5) and designed a demonstrative web application that allows users to check current bus delays from a given bus stop (Chapter 6).

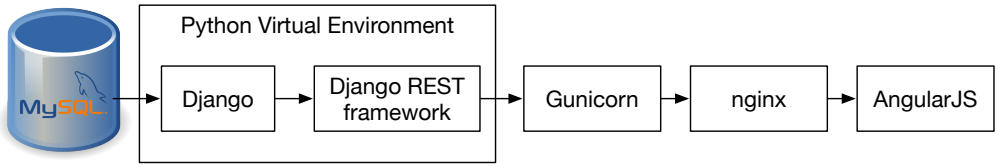


Figure 3.1: Architecture

3.4 Architecture Design

The overall architecture is shown in Figure 3.1. We discuss each component in detail in the following chapters.

Chapter 4

Data Collection and Generation

4.1 Overview

This chapter discusses the backend development environment setup, and the detailed data collection and processing steps to generate the reference, current, and historical timetables.

4.2 Development Environment

4.2.1 Virtual Machine

We set up a Virtual Machine in the Doc's Private Cloud[29] with the specifications shown in Table 4.1. We allocated a large memory for memory intensive Databases operations.

4.2.2 MySQL Databases

We chose MySQL for backend data storage for the following reasons:

- **User Interface** MySQL has convenient database management tools that enable easy data browsing, such as phpMyAdmin[30] and Sequel

Spec	Value
Number of CPU Cores	8
CPU (in MHz)	1000
Memory (in GB)	30

Table 4.1: Virtual Machine Specifications

Pro [31].

- **Scalability** MySQL can handle memory-intensive computations efficiently once configured correctly.

Databases Optimisation

As some database operations involved joining large tables, we optimised the databases with the following settings in file */etc/mysql/my.cnf* to allow MySQL to access more memory in the Virtual Machine [32, 33].

```
[mysqld]
innodb_io_capacity = 2000
innodb_read_io_threads = 64
innodb_thread_concurrency = 0
innodb_write_io_threads = 64
innodb_buffer_pool_size=20G
join_buffer_size=2G
sort_buffer_size=1G
read_buffer_size=1G
read_rnd_buffer_size=1G
max_connection=200
```

4.3 Generating Reference Timetable

table indexing

For each day of the week, there are a predefined number of running the give route. We used the `VehicleJourneys` as a starting point to retrieve the departure time of the actual vehicle from the terminal. We then retrieved the corresponding `JourneyPattern`, and obtained the travel time between each neighbouring stops on the route for the given vehicle journey, to compute the cumulative travel times throughout the route.

The above computation was performed on each XML file to generate the actual arrival time and travel time for each vehicle trip at each stop in the route throughout the day. We grouped the bus travel time by the hour that the arrival time falls in, and find the average bus travel time between every pair of the neighbouring stops for the given hour of the given day of the week. The results of this computation was stored in the `delay_tfl_timetable` table (Table 10.4).

4.3.1 Detailed Steps

We used the `ElementTree` XML API in Python [34] to extract the official bus travel times between stops from the Journey Planner Bus Timetables[22].

Each XML file contains bus schedule information for one route. We carried out the following steps on each XML file:

1. Obtain the route from `Services` → `Service` → `Lines` → `Line` → `LineName`
2. For each `VehicleJourneys` → `VehicleJourney`, extract the followings:
   (a) The departure time from `DepartureTime`
   (b) The days of the week that this journey operates on from `OperatingProfile` → `RegularDayType` → `DaysOfWeek`
   (c) The corresponding journey pattern reference from `JourneyPatternRef`
3. Each `JourneyPatternRef` maps to one Journey Pattern Section. This mapping is stored in `Services` → `Service` → `StandardService` → `JourneyPattern`.

   Retrieve the corresponding Journey Pattern Section Reference as such:
   (a) Each `JourneyPattern` element contains an element id, and a sub-element `JourneyPatternSectionRefs`.
   (b) Map each Journey Pattern id to its corresponding `JourneyPatternSectionRefs` for reference.
   (c) Consult the above mapping to retrieve the Journey Pattern Section Reference for each Journey Pattern Reference found in Step 2(c).
4. Next, obtain the bus travel time between every pair of neighbouring stops in the route from the `JourneyPatternSections`. The detailed steps are as the following:
   (a) For each `JourneyPatternSectionRefs`, find the corresponding `JourneyPatternSections` → `JourneyPatternSection` with the same id.
   (b) Each `JourneyPatternSection` contains multiple `JourneyPatternTimingLink` sub-elements. Each sub-element contains information for one pair of neighbouring bus stops. Retrieve the following information from each sub-element:

i. `From SequenceNumber`
ii. `From` → `StopPointRef`
iii. `To SequenceNumber`
iv. `To` → `StopPointRef`
v. `RunTime`

5. At this point, we have obtained the departure time from the terminal stop for each vehicle journey (Step 2(a)), and the bus travel time between every pair of the neighbouring stops (Step 4(b)v.). We calculated the cumulative bus travel time for each stop, and derived the bus arrival time at each stop for the given vehicle journey.

6. We grouped the bus travel time by the hour that the arrival time falls in, and computed the average bus travel time between every pair of the neighbouring stops for the given hour of the given day of the week.

4.4 Generating Current Timetable

4.4.1 Collecting Bus Arrival Times

Building the Query URL

We collected bus arrival data for analysis from the live bus arrivals API. The base URL used in this project was `http://countdown.api.tfl.gov.uk/interfaces/ura/instant_V1`.

We supplied the following parameters which specify the fields returned by the API.

- *StopID.* This is the alphanumeric identifier of a bus stop. It is also known as stop_code_lbsl.
- *LineName.* This is the route number that is displayed on the front of the bus on any publicity advertising the route.
- *DirectionID.* The direction of the bus.
- *VehicleID.* The unique identifier of the vehicle.
- *TripID.* The identifer of the specific trip that the prediction is for.
- *EstimatedTime.* This is the predicted time of arrival for the vehicle at a specific stop.

- *ExpireTime*. This is the time at which the corresponding prediction is no longer valid and should stop being displayed.

The resulting query URL is `http://countdown.api.tfl.gov.uk/interfaces/ura/instant_V1?ReturnList=StopID,LineName,DirectionID,VehicleID,TripID,EstimatedTime,ExpireTime`.

Storing Arrival Times

The TfL Live Bus Arrival Feed is updated every 30 seconds to give a more accurate predictions of the bus arrival times. We send an HTTP request to the above URL every 30 seconds.

Each data entry in the return result contains an estimated arrival time for each bus journey at a given bus stop. We assume that the actual bus arrival time is the the midpoint between the last estimated arrival time, and the system time when the clear signal ($ExpireTime = 0$) is received.

Since sometimes the clear signal is lost for certain entries, we assumed the actual bus arrival time is the latest recorded estimated arrival time, when there are no more updates 15 minutes after the expire time. This means that we have not received any new updates for the given bus at a given stop 15 minutes after the last estmated arrival time.

As we would like to only store the actual bus arrival times, we keep a local copy of the current query result using the Python `pickle` module[35], and only update the databases when the most arrival time for the given bus at the given stop has expired for more than 15 minutes.

Here are the detailed steps:

1. Load the local arrivals objects if there exists a copy.
2. Pull the new TfL arrivals predictions.
3. Update the loacl arrivals objects with the new predictions.
4. For arrival entries that have expired for more than 15 minutes, remove these entries from the local copy, and store them in the `delay_arrivals` table in the databases.

The above steps were implemented in a Python script. Each run of the steps takes approximately 10 to 15 seconds. We re-run the script 15 seconds after the previous run finishes.

id	route	start_stop	end_stop
18433	30	10002	11469
44878	N19	10002	11469
8653	19	10002	11469

Table 4.2: Sample data in delay_neighbours Table

4.4.2 Generating Bus Sequences and Neighbouring Stops

We imported the bus routes data introduced in Section 2.3.2 into the `delay_bus_sequences` table (Table A.2). Every entry contains information on the route name, route direction, and the sequences of stops in the route. As the sequence information for some routes have gaps, we preprocessed the data by updating the sequence number of the following stops to fill up the gaps. This preprocessing step was done after a few examples verified with the up-to-date TfL routes in the TfL Journey Planner.

