

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

MENG INDIVIDUAL PROJECT INTERIM REPORT

---

# Active Delay Warning Transport Application

---

*Author:*  
Yangfan ZHANG

*Supervisor:*  
Dr. Peter MCBRIEN

June 5, 2015

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

## Abstract

abstract

Abstract here. The abstract is a very brief summary of the report's contents. It should be about half a page long. Somebody unfamiliar with your project should have a good idea of what it's about having read the abstract alone and will know whether it will be of interest to them. Note that the abstract is a summary of the entire project including its conclusions. A common mistake is to provide only introductory elements in the abstract without saying what has been achieved.

# 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 during the project days and removed many of my 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

|  |           |
|--|-----------|
| <b>Acknowledgements</b>                        | <b>1</b>  |
| <b>1 Introduction</b>                          | <b>1</b>  |
| <b>2 Background</b>                            | <b>3</b>  |
| 2.1 London Bus Network . . . . .               | 3         |
| 2.1.1 Bus Network Performance . . . . .        | 3         |
| 2.2 Buses Status Updates . . . . .             | 4         |
| 2.3 Transport for London Open Data . . . . .   | 5         |
| 2.3.1 Live Bus Arrivals API Stream . . . . .   | 5         |
| 2.3.2 Bus Stop Locations and Routes . . . . .  | 6         |
| 2.3.3 Journey Planner Bus Timetables . . . . . | 6         |
| 2.3.4 Summary on Available Data . . . . .      | 7         |
| 2.4 Current Travel Applications . . . . .      | 8         |
| <b>3 Literature</b>                            | <b>9</b>  |
| 3.1 Overview . . . . .                         | 9         |
| 3.2 Regression . . . . .                       | 9         |
| 3.3 Artificial Neural Network . . . . .        | 9         |
| 3.4 Kalman Filters . . . . .                   | 10        |
| 3.5 K-Nearest Neighbours Regression . . . . .  | 10        |
| 3.6 Analytical Approaches . . . . .            | 10        |
| 3.7 Summary . . . . .                          | 10        |
| <b>4 Concept Design</b>                        | <b>11</b> |
| 4.1 Objectives . . . . .                       | 11        |
| 4.2 Bus Travel Times . . . . .                 | 11        |
| 4.2.1 Reference Timetable . . . . .            | 12        |
| 4.2.2 Current Timetable . . . . .              | 12        |
| 4.2.3 Historical Timetable . . . . .           | 12        |
| 4.3 Contributions . . . . .                    | 12        |

|          |  |           |
|----------|--|-----------|
| 4.4      | Architecture Design . . . . .  | 13        |
| <b>5</b> | <b>Data Collection and Generation</b>  | <b>14</b> |
| 5.1      | Development Environment . . . . .  | 14        |
| 5.1.1    | Virtual Machine . . . . .  | 14        |
| 5.1.2    | MySQL Databases . . . . .  | 14        |
| 5.2      | Generating Reference Timetable . . . . .   | 15        |
| 5.2.1    | Detailed Steps . . . . .   | 15        |
| 5.3      | Generating Current Timetable . . . . .   | 17        |
| 5.3.1    | Collecting Bus Arrival Times . . . . .   | 17        |
| 5.3.2    | Generating Bus Sequences and Neighbouring Stops . .                                | 18        |
| 5.3.3    | Generating the Current Average Travel Time Between<br>Neighbouring Stops . . . . . | 19        |
| 5.4      | Generating Historical Timetable . . . . .  | 20        |
| <b>6</b> | <b>Delay Data Service</b>  | <b>21</b> |
| 6.1      | Overview . . . . .   | 21        |
| 6.2      | Django Framework . . . . .   | 21        |
| 6.3      | Deployment Setup . . . . .   | 22        |
| 6.3.1    | Python Virtual Environment . . . . .   | 22        |
| 6.3.2    | Web Server . . . . .   | 22        |
| 6.4      | API Endpoints . . . . .  | 22        |
| 6.4.1    | Historical & Current Timetables . . . . .  | 22        |
| 6.4.2    | Reference Timetable . . . . .  | 24        |
| 6.4.3    | Arrival . . . . .  | 25        |
| 6.5      | Summary of Data Service . . . . .  | 25        |
| <b>7</b> | <b>Mobile Application for Active Delay Warning</b>                                 | <b>27</b> |
| 7.1      | Motivation . . . . .   | 27        |
| 7.2      | Implementation . . . . .   | 27        |
| 7.2.1    | AngularJS . . . . .  | 27        |
| 7.3      | Deployment . . . . .   | 27        |
| 7.4      | Summary . . . . .  | 27        |
| <b>8</b> | <b>Technical Details</b>   | <b>28</b> |
| 8.1      | Daemen Script Management - Supervisor . . . . .                                    | 28        |
| 8.2      | Scheduled Tasks Management - Jenkins . . . . .                                     | 28        |
| 8.2.1    | Current Timetable Update . . . . .   | 28        |
| 8.2.2    | Historical Timetable Update . . . . .  | 28        |
| 8.2.3    | Arrivals Daily Backup . . . . .  | 28        |
| 8.2.4    | Software Project Management - Trello . . . . .                                     | 29        |

