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 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 Automatic Vehicle Location one. This report explores and analyses the tools and applications currently available at Transport for London's bus emergency command and control unit. The report then proposes a prototypical tool for monitoring the bus delays in real time in the network. This tool offers an objective source of processed information to bus operators and control room staff. However, further work needs to be done in order for the system to be placed in production environment and relied upon. This is because arterial urban delay detection is very complex and unpredictable as the report will justify. The report concludes with some suggestions of how this project can be improved and driven further.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Konstantin Vladimirov Draganov

September 2014 - April 2015

Word count: 17934

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 as well as other 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 world. It is responsible for more than 2.4 billion passenger journeys a year [14]. The constant population growth of England's capital has been also driving the expansion and improvement of the transport networks across the city [30].This leads to more pressure being put on the infrastructure which includes not only the road network and the bus fleet, but on the technological systems that aid its operation as well.

Transport for London (TFL) is in charge of its operation and its bus network is recognised as one of the top in the world in terms or reliability, affordability and cost-effectiveness [14]. Maintaining such a large scale network requires careful planing and monitoring. Being able to operate such a high reliability service 24/7, 364 days in the year, requires employing new technologies. This also helps keep costs down and therefore makes the service more affordable and accessible for the general travelling public.

Each bus in the TFL network is equipped with state of the art GPS enabled automatic vehicle location (AVL) system named iBus [35]. This AVL system has led to improved fleet management and has enabled the creation and improvement of multiple applications [51]. The system is generating large sets of both real-time and historical data, which aids 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. This includes both planning for future demand and growth, as well as emergencies and innervation during the daily operation of the network.

However, there are still some situations and problems which require CentreComm staff to carry out manual analysis of the available data. This means that there is lack of readily available preprocessed information. Having to manually monitor thousands of buses continuously is very impractical. That is the reason why currently CentreComm operators rely heavily on individual bus operators and drivers to alert them of possible problems. Once alerted about a potential disruptions in the network, they (CentreComm) can start their own investigation - 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 could often result in time and resources being spent (wasted) on investigating non existent problems. Worse, it often leads to time and resources being spend on investigating and dealing with problems of less importance than others, because of wrong interpretation (exaggeration) by bus drivers and/or operators involved in these situations. This project tries to address these inefficiencies and to propose a prototypical tool for real-time monitoring of the bus delays in London's 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 automate and improve the work flows currently in place. This tool has to work and analyse the data that has been made available by TFL. This analysis is required to happen in real time as more data is being made available. The reason for this being that it would be used as an objective source of information for the delays in the bus network at each point in time. In addition to this, the project needs to perform analysis of what visualisation will be useful, suitable and usable for the output.

1.2 Aims

The main aim of this project is to design and implement a real-time visualisation tool which will be used highlight disrupted routes or parts of the TFL bus network which experience delays. Potentially, the system could alert (be proactive) of possible delays even before the bus drivers or operators have noticed and contacted CentreComm for assistance. This aim 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 needs to be able to analyse the input data sets, calculate and output a list of the disruption that are observed in the network. It has to present information regarding the location (route section) in the transport network and their severity.
- The second part of the main aim is to visualise the generated output in a way that is easy to use and understand. It is also important to note that the visualisation should be capable of updating itself whenever the list of delays have changed. This needs to happen in real time as well.

1.3 Objectives

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

- Obtain an in depth understanding of the problem and current work-flows that are in place at CentreComm.
- Research similar work in the literature that has already been done and how it relates to our problem.
- Obtain samples of the available data and gain an in-depth understanding of it (e.g. what it means).

- Gather, analyse and formalise user requirements during discussions and meetings with CentreComm staff and stakeholders.
- Design and develop initial prototype, based on the output from the above objective, which is to be further refined and improved upon 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, here I have outlined the project structure. The report will continue in the next chapter by providing the reader with a detailed background knowledge needed for the rest of the report, as well as an in-depth review of the related work that is found in the literature. This will include brief 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 will need to operate with. In the subsequent chapter, I will 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 (Chapter 3) that have been identified and gathered from CentreComm. The report then goes on to outline the design (Chapter 4) and the implementation (Chapter 5) of the proposed system, followed by Chapters 6 and 7 which address testing and evaluation of the prototype respectively. I conclude the report with a summary of what has been achieved and guidance how the work presented in this thesis could be further developed and improved.