In order to find out the average travel time between any pair of neighbouring stops, we needed a list of all the neighbouring stops serving by various routes for reference. We extracted this information from the bus routes data, and stored it in the `delay_neighbours` table (Table A.3). In the sample entries shown in Table 4.2, we can see that there are three different routes serving between stop 10002 and 11469. When calculating the average travel time between these two stops at a given hour, we used all bus trip information for these three routes.

4.4.3 Generating the Current Average Travel Time Between Neighbouring Stops

To generate the current average travel time, we first isolated the arrival times collected in the recent one hour.

We then performed the following steps on the recent arrivals data:

1. For each bus traveling between each pair of the neighbouring bus stop,
   (a) Find out the arrival times for the same bus at the start stop and the end stop of the neighbouring pair.
   (b) Calculate the difference of these two arrival times, and save it as one entry of the travel time.

2. Compute the the average travel time of all bus trips took place between every pair of neighbouring stops, and save it in the current timetable.

3. Save the current day of week, and hour of the day in the timetable for reference.

We ran the above steps every hour to refresh the current timetable. Before every update, we stored the current timetable into a log for generating the historical timetable.

4.5 Generating Historical Timetable

The historical timetable is generated from the current timetables. We processed the current timetable log by computing the average bus travel time by the start stop, end stop, day of the week, and hour of the day. This computation is performed weekly on Sunday midnight to update the historical timetable.

4.6 Summary

The data collection and generation forms is the key stage to produce the reference, current, and historical timetables. After we crafted and tested the scripts and procedures to update the timetables, we proceeded to build the data service API. This is dicussed in the next chapter.

Chapter 5

Delay Data Service

5.1 Overview

To enable structured query of the reference, current, and historical timetables, we created a REST data service API. These endpoints are available within the imperial network for other developers to use.

5.2 Django Framework

We used Django Framework [36] for the backend of this project. It is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. We chose it for the following advantages:

- It stores databases schema as data models [37], and allows for quick database migration. This makes deployment to the production server easy by using the built-in migration command [38].
- It has a compatible REST framework[40], which supports REST API routing[41] and query parameter parsing. Additionally, it offers a built-in user interface that displays the query result with pagination[42].
- Django has a Debug mode that display constructive error messages, make the development much easier.
- It is written in Python, which is our preferred language for its simple and short syntax.

5.3 Deployment Setup

5.3.1 Python Virtual Environment

In order to manage our Python project requirements neatly and make deployment simple, we set up virtual environment[43] [44] to manage the project requirements. This helps to separate the Python packages installed for different projects as well.

5.3.2 Web Server

We chose to use Nginx[46] as the web server and Gunicorn[45] as the Python Web Server Gateway Interface HTTP Server. Nginx was chosen over Apache Web Server as it was more widely used with Django and Gunicorn setup, therefore there was more support articles available online for reference [47].

5.4 API Endpoints

We built the backend using Django framework and Django REST framework, and exposed the following 3 endpoints for users to access data. The base URL for all endpoints is `http://delay.doc.ic.ac.uk:5000`.

5.4.1 Historical & Current Timetables

This API endpoint provides information on how long the bus on a given route at a given hour of the given day will take to reach the downstream stops from a given starting stop.

Query URL & Parameters

The base URL to access the historical and current timetables is `http://delay.doc.ic.ac.uk:5000/predictions/?`. It requires the following parameters:

- **day** Day of the week
- **hour** Hour of the day [0 - 23]
- **route** Route
- **run** Route direction

- **naptan_atco** The NaPTAN code for the starting downstream stops in the route

For example, the URL to retrieve historical and current bus travel time predictions for route 360 for downstream stops starting with NaPTAN code 490016263E at hour 15 on Tuesday is `http://delay.doc.ic.ac.uk:5000/predictions/?day=Tuesday&hour=15&route=360&run=2&naptan_atco=490016263E`.

Result Format

The result of the API query is a JSON list of bus stops, following the given route sequences in the given route direction, starting with the given stop. Each bus stop entry contains information on the bus stop, as well as the bus travel time in seconds. The following four fields are the most important ones:

- `average_travel_time` Historical average bus travel time from the previous stop to the current stop.
- `cumulative_travel_time` Historical average bus travel time from the given starting stop to the current stop.
- `average_travel_time` Current average bus travel time from the previous stop to the current stop.
- `cumulative_travel_time` Current average bus travel time from the given starting stop to the current stop.

Backend Computation

For each request, the Django backend performs the following computation steps to produce the response:

- Search the database for the bus sequence with the given route and run.
- If a NaPTAN code is given, filter out the stops after the given stop.
- For each pair of the neighbouring stops, search the databases for the corresponding historical travel time, and the current travel time values for the give hour of the day, and day of the week.
- Compute the cumulative travel time from the starting stop to each downstream stop.
- Return the result in JSON format.

5.4.2 Reference Timetable

This API endpoint provides information on the TfL official bus travel time for the buses on a given route at a given hour of the given day of week to reach downstream stops from a given starting stop.

Query URL & Parameters

The base URL to access the reference timetable is `http://delay.doc.ic.ac.uk:5000/tfl_timetable/?`.The required parameters are the same for the predictions endpoint, described in Section 5.4.1. For example, the URL to retrieve reference timetable entries for route 360 for downstream stops starting with NaPTAN code 490016263E at hour 15 on Tuesday is `http://delay.doc.ic.ac.uk:5000/tfl_timetable/?day=Tuesday&hour=15&route=360&run=2&naptan_atco=490016263E`.

Result Format

The result of this API query is similar to that of the prediction endpoint. It consists of a JSON list of bus stops, ordered by the given route sequences. Each bus stop entry contains information on the bus stop. Additionally, the `average_travel_time` field indicates the official bus travel time from the previous stop to the current stop in seconds, whereas the `cumulative_travel_time` field shows the official bus travel time from the givien starting stop to the current stop in seconds.

Backend Computation

The backend processes each request as such:

- Search the TfL timetable in databases for the entries for the given route, run and arrival hour.
- Filter the entries for the given day of the week, and the future stops of the given stop.
- Compute the cumulative travel time for the bus to reach each downstream stop from the given stop.
- Return the result in JSON format.

5.4.3 Arrival

This endpoint serves as a wrapper for the TfL Live Bus API endpoint. Given a set of GPS coordinates and the radius of circle, this API endpoint returns the arrival times of buses arriving at stops within the circle.

Query URL & Parameters

The base URL for this endpoint is `http://delay.doc.ic.ac.uk:5000/arrivals/?`. The required parameters include the latitude and the longitude of the GPS coordinates, and the radius in meters. An example query URL would be `http://delay.doc.ic.ac.uk:5000/arrivals/?latitude=51.495171603309615&longitude=-0.1883983612060547&radius=100`.

Result Format

The query result is a list of bus stops within the circle. Each bus stop entry has basic information about the stop, and a list of the buses arriving at the stop in the next 30 minutes, grouped by the bus routes. Within each route, the `estimatedTimeInSeconds` shows the arrival times of the buses arriving at the given stop. See Figure 5.1 for example response.

5.5 Summary of Data Service API

These three API endpoints provide easy access to the reference, current, and historical timetables produced. Other developers can use the data to produce bus delay predictions or build software applications for London bus passengers. We evaluated the accuracy of the timetables and the performance of the API in Chapter 8.