|           |  |           |
|-----------|--|-----------|
| <b>9</b>  | <b>Evaluation</b>                                  | <b>30</b> |
| 9.1       | Accuracy of Live Bus Arrivals API Stream . . . . . | 30        |
| 9.1.1     | Test for Accuracy . . . . .                        | 30        |
| 9.1.2     | Test Result . . . . .                              | 31        |
| 9.2       | Correctness and Performance of the API . . . . .   | 31        |
| 9.2.1     | Tool . . . . .                                     | 31        |
| 9.2.2     | Preparation . . . . .                              | 32        |
| 9.2.3     | Test for Correctness . . . . .                     | 32        |
| 9.2.4     | Load Test for Performance . . . . .                | 32        |
| 9.3       | Accuracy of Predictions . . . . .                  | 33        |
| 9.3.1     | Compare predictions to real data . . . . .         | 33        |
| 9.4       | User feedback on UI . . . . .                      | 33        |
| <b>10</b> | <b>Future Work</b>                                 | <b>34</b> |
| <b>11</b> | <b>Conclusion</b>                                  | <b>35</b> |
|           | <b>Acronyms</b>                                    | <b>38</b> |
|           | <b>Glossary</b>                                    | <b>39</b> |
|           | <b>References</b>                                  | <b>40</b> |

# Chapter 1

## Introduction

The London bus network carries 2.4 billion passengers a year [1]. On average, the buses come about 1 minute later than scheduled [6].

Transport for London (TfL) publishes the live expected bus arrival times. It is currently widely available on digital live bus arrivals signs at more than 2,500 bus stops [2]. Passengers can also access this information via SMS, the web, or mobile applications.

Passengers chose travel mode based on travel time and convenience.  
Find evidence for this assumption. How do people choose travel mode  
/ whether to take bus?

Passengers use the 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 consisting of a variety of travel mode, with an estimate travel time for each suggested journey. Such popular planners include Google maps [3], Citymapper [4] and TfL Journey Planner [5].

However, the accuracy of the bus arrival times published is affected by many external factors. For example, when there is heavy traffic, the buses are likely to be delayed by a difference significant enough for a change in passengers' route picking. Yet, this delay information is not reflected in the arrival time data or the estimated 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 was a 16.6 chance of waiting for more than 10 minutes [16].

Scenario, a passenger chose to take a bus, then only realises delay at the end of the journey. Would have chosen another mode of travel should he know of the delay earlier.

What is the problem? what are the objectives? what are the data?  
What is my approach? What are my contributions and results?

This can be avoided if passengers are informed of the delays in bus arrival times and estimated travel time in advance. Currently, there are no available data services or applications that generate predictions potential delays.

For exemple, a passenger chose to take a bus from point A to point b Such delays can predicted by analysing the historical delays. This is achieved by collecting data from the TfL live bus arrivals Application Program Interface (API) stream feed[2], and estimate the average journey time required between two locations on a given time of the day. Next, a bus arrival time table with delays during various time windows over a week can be crafted and fine tuned incrementally. We compared the travel times in this timetable with the official travel times published by the TfL Open Data to find out the delays in bus arrival times.