Chapter 2

Background

This chapter aims to introduce some concepts surrounding our problem domain, which aim to help the reader understand and easily follow the subsequent, more technical chapters. It also presents a review of the related work that has been done in this area. The sections below in this Chapter look at some of the key aspects and problems that arise. I then conclude by providing a summary of the alternatives that are presented.

2.1 London Bus Network

The 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 [11]. Each of these bus operator companies is then responsible for abiding to the contracts with TFL. This means that they (the different bus companies) are responsible to ensure that the services they are operating run according to the timetable as per the respective contract. There are two main types of schedules that are being used:

- Fixed - a bus stop need to be served at specific predefined times (e.g. 1:00pm, 1:20pm, 2:00pm etc.).
- Headway based - this means that buses should serve bus stops at regular intervals (e.g. a stop needs to be visited by a bus every 5 minutes).

However, under different circumstances some delays occurring on a given route are beyond the control of the individual 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 [13]) and even simply a severe congestion. In situations like this, bus operators have no authority or power to respond or overcome such problems on their own. This is where CentreComm comes into place to respond and deal with these issues. In situations like this the bus drivers or operators would need to alert and ask CentreComm to intervene. The emergency command and control room at TFL can do so by, for example implementing a short/long term diversions or curtailments (short turning) on/for some of the buses on the affected routes. They (CentreComm) can also seek assistance from London Traffic Control Centre[1] or even the Police under given circumstances.

Buses in the network can be classified by multiple factors however, for the purpose of this report, the main distinction that needs to be considered, apart from fixed and headway based schedules are **high** and **low** frequency bus routes [11]. High frequency routes are those where 5 or more buses attend a given stop per hour, whereas low frequency routes are those that have a stop being visited by 4 or less buses an hour.

2.2 CentreComm

CentreComm is TFL's emergency command and control room, responsible for all public buses in London. It has been in operation for more than 30 years [13] and it employs a dedicated team of professionals who work 24 hours a day, 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

[1] `http://www.tfl.gov.uk/corporate/about-tfl/what-we-do/roads`

unexpected and unpredicted changes and disruptions, maintaining the smooth, reliable and sage operation of London's busy bus network.

London bus network consists of around 680 bus routes operated by more than 8000 buses [2]. Each of these 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. It started more than 30 years ago [13] when it consisted of a couple of operators equipped with two way radios, pen and papers. Today CentreComm operators make use of numerous monitor screens, each displaying interactive maps (showing the location of each bus in the network) 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 deal with the growing bus network and its demands.

2.3 iBus AVL

Automatic vehicle location (AVL) systems provide vehicle tracking, usually in real time. Most often this is achieved by the integration of Global Positioning System (GPS), wireless communications (e.g. SMS, GPRS) and geographic information system (GIS) [38]. AVL systems employ wireless communications for the transmission of the GPS coordinates and other data of the vehicle as it moves in the transport network. Once received by the a central server or computer, this information allows the GIS software to map the location of the vehicle and it also enables further analysis to be performed based on the data.

All of London buses operating on the TFL bus network have been equipped with state of the art and award wining [10] AVL system named iBus [12]. 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 [23] and [51]. One of the key components of the system is the on-board unit (OBU) which is mounted on each of the buses in the TFL

bus fleet and consists of a computational unit connected to sensors and GPS transmitters (see figure 2.1 below taken from [23]). The OBU is responsible