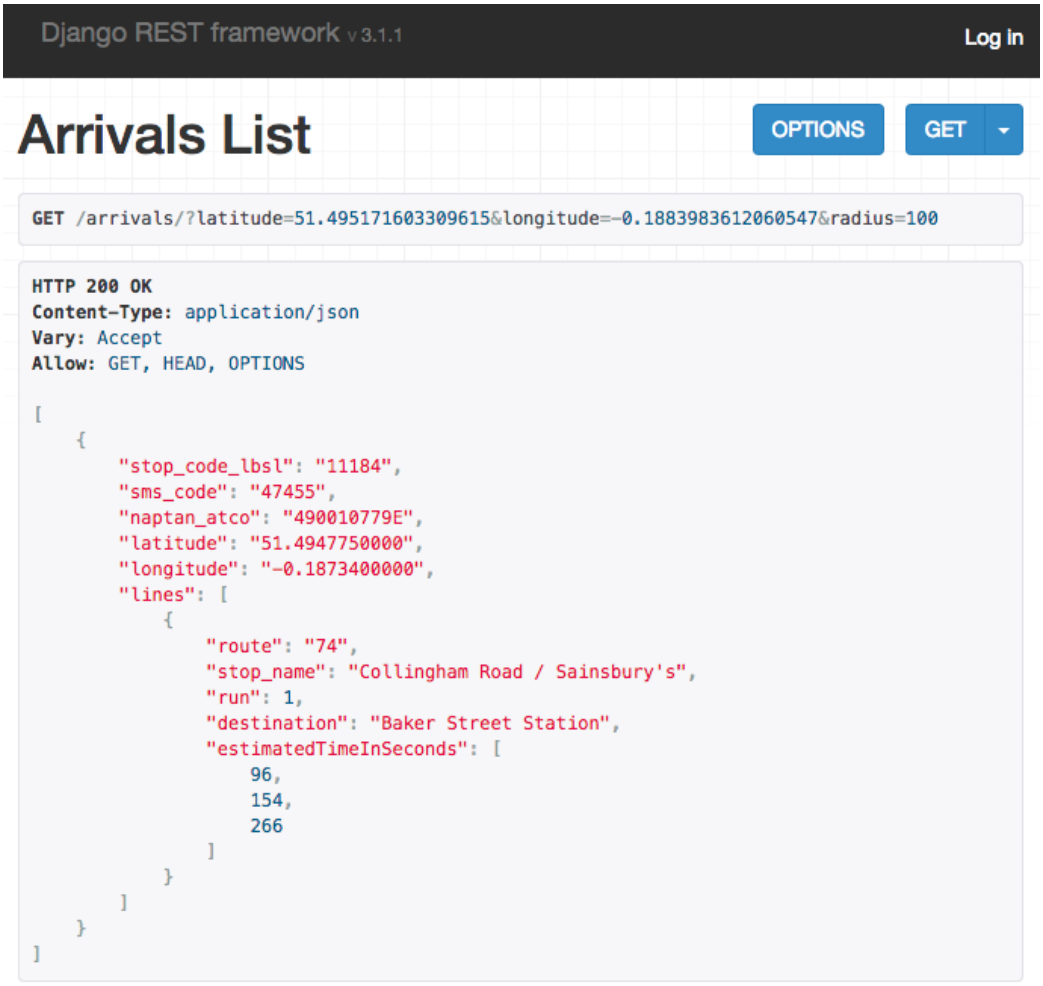


Figure 5.1: Sample Response for Arrivals API

Chapter 6

WebApp for Active Delay Warning

We designed a web application to demonstrate the potential use of our API data service. It allows users to search for nearby bus stops, view the available routes and live bus arrival times. For each route, the user can view the historical, current and reference bus travel time to reach each downstream bus stop.

6.1 Implementation

6.1.1 AugularJS Framework

We chose to use the AngularJS Framework [48] with UI Bootstrap [49] to build the frontend of the application. This was because AngularJS employs a clear Model-View-Controller structure, and has a wide range of packages to build extensions with. We used `Google Maps API` and `ui-router` extensively.

add references

6.1.2 Development & Deployment Pipeline

We used Yeoman [50], a web application scaffolding tool, to organise and manage scripts and files. We also used Grunt [51], a Javascript task runner to manage the build and delopment process.

For deployment, we created a Grunt task to run tests, minify the javascript files, and copy the minified version to the production server.

6.2 Frontend Walkthrough

The web application has a simple user interface. It divides the screen space into 2 parts, the top half consists of a map area, and the bottom half an information panel. We deployed the frontend web to `http://delay.doc.ic.ac.uk/`.

6.2.1 Landing Page

On load, the app will attempt to obtain the current location of the device, and center the map on it. If the current location is not available, then the default map centered on London will be loaded (Figure 6.1).

6.2.2 Nearby Bus Stops Arrival

A click on the map will place a grey location marker and make a request to the `arrivals` endpoint to search for the nearby bus stops within a given radius. The default radius is 100 meters. Users can use the slider to change the radius of the search area, ranging from 50 to 500 meters.

A list of the nearby bus stops and the routes available at each stop will be shown in the information panel, with the TfL bus arrivals displayed at the side showing the arrival times for the incoming buses(Figure 6.2). For each bus stop, a red marker will be placed in the map at the stop location with the stop code displayed under the marker.

6.2.3 Bus Delay Predictions

For each available route at a nearby bus stop, users could check the historical, current, and reference travel time to each downstream stops by clicking the red button to select a specific route for details. The app will make requests to the `Historical & Current` and `Reference` endpoints with the current day of the week, hour of the day, stop code, and direction of the bus route as parameters.

After the endpoints return results, a list of the downstream stops with the corresponding historical, current, and reference travel time from the current stop will be displayed in the informantion panel.

We placed an additional column of the bus travel times as a linear combination of the reference, historical, and current bus travel times. The `pre - ref` column indicates the difference between the combine timetable and the reference timetable travel times. A positive delFigure 6.3 shows bus travel time bus 70 starting from Kensington Road.

