King's College London
University of London

7CCS4PRJ Final Year Individual Project

Real-Time Visualisation of Bus Delays in London

iBus Disruption Monitor

A project in collaboration with Transport for London

Final Project Report

Author: Konstantin Vladimirov Draganov

Supervisor: Dr Steffen Zschaler

Course: MSci Computer Science

Student ID: 1101314

September 2014 - April 2015

Abstract

Automatic Vehicle Location (AVL) systems for bus fleets have been deployed successfully in many cities. They have enabled improved bus fleet management and operation as well as wide range of information for the travelling public. However there are still processes that can be improved or automated by utilising the data made available by the different systems including the AVL one. This report explores and analyses the tools and applications currently available at Transport for London (TFL) bus emergency and command unit. The report then proposes a prototypical tool for monitoring the bus delays in real time in the network. This tool offers objective source of processed information to bus operators and control room staff. However further work need to be done in order to place this tool in production environment. This is because arterial urban delay detection is very complex and unpredictable as will the report justify. The report concludes with some suggestions of how this project can be improved.

Originality Avowal

Konstantin Vladimirov Draganov

September 2014 - April 2015

Acknowledgements

First and foremost I offer my sincerest gratitude to my supervisor Dr Steffen Zschaler for the continuous support, guidance, encouragement, insightful comments and hard questions throughout the course of this exciting project.

I would also like to thank Andrew Highfield and Keith Elliot from TFL for providing me with all the needed data, information and feedback which has been immensely helpful.

Last but not the least, I would like to thank my parents Vladimir and Veselka also my sister Lilia and my partner Simona for their support and patience not only during the project, but throughout my life as well.

Contents

Chapter 1

Introduction

London bus network is one of the largest and most advanced bus networks in the worlds. It is responsible for more than 2.4 billion passenger journeys a year [11]. The constant population growth of England's capital has been also driving the expansion and improvement of the transport networks across the city. Transport for London (TFL) is in charge of its operation and its bus network if recognised as one of the top in the world in terms or reliability, affordability and cost-effectiveness [11]. The capital's bus network is continually increasing along with the city's population [21]. Maintaining such a large scale network requires careful planing and monitoring. Being able to maintain such high reliability service 24/7 364 days in the year requires employing new technologies. This also helps keep costs down and thus keeping the service more affordable and accessible for the general travelling public. Each bus in the TFL network has been equipped with state of the art GPS enabled automatic vehicle location (AVL) system named iBus[27]. This AVL system has led to improved fleet management and has enabled the creation and improvement of multiple applications [40]. The system is generating large sets of data both in real-time as well as historical data. This helps, the bus operators and the emergency control room at TFL responsible for maintaining the bus network (CentreComm), to better manage and maintain the smooth operation of the bus network.However there are currently problems for which CentreComm staff are required to manually analyse these data sets in order to discover in which parts of the networks problems are occurring. This is because there is lack of readily available preprocessed information. This is very impractical and time consuming and currently they ultimately rely on individual bus operators and drivers to notify them of possible problems. Once alerted of a possible disruption in the network they can start their own investigation into first verifying what they have been told by the bus drivers/operators and then into finding the cause and the actual severity of the problem. This often could lead to spending time and resources into investigating non existent problems. This is where this project comes in place to address this inefficiency and to propose, implement and evaluate a prototypical tool for real time monitoring of the bus network.

1.1 Scope

The scope of this project is to analyse the current work flow of CentreComm operators and their needs. The main goal is to design and implement a prototype which to aid the control room staff. This tool has to work and analyse the data that has been made available in real time. The main two problems that are in the scope of this project and need to be solved are:

- Analysing and identifying the disruptions in the bus network in real-time using the provided data.
- Producing a prioritised list of the disruptions classified using rules and heuristics gathered during meetings and discussions with the key stakeholders from TFL.
- Visualising the calculated list in an appropriate format.
- Evaluating the performance of the tool. This is essential as it need to be proven that the results could be trusted.

1.2 Aims

The main aim of this project is to design and implement a real-time visualisation tool which to highlight disrupted routes or parts of the TFL bus network where disruption might happen or have already happened even before the bus drivers or operators have noticed and alerted CentreComm. This could could be subdivided into two smaller aims:

- The first one which is independent of the other is to enable the processing of the data generated by the buses in the TFL's bus network. The tool need to be able to analyse the input data sets and calculate and output a list of the disruption that are observed in the network. It has to present information regarding the location (section) in the transport network and their severity.
- The second part of the main aim above is to visualise the generated output in an easy to use and understand way.

1.3 Objectives

The objectives that have been followed in order to successfully meet the above stated aims are:

- Getting in depth understanding of the problem and current work-flows that are in place.
- Researching similar work that has already been done and how it can relate to our problem.
- Obtaining samples of the available data and understanding what it means
- Gather and refine the user requirements during discussions and meetings with CentreComm staff and stakeholders.
- Design and develop initial prototype which to be further refined and improved after obtaining feedback from TFL.

- Test and evaluate that the tool works according to the user requirements and the design specifications.

1.4 Report Structure

In order to help the reader I have outlined the project structure here. The report would continue in the next chapter by providing the reader with all the background knowledge needed for the rest of the report. This would include brief of background on the current work-flow CentreComm operators follow and its inefficiencies. I will also give background on the iBus system and the data that the tool would need to operate with. I then explore related work that has already been done and how ours differs. This is followed by alternative approaches and models that could be utilised. Afterwards the report focusses on the specific requirements that have been identified and gathered from CentreComm. The report then goes on to discuss the design and the implementation of the proposed system. This is followed then by Chapter 5 and 6 which address testing and evaluation of the prototype. I conclude the report with a summary of what has been achieved and guidance how the work presented in this report could be further developed and improved.

Chapter 2

Background

This chapter aims to introduce some concepts surrounding our problem domain, which to help the reader to understand and follow easier the following more technical chapters. I also present a review of the related work that has been done in this area. The below sections look at some of the key aspects and problems that arise. I then conclude by providing a number of alternative methods for solving our problem.

2.1 London Bus Network

London bus network is one of the most advanced and renowned in the world. It runs 24 hours and it is extensive and frequent. Every route in the network is tendered to different bus company operator [8]. These operators agree with TFL that the routes they are operating would be served either according to a predefined fixed schedule (e.g. a bus stop need to be served at 1pm, 3pm etc.)or on a headway (e.g. a bus stop need to be served every 5 minutes). However under different circumstances some delays occurring on a given route are beyond the control of the different bus operator companies. A simple example could be a burst water/gas pipe on a street used by a bus route or any other incident (even terrorist attacks [10]) and even simply a severe congestion. In situations like this bus operators have no authority or power to overcome such problems on their own. They can only ask CentreComm to intervene. CentreComm can do so by for example implementing a short/long term diversions or curtailments (short turning) some of the buses on the affected routes.

Buses in the network can be classified by multiple factors, however for the purpose of this report the main distinction we need to consider are high and low frequency bus routes. High frequency routes are routes where there are 5 or more buses per hour attending a given bus stop. Low frequency routes are those that have 4 or less buses alighting at a stop.

2.2 CentreComm

CentreComm is TFL's emergency command and control room responsible for all public buses in London. It is has been in operation for more than 30 years [10] and it employs a dedicated team of professionals who work 24 hours 364 days in the year. They are dealing with more than a 1000 calls on a daily basis. The majority of these calls come from bus drivers or bus company operators regarding problems and incidents happening within the bus network. CentreComm staff implement planned long and short term changes in the bus network in response to different events taking place in the capital (including the 2012 Olympics). They are also responsible for reacting in real time to any unexpected and unpredicted changes and disruptions, maintaining the smooth, reliable and sage operation of London busy bus network.

London bus network consists of around 680 bus routes operated by more than 8000 buses [2]. Each of this buses is equipped with state of the art iBus system to help monitor and manage this enormous fleet. CentreComm's way of operation has been transformed beyond recognition since it has first opened and today. It started more than 30 years ago [10] and it consisted of a couple of operators equipped with two way radios and pen and papers. Today CentreComm operators make use of numerous screens each displaying interactive maps (displaying each bus location) and CCTV cameras in real-time. However there is still a lot of room for automation and improvement in their way of operation in order to effectively and efficiently maintain the growing bus network.

2.3 iBus AVL

Automatic vehicle location systems make use of the Global Positioning System (GPS) to enable the remote (using the internet) tracking of the locations of the vehicles in a given fleet. This system combines a number of technologies including GPS, cellular communications and more with the goal of improving and cutting the cost of fleet management.

All of London buses operating on the TFL bus network have been equipped with state of the art and award wining [7] AVL system named iBus [9]. This system has opened a range of new applications that could be built on top of it using the information that is made available. The iBus system consist of a number of computer and communication systems, sensors and transmitters as described in [17] and [40]. One of the key components of the system on-board unit (OBU) which mounted on each of buses in the TFL bus fleet and consists of a computational unit connected to sensors and GPS transmitters (see figure 1 below taken from [17]). This OBU is responsible for a number of tasks including a regular (approximately every 30 seconds) transmission of the bus location. This information is currently used by the different bus operators for fleet management as well as by CentreComm for real-time monitoring of the buses and their locations. There have been a number of other applications and systems that have been implemented and put in to use as a result of the data that is generated by the iBus AVL. Some examples include Countdown (real-time passenger information), improved bus priority at traffic signals and more [17]. This has led to improved and more affordable transport service.

CentreComm operators have access to an online system showing each bus location in the network on a map in real-time. This system allows them to see whether a given bus is behind, ahead or on time according to its schedule. However this does not show or alert the control room staff if a bus or a route

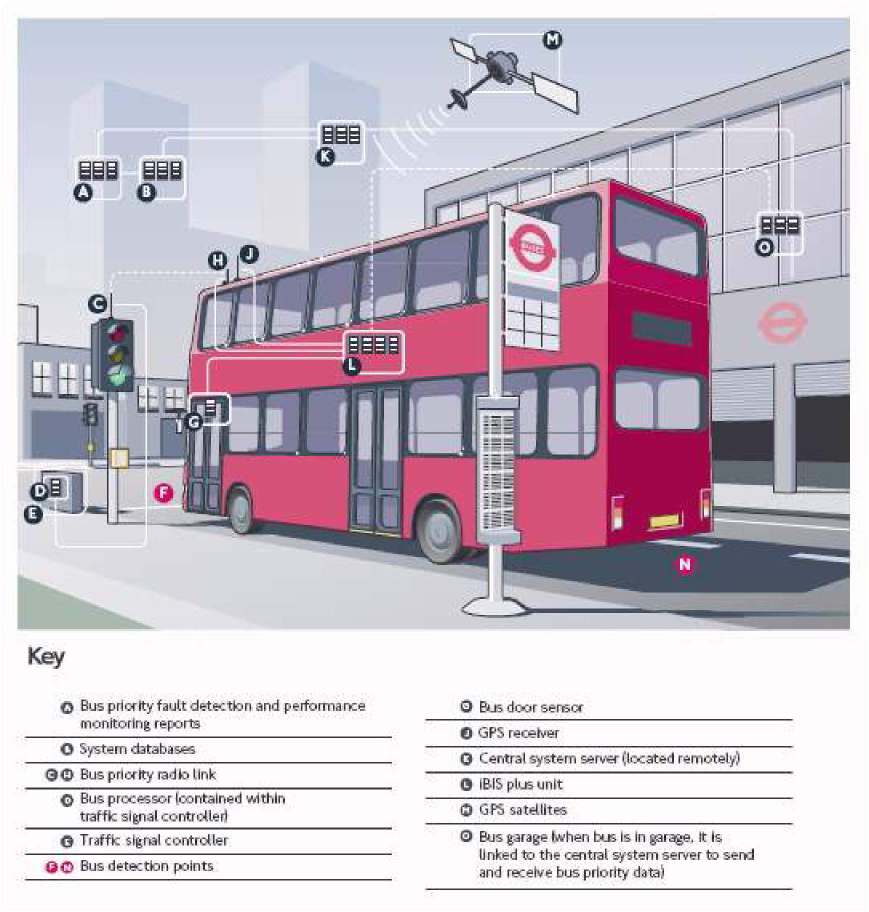

A
B
K
M
H
J
C
L
G
D
E
F
N
O
Key
A Bus priority fault detection and performance monitoring reports
B System databases
E H Bus priority radio link
D Bus processor (contained within traffic signal controller)
C Traffic signal controller
F N Bus detection points
G Bus door sensor
J GPS receiver
K Central system server (located remotely)
L iBIS plus unit
M GPS satellites
O Bus garage (when bus is in garage, it is linked to the central system server to send and receive bus priority data)

Figure 2.1: Overview of iBus System (Source: Transport for London, 2006a)

is disrupted. Control room staff also can see when was a given bus expected to arrive at a given bus stop and when it actually arrived, but this again is only per individual bus. Currently the work-flow is such that CentreComm need to go and analyse all this information manually (once bus drivers or bus company operator have contacted/alerted the control room) in order to figure out if a there is a problem and how severe it is. This is very inefficient, tedious and error prone process. Here is where our project comes into place to address the lack of preprocessed information.