# 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 is carrying 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 between 2005 and 2010, about 14% of the trips made by London residents were by bus [9]. They spent on average 14 minutes per day on these bus trips.

There are currently 19,345 bus stops, and 680 routes served by 8,765 buses daily in London[10].

produce graph for this, number of stops, routes and buses

#### 2.1.1 Bus Network Performance

TfL published the following figures in the second quarter 2014/2015 buses performance data [6].

For the 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 the buses to come within 10 minutes 83.4% of the time, there was 15.1% chance of waiting for 10-20 minutes, 1.3% chance of waiting for 20-30 minutes, and 0.2% chance of waiting for more than 30 minutes.

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

For the night buses, 84.5% of the services were on time. The average excess wait was 0.68 minutes.

produce graph here

# STATUS UPDATES

The screenshot shows the TfL Buses Status Update Service interface. At the top, there is a search bar with the placeholder text "Find a bus stop or route". The search bar contains the number "14" and a blue "Go" button. Below the search bar, the text "Showing disruptions on route 14 towards Putney Heath / Green Man" is displayed, along with a link "Clear route". Below this, there are two yellow boxes with a warning icon and a plus sign. The first box says "Status alert for route 14" and the second box says "Additional information".

Figure 2.1: TfL Buses Status Update Service

The bus arrivals might be affected by traffic congestion, staff availability, engineering problems, or mechanical breakdown [16].

## 2.2 Buses Status Updates

To inform passengers of the bus service disruptions or diversions, TfL provides a bus status updates service online [7]. Passengers can retrieve relevant bus service status for a given bus stop or route (Figure 2.1).

The textual information include service disruptions, diversion, suspensions, and delays due to heavy traffic.

While this service informs passengers of current abnormal bus schedules, it requires passengers to visit the site to check for specific information, and does not provide an estimation on the travel time under the special conditions.

TfL also publishes live status news and updates on Twitter[8], but it is not convenient for passengers to search specific information relevant to their journeys.

Many current popular journey planners such as Google Maps[3], Citymapper London[4], and TfL Journey Planner incorporate the bus delay information in the suggested journeys as a textual alert. However, these apps do not recompute the estimated journey time for passengers to make an informed

decision.

As a result, the lack of data service on bus delay predictions and travel times estimations presents a significant barrier to bus passengers who wish to plan their bus journeys for specific appointments.

## 2.3 Transport for London Open Data

Open data is defined as data which can be used, re-used and re-distributed freely by anyone - subject only at most to the requirement to attribute and share-alike. There may be some charge, usually no more than the cost of reproduction [11].

It is TfL's strategy to provide free and open data. It began in 2007 using embedded widgets[12]. In 2015, there are around 30 feeds and APIs available [13].

There are 3 forms of data:

- static data files which rarely change,
- feeds that refreshed at regular intervals,
- and APIs that enable a query to receive a bespoke response, depending on the parameters supplied.

Over 5,000 developers have registered for the open data[13], and around 200 travel apps are powered by it [1].

### 2.3.1 Live Bus Arrivals API Stream

The Live Bus Arrivals API Stream 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 stream 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 5.3.1.

### 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 time. See Section 5.3.2 for details.

### 2.3.3 Journey Planner Bus Timetables

The Journey Planner Bus Timetables[22] contains information on official bus schedules including stops, routes, departure 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 official bus timetables from these XML files, as discussed in Section 5.2.

### 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. Hence, we used these available data to create predictions for bus travel timetables, as discussed in Chapter 5.