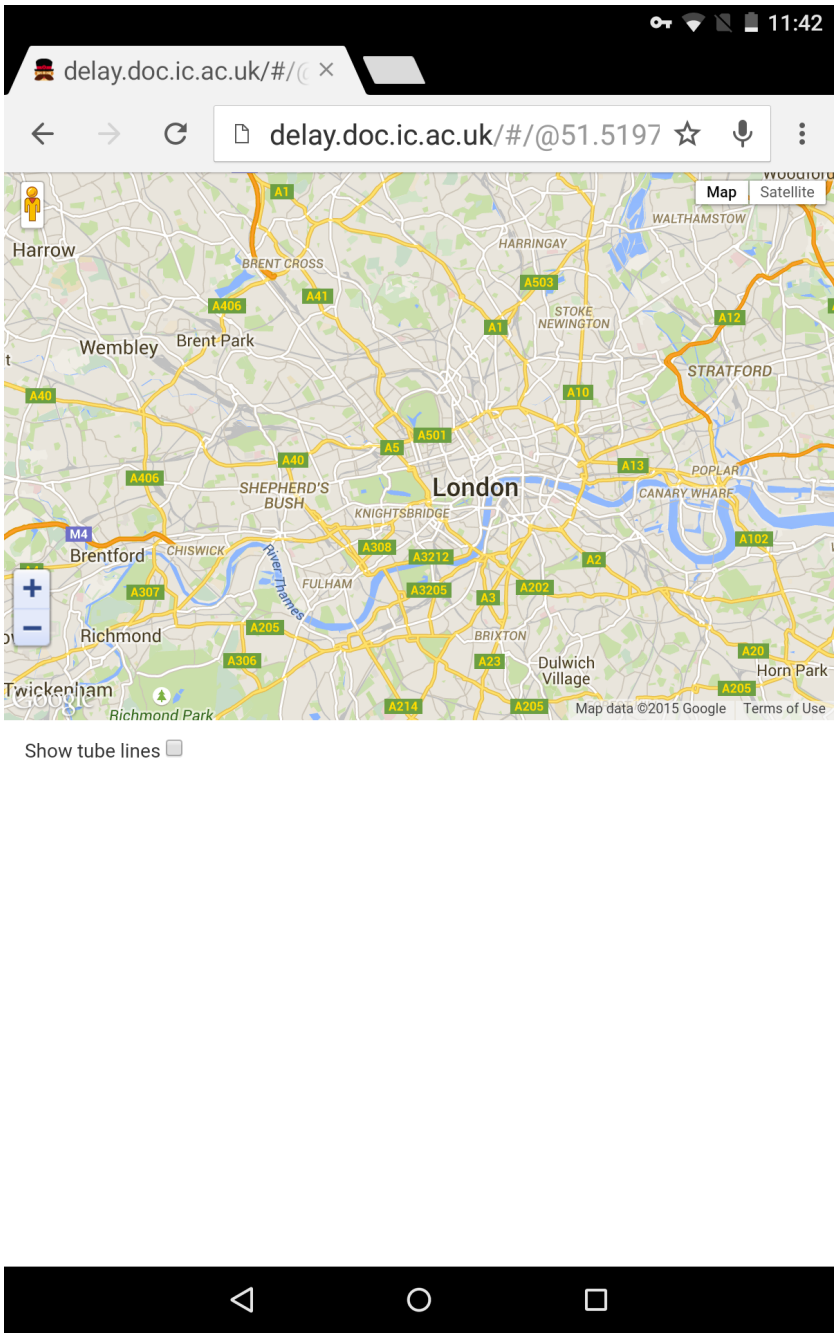


Figure 6.1: Web Application Langding Page

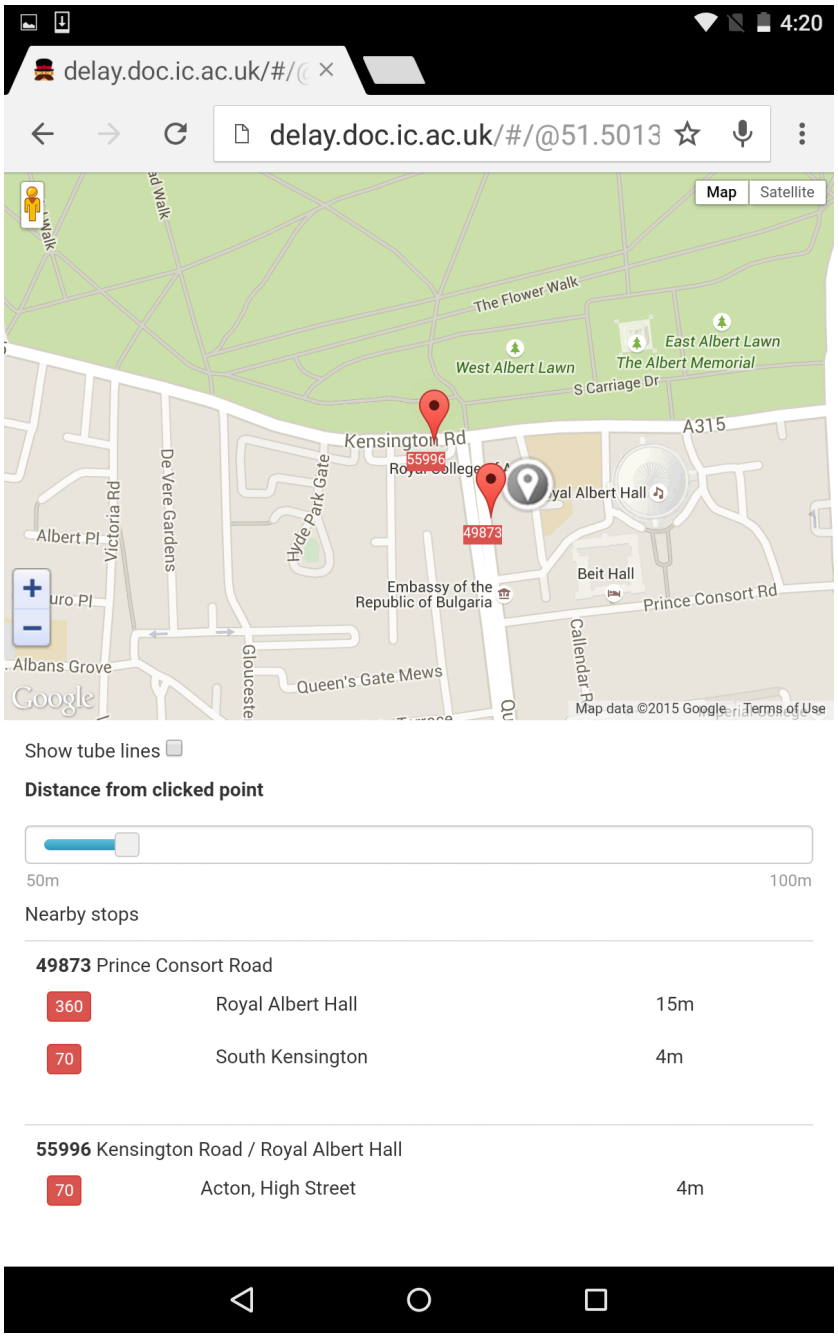
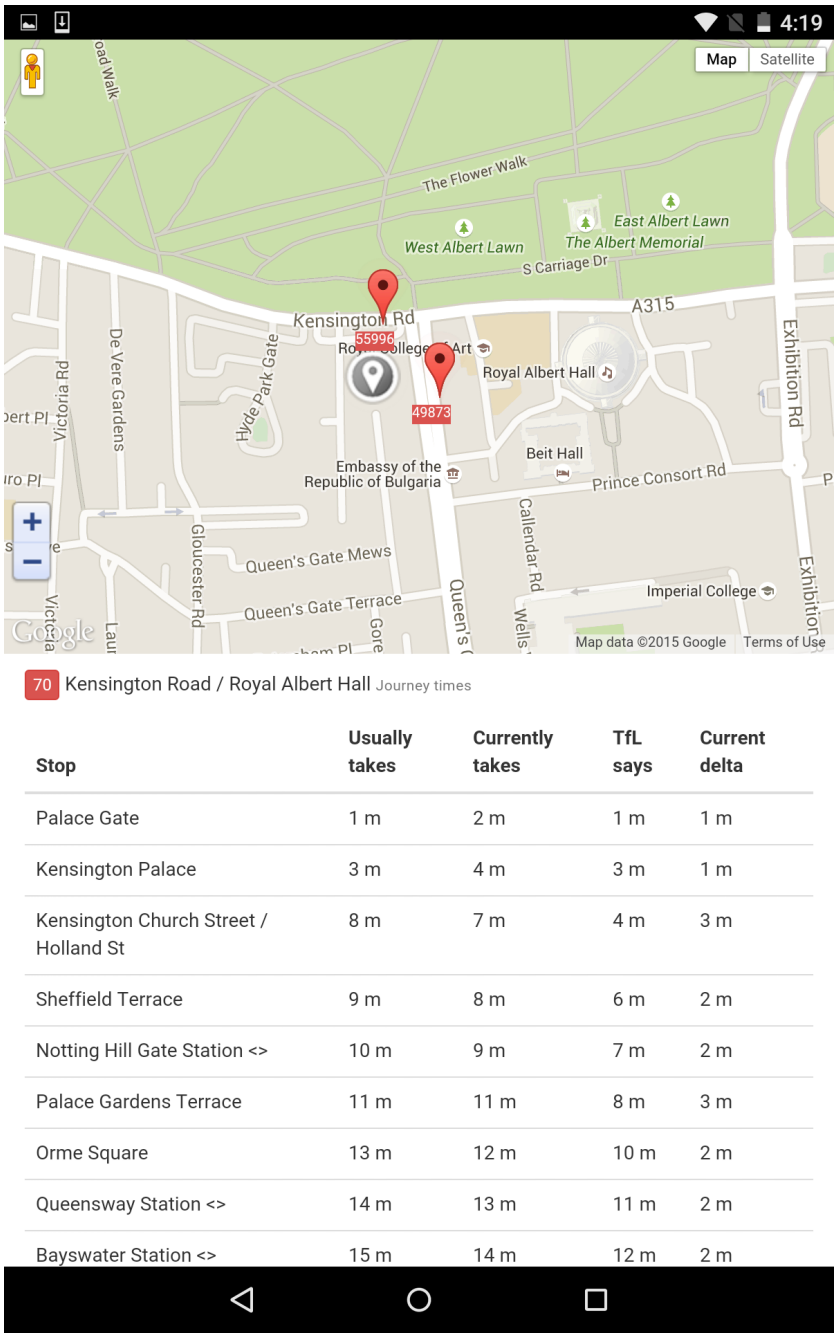


Figure 6.2: Web Application Nearby Bus Stops



Stop	Usually takes	Currently takes	TfL says	Current delta
Palace Gate	1 m	2 m	1 m	1 m
Kensington Palace	3 m	4 m	3 m	1 m
Kensington Church Street / Holland St	8 m	7 m	4 m	3 m
Sheffield Terrace	9 m	8 m	6 m	2 m
Notting Hill Gate Station <>	10 m	9 m	7 m	2 m
Palace Gardens Terrace	11 m	11 m	8 m	3 m
Orme Square	13 m	12 m	10 m	2 m
Queensway Station <>	14 m	13 m	11 m	2 m
Bayswater Station <>	15 m	14 m	12 m	2 m

Figure 6.3: Web Application Timetables for a Route

The coefficients used in the linear combination are derived from regression. This is discussed in Section 8.3.

6.3 Summary

The web application shows how the delay prediction data service could be used to inform users of potential delays in buses arriving at any downstream stops on a selected route. This will help bus passengers to make informed decisions when choosing travel mode and bus routes to save time.

Chapter 7

Technical Details

This chapter discusses the tools we used for process control, scheduled task management, and general project management.