2.4 iBus AVL Data

The data that is used for this project is provided by the Technical Service Group (TSG) at TFL. The data consists of comma separated value files. There is an individual file for every different bus operator which contains the data for all buses currently operated by this company. Initially every bus in the network would transmit its unique identification number and GPS coordinates approximately every 30 seconds [17]. This information is then preprocessed by a central server. This results in more information being derived as the central server has knowledge of the whole network and the bus schedules and headways. This results in the CSV feed files that have been provided to us for this project. An example of the content of the raw feed file and a formatted version is presented in section A.4 of Appendix A. Below I have provided a detailed explanation of each field in these files [30].

- **Vehicle Id** - this is a unique id of the vehicle.
- **Bonnet Code** - this is the bonnet code of the bus.
- **Registration Number** - this is the number of the registration plate of the bus.
- **Time of Data** - this refers to the date and time of when this data is received from the respective vehicle.
- **Base Version** - this the version of the system that is run by the respective bus.
- **Trip Id** - stores the internal trips id [1,2]. This would increment every time a bus starts new run either at end of its current run or if it is curtailed.
- **LBSL[3] Trip Number** - this is LBSL trip number[1]. This is similar to the Trip Id however this is a global trip id thus it is incremented whenever a bus in the network start a new trip.

[1]This is only valid when bus is properly logged.
[2]Not available in route variant.
[3]London bus services limited [35]

- **Trip Type** - the type of the trip as follows:
    1. - From depot to start stop of the block.
    2. - To new starting point.
    3. - Normal trip with passengers.
    4. - From the last stop of the block to the depot.
    5. - Without passengers.
    6. - Route variant.
    7. - Vehicle not logged in either block or route.
- **Contract Route** - the route name[1,2].
- **Last Stop ShortDesc** - this is the lbsl code of the last stop visited by the respective bus[1,2].
- **Schedule Deviation** - this is the standard deviation from the schedule, calculated using the bus position telegram, for the respective bus[1,2].
- **Longitude** - this is longitude of the place from where the vehicle is sending the telegram[4].
- **Latitude** - this is latitude of the place from where the vehicle is sending the telegram[4].
- **Event Id** - the last event Id.
- **Duration** - currently not being populated.

The information we are interested is the deviation from the schedule. This value is calculated the same way for both low[5] and high[6] frequency buses by knowing the bus schedule. It must be noted that it is possible vehicles would start the route already with some deviation from the schedule.

[4]GPS raw data divided by 3,600,000.
[5]Less than 5 buses per hour.
[6]5 or more buses per hour.

2.5 Related Work

The main problem posed by this project of detecting disruptions in the bus network can be also translated to short term traffic congestion detection. This is valid as we are not interested in individual bus delays. Some examples of such single instances of bus disruptions are customer incident on board of a single bus or technical fault with this bus etc. We are interested into finding routes/sections in the network which are delay/disrupted and this beyond the control of individual bus operators. Most often such problems are due some sort of congestion or road closures/repairs. However road problems are in most cases linked with increased traffic congestions as roads are used by other vehicles as well.

The literature is full of research towards accurately predicting bus arrival times with various computational models being used [1]. Also there is a lot of work done toward detecting calculating travel times in non urban environment. This includes approaches based on AVL probe data and automatic vehicle identification (AVI) as well as induction loops [37]. There are also plenty research done towards traffic congestion detection based on AVL probe data [37]. However there are significant challenges due to the nature of the urban environment it self. Densely populate areas are influenced by many factors which could be affecting the general traffic flow. Another problem posed by urban environments is the irregularity of the AVL data transmission because of for example weak or lost signal at times (e.g. due to high buildings) or even there could be some noise which reduces the accuracy of the stated location.

There is however little research to my knowledge which focusses on the issues of detecting and short-term forecasting of traffic congestion/disruptions in arterial urban environment [37] [5]. As it can be seen from the tables in figures A.1 to A.4 taken from [37]. From these tables we can easily see that most of that has been done has focussed on motorways and also has employed data from static detector points (e.g automatic vehicle identification). Only in recent years we can see that more attention has been given to the use of GPS and AVL data. This is probably due to increased popularity and mass usage of such technologies. The literature is abundant of research on addressing the issues of bus prioritisation at traffic signals and junctions [16] and [26], but this is not directly related to our problem domain and thus is not discussed further in this report.

The approaches and research that we have examined show us that the the main model that have been used for detecting and forecasting short-term traffic congestion/disruptions are: According to [41] traffic forecasting models could be categorised as follows:

- Statistical models
- Computer Simulations
- Mathematical Optimisation
- Artificial Intelligence

The statistical models could be further subdivided to:

- Historical
- Time Series
- Nonparametric regression
- Hybrid

From the above the most widely used and well defined are the statistical models. From them the historical approaches are relatively easy for implementation and have fast execution speed, but have difficulty with dealing with incidents. Time series models have many applications and are well formulated. Because of their wide usage and advantages and the available AVL data for this project we focus our attention on time series analysis for the rest of this report.

2.6 Time Series

Time series is a sequence of data readings taken during successive time intervals. This could be a continuous recording of readings or a set of discrete readings. In the context of our project we have the continuous process of bus readings (generated by the AVL system) being generated and transmitted. Analysis of time series data could be performed in order to extract and calculate some meaningful statistics from the data [32]. This could also result in producing a forecast of the data of interest for the next (future/unobserved) period of time based on the past observations. In order to highlight trends and make predictions we need to employ time series analysis techniques. Below I have presented some of the available techniques that could be employed when analysing time series data. It is not an exhaustive review of all available methods and models as I have tried to keep the discussion relevant to this project.

2.6.1 Moving Averages

Moving average also called rolling or running average is a statistical calculation method. It is universal analysis technique. It is type of mathematical convolution [32]. It helps to analyse data series by calculating a series of averages of subsets of the data. It can also be referred to as rolling or moving mean and it acts like a filter. Moving averages is commonly used in time series data analysis. It can be used in order to smooth out a time series data with the aim of highlighting or estimating the underlying trend of the data. The other main usage is as a forecasting method again for time series. The main strength of these methods is that they are easy to understand. Moving averages are often used as the building block for more complex time series analysis. Below are the three main type of moving averages.

Simple Moving Average

Simple moving average (SMA) is calculated by adding all the observations for a given period of time and dividing this sum by the number of observations. This is popular statistical technique which is mainly used to calculate the trend direction. The simple average is only useful for estimating the next forecast when the data does not contain any trends. Each observation is weighted equally. If we consider shorter term averages they would react quicker to changes while long term averages would have greater lag. The equation for calculating SMA is given below as equation 2.1.

$$SMA = \frac{\sum_{0 \le i \le n} \text{Value(i)}}{n} \tag{2.1}$$

Weighted Moving Average

The problem with the simple moving average is that it weighted all data points equally. Meaning that both older and newer data would have the same effect on the average. This however is not the case when using weighted moving average (WMA). In WMA model each data point would be weighted differently according to when the observation was made. For example if we consider a n period moving average we could multiply the oldest observation by

$$\frac{2 * 1}{n(n+1)}$$

and the newest by

$$\frac{2 * n}{n(n+1)}$$

. This would mean that recent data have bigger impact on the result. It is important that the weights add up to 1. The general equation for calculating WMA is given below as equation 2.2.

$$WMA = \frac{\sum_{0 \le i \le n} (\text{Weight(i)Value(i)})}{\sum_{0 \le i \le n} Weight(i)} \tag{2.2}$$

Exponential Moving Average

Exponential smoothing was first suggested by Robert Goodell Brown [13].

Exponential moving average (EMA) or also called exponential smoothing is very similar to WMA. The main difference is that in order to calculate it we do not need to keep all the data, but we could only store the latest value and the previous forecast only. Exponential moving average weights the data points exponential which means that the oldest data would have minimalistic effect on the result.There exist few exponential smoothing techniques including single, double and triple exponential moving average. Equations 2.3 and 2.4 give the simplest form for calculating single exponential smoothing. In this equation α is called the smoothing factor and it is usually a value between 0 and 1. The closer α is to 0 have greater smoothing factor, but are less responsive to recent changes thus have greater lag. Values of α that are near to 1 have less smoothing effect, but are very reactive to recent changes in the data.

$$EMA_1 = Value_1 \tag{2.3}$$

$$\text{for } t > 1,\ EMA_t = \alpha Value_t + (1-\alpha)EMA_{t-1} \tag{2.4}$$

Summary

If we compare the three types (simple, exponential and weighted) we can clearly see that the simple moving average offers the most smoothing however there is the trade-off of the biggest lag.The exponential moving average will react faster to changes and sits closer to the actual readings, however it might overreach at times. The weighted moving follows the movement even more closely than the exponential moving average. Determining which one to use depends on the objective. If a trend indication with better smoothing and little reaction for shorter movements then the simple average should produce the best results. However if smoothing is desired where you can still see shorter movements then it is better to use either WMA or EMA.

2.6.2 Peak Detection

Detection and analysis of peaks (spikes) and valleys in time series is important in many applications (e.g signal processing, bioinformatics and many more). Peaks and valleys usually represent either significant events or errors in time series data. In our domain we are mostly interested in detecting high sudden changes in the traffic congestion conditions (e.g along route/sections in the bus network). Peaks could be easily identified by visualising the data, however we are interested in automating this process. In the literature has many examples of peak detection application one such for example is [31] used for spike detection in microarray data (anther example could be found in [3]). We do not go into details of any particular algorithms here because of the time constraints of this project (a study peak detection algorithms could be found in [36]). However it is worth to note that spike detection and analysis could be used to classify events that are detected. A simple example could be to distinguish if given detected congestion/disruption is incident related (e.g. sudden) or it is general traffic jam (e.g. rush hour).

2.7 Other

Other approaches that could be used include Kalman filtering [18] [14], Markov Chains [25] [29], Machine learning [15], Bayesian networks [39].

2.8 Summary

In this chapter I have aimed to provide the reader with broad view of the background of our problem and its domain. Detailed overview of the operations and the technologies and work flows used by TFL's bus command and control centre has been provided. This has led to detailed review and discussion of the related work that has been found in the literature. The chapter concluded with discussion on some of the available approaches. The exact approach taken and any implementation details are described in detail in Chapter 5 along with presentation and discussion of the data that has been provided by TFL for this project.

Chapter 3

Specification

In this chapter I have introduced and formalised the user and functional requirements. This is an important step of any project, especially computer science project,as it formalises the problems that the project is trying to address as outlined by the project aims and objectives in chapter 1. It also allows to be used as a measure for evaluating the success of the project once it has been completed. The requirements presented below have evolved and have been refined throughout the project lifetime in response to feedback and discussions carried out with the key stakeholders.

3.1 User Requirements

The user requirements provide a list of the functionalities that the user(s) expects to be able to perform and see the according results, in the end product. These are what is expected from the system, but are not concerned how they are designed or implemented. The main user requirements are listed as follows:

1. The tool must be able to produce a prioritised list of the disruption in the bus network that it has knowledge of.

2. The tool must be prioritising the disruptions according to the user defined rules (these are still discussed and gathered from the user) The tool must

be updating this list of disruptions whenever there is more data. This should happen as real-time as possible.

3. The tool must be able to provide detailed information for every detected disruption. This has to include the specific section and route that are affected and its severity.

4. The user must be able to interact with the system in order to lower or increase the priority of a given disruption (even ignore one).

3.2 Functional Requirements

These requirements specify in more details what the expected behaviour and functionality of the system/tool is. They are built on top of the user requirements as an input and are detailed list of what the system should be able to accomplish technically. The tool/system must:

1. Have an appropriate and useful representation of the bus network in order to be able to monitor and detect problems in it.

2. Be able to read and process CSV files as this is the primary input of the AVL feed files (more detailed discussion on the exact input and its format is presented in Chapter 5).

3. Listen/monitor for new incoming data and process it in real-time.

4. Be able to update itself whenever new data is detected and processed.

5. Be able to run without intervention 24/7.

6. Keep track of the disruptions detected and track how they evolve and develop.

7. Be able to keep information for a given window of time (e.g. data feeds from the last 2 hours).

8. Be able to output a prioritised list of disruptions.

9. Visualise the generated output appropriately. It should be compatible to run under Firefox or Internet Explorer as this are the main browsers used by CentreComm/TFL staff.