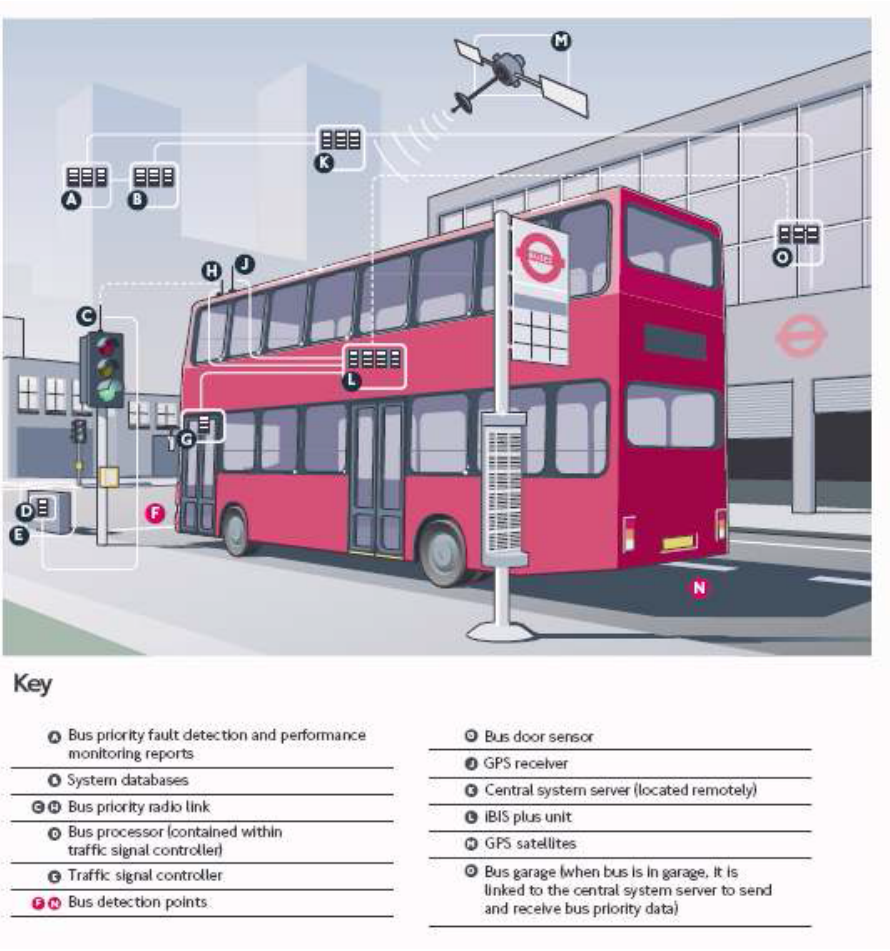

Figure 2.1: Overview of iBus System [23]

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 [23]. This has led to improved and more affordable transport service.

CentreComm operators have access to an online GIS system showing each bus location in the network on a map in real time. This system also 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 is disrupted. Control room staff can also see when was a given bus expected to arrive at a given bus stop and when it actually arrived. Again, this is only per individual bus and there are no automatic alerts set, in case a given route or stop is experiencing delays. Currently the work-flow is such that on-duty CentreComm staff needs 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 there is a problem and how severe it actually is. This is a very inefficient, tedious and error-prone process. Here is where this project comes into place to address the lack of preprocessed information and automated alerts.

2.4 Related Work

The literature is full of research towards accurately predicting bus arrival times with various computational models being used [1]. Predicting bus arrival is complex as many factors need to be considered such as, bus dwell time at stops, general congestion and others [24]. This is also closely related to the issues of bus prioritisations for which numerous examples can be found in the literature. Some examples of work done towards bus prioritisation at traffic signals and junctions can be found in [22] and [34]. However, these are not directly related to our problem domain and thus are not discussed further in this report. This is because we are not interested to know when is the next bus due to arrive at a stop or should we give priority to a bus at traffic signal. We want to know if buses experience increased travel time, because of which they get delayed travelling through some parts of the network.

The main problem posed by this project of detecting disruptions in the bus network can be also translated to short term traffic congestion detection and/or

travel time calculation. 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, technical fault with this bus or other issue(s) which are affecting a single vehicle rather than the entire route or network. This project is concerned with finding routes/sections in the network which are delayed/disrupted and this is beyond the control of individual bus operators. Most often such problems are due to 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. In order to address this problem we have focussed our attention towards work which relates to calculating the bus travel times or gives short-term traffic congestion predictions in a given transport network.

The literature contains plenty of work done toward detecting and 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 [47]. There is also plenty of research done towards traffic congestion detection based on AVL probe data [47]. However, there are significant challenges due to the nature of the urban environment itself. Densely populated areas are influenced by many factors which can 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 the presence of some noise which reduces the accuracy of the transmitted location.