7.1 Process Control System - Supervisor

We needed to monitor two daemon tasks. The first task was the script to send requests to TfL Live Bus Arrivals API and save the arrivals to the Databases (Section 4.4.1). The second task was to keep Gunicorn running (Section 5.3.2).

We used Supervisor[52] to monitor these two tasks. It watches the two processes and restart them if they fail, and ensure they start on system boot.

Suervisor was chosen as it matches our requirement exactly, as it is meant to be used to control processes related to a project or a customer, and is meant to start like any other program at boot time.

7.2 Scheduled Tasks Management - Jenkins

We used Jenkins[55] to managed our predictions timetable update schedule. Figure 7.1 shows some of our schedule tasks. Jenkins was chosen as it provides a dashboard that gives a quick glance of the build state and saves build histories for future reference.

7.2.1 Current Timetable Update

The current timetable is re-generated every hour. We set up a Jenkins job to re-run the relevant current timetable update SQL script every hour, and

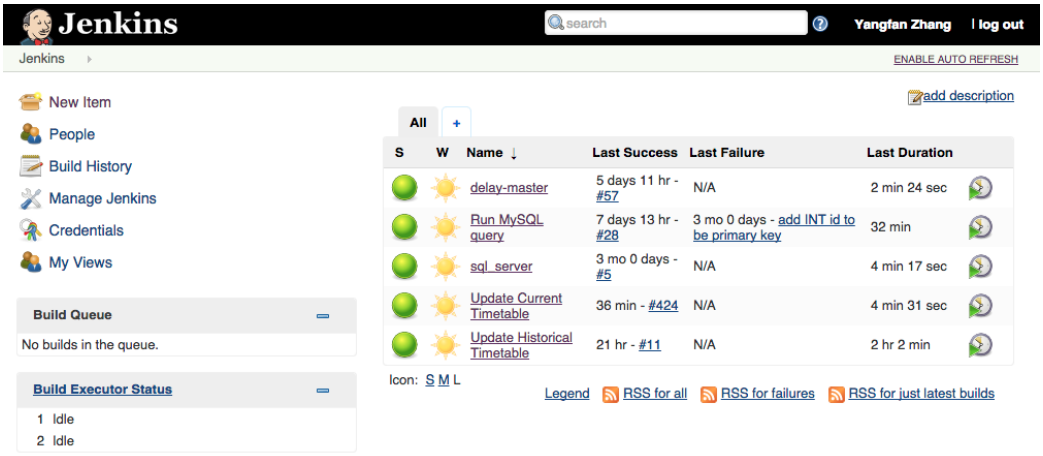


Figure 7.1: Jenkins Dashboard

save the previous timetable into a log for updating the historical timetable at a later time. This job takes less than 10 minutes on average.

7.2.2 Historical Timetable Update

Similarly, we set up a Jenkins job to update the Historical Timetable daily at 3.30am when the server is less busy.

7.2.3 Arrivals Daily Backup

The arrivals table grows rapidly by approximately 150 thousand rows per hour. In order to allow fast access to the arrivals within the past one hour to construct the current timetable, we decided to save the arrivals for the current day in a new table, while keeping the past arrivals entries in an archive table. This required us to backup the arrivals entries daily to the arive.

Again, we created a Jenkins job to run the back up script daily. In order not to affect the historical timetable update, we set this job as a downstream task of the historical timetable update.

7.3 Software Project Management - Trello

We used Trello [56] for project management. It was useful to keep track of the tasks backlog, the tasks at hand, and the tasks to be further tested (Figure 7.2). We chose Trello because it is free, and has a simple user interface.

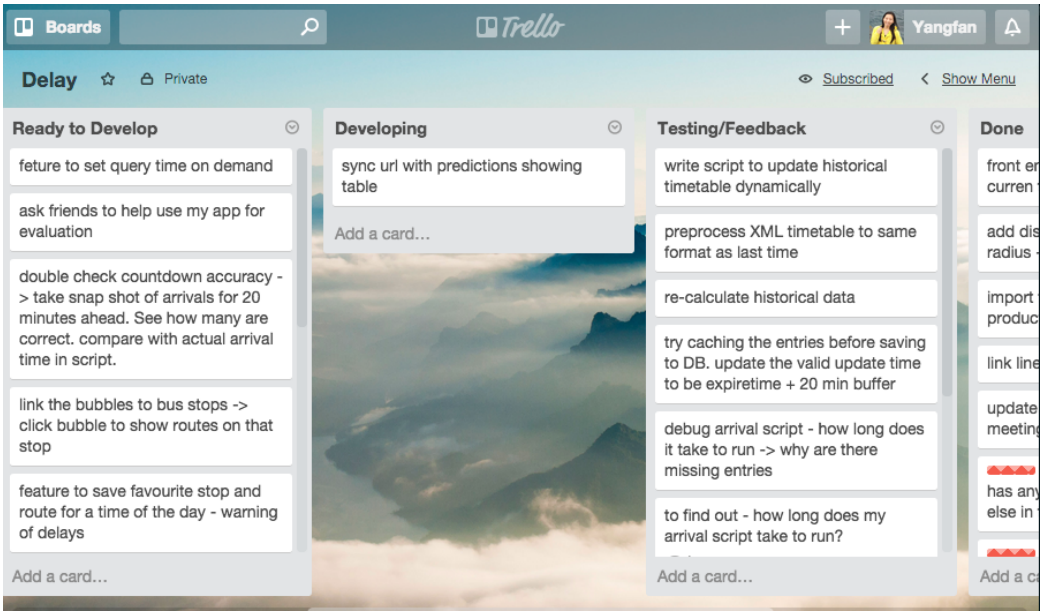


Figure 7.2: Project Trello Board

7.4 Summary

The appropriate use of the above tools makes it easy to keep track of daemon processes and regularly scheduled tasks, and manage the pace of development. This makes development and deployment much smoother.

Chapter 8

Evaluation

This chapter documented the evaluations of the API endpoints. We first quantified the accuracy of the TfL Live Bus Arrivals API. We then tested the correctness and performance of our data service API endpoints. Lastly, we evaluated the accuracy of the generated bus timetables.

8.1 Accuracy of TfL Live Bus Arrivals API

The Tfl Live Bus Arrivals API updates the bus arrival times every 30 seconds. This means that the accuracy of the bus arrival predictions for the current stop will improve gradually as the bus approaches the stop. This is because any error in earlier predictions would be incrementally corrected at every update as the location of the bus gets refreshed.

8.1.1 Test for Accuracy

We evaluated the accuracy of these predictions by the following steps:

- We took a snapshot of all the bus arrivals predictions for the next 30 minutes at a given recorded time (2015-06-04 11:24:37).
- We grouped these prediciton entries by the difference between the predicted arrival time and the recorded time at a 5-minute interval. For example, the predictions for the buses arriving in the next 5 minutes, next 5 to 10 minutes, and next 10 to 15 minutes, etc. We added an additional group for the next 0 - 3 minutes for reference.
- After 3 hours, we compared each arrival prediction entry in the snapshot table against the actual arrival data we stored for generating the

Predictions for	Average Delta (Seconds)
next 0 - 3 mintues	-34.8112
next 0 - 5 mintues	-45.4951
next 5 - 10 mintues	-97.2584
next 10 - 15 mintues	-134.3606
next 15 - 20 mintues	-169.5990
next 20 - 25 mintues	-174.7177
next 25 - 30 minutes	-166.1368

Table 8.1: Live Bus Arrivals API Accuracy - A negative delta indicates the bus came later than the predicted time

current timetable. The difference between the predicted and the actual arrival time gives an indication of the prediction accuracy. We stored this difference as the delta value. A negative delta indicates the bus came later than the predicted time.

- We calcuated the average delta value for each group, and created Table 8.1.

8.1.2 Test Result

Table 8.1 shows that buses usually came later than the predictions provided by the Live Bus Arrivals API. For the buses that were predicted to arrive in the next 5 minutes, they usually came less than one minute later than the predicted arrival time. We took this finding into consideration when testing the accuracy of our current and historical timetable by only choosing buses that are due within the next 5 minutes as tracking targets.

8.2 Correctness and Performance of the API

8.2.1 Tool

We used *siege*[53] to conduct the load test of our API endpoints. Our aim was to test the number requests the server could handle reliably at one time, and the response time it took.