10. Display on request detailed information for the requested disruption. This should include a graph representing the route/section average disruption time.

11. Be easily configurable and maintainable.

Chapter 4

Design

For the purpose of the successful completion of this project I have decided to employ agile software development methodologies with evolutionary prototyping. The reason for taking this approach are the strengths of the agile software developmental methodology which is that it is incremental, cooperative, flexible and adaptive [20]. The research nature (as seen in chapter 2 of this report) of this project itself makes it sensible to use evolutionary prototyping [6] as this helps minimise the impact of misunderstanding or miscommunication of the requirements. The risk of which are relatively high as the goals of this project are relatively new and there is not much similar work done. This technique would also give better idea of what the end product would be capable of and would look like to the client. With evolutionary prototyping the system is continuously refined and improved. Each iteration builds on top of the previous thus meaning that with each increment we add more functionality and features or/and refine/improve what has already been implemented. Simply stated this means that with each iteration we are one step closer to the end product. This allows us to add features which were not previously considered or remove ones that are no longer viable or needed. In addition this approach also allows us to engage with the key stakeholders very early in the project life-cycle. This would provide us with valuable feedback which again brings a lot of advantages.

- The delivery of the tool is speeded up and also minimises the risks of failing to deliver a working product before the project deadline [6].
- Users would engage with the product early in the project lifetime. This however poses some risk like the users requesting more features which were not previously mentioned or discussed. This means that we need to maintain some balance as this project has a fixed deadline and limited resources.
- Increased chances of fully understanding and meeting the user requirements and expectations from the tool.

Each increment (iteration) consists of the following stages:

1. Requirements specification & refinement
2. Design
3. Prototype implementation & Testing
4. User testing and feedback provision
5. Evaluate

The requirements and specification step would document what the tool should look and work like at the end of each iteration. These would be following the aims and objectives we have defined in Chapter 1 of this report. After a prototype has been implemented and tested by the user we will evaluate the progress and make any changes in out requirements, design and project plan accordingly. Thus after a number of iteration we should have a prototype which to resemble the desired product as required by the user.

In the below sections I have presented the system at an abstract level. This follows from careful analysis and consideration of the requirements stated in the previous chapter. I have tried to highlight all major key components and classes which are described formally. This allows us to better organise and structure our problem. The diagrams presented follow the unified modelling language (UML) paradigm [24] and are platform-independent models (PIM) of the system. This allows us to focus on the design of the system itself without distracting our attention with platform specific decisions. Once these models are created we can than easily transform them into platform specific models (PSM) using the desired technologies. This technique is known as Model-Driven Architecture (MDA) software design approach [23].

4.1 Use cases

Based on the user requirements stated in the previous chapter a use case diagram has been derived and presented in figure A.1 in Appendix A. The use case diagram depicts the way users(actors) interact with the tool (system). There a three types of users of the system (every next type is extension of the previous as it can be seen from the diagram):

- **Normal** users are people with general access to the system. They have read-only right and can interact with the system by requesting a list of disruptions and more details for a selected from them disruption. They have also access to the disruption history of the network. This use case was not in the initial requirements and was identified as a useful feature during demonstrations of the prototype to TFL.

- **Operator** is assumed to be a CentreComm staff member who is required to authenticate into the system. This would allow them the extra functionalities of adding comments to disruptions for others to see. Such users also have the functionality, of hiding and showing disruptions that are currently detected in the bus network, available to them.

- **Administrator** users have the most privileges of all type of users of the system. In addition to the above actions they are also allowed to view and change the configuration settings of the system. This is to allow easier configuration of the application.

4.2 System architecture

In figure A.2 in Appendix A the architecture diagram of the system could be seen. The overall architecture is following a four-tier architecture with an model view controller (MVC) pattern for the user interface. This architecture allows us to decompose to system into separate subsystems where lower tiers do not depend on higher tiers. This system allows for the implementation details of subsystems to be changed without affecting other components if the interfaces do not change.

As it can be seen from the diagram the system is divided in four main layers:

- Representation Layer - responsible for visualisation of the user interface. It consists of a number of user views.
- Representation Control Layer - responsible for the control/transition between user interface windows. In this layer I have made use of the front controller pattern [12] as it allows us to combine the common logic in one controller.
- Application Logic Layer - this layer is the functional core of the system. This is where all the business logic is encoded and corresponding calculations are done. The most important part of the system is the Disruption Engine which is responsible for:

    1. Monitoring for new feeds and processing them.
    2. Updating the bus network status (calculating delays and detecting disruptions).
    3. Writing the changes to the system database.

  The Disruption Model is the other major component of the system. It is responsible for retrieving the disruptions and their details from the database and providing them to the representation control tier.
- Data Layer - this the the data repository layer. It consists of comma separated values (CSV) files representing the AVL feeds that are being

pushed to the system. Here we also have the system database which contains all the configuration settings parameters and the output of the engine (disruptions and their details that are detected by the tool).

This architecture diagram represents coarse grained view of the system to be implemented. Each of the components presented above could be implemented as a number of smaller components and modules depending on the specific technologies.

4.3 State Machine

State machine diagrams are useful in explaining what in what states the system could be and how it transitions from one state to the other. The state machine diagram for our system is presented in figure A.5 in Appendix A. This diagram represent the states in which the disruption engine could reside and the available transitions. As it can be seen from the figure once the engine is initialised correctly it enters a continuous loop. This loop represents the engine waiting for new feeds to be detected by the system. Once detected they are processed and the bus network state is updated. If the change of the bus network has not changes from what was previously observed the tool does not make any changes to the database, else it would write all the changes that have been detected and calculated. In case the system fails to connect or write the changes to the database the system would terminate.

4.4 Class organisation

Class decomposition diagrams are the main building block of object oriented programming paradigm. They are widely used tool for the organisation and design of a software system. The diagram consist of classes, which encapsulate some attributes (state) and methods (functionalities), and the associations between the individual class. Each class is depicted by a box with the name of the class on top and its member attributes and methods below. Associations are represented by lines connecting those classes where a relation exists. The class diagram for our system is presented in figure A.3 in Appendix A. In the below subsections I have provided some explanation of the most important classes in the class diagram.

4.4.1 iBusMonitor

This could be viewed as the entry point of the tool engine. It most notable attribute is the link to the configuration file specifying the database connection properties. It can have a number of bus network and is responsible for initialising the tool and monitoring. This means it encapsulates the logic for listening for new feed files being published.

4.4.2 BusNetwork

This class represent a given bus network. In the context of this project this can be viewed as the TFL bus network. This object encapsulates all attributes that relate to the network state and its behaviour. Each bus network consists of at least one bus stop and at least one route otherwise it does not make any sense to have a network without any stops or routes.

4.4.3 BusStop

The bus stop class is representation of a bus stop in the bus network. It consists of a number of expected attributes that a stop would have. We also make the assumption that a single stop can belong to only one bus network.

4.4.4 Route

This class is one of the most important in the context of this project as it encapsulates the state of a single route in the network. Each route is associated with at least one run (in most cases each route would have two runs In/Outbound) and a number of observations.

4.4.5 Observation

The observation class captures the state and functionality of a single observation. By observation we mean a single reading extracted from the AVL data input. This reading is expected to be coming from a single bus logged on a given bus route, thus it would belong to this route.

4.4.6 Run

This object represent a route's run state and methods. Its main properties consist of list of consecutive readings made on this run for each logged bus. It also provides interface for detecting and updating disruptions on this run, thus it needs to keep track of the disruptions that were previously seen along this run.

4.4.7 Section

This is the most basic part of a bus route apart from the bus stop. Each section represents the part of the route between two consecutive bus stops along this route. This means it is characterised by a start and end stop and the sequence of this section along the route. In this class we calculate the delay per individual section (more on how this is done in the following chapter).

4.4.8 Disruption

This class simply captures all the attributes of a disruption. Each disruption would have an identification number, sections between the disruption is observed and the corresponding delay and trend. It also provides methods for updating and saving the details to the database.

Chapter 5

Implementation

This chapter aims to present the reader with explanation of the key implementation aspects. These include major challenges, decisions and problems that have been encountered and taken during the course of this project. I have tried to avoid going into too much technical details except where this is essential and provides the reader with better insight and understanding of the material.

The system I have developed as a prototype which to satisfy our aims consists of two sub systems. These are the disruption engine which address the first aim of detecting disruptions in the bus network and a web front end application which to visualise the calculated disruptions. Below I have presented the major implementation decisions and challenges that were faced during the development of this system. We begin with the disruption engine as this is where core functionality lays. Then we cover the visualisation part which can be viewed as an extension of the disruption engine.

5.1 Disruption Engine

The disruption engine is implemented based on the design given in the previous chapter. The main implementation language used to the implementation is Scala. Scala is both functional and object oriented language [22]. It is a type-safe Java Virtual Machine (JVM) language [22]. This means that it is compatible with existing Java[1] code which allows for reuse of existing Java libraries. Scala was first introduced back in 2003, however it has been only in the past few years that it had gained more popularity. In addition Scala enables the programmer to write more concise and clear code than Java. The decision of using Scala has also been influenced by the fact that I have good knowledge of the object oriented programming paradigm as well as experience in Java. This allowed me to quickly learn Scala and put it into use.

The disruption engine need to have an accurate internal representation of the bus network. TFL's bus network and any other bus network usually consists of bus routes. Each bus route often has multiple runs (directions - e.g. inbound and outbound). In turn each run consists of a sequence of bus stops that the bus passes through. In addition to these typical bus network components for our implementation we also have the notion of a section. By this we mean a pair of consecutive bus stops along a given run of a given route. In figure 5.1 below we can see an example for route 15 outbound where we have depicted two section X (between Leman Street and Tower Of London Stops) and Y (from Tower Of London to Great Tower Street). This means that if we have n stops on a given run then we have $n-1$ sections on the same run.

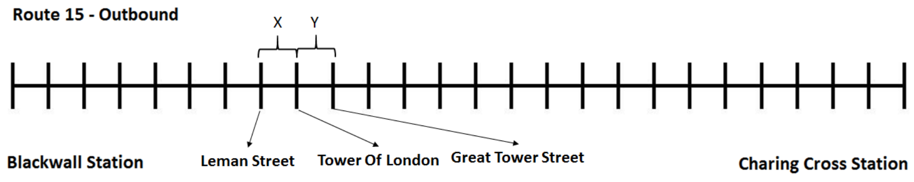


Figure 5.1: Example of a section

Our tool makes use of a PostgreSQL[2] database for storing all configuration parameters and also for storing the required information for the bus network representation. In figure A.8 in Appendix A I have presented the relational model on which the database is based. The initial versions of the prototype used a CSV files as a means for storing the output. The decision to switch the

[1]https://www.java.com/en/download/faq/whatis_java.xml
[2]http://www.postgresql.org/about/

flat file storage for a database is based on a number of things. The most important being is that using a database rather than CSV files we could keep historical data of the detected disruption for future analysis which is accomplished much easier using a proper database. Another advantage for the database approach is that it offers better concurrency support out of the box. Unlike the flat files which if manipulated concurrently could result in inconsistent state. Also having a database means that our disruption engine would require only details for establishing a connection to the respective database where it can read all other information that it requires. Otherwise it would require a number of other files and parameters to be defined which is not as maintainable as having one single database containing all of the configuration parameters as well as data for the bus network representation. For the above reasons I have decided to use PostgreSQL as it is advanced open source relation database management system. PostgreSQL has very good document and community support which big advantage for any technology. Another advantage for using PostgreSQL for our implementation is that it ensures reliability and data integrity [19]. Also it supports Listen[3] and Notify[4] functionality which could be used for real-time updating of the web application along with Server Sent Events (SSE) [38].

5.1.1 Bus Network representation

In order for the engine to have full bus network representation it requires information of the bus routes and the bus stops in the network. This information is freely available to anyone on TFL website [5]. It consists of two CSV files, one containing information on all bus routes and one for all bus stops in the network. The bus file contains a list of bus routes respectively has a one or more runs which consists of sequence of bus stops. In our implementation we assume that this information is preloaded into the database. This pre load consists of simply extracting the information from the CSV files obtained from

[3]http://www.postgresql.org/docs/9.1/static/sql-listen.html
[4]http://www.postgresql.org/docs/9.1/static/sql-notify.html
[5]http://www.tfl.gov.uk/info-for/open-data-users/our-feeds

TFL website into the respective tables (BusStops and BusRouteSequences). In addition to this we need to pre load also the sections which will allows us to store individual section information. Currently sections are being generated manually however this could be easily automated, but this is not in the scope of this project. Then once the engine is started it would read and load from the database all bus routes and respective sections. This results in the BusNetwork class storing a HashMap which maps a bus route name to the corresponding Route object. In turn the Route class maintains a list of all runs for the respective route. BusStop information (apart from the bus stop LBSL code which we use as an id) is only loaded from the database on request. This whole process is part of the initialisation of the monitoring tool along with pre loading some other environment configuration parameters from the database. It happens only once throughout the execution of the tool and takes place just after the engine is started.