There is however, little research to my knowledge, which focusses on the issues of detecting and short-term forecasting traffic congestion/disruptions in arterial urban environment [4, 47]. From the tables shown in figures A.1 to A.4 in Appendix A taken from [47] we can easily see that most of what 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 usage of these technologies.

In the literature various approaches to measure and predict travel time can be found. These models are categorised in four type groups according to [52] as follows:

- Statistical models - this type of models employ statistical tools and methods for analysis and forecasting. Some models of this type include:
  - Historical
  - Time Series
  - Nonparametric regression
  - Hybrid
- Computer Simulations - main models of this type are traffic simulations. They allow simulation of the traffic flow in a network resembling the characteristic of moving vehicles. Main advantage of these models is that they allow for the simulation of different scenarios. The main drawback however, is that they require traffic flow prediction information in advance [42]. Due to the optimisation nature of these approaches, usually high performance computers are employed (i.e. in [25] they make use of parallel computing).
- Mathematical Optimisation - this include dynamic traffic assignment (DTA) models. A good review of dynamic traffic assignment and simulation models can be found in [29].
- Artificial Intelligence (AI) - neural networks are an example of AI approach. They have received a lot of attention in terms of transportation systems applications. Some examples include traffic signal control, traffic flow modelling and transportation planning [9, 18, 42].

The advantages and disadvantages of the listed types and models are summarised in table showed in figure 2.2 below.

From the above mentioned, 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

Type	Model	Advantages	Disadvantages
Statistical models	Historical Profile	-Relatively easy to implement -Fast execution speeds	-Difficult to respond to traffic incidents
	Time Series -Moving Averages (MA) - Autoregressive (AR) -State Space/Kalman filter	-Many applications -Well-defined model formulation	-Difficult to handle missing data
	Nonparametric Regression -Dynamic clustering/pattern recognition	-Pattern recognition -No assumption of underlying relationship	- Complexity of search for "neighbors"
	Hybrid -Clustering & Linear Regression-Fuzzy logic	-Smaller and more efficient network	-Not yet many implementations
Computer simulation	Traffic simulation	-Possible to simulate various situations	- Requires traffic flow prediction in priori
Mathematical optimization	Dynamic Traffic Assignment	-Various types of models available and well known	-Not suitable for micro-simulation
Artificial Intelligence	Neural Networks	-Suitable for complex, non-linear relationships	-Forecasting in black box -Training procedure

Figure 2.2: Advantages and disadvantages of the different traffic forecasting types and models [52].

dealing with incidents. Time series models have many applications and are well formulated. Because of these reasons and also the nature of the data that has been made available (detailed description of which can be found in Chapter 5) for this project, we will focus our attention on time series analysis for the rest of this report.

2.5 Time Series

Time series is a sequence of data readings taken during successive time intervals [40]. This could be a continuous recording of readings or a set of discrete readings. In the context of this project we have the continuous process of bus readings (generated by the AVL system) being transmitted which generate a discrete set of observations. This results in a data set of measurement values which consists of the actual values with some noise. Time series data contains four main components (illustrated in figure 2.3) [6]:

- **Trend** - this is the long term pattern that the given time series data follows. The trend can have positive or negative value depending on

whether the data exhibits an increase or decrease respectively in the long term pattern. Time series data with no trend (it does not show nether increase or decrease) is said to be stationary.

- **Cyclical** - this is when we can see that the data show up and down movement around a given trend is referred to as cyclical pattern. Main characteristic of the cycle is its duration which can depend on the type of measurement.
- **Seasonal** - seasonality occurs when the time series exhibits regular repetetive fluctuations. For instance, temperatures peaks during summer months (in the northern hemisphere) and drop during winter months.
- **Irregular** - also known as the error component. These are random increases or decreases for a specific time period.