Siege takes in the number of users, the delay time between each page load, the test running time, and the list of URLs to send requests to as parameters. It then simulates the user behaviours to fire requests to the list

of given URLs. At the end of the test, *Siege* generates a report for the test, including metrics such as the transaction rate, and the response time of the target URLs.

8.2.2 Preparation

We generated a list of API URLs with random parameters for testing. We then ran the siege tests by fixing the test running time to be 1 minute, with 1 second of delay between each page load.

8.2.3 Test for Correctness

We first tested for the correctness of our API endpoints. This involves firing requests with random day of the week and hour of day, route, run, and starting bus stop code to check whether the server can return a response correctly.

At this stage, we found some specific URLs that resulted in a 500 error code, and debugged the backend code to return correct results. For example, this happened when we requested for a reference travel time for a bus route at an hour out of its operating time. As there was no corresponding entry in the databases, we fixed the code by returning an empty list by default.

After we received 200 status code for all requests in a few test runs consistently, we proceeded to perform the load tests by fixing the day of the week and hour of the day to be the current values when the test was run.

8.2.4 Load Test for Performance

We carried out the siege test with various number of concurrent users ranging from 50 to 500, on a list of URLs with randomly selected parameters. We collected the test results and plotted the figures for availability, response time, and transaction rates [54]. The metrics are defined as:

- `Response time` is the average time it took to respond to each simulated users requests.
- `Transaction rate` is the average number of transactions the server was able to handle per second, in a nutshell: transactions divided by elapsed time.
- `Successful transactions` is the number of times the server returned a code less than 400. Accordingly, redirects are considered successful transactions.

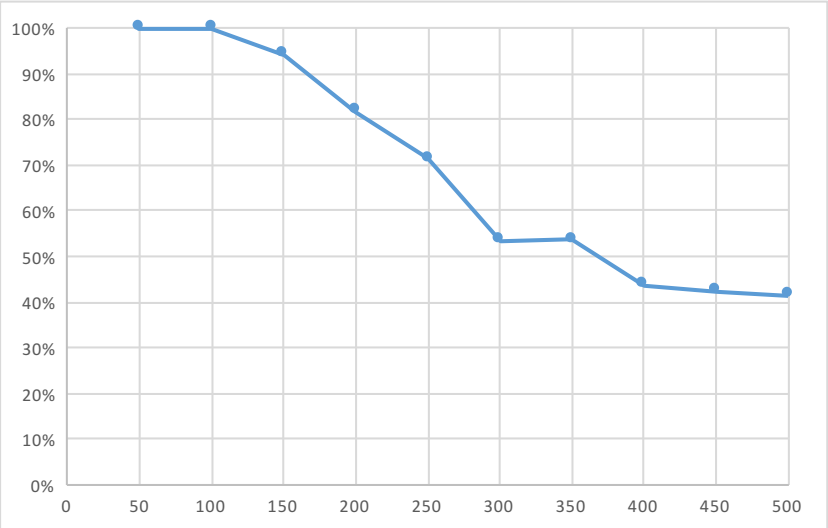


Figure 8.1: Historical & Current Timetables API Availability against no. of simultaneous users

- `Avalability` is the percentage of successful transactions over the total number of server hits.

Historical & Current Timetables API Performance

Figure 8.1 shows that the `availability` drops below 100% when there were more than 100 users accessing the API simultaneously. We used this as a benchmark for the number of uses the current server can reliably handle. Figure 8.2 shows that the average `response time` for 100 users is approximately 15 seconds, and increases as the number of users increases. Figure 8.3 shows that the `transaction rate` for 100 users is about 5.5 transactions per second.

The slow response time was probably due to the number of databases query required to form a response. One databases query was needed for every pair of neighbouring stops in the downstream stop sequence to retrieve the timetable entry. This could potentially be reduced by

how to reduce this?

Reference Timetable API Performance

Figure 8.4 shows that the `availability` drops below 100% when there were more than 150 users sending requests to the Reference Timetable API endpoint concurrently. The average `response time` for 150 users is about 4.5

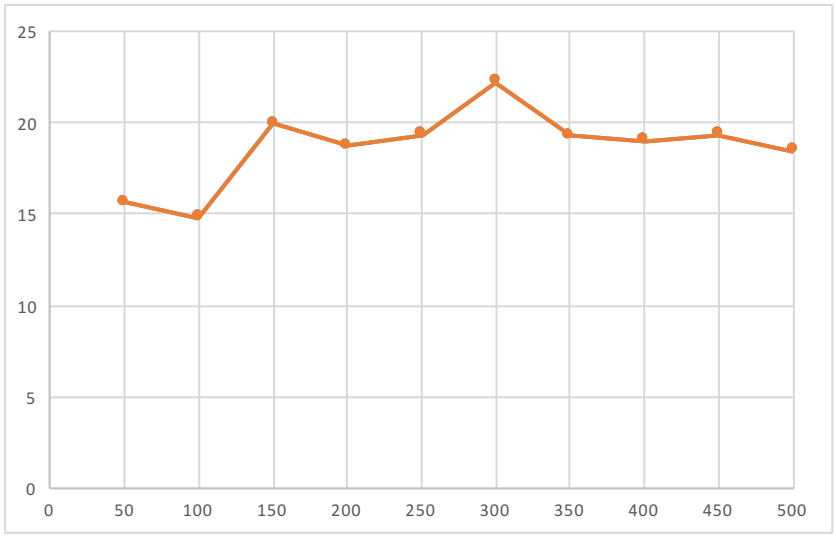


Figure 8.2: Historical & Current Timetables API Response Time (seconds) against no. of simultaneous users

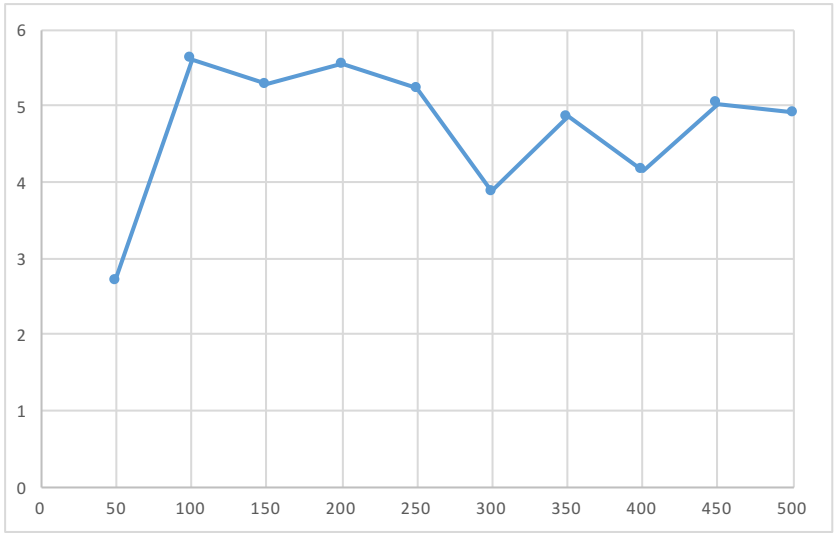


Figure 8.3: Historical & Current Timetables API Transaction Rate against no. of simultaneous users

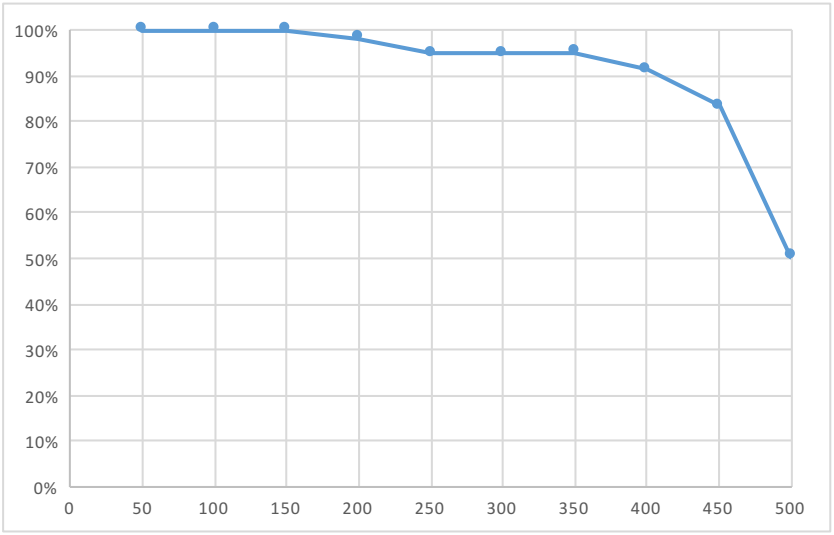


Figure 8.4: Reference Timetables API Availability against no. of simultaneous users

seconds (Figure 8.5), and the `transaction rate` is approximately 27 transactions per second (Figure 8.6). The performance of this endpoint is better as every request involves only one databases query.