5.1.2 Monitoring and processing new feeds

Once the system is initialised the tool will continuously monitor a specified directory (configurable from the database) for new feeds being written (pushed). Once new feed files are detected they are picked up and processed by the engine. The processing consists of extracting the data of interest and calculating the time lost by buses on average for each section. Extracting the data means reading the CSV feed file line by line. Each line would be a data (observation) for a given bus thus we associate each observation with the route that is currently logged on. Once the observations are extracted they are sorted by the time of the data field and any data that is older than a given predefined threshold (e.g. 120 minutes - this is configurable) is discarded. Once a given feed file is processed it is moved to a predefine processed feed directory. This completes the feed file processing step. Feed processing is performed on batches of feeds (see the state machine diagram on figure A.5 in Appendix A). This means that once the engine detects new feed(s) in the directory it is monitoring it will process all new feed files.

5.1.3 Bus network state update

Once feed processing has finished the system need to update the bus network state. This consists of a number of steps. First we need to calculate the lost time per section. This process is done iteratively for every route in the network that has active buses (readings have been transmitted in some predefined interval of time e.g. 90 minutes). Each bus observation is then taken on a given route (see Figure 5.2). At least two or more observations are required in order to calculate the time loss for a section. In the example below (figure 5.2) we can take the first reading x_1 and the second reading x_2. The difference in the

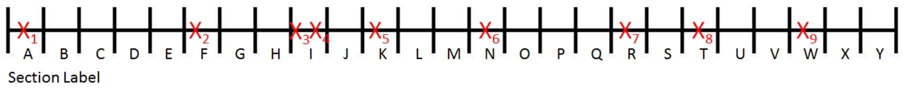
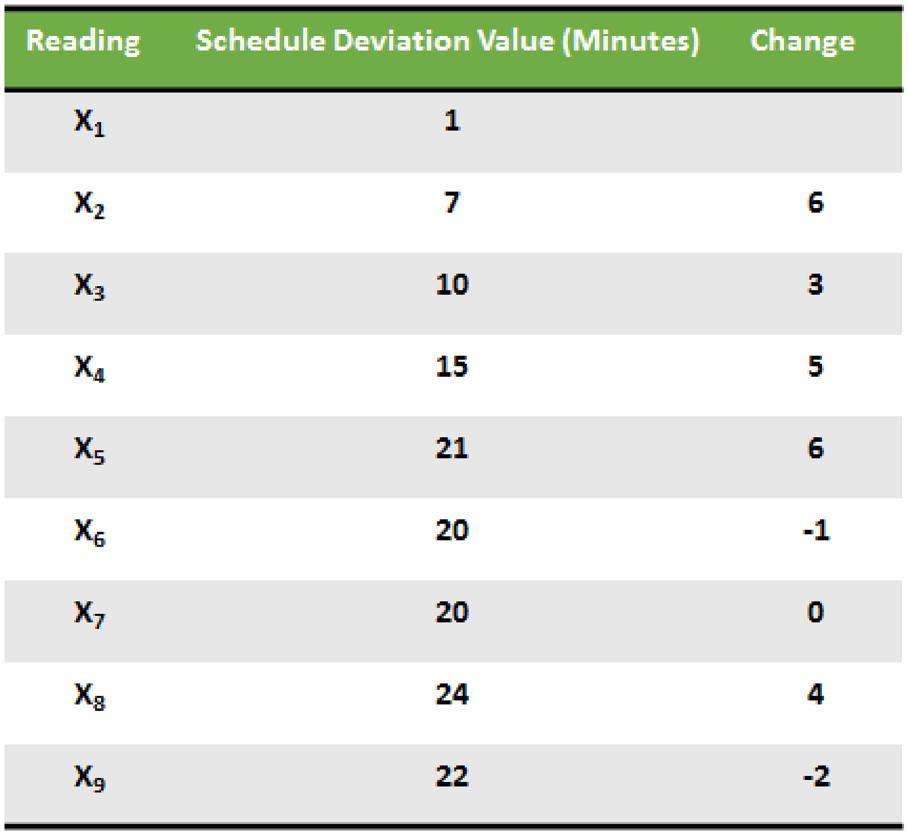


Figure 5.2: Example of observations

schedule deviation between x_2 and x_1 is then calculated (see table figure 5.3).

Once we have calculated the schedule deviation change between two stops we need to assign it to the respective sections. From our example we can see that reading x_1 has been sent somewhere from section A and that x_2 from section F. This means that the bus have travelled through 6 sections (A to F) and during that time it has lost 6 minutes. The problem is that we do not know where exactly this delay has happened. It is possible that there was a delay just in one of the section or it could have been distributed along all or few sections. For this reason what we do is to distribute this lost time evenly along all sections in-between the two readings. In our example this would mean that we would assign 1 minute to each of sections A, B, C, D, E and F. Then we take the next pair of observation in this case x_2 and x_3 and do the same procedure. Every time we have new time loss value for a section we add it to the existing lost time for the section and associate it with the newest observation time of the data. In this case this means that section F will have time loss value of $1+\frac{3}{4}$ and time-stamp equal to the time of data of observation x_3. If we however take the case of reading x_3 and x_4 we know for sure that the



Reading	Schedule Deviation Value (Minutes)	Change
X_1	1	
X_2	7	6
X_3	10	3
X_4	15	5
X_5	21	6
X_6	20	-1
X_7	20	0
X_8	24	4
X_9	22	-2

Figure 5.3: Example schedule deviation calculations for the example in figure 5.2

delay has occurred in section I thus we assign the 5 minutes lost time only to section I. Repeating the above steps for each bus on a route we end up with a list of values representing the delay (time lost) and the time (time of the data of the latter observation in the examples given above this means we take the time of the data of x_2 and x_4 respectively) when this has occurred.

Once we have a list of these values for each section of each bus route we need to calculate the weighted moving average of this data. To do this we firstly need to sort this list of values for each section of a route run by the time of the observation in ascending order. Then we calculate the weighed moving average by assigning the oldest data weight of 1 and the newest data weight of n (n is the number of data entries of a section). Calculating the weighted moving average instead of a simple average we put more weight and the newer data and also help dampen the effect of a single irregularity (e.g. a bus has experienced a technical fault).

Doing the above steps we obtain a value representing the WMA time loss (delay) for each section (see figure 5.4). The next step then is to examine each route run and its sections in particular and check if any of them are disrupted. To accomplish this we firstly check if the total cumulative lost time, of the sections of the respective run, is greater than or equal to some predefine minimal threshold (this is configurable). In case this does not hold (e.g it is below the minimal threshold) the algorithm will move onto the next run and will consider this one as clear (without any problematic sections). Any negative values (e.g where buses have gained time) are treated as 0 as we are only interested if there are any problematic sections of the route.

However if for example we assume that the minimal threshold is 20 minutes and take the example from figure 5.4 we can clearly see that the cumulative lost time is greater than or equal 20. In such cases we need to look more closely at this route run and try to identify the sections which are causing the most delay. We do this by searching for a number of consecutive sections which have their sum of the delay time greater or equal to some predefined threshold (e.g. 20 minutes). However small delays (e.g of 1-2 minutes) are treated as 0 as there always some slight variations and we are only interested in major disruption which are beyond the control of individual bus operator companies. If we take the example presented in figure 5.4 we can see that there seems to be some significant problem between sections L and O. The sum of the delay is $7 + 14 + 12 + 5 = 36$ minutes which is greater than our example threshold of 20 minutes. This results in the engine detecting and outputting a disruption between those sections.

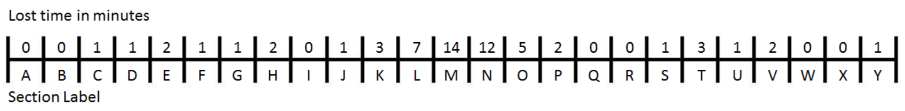


Figure 5.4: Example of a disruption

However we have to limit the number of consecutive section we look at a single time as for example we consider the below case (see Figure 5.5).We can have a long route run with very small loss of time per section (as in the below example of the order of 1 to 3 minutes) but overall they might add up to 20 or more minutes in particular if the route run is long (e.g has 30 or more stops). This however does not represent a single problematic hotspot and is responsibility of the bus operators to manage such cases and not CentreComm. For this reason our algorithm would mark the start of a disruption as the first section it encounters with a greater WMA delay value and it would continue expanding this disruption until it encounter a section with value less than the minimal threshold for section time loss value. Then it checks if the total delay for the detected sections is greater than or equal to the minimal disruption threshold. If true it outputs it as a disruption affecting those sections. This is done for the rest of the route run even if some disruptions are already detected at the beginning of the run of a particular route.

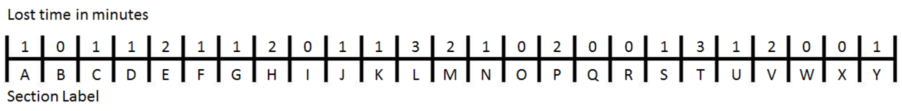


Figure 5.5: Example of no disruption

Every detected disruption or one that has changes its state is updated and written to the database. This allows for the user interface to update itself with the latest information. The engine also writes a snapshot of the sections state of every route run in the network which has active disruptions. We only write data for the sections which belong to runs with disruption due to performance considerations. For example if we take the London bus network which has 680 routes most of which have 2 runs (outbound and inbound) and each run on average has 30 stops it means we have more than 40000 sections. Updating them each time would take significant computational time (e.g. couple of seconds depending on the machine and the location of the database) and also would be waste of space. For this reason our implementation only updates sections which belong to disrupted runs as this information is utilised for displaying detailed graphs in the user interface.

5.2 Graphical User Interface

The engine implementation described above address only half of our requirements. In order to fully meet the project goals we need to be able to visualise the detected disruption in the network. For this reason I have decided to implement the visualisation part of the system as a separate web application. The advantages of this approach are that this provides universal access to the information. We only need to deploy this application once and it can be accessed from multiple clients running different operating systems and even devices. I have decided to use Ruby running on Rails framework. The reason for this is its recent popularity and wide support in terms of third party libraries and community information and guides.

The Rails framework [REFERENCE] is full stack open source web framework implemented in Ruby. It is relatively easy to use and maintain and it support a wide range of software engineering paradigms and patterns. The main ones being Model View Controller (MVC), Don't Repeat Yourself (DRY), Convention Over Configuration (CoC) and the Active Record pattern [12]. I have also made use of Foundation Cascading Style Sheets (CSS) front-end framework. This framework has enabled quick prototyping as it has a rich library of predefined components. However its main advantage over some other similar frameworks is its notion of grid which allows for quick and easy implementation of responsive websites. Making the web application with responsive design allows us to reach more platforms and client systems with a single application. This allows us with little additional implementation effort and time to achieve results which look good on both large screens (desktop computers, laptops etc.) as well as small mobile devices (e.g. smart phones, tablets etc.). This framework also has the advantages of good support in terms of documentation and add-ons and it is lightweight. Sample of what the system user interface looks like can be seen in appendix A [REFERENCE TO APPENDIX].

The ruby on rails web application is implemented following the MVC pattern. In our case it consists of few simple models representing the corresponding database tables, controllers and a number of views (some of which are partials). The models are implemented as Active Records which provide interfaces to the respective database tables. They provide create, read, update and delete (CRUD) functionalities. The controllers are responsible for processing the client request by parsing and checking for any parameters, credentials where necessary and responding with the requested information. The responses are in most cases rendered views containing the requested information.

One important aspect of the user interface is how to update the disruption list in real time whenever there are changes. This is achieved by implementing Ajax short polling. This means that we use asynchronous JavaScript on the client side to make request to the server at predefined intervals. Alternative methods for implementing the updating of the list include Web Sockets, Ajax Long Polling, Server Sent Events (SSE). The main advantages of using Ajax short polling over Web Sockets and SSE are that Ajax Short Polling is supported by all major web browsers unlike SSE which lack Internet Explorer support and it is easy and quick to implement it. The drawback of using Ajax polling is that in order to achieve near real time update the clients will need to make frequent request to the server which wastes network and server resources. SSE is probably the best approach in this scenario as it establishes a persistent long-term connection on which the server is able to push new data once it becomes available to all connected clients. However SSE are not supported by Internet Explorer which is one of the main web browsers CentreComm staff use and also SSE require a concurrency enabled server. Web Sockets provide a persistent two way connection between the client and the server however in our case we are mainly interested in pushing new data from the server to the client. There is very little information the clients need to send to the server and thus we have excluded this approach as viable one.