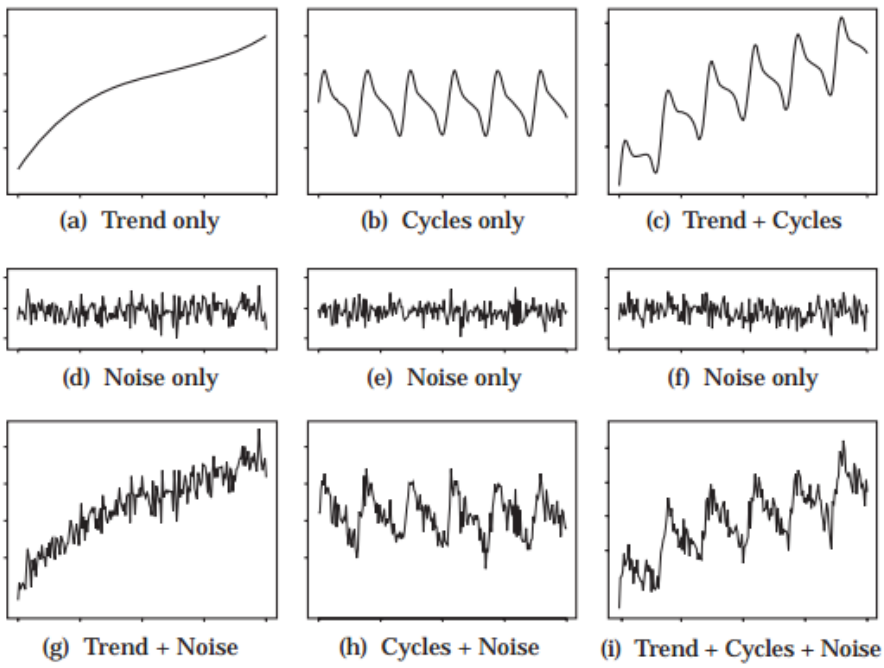

Figure 2.3: Time series data components example (taken from [50])

Analysis of time series data could be performed in order to extract and calculate some meaningful statistics from the data [40]. 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. This however, is not an exhaustive review of all available methods and models as I have tried to keep the discussion relevant to this project.

2.5.1 Moving Averages

One technique commonly used in time series analysis is moving averages which represent a form of smoothing. Smoothing means to dampen the effect of noise and irregularities in the original time series. Moving average, also called rolling or running average is a statistical calculation method. It helps to analyse data series by calculating a series of averages of subsets of the data. Moving averages are commonly used in time series data analysis when the data is fairly stable and does not have significant trend, cyclical or seasonal effects. 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 I have presented some of the main types of moving averages that are used in practice. [7, 40]

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 period window (meaning less observations are considered e.g. only the last 5 or 10) for our averages they would react quicker to changes. While if we work with bigger period windows the averages would

have greater lag. The equation for calculating SMA is given below in equation 2.1. In this n is the size of the window (e.g. the number of reading we are considering) and $Value(i)$ is the actual value of observation i.

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

In the table on figure 2.4 we can see sample time series data with window size n equal to 3. The graph in figure 2.5 shows the plotted actual observation values and the SMA calculated predictions.

Time period	Observation Value	SMA (n=3)
2007	10	N/A
2008	12	N/A
2009	16	N/A
2010	13	12.67
2011	17	13.67
2012	19	15.33
2013	15	16.33
2014	20	17.00
2015	22	18.00
2016	N/A	19.00