8.3 Accuracy of Predictions

To evaluate the accuracy of the bus journey time predictions, we ran tests to compare the actual bus travel times with the historical, current, and reference timetables respectively. We achieved this by first saving the predicted bus journey times from the two endpoints for a given vehicle. We then searched the databases for the actual arrival time records for the vehicle at each downstream stops. For each test case, we ran through the steps below:

1. **Saving predicted bus journey times**:
   (a) Fix day of the week and hour of the day to be the current value when the test was run.
   (b) Generate random parameters for the route, bus direction, current starting stop on the route.
   (c) Generate the URL and send request to obtain the next vehicle arrival time for the given route at the given stop from the TfL Live Bus Arrivals API. If the next vehicle is due in more than 5

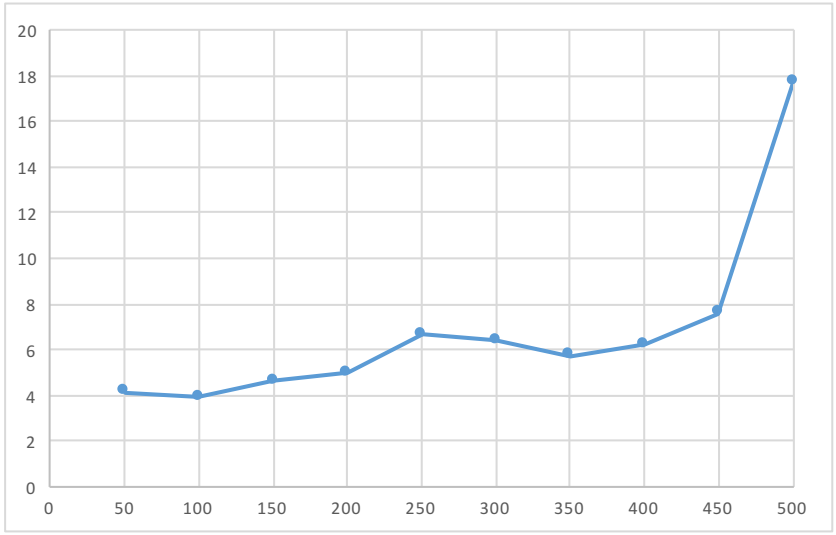


Figure 8.5: Reference Timetables API Response Time (seconds) against no. of simultaneous users

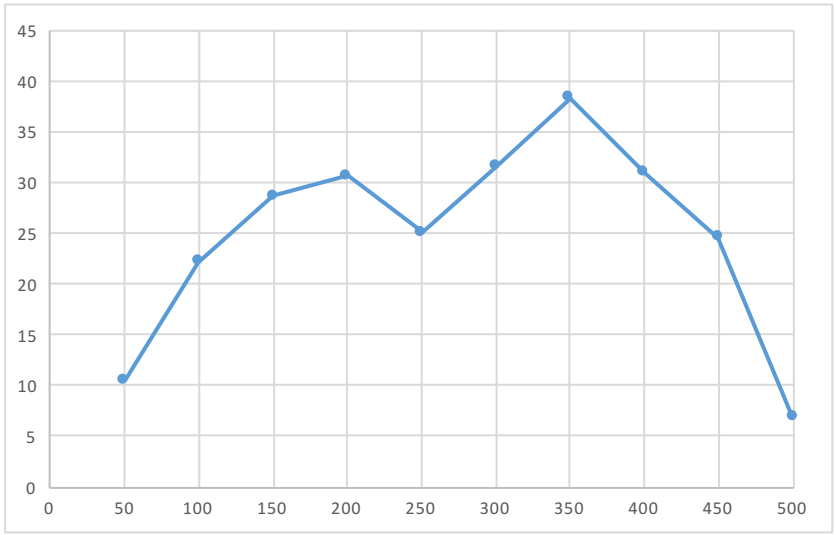


Figure 8.6: Reference Timetables API Transaction Rate against no. of simultaneous users

minutes later, repeat steps 1(a) and 1(b) until there is a case with a next vehicle due within 5 minutes.

(d) Generate the historical & current, and reference timetable URLs with the same set of parameters.

(e) Send requests to the above URLs to obtain a snapshot of the reference, historical and current bus travel timetables for the given route at the given hour of the given day.

(f) Save the responses for comparison with the actual arrivals data later.

2. Wait for a few hours to allow the target vehicle to travel to the terminal. The arrival times for the target vehicle at each downstream stop should have been recorded in the database.

3. **Comparing predictions to actual travel times**:

(a) For each target vehicle, retrieve its actual arrival times at each downstream stop from the given starting stop.

(b) Compute the actual bus travel time in seconds between the given starting stop and each downstream stop.

(c) Compare the actual bus travel times with the reference, historical, and current bus travel times for each pair of the starting stop and ending stop respectively.

We collected 154 test cases throughout the day on 12 June 2015. Each test cases consists of a list of travel times from the starting stop to each downstream stop. There were 3362 pair of stops in total. We ploted the probability density graph to compare the reference, historical and current bus travel times with the actual bus travel times.

insert graph

We found that the individual predictions do not give satisfactory accuracy.

Chapter 9

Future Work

9.1 API Performance

Currently, only one server is used to collect the data, process the data to produce timetables, and handle API queries. We could improve the performance of the API endpoints by using at least two servers, with one in charge of data collection and processing, and the other to handle live user queries.

9.2 Prediction Accuracy

9.3 WebApp Extensions

We could inclue features to alert users of bus delays actively for extension. The application should track the device location, and alert users of any delays occurred on the frequently travelled routes.

The user interface could be furture improve the make the experience smoother.

Chapter 10

Conclusion

In this project, we built a data service to provide the average bus travel time for each downstream stop for a given route on a given day of the week and hour of the day. This prediction data API can support about 100 simultaneous users.

We also designed a demonstrative web application to showcase the potential use of the data service.

Appendices

Appendix A

Database Table Schema

Column Name	Type	Default
id(Primary)	int(11)	Auto Increment
stop_code_lbsl	varchar(64)	
route	varchar(64)	
vehicle_id	varchar(64)	
trip_id	varchar(64)	
arrival_date	date	
arrival_time	timestamp	NULL
expire_time	timestamp	NULL
recorded_time	timestamp	Current Timestamp

Table A.1: delay_arrivals Table Schema

Column Name	Type	Comments
id(Primary Key)	int(11)	Auto Increment
route	varchar(64)	The route name
run	int(11)	The route direction
sequence	int(11)	The sequence of the bus stop in the route
stop_code_lbsl	varchar(64)	The internal bus stop identifier
bus_stop_code	varcher(64)	The public code for the bus stop
naptan_atco	varchar(64)	The national identifier of the bus stop
stop_name	varchar(64)	The name of the bus stop

Table A.2: delay_bus_sequences Table Schema

Column Name	Type	Comments
id(Primary Key)	int(11)	Auto Increment
route	varchar(64)	The bus route
start_stop	varchar(64)	The stop_code_lbsl for the start stop
end_stop	varcher(64)	The stop_code_lbsl for the end stop

Table A.3: delay_neighbours Table Schema

Column Name	Type	Comments
id(Primary Key)	int(11)	Auto Increment
route	varchar(64)	The bus route
day	varchar(32)	The day of week for the vehicle journey
run	int(11)	The route direction
sequence	int (11)	The sequence of the bus stop in the route
stop_name	varchar(64)	The name of the bus stop
naptan_atco	varchar(64)	The national identifier of the bus stop
average_travel_time	datetime(6)	The average travel time in seconds for the bus trips from the previous stop in the route to the current stop during the given arrival hour
cumulative_travel_time	int(11)	The average travel time in seconds from the terminal to the current stop for the bus trips arrived at the current stop in the given arrival hour

Table 10.4: delay_tfl_timetable Table Schema

Acronyms

API Application Program Interface. 2, 4

CSV Comma Separated Values. 4

TfL Transport for London. 1, 2, 4

XML Extensible Markup Language. 4

Glossary

TransXChange the UK nationwide standard for exchanging bus schedules and related data. 4, 5