The disruptions at any point in time are visualised in tabular form. This provides instant awareness of what the network state is to the CentreComm operators. Disruptions are prioritised by their severity and are color coded. Further details for disruption are displayed on request of the user. This includes a graph representation of the delay observed on each section along the disrupted route. On this graph the cumulative delay for this route is also visible. When hovered the graph displays details for this section including the start and end stop of it and the number of observations.

5.2.1 Problems

[Problems: Diversions, Curtailments, Sparse data, Missing bus stops due to regular changes, undefined routes] One problem that was encountered is that there is no explicit information in the data that has been provided if a bus is on diversion. Diversions vary greatly in terms of length of the diversion (few or many stops) and duration of the implemented diversion (e.g. it can last half an hour or few weeks/months). What happens is such cases is that the bus would still transmit its position however the centre iBus server would not be able to calculate the schedule deviation. When such readings are observed by the engine they would simply be ignored. This is problematic and our system only ignores it in the current prototype as failure in computing the schedule deviation could also be caused by the bus not being properly logged into the iBus system (e.g. bus taking a break at end of route). This means that we lose some accuracy especially for longer diversions as any lost time would be distributed along the sections which have not been observed (where the bus has been on diversion). This however could also be the case if a bus skips or fails to send data. Again this could be because of number of reasons (e.g. weak signal due to high buildings etc.). Another problem with the available data is that we do lose some reading when the bus turns at the end of a run. We cannot simply assume that every bus would travel along a route from start to end and turn as there are occasions when buses are curtailed along the route. This can happen for a number of reasons including service regulation, heavy congestion/disruption along some sections of a route, bus driver going above the work hours allowed by law and more. Another problem of the approach taken is that because the system relies on monitoring the schedule deviation value and it does not treat differently calculations where during the first observation the bus is ahead or schedule and when it is already behind.

This means that it is possible that the bus driver is losing time on purpose for service regulations. Thus we can assume that our tool provides an upper bound of the disruption.

Chapter 6

Testing

Testing is important and integral part of any software development project. It is necessary to carry out testing throughout the implementation as well as once the system is completed in order to make sure it functions correctly according to the requirements and the design.

6.1 Unit Testing

As described in the previous chapters the system is composed of a number of classes each of them representing different part of the bus network with its own functionalities and responsibilities. Thought the development of the product a number of unit test have been carried out in order to ensure that each of the classes function correctly as per requirements and carries out the required task. Each module was tested separately before being connected to the rest of the system.

6.2 White Box Testing

Apart from the unit testing through the development of the tool we have made use of the debugging tool provided by IntelliJ IDEA integrated development environment (IDE) [1]. This was the main IDE used for the development of

[1] https://www.jetbrains.com/idea/

this project and has provided many useful tools and support. The built-in debugger allowed us to carry out inspection of the code during its execution. This means that we could halt the execution (in real-time) at specific point during the calculations and examine the state of the program (the values of all variables).

6.3 Functional Testing

The proposed prototype system in this project consisted of an engine responsible for calculating the disruption and a separate front end responsible for visualising the output from the engine. Because of this separation most of the functional testing was performed on the front end web application. This was carried out by manually testing the functionality of the web application and verifying that it resulted in the correct output. The user interface has been tested by me throughout development, some feedback has been received on it during demonstration and discussions with CentreComm and my supervisor. In addition to that I have asked family and friends to spend time and test the functionality of the user interface for this I provided them with a list of use cases and the expected behaviour. This has proved useful in identifying some issues with the functionality and the compatibility of the product with different web browsers and systems. All of the identified problems have been addressed and rectified after that.

6.4 Stress Testing

On of the main requirements for detecting the disruptions in the bus network is for this to be calculated in real-time. Because of this we need to make sure that the implementation is able to cope with large amounts of data quickly. Also as the system will run continuously with more data being made available for the system throughout its execution it need to be ensured that any memory that is occupied by data that is irrelevant (old) be discarded appropriately without creating any memory leaks.

The approach that I have taken in order to test the system for such performance issue consisted of firstly creating an automated script for simulating the feed pushing to the tool. This program takes two main input parameters. One is the directory in which the feeds for the simulation are being stored and the second main input is the rate at which these files are copied to the directory which is used by the disruption engine for monitoring. Currently as discussed previously in this report the AVL data files which were provided for this project are generated every 5 minutes. However this could be changed if the tool goes into production. For this reason we need to make sure that the proposed prototype is capable of dealing with large sets of data being pushed every 30 seconds (as this is the current rate iBus equipped buses transmit data). Apart from the rate at which new data is pushed to the tool the performance of our system depends mainly on how much data is pushed (e.g. during the night ours there are less active buses thus less data generated compared to daytime) and also on how many disruptions are detected in the system. The earlier is clearly obviously that if there is more data to be processed the system would naturally take longer. However the second statement is because we only update the database information for a route, run and section only if there is changes. This means that the update time could vary greatly depending on the number of changes at each network update. In order to minimise the impact of this the system is implemented in such a way that it makes all database updates after a network update in one transaction. This means that the main performance bottleneck of the system becomes the updating of the database. The initial versions of the prototype did not make use of a database, but rather wrote the output into flat CSV files. The reason for adding a database in place of the flat files is that the implementation change, so that it could keep historical state of the network which could later be used for further analysis and reports which are features that have received a lot of positive feedback from TFL.

Simulations have been run to make sure the system is capable of processing and updating itself in real-time. This tests have been carried out on a laptop running Intel Core i7-3610QM with 16GB DDR3 1600Mhz of ram and Windows

10 x64 bit operating system. The simulations consisted of feeding the tool with a week worth of data that was provided by TFL. The data was made of 21629 separate (each one for a single bus operator) AVL feeds which have been group into 2814 batches according to their timestamps. The total size of the sample is 1.07 Gigabyte (1076394754 bytes) and the average size of a single feed file is 49.77 Kilobytes (49766.27 bytes). We ran a number of simulations with this data keeping all parameters the same and re-initialising the system before each run. The only parameter we altered, to make sure the system is capable of handling data at higher rates, is the rate at which the feed simulation program pushed the AVL files. The results could be seen on figure A.10 in Appendix A. From the results we can conclude that even on a normal computer in a development environment the system is capable of producing output in almost real-time. The memory usage also could be seen in the results shown on figure A.10 in Appendix A. The memory was measured through the execution of the tool and readings were taken every minute. The memory usage is highly dependent on the amount of the data being processed and also depends on when the Java Virtual Machine(JVM[2]) garbage collector executes. The latter is because the system might have already removed all references to some object, however this memory would not be unallocated until the JVM garbage collector is called.

The system current implementation is updating each route in the network consecutively. However the system has been implemented with concurrent execution of this calculation in mind. This means that it could easily be updated such that each route is concurrently processed. This is also the case for the parsing of the CSV feed files and observation extraction. As the aim of this project is to built a prototype I have decided not to spend the limited project time on such optimisations as it is more important to prove if this is viable solution. As it can be seen from the stress tests carried out even without such parallelism in place the system produces results in reasonable amount of time.

[2]http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

Chapter 7

Results & Evaluation

Evaluation is important step of every software project yet it is sometime neglected. For our project we are only ably to objectively evaluate the performance of the disruption detection engine. Evaluation of the visualisation is very subjective and it will vary greatly between one user and another. For this reason we only rely on feedback from users especially CentreComm operator which we can use as a measure for the success of the particular visualisation we have implemented to complete this project. The disruption engine however needs to be properly evaluated. In order to evaluate it we need to answer three questions:

- What is being evaluated?
- How is it evaluated?
- What are the results have been obtained?

7.1 Engine

AS DISCUSSED IN THE BACKGROUND THERE ARE NO STUDIES DONE ON USING AVL DATA FOR DETECTING CONGESTION ON ARTERIAL URBAN TRANSPORT NETWORK

EXPLAIN THAT WE HAVE CREATED A SIMULATION FEED THREAD FOR COPYING THE FEEDS

To answer the first question we need to measure the ability of the prototype to detect accurately and in real-time disruptions in the network (this is one of the main aims of the project). This means that we need to consider two aspects:

1. Is the tool able to detect all disruptions that need to be detected?
2. Are those disruption detected in real-time?

In order to check if the system we have proposed can satisfy those requirements we need to run simulations and check if they result generated by the prototype match our expectations. One option to accomplish this is to use real data that was provided by TSG of TFL. However the problem with this approach is that we do not have a complete list of what was the actual state of the network at that time (the period of the sample data provided). It would be infeasible for us to manually analyse it in order to calculate what, when and for how long it should be picked up. This leads us to the seconds option which is to manually generate some test data which conforms to the iBus AVL data.

We have limited our test data to four scenarios as follows:

1. The schedule deviation value remain fairly constant with very minor changes across the route for every bus on that route. This scenarios should yield no disruption detections.
2. Single bus incident (e.g. bus breakdown, customer incident, error reading etc.). This again should not be picked up by the system.
3. General traffic scenario. This means that there is gradual schedule deviation increase at some point of the route run. This is the most typical real life scenario (e.g. rush hour traffic build up) and it should be detected by the system.

4. Incident (e.g. traffic collision) along a route. The data for this would represent a sudden increase in the schedule deviation. This needs to be detected by the tool.

Once we have generated the data for the above scenarios we need to run the system and feed it with the artificially generated input. We run each of the above scenarios separately and the system is re-initialised before each run. This allows us to have the system in the exactly same state for each test.

This tests allowed us to run multiple simulation in order to adjust the weighted moving average parameters. These are the weights and the moving average window. This parameter values are critical for the accuracy of our tool. They have direct impact on what is being detected by the tool and with what lag. We also altered the data validity parameter which represents when we discard any observations.

In order to calibrate our prototype and its output we ran a numerous simulation with the describer scenarios and different values for our parameters. Using our test generated data we have obtained best results with having the data validity set to 90 minutes, but then only have moving average window of 5. This means that we only consider the last 5 observations for a given section which have occurred in the last hour and a half. We also use weight of 1 to 5 (1 for the oldest and 5 for the most recent). This allowed for a balance output, as the system tended to overreact if we used an exponentially growing weights or it lagged behind if we used a greater moving average window. Also keeping data for the last 90 minutes in memory should work well for both low and high frequency buses, as from the data provided we do not have any indication of the type of the service.

7.2 Visualisation

The user interface Usability

Chapter 8

Professional & Ethical Issues

Throughout every stage of this project I have made every effort to follow the rules and guidelines that are set out by the British Computer Society (BCS) Code of Conduct & Code of Practice [4]. These are rules and professional standards that govern the individual decisions and behaviour. The main rules that apply almost to every software development project states the individual should:

- "have due regard for public health, privacy, security and wellbeing of others and the environment."[4]
- "have due regard for the legitimate rights of Third Parties"[4].

The whole project has been planned, designed and developed with both of these rules, as well as other rules and standards, in mind. The system makes use of a number of third party libraries and framework. However I have made explicitly the use of any such libraries and provided the according reference to the source of the original idea/product. I have also given references to any work or ides that I have made use of throughout the project.

I have also tried to make sure that the applications that were developed as part of this project do not pose any harm neither to the computers they are running on or interacting with nor to their users. The tool is expected to run 24/7 with a large number of files being processed every day. This means that we need to make sure does not contain any memory leaks as discussed in previous chapters. I have also used appropriate method to safeguard the database from any SQL injection [33] which could potentially alter the data unintentionally or without the appropriate permission. However it should be noted that this is a prototypical system and not a fully working and security proof production version.

Chapter 9

Conclusion & Future Work

[INTRO PARAGRAPH - NOT SECTION]

9.1 Future Work

This project has examined carefully the needs of CentreComm for automation of their current work flow which could lead to better operation, management and cut in costs of controlling London's bus network. It has reviewed the literature for related work and approaches and has resulted in a prototypical tool being designed and implemented. This tool has provided some initial results as seen in the previous chapter. There is however a lot that could be improved and built upon on it. In this section I provide some suggestions and direction for taking this work further.

One simple extension that could take this project one step ahead is to try to implement a peak/valley detection algorithm which could distinguish between incidents and just increased traffic congestion. This was briefly discussed in chapter 2 of this report however due to time limits of this project has not been looked in more depth or considered for design and implementation.

Another feature that has received a lot of positive feedback from different levels in TFL is the ability to generate complex historical reports of what has been the network state. According to TFL this would provide more objective view of what has happened in the transport network and who should take the responsibility.

More historical data could also be employed and correlated with the real-time data received. Examples of such data may include weather data, time of the day/week, workdays compared to weekends and public holidays etc. This could improve the accuracy of the system as well as bring to light some persisting problems under given environments.