Figure 2.4

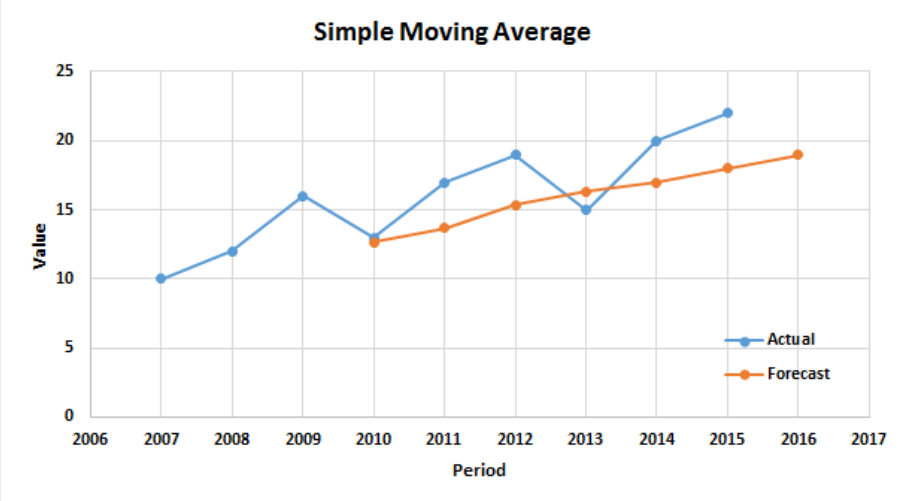

Figure 2.5

Another form of SMA is centered moving average (CMA). Both are very similiar in terms that they use the same method for calculating the average value, but they differ in that the CMA calculates an average of n periods' data and associates it with the midpoint of the periods. An example can be seen in figures 2.6 and 2.7.

Time period	Observation Value	CMA (n=3)
2007	10	N/A
2008	12	12.67
2009	16	13.67
2010	13	15.33
2011	17	16.33
2012	19	17.00
2013	15	18.00
2014	20	19.00
2015	22	N/A
2016	N/A	N/A

Figure 2.6

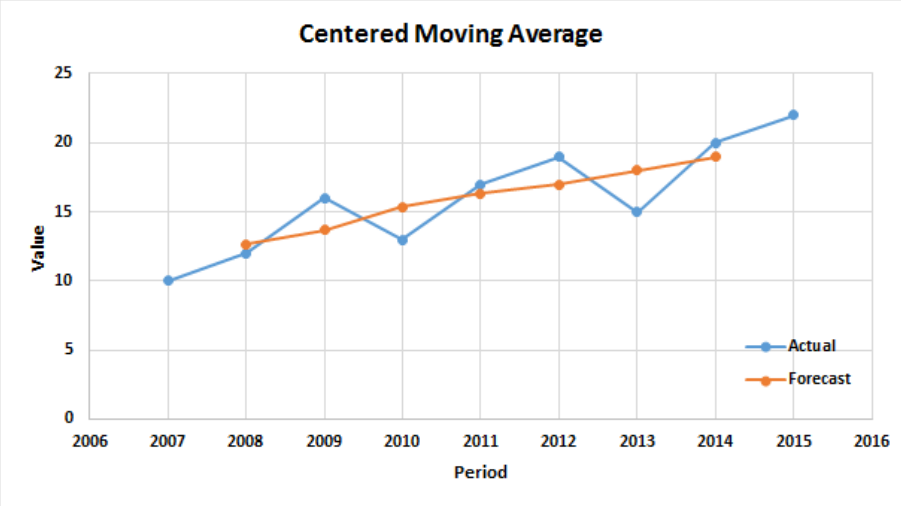

Figure 2.7

Weighted Moving Average

The problem with the simple moving average is that it weighs 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 the period of time when the observation was made. For example, if we consider a n period moving average we can calculate the weight for the value taken in period i where $0 \leq i \leq n$ by the following formula:

$$\text{Weight(i)} = \frac{2i}{n(n+1)}$$

This would mean that recent data has bigger impact on the result. However, it should be noted that the weighting formula given is only an example as it is the most natural and widely used weigthing scheme for WMA. It is possible to use different weighting formula, one such alternative could be:

$$\text{Weight(i)} = \frac{2^i}{\sum_{0 \leq x \leq n} 2^x}$$

this would result in putting more weight on more recent data (e.g. older data is having less effect). The general equation for calculating WMA is given below as equation 2.2, where n is the number of observations (the size of the window).

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

An example of the application of WMA is shown in the table in figure 2.8. The forecast is plotted against the actual values and is shown in the graph in figure 2.9

Time period	Observation Value	Weight moving total (n=3)	WMA (n=3)
2007	10	N/A	N/A
2008	12	N/A	N/A
2009	16	N/A	N/A
2010	13	82	13.67
2011	17	83	13.83
2012	19	93	15.50
2013	15	104	17.33
2014	20	100	16.67
2015	22	109	18.17
2016	N/A	121	20.17