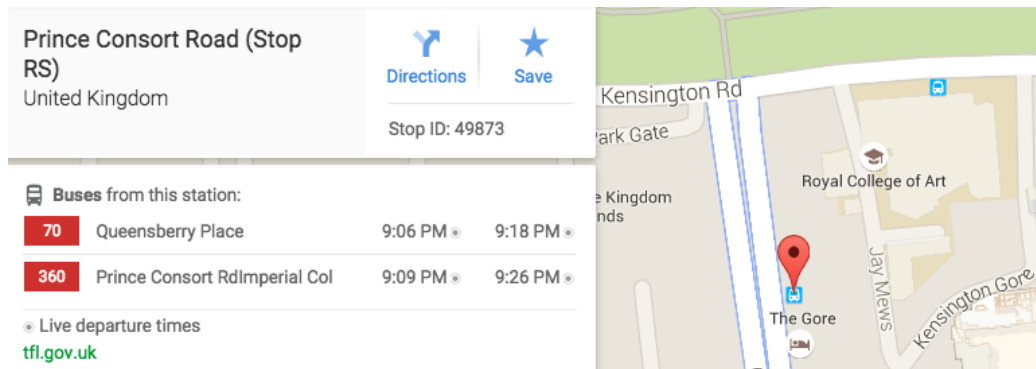


Figure 2.2: Google Maps Using TfL Bus Arrival Data

## 2.4 Current Travel Applications

Currently, the most popular travel applications powered by the TfL Open Data 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.

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.

# Chapter 3

## Literature

### 3.1 Overview

Conventional methods used to predict bus arrival times include regression models, Kalman filters models, Artificial Neural Network (ANN) models, K-nearest neighbours models, and analytical approaches.

### 3.2 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 intersections). Therefore, defining a set of uncorrelated independent variables is the main challenge of building regression models.

### 3.3 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 intersections and schedule adherence to predict bus travel time[24].

### **3.4 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].

### **3.5 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].

### **3.6 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 firstly 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].

### **3.7 Summary**

While there are many available methods, we have not seen any implementations for London in specific. Additionally, since the TfL makes bus arrival times predictions public, we decided to develop an analytical approach that targets at predicting bus travel time for the city of London only.



# Chapter 4

## Concept Design

### 4.1 Objectives

We aim to improve the prediction of bus travel time downstream from location of last observation in mixed traffic operations. We achieve this by providing an API data service of bus travel time predictions, and a demonstrative web application to show case the use of such a data service.

### 4.2 Bus Travel Times

The bus travel time between 2 neighbouring stops 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 travel times by discovering and analysing these contributing factors is complicated.

We decided to bypass looking at these factors, and examine the historical and current bus travel times instead. We assumed that for a specific short time frame, the external factors remain largely unchanged. In this case, the bus travel time between the given 2 neighbouring stops is similar to the previous trips performed in the same time frame.

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

- **Reference Timetable** 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 travel 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 reference timetables would indicate a bus delay.

### **4.2.1 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 5.2.

### **4.2.2 Current Timetable**

We collect the live bus arrival times for the past 1 hour, and store the final bus arrival times for each bus at each stop. We then find out the travel time of each bus between every pair of neighbouring stops for the given hour. Next, we calculate 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 5.3.

### **4.2.3 Historical Timetable**

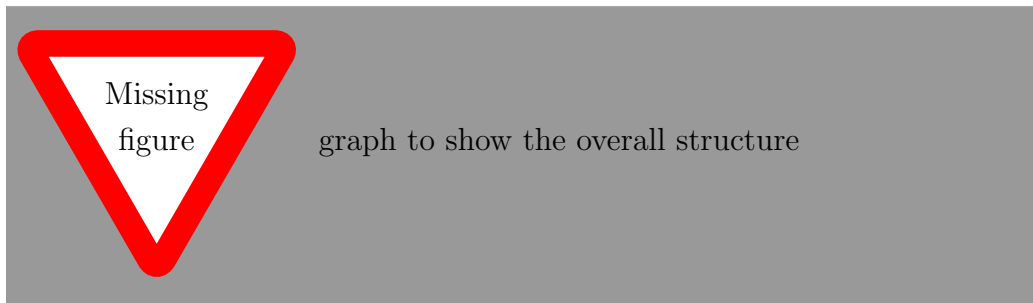
We store the current timetable generated at each hour, and group them by the hour of the day for the same day of the week. We then calculate the average bus travel time between each pair of neighbouring stops for each hour of the day in each day of the week. For example, the average bus travel time between 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 could be found in Section 5.4.

## **4.3 Contributions**

We provided the above mentioned three timetables as an API data service (Chapter 6) and designed a demonstrative mobile application that warns users of current bus delays from a given bus stop (Chapter 7).