The increased popularity and usage of various AVL systems being used by different fleets could be used along with the data present in this report. This could include taxi fleets [28], delivery service fleet, emergency services and many more. Having more information and especially from different sources could result in one central congestion monitoring system which could more accurately and in shorter-terms calculate and predict traffic conditions in the arterial road network of the city. There is the potential for employing data from GPS enabled devices that most people of the general travelling public own these days. This something is already being investigated [34] and I can see it will get more attention in the near future.

9.2 Conclusion

This report proposed a prototypical tool for detecting disruptions in London bus network. This is achieved by employing moving average smoothing technique for analysis of the time series data presented. The main strength of this approach is its simplicity. This is growing area of interest for intelligent transportation systems (ITS) and I expect to receive much more attention in near future with the rise in AVL data availability from different sources. I have also given some directions and proposal for improving and driving this work further.

This project has taught me a lot about transportation systems and networks. It has given me great insight into the complex operations that go into operating and managing large bus networks as is London's one. In addition to that I have also learnt a lot about different statistical techniques available for analysis of time series data. Such techniques are very useful and could be applied not only in the domain of this project, but in every problem where there is time series data which needs to be analysed.

The software development approach taken for developing the prototype for this project seemed to work well. It has allowed me to gather useful feedback early on and to refine the project requirements as initially the user requirements were very broad and vague. This is probably due to the fact that there is not much closely related work previously done.

One thing that did not work so well is the evaluation of the software system. This is mainly owned to delay in planning and carrying out this evaluation. However this is something I have now learnt and would help me better plan my future projects I undertake.

References

[1] Mehmet Altinkaya and Metin Zontul. Urban bus arrival time prediction: A review of computational models. *International Journal of Recent Technology and Engineering (IJRTE)*, 2(4):164–169, 2013.

[2] Greater London Authority. Number of buses by type of bus in london. `http://data.london.gov.uk/dataset/number-buses-type-bus-london`, 2014. Online; accessed 22-October-2014.

[3] Hamed Azami and Saeid Sanei. Spike detection approaches for noisy neuronal data: Assessment and comparison. *Neurocomputing*, 133(0):491 – 506, 2014.

[4] BCS. Bcs code of conduct. `http://www.bcs.org/category/6030`, June 2011. Online; accessed 6-April-2015.

[5] A.I. Bejan, R.J. Gibbens, D. Evans, A.R. Beresford, J. Bacon, and A. Friday. Statistical modelling and analysis of sparse bus probe data in urban areas. In *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pages 1256–1263, Sept 2010.

[6] J. L. Connell and L. Shafer. *Structured Rapid Prototyping: An Evolutionary Approach to Software Development.* Yourdon Press, Upper Saddle River, NJ, USA, 1989.

[7] Transport for London. Transport for london's ibus wins innovation award. `https://www.tfl.gov.uk/info-for/media/press-releases/`

`2008/march/transport-for-londons-ibus-wins-innovation-award`, March 2008. Online; accessed 2-April-2015.

[8] Transport for London. `https://www.tfl.gov.uk/cdn/static/cms/documents/uploads/forms/lbsl-tendering-and-contracting.pdf`, 2008. Online; accessed 2-April-2015.

[9] Transport for London. All london's buses now fitted with ibus. `https://www.tfl.gov.uk/info-for/media/press-releases/2009/april/all-londons-buses-now-fitted-with-ibus`, April 2009. Online; accessed 2-April-2015.

[10] Transport for London. `https://www.tfl.gov.uk/info-for/media/press-releases/2009/may/centrecomm-celebrates-30-years-keeping-londons-buses-moving`. Online, May 2009. Online; accessed 2-December-2014.

[11] Transport for London Media. `https://www.tfl.gov.uk/info-for/media/press-releases/2014/may/annual-passenger-journeys-on-london-s-buses-top-2-4-billion`. Online, May 2014. Online; accessed 2-December-2014.

[12] Martin Fowler. *Patterns of enterprise application architecture*. Addison-Wesley, Boston, 2003.

[13] Everette S. Gardner. Exponential smoothing: The state of the art. *Journal of Forecasting*, 4(1):1–28, 1985.

[14] Jianhua Guo, Wei Huang, and Billy M. Williams. Adaptive kalman filter approach for stochastic short-term traffic flow rate prediction and uncertainty quantification. *Transportation Research Part C: Emerging Technologies*, 43, Part 1(0):50 – 64, 2014. Special Issue on Short-term Traffic Flow Forecasting.

[15] Ryan Jay Herring. Real-time traffic modeling and estimation with streaming probe data using machine learning. 2010.

[16] N.B. Hounsell and B.P. Shrestha. Avl based bus priority at traffic signals: a review of architectures and case study. *European Journal of Transport and Infrastructure Research*, 5(1):13–29, June 2005.

[17] N.B. Hounsell, B.P. Shrestha, and A. Wong. Data management and applications in a world-leading bus fleet. *Transportation Research Part C: Emerging Technologies*, 22(0):76 – 87, 2012.

[18] Siem Jan Koopman. Exact initial kalman filtering and smoothing for nonstationary time series models. *Journal of the American Statistical Association*, 92(440):1630–1638, 1997.

[19] Reuven M Lerner. Open-source databases, part iii: choosing a database. *Linux Jurnal*, 2007.

[20] J.A. Livermore. Factors that impact implementing an agile software development methodology. In *SoutheastCon, 2007. Proceedings. IEEE*, pages 82–86, March 2007.

[21] BBC News. Extra 500 buses planned for growing capital before 2021. `http://www.bbc.co.uk/news/uk-england-london-30285777`, 2014. Online; accessed 2-December-2014.

[22] Martin Odersky, Lex Spoon, and Bill Venners. *Programming in scala.* Artima Inc, 2008.

[23] Object Management Group (OMG). Mda. `http://www.omg.org/mda/`, August 2014. Online; accessed 2-April-2015.

[24] Object Management Group (OMG). Uml. `http://www.uml.org/`, April 2015. Online; accessed 2-April-2015.

[25] Yan Qi and Sherif Ishak. A hidden markov model for short term prediction of traffic conditions on freeways. *Transportation Research Part C: Emerging Technologies*, 43, Part 1(0):95 – 111, 2014. Special Issue on Short-term Traffic Flow Forecasting.

[26] "Clarke R., Bowen T., and J" Head. Mass deployment of bus priority using real-time passenger information systems in london. In *Proc. European Transport conference, 2007.*, Leeuwenhorst, Netherlands, 2007.

[27] Clarke R., Bowen T., and Head J. Mass deployment of bus priority using real-time passenger information systems in london, 2007.

[28] Mahmood Rahmani, Haris N Koutsopoulos, and Anand Ranganathan. Requirements and potential of gps-based floating car data for traffic management: Stockholm case study. In *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pages 730–735. IEEE, 2010.

[29] Mohsen Ramezani and Nikolas Geroliminis. On the estimation of arterial route travel time distribution with markov chains. *Transportation Research Part B: Methodological*, 46(10):1576 – 1590, 2012.

[30] Steve Robinson. Ticket info on buses in service, 2010. Proposed Change Paper Provided by TFL.

[31] A. Sboner, A. Romanel, A. Malossini, F. Ciocchetta, F. Demichelis, I. Azzini, E. Blanzieri, and R. DellâĂŹAnna. Simple methods for peak and valley detection in time series microarray data. In Patrick McConnell, SimonM. Lin, and Patrick Hurban, editors, *Methods of Microarray Data Analysis V*, pages 27–44. Springer US, 2007.

[32] Robert H Shumway and David S Stoffer. *Time series analysis and its applications: with R examples.* Springer Science & Business Media, 2010.

[33] Zhendong Su and Gary Wassermann. The essence of command injection attacks in web applications. *SIGPLAN Not.*, 41(1):372–382, January 2006.

[34] Arvind Thiagarajan, James Biagioni, Tomas Gerlich, and Jakob Eriksson. Cooperative transit tracking using smart-phones. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, pages 85–98. ACM, 2010.

[35] Trapeze. London bus services limited (lbsl). `http://www.trapezegroup.eu/london-bus-services-limited-lbsl`. Online; accessed 2-April-2015.

[36] D Ventzas and N Petrellis. Peak searching algorithms and applications. In *The IASTED International Conference on Signal and Image Processing and Applications˜ SIPA 2011*, 2011.

[37] Eleni I. Vlahogianni, Matthew G. Karlaftis, and John C. Golias. Short-term traffic forecasting: Where we are and where weâĂŹre going. *Transportation Research Part C: Emerging Technologies*, 43, Part 1(0):3 – 19, 2014. Special Issue on Short-term Traffic Flow Forecasting.

[38] W3C. Server-sent events. `http://www.w3.org/TR/eventsource/`. Online; accessed 2-April-2015.

[39] Jian Wang, Wei Deng, and Yuntao Guo. New bayesian combination method for short-term traffic flow forecasting. *Transportation Research Part C: Emerging Technologies*, 43, Part 1(0):79 – 94, 2014. Special Issue on Short-term Traffic Flow Forecasting.

[40] Alan Wong and Nick Hounsell. Using the ibus system to provide improved public transport information and applications for london. Paper 01753, July 2010.

[41] Jinsoo You and Tschangho John Kim. Empirical analysis of a travel-time forecasting model. *Geographical Analysis*, 39(4):397–417, 2007.

Appendix A

User Guide

You must provide an adequate user guide for your software. The guide should provide easily understood instructions on how to use your software. A particularly useful approach is to treat the user guide as a walk-through of a typical session, or set of sessions, which collectively display all of the features of your package. Technical details of how the package works are rarely required. Keep the guide concise and simple. The extensive use of diagrams, illustrating the package in action, can often be particularly helpful. The user guide is sometimes included as a chapter in the main body of the report, but is often better included in an appendix to the main report.

A.1 Installation

Link to gitHub repository

A.2 Execution

Link to gitHub repository

A.3 Dependencies

Link to gitHub repository Scala and Java versions, scalatest library, PostgreSQL, scala xml library, scala geo coordinate library, scala logger Ruby on Rails, Ruby version, Gems that have been used Foundation, jQuery - **Include the specific versions**

Appendix B

Source Code

Complete source code listings must be submitted as an appendix to the report. The project source codes are usually spread out over several files/units. You should try to help the reader to navigate through your source code by providing a “table of contents” (titles of these files/units and one line descriptions). The first page of the program listings folder must contain the following statement certifying the work as your own: “I verify that I am the sole author of the programs contained in this folder, except where explicitly stated to the contrary”. Your (typed) signature and the date should follow this statement.

All work on programs must stop once the code is submitted. You are required to keep safely several copies of this version of the program - one copy must be kept on the departmental disk space - and you must use one of these copies in the project examination. Your examiners may ask to see the last-modified dates of your program files, and may ask you to demonstrate that the program files you use in the project examination are identical to the program files you had stored on the departmental disk space before you submitted the project. Any attempt to demonstrate code that is not included in your submitted source listings is an attempt to cheat; any such attempt will be reported to the KCL Misconduct Committee.

You may find it easier to firstly generate a PDF of your source code using a text editor and then merge it to the end of your report.

There are many free tools available that allow you to merge PDF files.