Figure 2.8

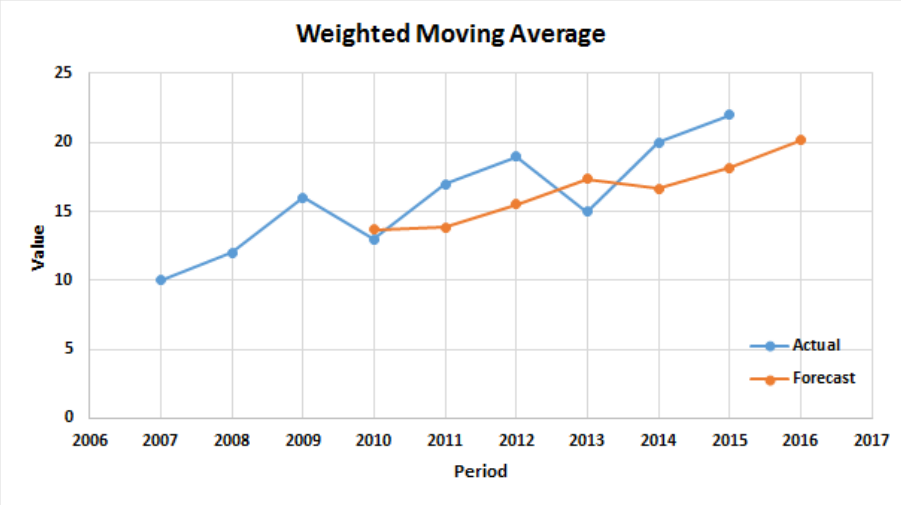

Figure 2.9

Exponentially Weighted Moving Average

Exponential smoothing was first suggested by Robert Goodell Brown [16]. Exponentially weighted moving average (EWMA) or also called exponential smoothing or simply exponential moving average (EMA) 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 exponentially which means that the oldest data would have minimalistic effect on the result. There exist a few exponential smoothing techniques including single,

double and triple exponential moving average [17]. Equations 2.3 and 2.4 give the simplest way 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, the greater smoothing effect it has. This however, makes it less responsive to recent changes thus produces greater lag. Values of α that are near to 1 have less smoothing effect, but are very reactive to recent changes in the data.

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

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

Example of the application of EMA with different values of α (0.2 and 0.8) is shown in figures 2.10 and 2.11. From this simple example it can be clearly seen the effect the value of α has on the output value. From the graph (figure 2.11) it can be seen that the smaller α value of 0.2 has greater smoothing effect, but greater lag. The bigger value of this constant increases the reactivity to recent changes, but produces less smoothed line.

Time period	Observation Value	Previous EMA (α = 0.2)	EMA (α = 0.2)	Previous EMA (α = 0.8)	EMA (α = 0.8)
2007	10	N/A	10.00	N/A	10.00
2008	12	10	10.40	10	11.60
2009	16	10.4	11.52	11.6	15.12
2010	13	11.52	11.82	15.12	13.42
2011	17	11.816	12.85	13.424	16.28
2012	19	12.8528	14.08	16.2848	18.46
2013	15	14.08224	14.27	18.45696	15.69
2014	20	14.265792	15.41	15.691392	19.14
2015	22	15.4126336	16.73	19.1382784	21.43
2016	N/A	16.73010688	N/A	21.42765568	N/A