## 4.4 Architecture Design

- Used Django Framework to build the backend that retrieves data from a MySQL database.
- Used AngularJS with Twitter Bootstrap for frontend development



# Chapter 5

## Data Collection and Generation

### 5.1 Development Environment

#### 5.1.1 Virtual Machine

We set up a Virtual Machine in the Doc’s Private Cloud[29] with the specifications shown in Table 5.1. We allocated a large memory for memory intensive Databases operations.

#### 5.1.2 MySQL Databases

We chose to use MySQL to store the data for the following benefits it offers:

- **User Interface** MySQL has convenient database management tools such as phpMyAdmin[30] and Sequel Pro[31] for easy data browsing.
- **Scalability** MySQL can handle memory heavy computation efficiently, given the correct configuration.

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

Table 5.1: Virtual Machine Specifications

## Databases Optimisation

As some databases operations involve 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
```

## 5.2 Generating Reference Timetable

For each day of the week, there are a predefined number of VehicleJourneys 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 11.1).

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

We retrieve the corresponding Journey Pattern Section Reference as such:

- (a) Each **JourneyPattern** element contains an element id, and a subelement **JourneyPatternSectionRefs**.
  - (b) We map each Journey Pattern id to its corresponding **JourneyPatternSectionRefs** for reference.
  - (c) We consult the above mapping to retrieve the Journey Pattern Section Reference for each Journey Pattern Reference found in Step 2(c).
4. Next, we obtained 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**  $\rightarrow$  **JourneyPatternSection** with the same id.
    - (b) Each **JourneyPatternSection** contains multiple **JourneyPatternTimingLink** subelements. Each **JourneyPatternTimingLink** contains information for one pair of neighbouring bus stops. We retrieved the following information:
      - i. **From** **SequenceNumber**
      - ii. **From**  $\rightarrow$  **StopPointRef**
      - iii. **To** **SequenceNumber**

iv. **To**  $\rightarrow$  **StopPointRef**

v. **RunTime**

5. At this point, we have 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 calculate the cumulative bus travel time for each stop, and derive the bus arrival time at each stop for the given vehicle journey.
6. We group 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.

## 5.3 Generating Current Timetable

### 5.3.1 Collecting Bus Arrival Times

#### Building the Query URL

We collect 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](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 identifier 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](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 will assume the actual bus arrival time is the last estimated arrival time, when it is more than 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 estimated 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 pickle module in Python[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.

We achieved this by the following steps:

1. Load the local arrivals objects if there exists a copy.
2. Pull the new TfL arrivals predictions.
3. Update the local 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.

These 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.

### 5.3.2 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 11.3). 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



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

Table 5.2: Sample data in delay\_neighbours Table

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 need 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 11.4). In the sample entries shown in Table 5.2, 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.

### 5.3.3 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) We found out the arrival times for the same bus at the start stop and the end stop of the neighbouring pair.
  - (b) We calculated the difference of these two arrival times, and saved it as one entry of the travel time.
2. We computed the the average travel time of all bus trips took place between every pair of neighbouring stops, and saved it in the current timetable.
3. We also saved 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.

## **5.4 Generating Historical Timetable**

The current timetable generated every hour is stored in a log. We processed this 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.

# Chapter 6

## Delay Data Service

### 6.1 Overview

To enable structured query of the predictions generated, we created a REST API service for the prediction data. We hope to make these endpoints freely available for other developers to use.

### 6.2 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 given migration command [38].
- It has a nice 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.

## 6.3 Deployment Setup

### 6.3.1 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.

### 6.3.2 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].

## 6.4 API Endpoints

Should I put screenshots for 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`.

### 6.4.1 Historical & Current Timetables

This API endpoint is built to provide information on how long the bus on a given route at a given hour of the given day will take to reach the future 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/?.`

The following parameters are required to obtain a specific set of predictions:

- **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](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.

## 6.4.2 Reference Timetable

This API endpoint is designed to provide 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 6.4.1.

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 given 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.

### 6.4.3 Arrival

This endpoint serves as a wrapper for the TfL Live Bus API endpoint. Given a set of Global Positioning System (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 6.1 for example response.

## 6.5 Summary of Data Service

Building these three API endpoints helped us meet our first objective by supplying data on bus travel time predictions. We evaluated the accuracy and performance in Chapter 9.

Django REST framework v3.1.1

Log In

Arrivals List

OPTIONSGET

GET /arrivals/?latitude=51.495171603309615&longitude=-0.1883983612060547&radius=100

HTTP 200 OK

Content-Type: application/json

Vary: Accept

Allow: GET, HEAD, OPTIONS