B.1 Engine

SourceCode/Disruption.scalaframe

```scala
package lbsl

import java.sql._
import java.util.Date

import _root_.utility.{DBConnectionPool, Environment}
import org.slf4j.LoggerFactory

/**
 * Created by Konstantin on 04/02/2015.
 */
class Disruption(private var sectionStartIndex: Integer,
                 private var sectionEndIndex: Integer,
                 private var sectionStart: String,
                 private var sectionEnd: String,
                 private var delaySeconds: Double,
                 private var totalDelaySeconds: Double,
                 private val timeFirstDetected: Date) {

  private var id: Integer = null
  private val logger = LoggerFactory.getLogger(getClass().getSimpleName)
  private var trend: Integer = Disruption.TrendWorsening

  def getSectionStartIndex: Integer = sectionStartIndex

  def getSectionEndIndex: Integer = sectionEndIndex

  def getSectionStartBusStop: String = sectionStart

  def getSectionEndBusStop: String = sectionEnd

  def getTotalDelay: Integer = {
    return totalDelaySeconds.toInt
  }

  def getTotalDelayInMinutes: Integer = {
    return getTotalDelay / 60
  }

  def getDelay: Integer = {
    return delaySeconds.toInt
  }

  def getDelayInMinutes: Integer = {
    return getDelay / 60
  }

  def getTimeFirstDetected: Date = timeFirstDetected

  def getTrend: Integer = trend

  def update(newDelaySeconds: Double, newTotalDelaySeconds: Double):
      Unit = {
    update(sectionStartIndex, sectionEndIndex, sectionStart,
        sectionEnd, newDelaySeconds, newTotalDelaySeconds)
  }

  def update(newSectionStartIndex: Integer, newSectionEndIndex:
      Integer, newSectionStart: String, newSectionEnd: String,
      newDelaySeconds: Double, newTotalDelaySeconds: Double): Unit = {
    //TODO: Consider the section size for the trend as well
```

```
    val oldSectionSize = this.sectionEndIndex - this.sectionStartIndex
    this.sectionStartIndex = newSectionStartIndex
    this.sectionEndIndex = newSectionEndIndex
    this.sectionStart = newSectionStart
    this.sectionEnd = newSectionEnd
    this.totalDelaySeconds = newTotalDelaySeconds
    if (newDelaySeconds > delaySeconds) {
      trend = Disruption.TrendWorsening
    } else if (newDelaySeconds < delaySeconds) {
      trend = Disruption.TrendImproving
    } else if (oldSectionSize < this.sectionEndIndex -
        this.sectionStartIndex) {
      trend = Disruption.TrendWorsening
    } else {
      trend = Disruption.TrendStable
    }
    this.delaySeconds = newDelaySeconds
  }

  //   TODO:Extend this to capture all cases
  def equals(that: Disruption): Boolean = {
    if (this.sectionStartIndex == that.sectionStartIndex ||
      this.sectionEndIndex == that.sectionEndIndex) {
      return true
    } else if (this.sectionStartIndex >= that.sectionStartIndex &&
        this.sectionStartIndex <= that.sectionEndIndex) {
      return true
    } else if (this.sectionEndIndex <= that.sectionEndIndex &&
        this.sectionEndIndex >= that.sectionStartIndex) {
      return true
    }
    return false
  }

```

```
def clear(date: Date): Unit = {
  updateDBEntry(date)
}

def save(route: String, run: Integer): Unit = {
  if (id == null) {
    id = Disruption.getNextId()
    newEntry(route, run)
  } else {
    updateDBEntry()
  }
}

private def updateDBEntry(clearedAt: Date = null): Unit = {
  var preparedStatement: PreparedStatement = null
  val query = "UPDATE \"Disruptions\" SET \"fromStopLBSLCode\" = ?,
      \"toStopLBSLCode\" = ?, \"delayInSeconds\" = ?, trend = ?,
      \"routeTotalDelayInSeconds\" = ?, \"clearedAt\" = ? WHERE id =
      ? ;"
  try {
    preparedStatement =
        Environment.getDBTransaction.connection.prepareStatement(query)
    preparedStatement.setString(1, sectionStart)
    preparedStatement.setString(2, sectionEnd)
    preparedStatement.setDouble(3, delaySeconds)
    preparedStatement.setInt(4, trend)
    preparedStatement.setDouble(5, totalDelaySeconds)
    if (clearedAt != null) {
      preparedStatement.setTimestamp(6, new
          Timestamp(clearedAt.getTime))
    } else {
      preparedStatement.setNull(6, Types.NULL)
    }
    preparedStatement.setInt(7, id)
```

```
      preparedStatement.executeUpdate()
    }
    catch {
      case e: SQLException => logger.error("Exception: with query ({})
          ", preparedStatement.toString, e)
    } finally {
      if (preparedStatement != null) {
        preparedStatement.close()
      }
    }
  }

  private def newEntry(route: String, run: Integer): Unit = {
    if (delaySeconds > 2400 || totalDelaySeconds > 2400) {
      logger.debug("New disruption added on route {} run {}with delay
          {}mins and total delay {}mins.", scala.Array[Object](route,
          run.toString, (delaySeconds / 60).toString,
          (totalDelaySeconds / 60).toString))
    }
    var preparedStatement: PreparedStatement = null
    val query = "INSERT INTO \"Disruptions\" (id, \"fromStopLBSLCode\",
        \"toStopLBSLCode\", route, run, \"delayInSeconds\",
        \"firstDetectedAt\", trend, \"routeTotalDelayInSeconds\")
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?);"
    try {
      preparedStatement =
          Environment.getDBTransaction.connection.prepareStatement(query)
      preparedStatement.setInt(1, id)
      preparedStatement.setString(2, sectionStart)
      preparedStatement.setString(3, sectionEnd)
      preparedStatement.setString(4, route)
      preparedStatement.setInt(5, run)
      preparedStatement.setDouble(6, delaySeconds)
      preparedStatement.setTimestamp(7, new
        Timestamp(timeFirstDetected.getTime))
      preparedStatement.setInt(8, trend)
      preparedStatement.setDouble(9, totalDelaySeconds)
      preparedStatement.executeUpdate()
    }
    catch {
      case e: SQLException => logger.error("Exception: with query ({})
          ", preparedStatement.toString, e)
    } finally {
      if (preparedStatement != null) {
        preparedStatement.close()
      }
    }
  }
}

object Disruption {

  private var id: Integer = null
  // NRT delay classifications:
  // Moderate - 0 - 20 min
  // Serious - 21 - 40 min
  // Severe - 41 - 60 min

  // final val SectionModerate: Integer = 10
  // final val SectionSerious: Integer = 20
  // final val SectionSevere: Integer = 40
  //
  // final val RouteSerious: Integer = 30
  // //40
  // final val RouteSevere: Integer = 50 //60

  final val TrendImproving = 1
  final val TrendStable = 0
```

```scala
final val TrendWorsening = -1


def getNextId(): Integer = {
  this.synchronized {
    if (id == null) {
      getMaxId()
    }
    id += 1
    return id
  }
}

private def getMaxId() {
  var connection: Connection = null
  var statement: Statement = null
  try {
    connection = DBConnectionPool.getConnection()
    statement = connection.createStatement()
    val rs = statement.executeQuery("SELECT max(id) FROM
        \"Disruptions\"")
    while (rs.next()) {
      id = rs.getInt(1)
    }
  }
  catch {
    case e: SQLException =>
        LoggerFactory.getLogger(getClass().getSimpleName).error("Exception:",
        e)
  } finally {
    if (statement != null) {
      statement.close()
    }
    if (connection != null) {
        DBConnectionPool.returnConnection(connection)
      }
    }
  }
}
```