References

[1] Transport for London Annual Report and Statement of Accounts 2013/14, http://tfl.gov.uk/cdn/static/cms/documents/annual-report-2013-14.pdf [visited on 27/01/2015]

[2] Transport for London Live Bus Arrivals, http://www.tfl.gov.uk/modes/buses/live-bus-arrivals [visited on 27/01/2015]

[3] Google Maps, https://www.google.co.uk/maps [visited on 30/01/2015]

[4] CityMapper London, https://citymapper.com/london [visited on 30/01/2015]

[5] TfL Plan A Journey, http://tfl.gov.uk/plan-a-journey/ [visited on 30/01/2015]

[6] Transport for London Buses Network Performance Second Quarter 2014/15, http://tfl.gov.uk/cdn/static/cms/documents/network-performance-latest-quarter.pdf [visited on 27/01/2015]

[7] Transport for London Buses Status Updates, http://www.tfl.gov.uk/bus/status/ [visited on 22/05/2015]

[8] TfL Bus Alerts Twitter, https://twitter.com/TfLBusAlerts/status/601795342049398784 [visited on 22/05/2015]

[9] Travel in London, Supplementary Report: London Travel Demand Survey (LTDS), http://www.tfl.gov.uk/cdn/static/cms/documents/london-travel-demand-survey.pdf [visited on 28/01/2015]

[10] Transport for London Bus Stops Locations and Routes Open Data, https://www.tfl.gov.uk/info-for/open-data-users/our-feeds [visited on 28/01/2015]

[11] APPSI Glossary, http://www.nationalarchives.gov.uk/appsi/appsi-glossary-a-z.htm

[12] p26. What is the Value of Open Data?, urlhttps://www.nationalarchives.gov.uk/documents/meetings/20140128-appsi-what-is-the-value-of-open-data.pdf

[13] Transport for London Open Data, https://www.tfl.gov.uk/info-for/open-data-users/our-open-data [visited on 28/01/2015]

[14] Quora: How do the electronic bus times countdown screens work at London bus stops? Answer by Dave Arquati, Transport planner in London for 6 years, http://qr.ae/7XnYAK [visited on 29/05/2015]

[15] Transport for London Live Bus & River Bus Arrivals API Interface Documentation, https://www.tfl.gov.uk/cdn/static/cms/documents/tfl-live-bus-river-bus-arrivals-api-documentation-v16.pdf [visited on 28/01/2015]

[16] Transport for London Buses Performance Data, https://www.tfl.gov.uk/corporate/publications-and-reports/buses-performance-data#on-this-page-5 [visited on 28/01/2015]

[17] TransXChange, https://www.gov.uk/government/collections/transxchange [visited on 13/05/2015]

[18] XML Schema Language, http://www.w3.org/TR/xmlschema-2/ [visited on 13/05/2015]

[19] TransXChange Downloads & Schema, https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/391783/Downloads_Schema-2015-01-05_-_Updated.pdf [visited on 13/05/2015]

[20] TransXChange Schema 2.1 xsd, http://www.transxchange.org.uk/schema/2.1/TransXChange_general.xsd [visited on 13/05/2015]

[21] TransXChange Schema Guide 2.1 & 2.2a, http://81.17.70.199/transxchange/schema/2.1/guide/TransXChangeSchemaGuide-2.1-v-45.pdf [visited on 13/05/2015]

[22] Transport for London Open Data Feeds, https://www.tfl.gov.uk/info-for/open-data-users/our-open-data [visited on 13/05/2015]

[23] Patnaik J., Chien S. and Bladikas A., (2004). Estimation of bus arrival times using APC data, *Journal of Public Transportation*, Vol. 7, No. 1, p.p. 1-20, http://nctr.usf.edu/jpt/pdf/JPT%207-1%20Chien.pdf [visited on 29/05/2015].

[24] Mazloumi E., Currie G., Rose G. and Sarvi M., (2009), USING SCATS DATA TO PREDICT BUS TRAVEL TIME, http://atrf.info/papers/2009/2009_Mazloumi_Currie_Rose_Sarvi.pdf [visited on 29/05/2015]

[25] Kalman R. E. (1960). A new approach to linear filtering and prediction problems, *Transactions of the ASME-Journal of Basic Engineering*, Vol 82D, p.p. 35-45.

[26] Chen M., Liu X., and Xia J., (2005). Dynamic prediction method with schedule recovery impact for bus arrival time, *Transportation Research Record*, 1923, pp. 208 217.

[27] Baker C. M. and Nied A. C. (2013). Predicting Bus Arrivals Using One Bus Away Real-Time Data, http://homes.cs.washington.edu/~anied/papers/AConradNied_OneBusAway_Writeup_20131209.pdf [visited on 25/05/2015]

[28] Sun D., Luo H., Fu L., Liu W., Liao X., and Zhao M., (2007). Predicting bus arrival time on the basis of global positioning system data, *Transportation Research Record*, No. 2034, p.p. 62-72.

[29] DoC's Private IAAS Cloud Service, http://www.doc.ic.ac.uk/csg/services/cloud [visited on 02/06/2015]

[30] phpMyAdmin, http://www.phpmyadmin.net/home_page/index.php [visited on 28/05/2015]

[31] Sequel Pro, http://www.sequelpro.com/ [visited on 28/05/2015]

[32] How large should be mysql innodb_buffer_pool_size?, http://dba.stackexchange.com/questions/27328/how-large-should-be-mysql-innodb-buffer-pool-size [visited on 02/06/2015]

[33] The InnoDB Buffer Pool, https://dev.mysql.com/doc/refman/5.6/en/innodb-buffer-pool.html [visited on 02/06/2015]

[34] The ElementTree XML API, url-https://docs.python.org/2/library/xml.etree.elementtree.html [visited on 30/05/2015]

[35] pickle Python object serialization, url-https://docs.python.org/3.1/library/pickle.html [visited on 01/06/2015]

[36] Django Framework, `https://www.djangoproject.com/` [visited on 13/05/2015]

[37] Django Framework Documentation - Models, `https://docs.djangoproject.com/en/1.8/topics/db/models/` [visited on 28/05/2015]

[38] Django Framework Documentation - Migrations, `https://docs.djangoproject.com/en/1.8/topics/migrations/` [visited on 28/05/2015]

[39] Django Framework Documentation - User Authentication, `https://docs.djangoproject.com/en/1.8/topics/auth/` [visited on 28/05/2015]

[40] Django REST Framework, `http://www.django-rest-framework.org/` [visited on 28/05/2015]

[41] Django REST Framework API Guide - Routers, `http://www.django-rest-framework.org/api-guide/routers/` [visited on 28/05/2015]

[42] Django REST Framework API Guide - Pagination, `http://www.django-rest-framework.org/api-guide/pagination/` [visited on 28/05/2015]

[43] Virtualenv, `https://virtualenv.pypa.io/en/latest/` [visited on 05/06/2015]

[44] Virtualenvwrapper, `http://virtualenvwrapper.readthedocs.org/en/latest/install.html` [visited on 05/06/2015]

[45] Gunicorn - Python WSGI HTTP Server for UNIX, `http://gunicorn.org/` [visited on 05/06/2015]

[46] Nginx Web Server, `http://nginx.org/` [visited on 05/06/2015]

[47] Deploying nginx + django + python 3, http://tutos.readthedocs.org/en/latest/source/ndg.html [visited on 05/06/2015]

[48] AngularJS, https://angularjs.org/ [visited on 06/06/2015]

[49] UI Bootstrap, https://angular-ui.github.io/bootstrap/ [visited on 06/06/2015]

[50] Yeoman, http://yeoman.io/ [visited on 08/06/2015]

[51] Grunt, http://gruntjs.com/ [visited on 08/06/2015]

[52] Supervisor: A Process Control System, http://supervisord.org/ [visited on 05/06/2015]

[53] Siege - an HTTP Load Testing and Benchmarking Utility, https://www.joedog.org/siege-home/ [visited on 03/06/2015]

[54] Siege Manual, https://www.joedog.org/siege-manual/ [visited on 10/06/2015]

[55] Jenkins, https://jenkins-ci.org/ [visited on 04/06/2015]

[56] Trello, https://trello.com/ [visited on 04/06/2015]