```
[
  {
    "stop_code_lbsl": "11184",
    "sms_code": "47455",
    "naptan_atco": "490010779E",
    "latitude": "51.4947750000",
    "longitude": "-0.1873400000",
    "lines": [
      {
        "route": "74",
        "stop_name": "Collingham Road / Sainsbury's",
        "run": 1,
        "destination": "Baker Street Station",
        "estimatedTimeInSeconds": [
          96,
          154,
          266
        ]
      }
    ]
  }
]
```

Figure 6.1: Sample Response for Arrivals API



# Chapter 7

## Mobile Application for Active Delay Warning

### 7.1 Motivation

Demo App to use the data for effective delay warning

- Write from the user point of view

- to demonstrate what the predictions can be used for non-technical users

- easy to use interface

- easy to understand data presentation

- personalised warning feature

### 7.2 Implementation

#### 7.2.1 AngularJS

### 7.3 Deployment

### 7.4 Summary

save users time make informed decisions when choosing travel mode

# Chapter 8

## Technical Details

### 8.1 Daemen Script Management - Supervisor

### 8.2 Scheduled Tasks Management - Jenkins

We used Jenkins[49] to managed our predictions timetable update schedule.

add screenshots for jenkins

#### 8.2.1 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 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.

#### 8.2.2 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.

#### 8.2.3 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

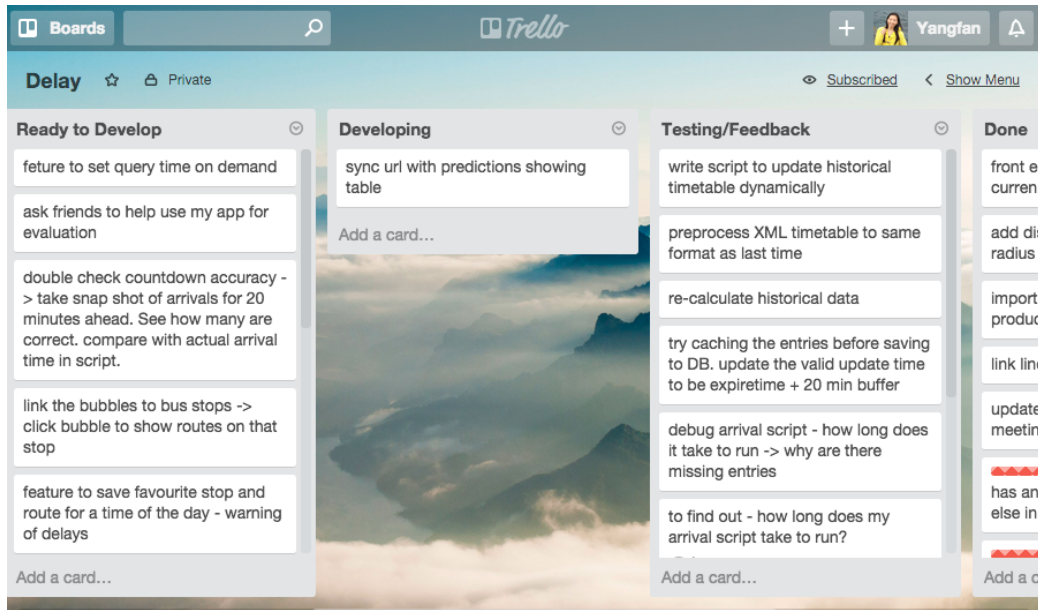


Figure 8.1: Project Trello Board

archive table. This required us to backup the arrivals entries daily to the archive.

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.

#### 8.2.4 Software Project Management - Trello

We used Trello [50] 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 8.1).

# Chapter 9

## Evaluation

### 9.1 Accuracy of Live Bus Arrivals API Stream

The TfL Live Bus Arrivals API Stream updates the bus arrival times every 30 seconds. This means that the bus arrival predictions for the current stop would get more and more accurate 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.

#### 9.1.1 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 prediction 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 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.

| 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 9.1: Live Bus Arrivals API Stream Accuracy - A negative delta indicates the bus came later than the predicted time

- We calcuated the average delta value for each group, and created Table 9.1.

### 9.1.2 Test Result

Table 9.1 shows that buses usually came later than the predictions provided by the Live Bus Arrivals API Stream. For the buses that were predicted to due 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.

## 9.2 Correctness and Performance of the API

### 9.2.1 Tool

We used *siege*[48] 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.

| No. Users | Transactions | Availability | Elapsed time (secs) | Data transferred (MB) | Response time | Transaction rate (trans/sec) | Throughput | Concurrency | Successful transactions | Failed transactions | Longest transaction | Shortest transaction |
|-----------|--------------|--------------|---------------------|-----------------------|---------------|------------------------------|------------|-------------|-------------------------|---------------------|---------------------|----------------------|
| 50        | 238          | 100.00%      | 59.46               | 1.82                  | 11.01         | 4.00                         | 0.03       | 44.05       | 238                     | 0                   | 17.76               | 0.54                 |
| 100       | 358          | 100.00%      | 59.81               | 2.68                  | 13.93         | 5.99                         | 0.04       | 83.4        | 358                     | 0                   | 20.29               | 0.3                  |