B.2 Web Front End

B.3 Database

SourceCode/iBusDisruptionDB.backupframe

```
--
-- PostgreSQL database dump
--

-- Dumped from database version 9.4.1
-- Dumped by pg_dump version 9.4.1
-- Started on 2015-04-15 11:38:31

SET statement_timeout = 0;
SET lock_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SET check_function_bodies = false;
SET client_min_messages = warning;

DROP DATABASE "iBusDisruption";
--
-- TOC entry 2077 (class 1262 OID 16399)
-- Name: iBusDisruption; Type: DATABASE; Schema: -; Owner: postgres
--

CREATE DATABASE "iBusDisruption" WITH TEMPLATE = template0 ENCODING =
    'UTF8' LC_COLLATE = 'English_United States.1252' LC_CTYPE =
    'English_United States.1252';



ALTER DATABASE "iBusDisruption" OWNER TO postgres;


\connect "iBusDisruption"


SET statement_timeout = 0;
SET lock_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SET check_function_bodies = false;
SET client_min_messages = warning;


--
-- TOC entry 5 (class 2615 OID 2200)
-- Name: public; Type: SCHEMA; Schema: -; Owner: postgres
--


CREATE SCHEMA public;



ALTER SCHEMA public OWNER TO postgres;


--
-- TOC entry 2078 (class 0 OID 0)
-- Dependencies: 5
-- Name: SCHEMA public; Type: COMMENT; Schema: -; Owner: postgres
--


COMMENT ON SCHEMA public IS 'standard public schema';
```

```
--
-- TOC entry 187 (class 3079 OID 11855)
-- Name: plpgsql; Type: EXTENSION; Schema: -; Owner:
--

CREATE EXTENSION IF NOT EXISTS plpgsql WITH SCHEMA pg_catalog;



--
-- TOC entry 2080 (class 0 OID 0)
-- Dependencies: 187
-- Name: EXTENSION plpgsql; Type: COMMENT; Schema: -; Owner:
--

COMMENT ON EXTENSION plpgsql IS 'PL/pgSQL procedural language';



SET search_path = public, pg_catalog;

SET default_tablespace = '';

SET default_with_oids = false;

--
-- TOC entry 174 (class 1259 OID 16414)
-- Name: BusRouteSequences; Type: TABLE; Schema: public; Owner:
    postgres; Tablespace:
--

CREATE TABLE "BusRouteSequences" (
   route character varying(10) NOT NULL,
   run smallint NOT NULL,
   sequence smallint NOT NULL,
   "busStopLBSLCode" character varying(10)
```

```
);



ALTER TABLE "BusRouteSequences" OWNER TO postgres;


--
-- TOC entry 173 (class 1259 OID 16408)
-- Name: BusStops; Type: TABLE; Schema: public; Owner: postgres;
    Tablespace:
--


CREATE TABLE "BusStops" (
    "lbslCode" character varying(10) NOT NULL,
    code character varying(10),
    "naptanAtcoCode" character varying(20),
    name character varying(100),
    "locationEasting" integer,
    "locationNorthing" integer,
    heading character varying(4),
    "stopArea" character varying(15),
    virtual boolean
);



ALTER TABLE "BusStops" OWNER TO postgres;


--
-- TOC entry 176 (class 1259 OID 16434)
-- Name: DisruptionComments; Type: TABLE; Schema: public; Owner:
    postgres; Tablespace:
--


CREATE TABLE "DisruptionComments" (
    id integer NOT NULL,
    "disruptionId" integer,
    comment text,
    "operatorId" integer,
    "timestamp" timestamp with time zone DEFAULT now() NOT NULL
);


ALTER TABLE "DisruptionComments" OWNER TO postgres;

--
-- TOC entry 186 (class 1259 OID 24731)
-- Name: DisruptionComments_id_seq; Type: SEQUENCE; Schema: public;
    Owner: postgres
--

CREATE SEQUENCE "DisruptionComments_id_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER TABLE "DisruptionComments_id_seq" OWNER TO postgres;

--
-- TOC entry 2081 (class 0 OID 0)
-- Dependencies: 186
-- Name: DisruptionComments_id_seq; Type: SEQUENCE OWNED BY; Schema:
    public; Owner: postgres
--

ALTER SEQUENCE "DisruptionComments_id_seq" OWNED BY
    "DisruptionComments".id;
```

```
--
-- TOC entry 175 (class 1259 OID 16431)
-- Name: Disruptions; Type: TABLE; Schema: public; Owner: postgres;
    Tablespace:
--

CREATE TABLE "Disruptions" (
    id integer NOT NULL,
    "fromStopLBSLCode" character varying(10),
    "toStopLBSLCode" character varying(10),
    route character varying(10),
    run smallint,
    "delayInSeconds" double precision,
    "firstDetectedAt" timestamp with time zone,
    "clearedAt" timestamp with time zone,
    hide boolean DEFAULT false NOT NULL,
    trend smallint DEFAULT 0 NOT NULL,
    "routeTotalDelayInSeconds" double precision
);


ALTER TABLE "Disruptions" OWNER TO postgres;

--
-- TOC entry 183 (class 1259 OID 24637)
-- Name: Disruptions_id_seq; Type: SEQUENCE; Schema: public; Owner:
    postgres
--

CREATE SEQUENCE "Disruptions_id_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER TABLE "Disruptions_id_seq" OWNER TO postgres;

--
-- TOC entry 2082 (class 0 OID 0)
-- Dependencies: 183
-- Name: Disruptions_id_seq; Type: SEQUENCE OWNED BY; Schema: public;
     Owner: postgres
--

ALTER SEQUENCE "Disruptions_id_seq" OWNED BY "Disruptions".id;


--
-- TOC entry 172 (class 1259 OID 16400)
-- Name: EngineConfigurations; Type: TABLE; Schema: public; Owner:
     postgres; Tablespace:
--

CREATE TABLE "EngineConfigurations" (
    key character varying(60) NOT NULL,
    value character varying(500),
    editable boolean DEFAULT false NOT NULL
);


ALTER TABLE "EngineConfigurations" OWNER TO postgres;

--
-- TOC entry 178 (class 1259 OID 16445)
-- Name: Operators; Type: TABLE; Schema: public; Owner: postgres;
    Tablespace:
--

CREATE TABLE "Operators" (
    username character varying(100),
    password character varying(128),
    admin boolean,
    id integer NOT NULL
);


ALTER TABLE "Operators" OWNER TO postgres;

--
-- TOC entry 177 (class 1259 OID 16443)
-- Name: Operators_id_seq; Type: SEQUENCE; Schema: public; Owner:
    postgres
--

CREATE SEQUENCE "Operators_id_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER TABLE "Operators_id_seq" OWNER TO postgres;

--
-- TOC entry 2083 (class 0 OID 0)
-- Dependencies: 177
-- Name: Operators_id_seq; Type: SEQUENCE OWNED BY; Schema: public;
```

```
    Owner: postgres
--

ALTER SEQUENCE "Operators_id_seq" OWNED BY "Operators".id;



--
-- TOC entry 180 (class 1259 OID 24611)
-- Name: Sections; Type: TABLE; Schema: public; Owner: postgres;
    Tablespace:
--

CREATE TABLE "Sections" (
   id integer NOT NULL,
   route character varying(10),
   run smallint,
   "startStopLBSLCode" character varying(10),
   "endStopLBSLCode" character varying(10),
   sequence smallint
);


ALTER TABLE "Sections" OWNER TO postgres;

--
-- TOC entry 182 (class 1259 OID 24625)
-- Name: SectionsLostTime; Type: TABLE; Schema: public; Owner:
    postgres; Tablespace:
--

CREATE TABLE "SectionsLostTime" (
   "sectionId" integer NOT NULL,
   "lostTimeInSeconds" integer,
   "timestamp" timestamp with time zone DEFAULT now() NOT NULL,
    "serialId" integer NOT NULL,
    "numberOfObservations" integer DEFAULT 0 NOT NULL
);


ALTER TABLE "SectionsLostTime" OWNER TO postgres;

--
-- TOC entry 181 (class 1259 OID 24623)
-- Name: SectionsLostTime_sectionId_seq; Type: SEQUENCE; Schema:
     public; Owner: postgres
--

CREATE SEQUENCE "SectionsLostTime_sectionId_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER TABLE "SectionsLostTime_sectionId_seq" OWNER TO postgres;

--
-- TOC entry 2084 (class 0 OID 0)
-- Dependencies: 181
-- Name: SectionsLostTime_sectionId_seq; Type: SEQUENCE OWNED BY;
     Schema: public; Owner: postgres
--

ALTER SEQUENCE "SectionsLostTime_sectionId_seq" OWNED BY
     "SectionsLostTime"."sectionId";
```

```
--
-- TOC entry 184 (class 1259 OID 24691)
-- Name: SectionsLostTime_serialId_seq; Type: SEQUENCE; Schema:
    public; Owner: postgres
--

CREATE SEQUENCE "SectionsLostTime_serialId_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER TABLE "SectionsLostTime_serialId_seq" OWNER TO postgres;

--
-- TOC entry 2085 (class 0 OID 0)
-- Dependencies: 184
-- Name: SectionsLostTime_serialId_seq; Type: SEQUENCE OWNED BY;
    Schema: public; Owner: postgres
--

ALTER SEQUENCE "SectionsLostTime_serialId_seq" OWNED BY
    "SectionsLostTime"."serialId";



--
-- TOC entry 179 (class 1259 OID 24609)
-- Name: Sections_id_seq; Type: SEQUENCE; Schema: public; Owner:
    postgres
--

CREATE SEQUENCE "Sections_id_seq"
```

```
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER TABLE "Sections_id_seq" OWNER TO postgres;

--
-- TOC entry 2086 (class 0 OID 0)
-- Dependencies: 179
-- Name: Sections_id_seq; Type: SEQUENCE OWNED BY; Schema: public;
     Owner: postgres
--

ALTER SEQUENCE "Sections_id_seq" OWNED BY "Sections".id;


--
-- TOC entry 185 (class 1259 OID 24699)
-- Name: schema_migrations; Type: TABLE; Schema: public; Owner:
     postgres; Tablespace:
--

CREATE TABLE schema_migrations (
    version character varying NOT NULL
);


ALTER TABLE schema_migrations OWNER TO postgres;

--
-- TOC entry 1930 (class 2604 OID 24733)
-- Name: id; Type: DEFAULT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY "DisruptionComments" ALTER COLUMN id SET DEFAULT
    nextval('"DisruptionComments_id_seq"'::regclass);



--
-- TOC entry 1927 (class 2604 OID 24639)
-- Name: id; Type: DEFAULT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY "Disruptions" ALTER COLUMN id SET DEFAULT
    nextval('"Disruptions_id_seq"'::regclass);



--
-- TOC entry 1932 (class 2604 OID 16448)
-- Name: id; Type: DEFAULT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY "Operators" ALTER COLUMN id SET DEFAULT
    nextval('"Operators_id_seq"'::regclass);



--
-- TOC entry 1933 (class 2604 OID 24614)
-- Name: id; Type: DEFAULT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY "Sections" ALTER COLUMN id SET DEFAULT
    nextval('"Sections_id_seq"'::regclass);
```

```
--
-- TOC entry 1934 (class 2604 OID 24628)
-- Name: sectionId; Type: DEFAULT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY "SectionsLostTime" ALTER COLUMN "sectionId" SET
    DEFAULT nextval('"SectionsLostTime_sectionId_seq"'::regclass);



--
-- TOC entry 1936 (class 2604 OID 24693)
-- Name: serialId; Type: DEFAULT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY "SectionsLostTime" ALTER COLUMN "serialId" SET
    DEFAULT nextval('"SectionsLostTime_serialId_seq"'::regclass);



--
-- TOC entry 1945 (class 2606 OID 24648)
-- Name: pk_Disruptions; Type: CONSTRAINT; Schema: public; Owner:
    postgres; Tablespace:
--

ALTER TABLE ONLY "Disruptions"
   ADD CONSTRAINT "pk_Disruptions" PRIMARY KEY (id);



--
-- TOC entry 1953 (class 2606 OID 24698)
-- Name: pk_SectionLostTime; Type: CONSTRAINT; Schema: public; Owner:
    postgres; Tablespace:
--
```

```
ALTER TABLE ONLY "SectionsLostTime"
    ADD CONSTRAINT "pk_SectionLostTime" PRIMARY KEY ("serialId");



--
-- TOC entry 1949 (class 2606 OID 16450)
-- Name: pk_UserId; Type: CONSTRAINT; Schema: public; Owner: postgres;
     Tablespace:
--

ALTER TABLE ONLY "Operators"
    ADD CONSTRAINT "pk_UserId" PRIMARY KEY (id);



--
-- TOC entry 1947 (class 2606 OID 24748)
-- Name: pk_disruptionComments; Type: CONSTRAINT; Schema: public;
     Owner: postgres; Tablespace:
--

ALTER TABLE ONLY "DisruptionComments"
    ADD CONSTRAINT "pk_disruptionComments" PRIMARY KEY (id);



--
-- TOC entry 1943 (class 2606 OID 16425)
-- Name: pk_id; Type: CONSTRAINT; Schema: public; Owner: postgres;
     Tablespace:
--

ALTER TABLE ONLY "BusRouteSequences"
    ADD CONSTRAINT pk_id PRIMARY KEY (route, run, sequence);
```

```
--
-- TOC entry 1939 (class 2606 OID 16407)
-- Name: pk_key; Type: CONSTRAINT; Schema: public; Owner: postgres;
    Tablespace:
--

ALTER TABLE ONLY "EngineConfigurations"
   ADD CONSTRAINT pk_key PRIMARY KEY (key);



--
-- TOC entry 1941 (class 2606 OID 16412)
-- Name: pk_lbslCode; Type: CONSTRAINT; Schema: public; Owner:
    postgres; Tablespace:
--

ALTER TABLE ONLY "BusStops"
   ADD CONSTRAINT "pk_lbslCode" PRIMARY KEY ("lbslCode");



--
-- TOC entry 1951 (class 2606 OID 24616)
-- Name: pk_sections; Type: CONSTRAINT; Schema: public; Owner:
    postgres; Tablespace:
--

ALTER TABLE ONLY "Sections"
   ADD CONSTRAINT pk_sections PRIMARY KEY (id);



--
-- TOC entry 1954 (class 1259 OID 24705)
-- Name: unique_schema_migrations; Type: INDEX; Schema: public; Owner:
    postgres; Tablespace:
```

```
--

CREATE UNIQUE INDEX unique_schema_migrations ON schema_migrations
    USING btree (version);



--
-- TOC entry 1958 (class 2606 OID 24749)
-- Name: fk_DisruptionCommentsDisruptionId; Type: FK CONSTRAINT;
    Schema: public; Owner: postgres
--

ALTER TABLE ONLY "DisruptionComments"
   ADD CONSTRAINT "fk_DisruptionCommentsDisruptionId" FOREIGN KEY
       ("disruptionId") REFERENCES "Disruptions"(id) ON UPDATE CASCADE
       ON DELETE RESTRICT;



--
-- TOC entry 1956 (class 2606 OID 24706)
-- Name: fk_DisruptionFromBusStop; Type: FK CONSTRAINT; Schema:
    public; Owner: postgres
--

ALTER TABLE ONLY "Disruptions"
   ADD CONSTRAINT "fk_DisruptionFromBusStop" FOREIGN KEY
       ("fromStopLBSLCode") REFERENCES "BusStops"("lbslCode") ON
       UPDATE CASCADE ON DELETE RESTRICT;



--
-- TOC entry 1957 (class 2606 OID 24711)
-- Name: fk_DisruptionToBusStop; Type: FK CONSTRAINT; Schema: public;
    Owner: postgres
```

```
--

ALTER TABLE ONLY "Disruptions"
    ADD CONSTRAINT "fk_DisruptionToBusStop" FOREIGN KEY
        ("toStopLBSLCode") REFERENCES "BusStops"("lbslCode") ON UPDATE
        CASCADE ON DELETE RESTRICT;



--
-- TOC entry 1962 (class 2606 OID 24726)
-- Name: fk_SectionBusRouteSequence; Type: FK CONSTRAINT; Schema:
    public; Owner: postgres
--

ALTER TABLE ONLY "Sections"
    ADD CONSTRAINT "fk_SectionBusRouteSequence" FOREIGN KEY (route,
        run, sequence) REFERENCES "BusRouteSequences"(route, run,
        sequence) ON UPDATE CASCADE ON DELETE RESTRICT;



--
-- TOC entry 1961 (class 2606 OID 24721)
-- Name: fk_SectionEndStop; Type: FK CONSTRAINT; Schema: public;
    Owner: postgres
--

ALTER TABLE ONLY "Sections"
    ADD CONSTRAINT "fk_SectionEndStop" FOREIGN KEY ("endStopLBSLCode")
        REFERENCES "BusStops"("lbslCode") ON UPDATE CASCADE ON DELETE
        RESTRICT;



--
-- TOC entry 1963 (class 2606 OID 24632)
-- Name: fk_SectionIdLostTime; Type: FK CONSTRAINT; Schema: public;
    Owner: postgres
--


ALTER TABLE ONLY "SectionsLostTime"
   ADD CONSTRAINT "fk_SectionIdLostTime" FOREIGN KEY ("sectionId")
       REFERENCES "Sections"(id) ON UPDATE CASCADE ON DELETE RESTRICT;



--
-- TOC entry 1960 (class 2606 OID 24716)
-- Name: fk_SectionStartStop; Type: FK CONSTRAINT; Schema: public;
    Owner: postgres
--


ALTER TABLE ONLY "Sections"
   ADD CONSTRAINT "fk_SectionStartStop" FOREIGN KEY
       ("startStopLBSLCode") REFERENCES "BusStops"("lbslCode") ON
       UPDATE CASCADE ON DELETE RESTRICT;



--
-- TOC entry 1955 (class 2606 OID 16426)
-- Name: fk_busStop; Type: FK CONSTRAINT; Schema: public; Owner:
    postgres
--


ALTER TABLE ONLY "BusRouteSequences"
   ADD CONSTRAINT "fk_busStop" FOREIGN KEY ("busStopLBSLCode")
       REFERENCES "BusStops"("lbslCode") ON UPDATE CASCADE ON DELETE
       RESTRICT;



--
```

```
-- TOC entry 1959 (class 2606 OID 24754)
-- Name: fk_disruptionCommentsOperatorId; Type: FK CONSTRAINT; Schema:
    public; Owner: postgres
--

ALTER TABLE ONLY "DisruptionComments"
   ADD CONSTRAINT "fk_disruptionCommentsOperatorId" FOREIGN KEY
       ("operatorId") REFERENCES "Operators"(id) ON UPDATE CASCADE ON
       DELETE RESTRICT;



--
-- TOC entry 2079 (class 0 OID 0)
-- Dependencies: 5
-- Name: public; Type: ACL; Schema: -; Owner: postgres
--

REVOKE ALL ON SCHEMA public FROM PUBLIC;
REVOKE ALL ON SCHEMA public FROM postgres;
GRANT ALL ON SCHEMA public TO postgres;
GRANT ALL ON SCHEMA public TO PUBLIC;



-- Completed on 2015-04-15 11:38:32


--
-- PostgreSQL database dump complete
--
```