| 150       | 455          | 98.48%       | 59.56               | 3.37                  | 15.78         | 7.64                         | 0.06       | 120.56      | 455                     | 7                   | 29.68               | 2.45                 |
| 300       | 461          | 75.33%       | 59.27               | 3.28                  | 14.22         | 7.78                         | 0.06       | 110.57      | 461                     | 151                 | 29.42               | 0.17                 |
| 500       | 522          | 58.98%       | 59.86               | 3.88                  | 16.35         | 8.72                         | 0.06       | 142.58      | 522                     | 363                 | 35.23               | 0.96                 |

Figure 9.1: Result of Performance Test with Siege

## 9.2.2 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.

## 9.2.3 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.

## 9.2.4 Load Test for Performance

We found out that currently, the server could reliably handle about 150 users to send requests concurrently, with an average response time of 15.78 seconds. Figure 9.1 shows the results of our tests.

plot graph from results

Transactions is the number of server hits. In the example, 25 simulated users [

Elapsed time is the duration of the entire siege test. This is measured from the

Data transferred is the sum of data transferred to every siege simulated user. I

Response time is the average time it took to respond to each simulated users request.

Transaction rate is the average number of transactions the server was able to handle.

Throughput is the average number of bytes transferred every second from the server.

Concurrency is average number of simultaneous connections, a number which rises with the number of users.

Successful transactions is the number of times the server returned a code less than 400.

## 9.3 Accuracy of Predictions

How accurate are the predictions?

### 9.3.1 Compare predictions to real data

We can compare the predictions generated to the live bus arrivals data stored in the arrivals table. We can calculate the standard deviation of the difference in the predicted time and actual arrival time. This value will be the direct indicator of the accuracy of the predictions.

## 9.4 User feedback on UI

On the How effective is it at warning delay?

# Chapter 10

## Future Work

- improve the performance of API endpoints by separating the data collection DB from the API request handling DB



# Chapter 11

## Conclusion

¡Conclusion here!

| 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 11.1: delay\_tfl\_timetable 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 11.2: 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   | varchar(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 11.3: 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        | varchar(64) | The stop_code_lbsl for the end stop   |

Table 11.4: delay\_neighbours 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?, url<https://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](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](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](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](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](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](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](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] Siege - an HTTP Load Testing and Benchmarking Utility, <https://www.joedog.org/siege-home/> [visited on 03/06/2015]
- [49] Jenkins, <https://jenkins-ci.org/> [visited on 04/06/2015]
- [50] Trello, <https://trello.com/> [visited on 04/06/2015]