Figure 2.10

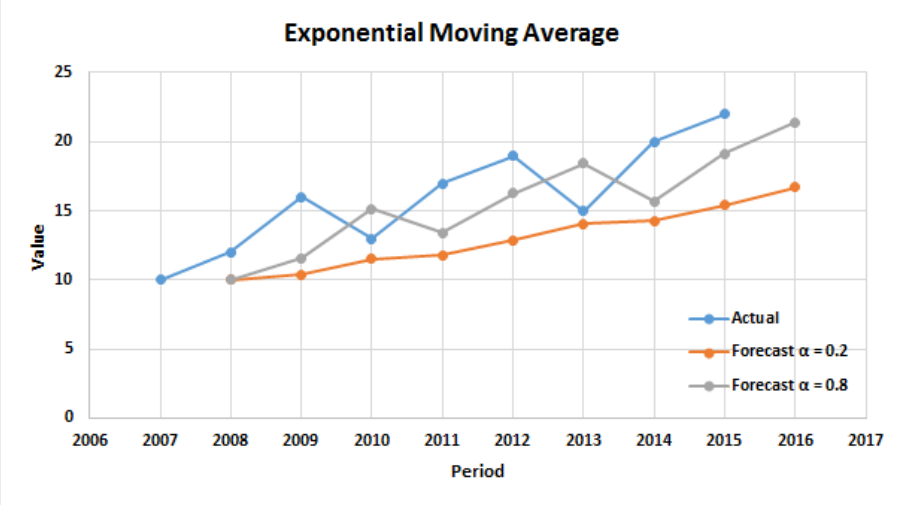

Figure 2.11

Summary

If we compare the presented moving average methods (SMA, CMA, WMA and EMA) it can clearly be seen that the SMA and CMA offer the most smoothing. However, this comes with the trade-off of an increased lag (e.g. it takes longer to reflect recent changes).

The weighted moving average performance is influenced by the choice of window size, as well as the choice of weights. There is no single rule what weights one should use and most often this is based on intuition and simulations in order to get optimal results.

As we have seen in the given examples, the exponential moving average performance depends heavily on the chosen values for the α constant. EMA offers the advantage of not having to keep all data point values in memory for the periods of our window, whereas all the other presented techniques require us to specify a widow size for our moving average and also to have the data for these periods available in memory. The choice of window size for the SMA, CMA and WMA has direct impact on the sensitivity (speed of reaction) of the method to changes. Increased size of the window results in less reactive moving average and increase in the opposite.

If a trend indication with better smoothing and little reaction for shorter

movements is required, 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. Using either of those techniques, requires us to make some choices regarding what parameters (window size and weight for WMA and value of α for EMA) to use in order to obtain best results.

2.5.2 Peak Detection

Peaks and valleys represent significant events where the graph changes from increasing to decreasing behaviour and vice versa (decreasing to increasing). In the domain of our project we are mostly concerned with peak detection as it is not of interested to us to know if buses are gaining time (e.g. are going ahead of schedule). Mathematically viewed peaks and valleys represent local maxima and local minima respectively [41].

Detection and analysis of peaks (spikes) and valleys in time series is important in many applications (e.g signal processing, bioinformatics, medicine and many more) [46]. 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). In figure 2.12 we can see an example of such peak (highlighted in red) we would be interested to detect.

Peaks could be easily identified by visualising the data as in the above example. However, to be relevant and useful in real application this process should be automated. Various peak searching algorithms are studied and presented in [46].

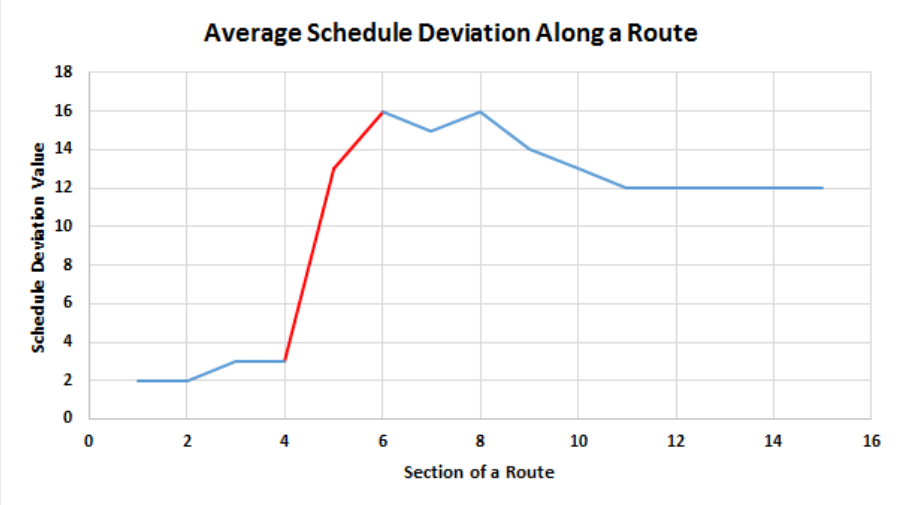

Figure 2.12: Time series data peak example

2.6 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, 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 found in the literature. The chapter concluded with discussion on some of the available time series analysis methods that can be used in our project. The exact approach taken and any implementation specifics 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 from 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) expect(s) to be able to perform and see/obtain the according results, in the end product system. 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 disruptions 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).

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

4. 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.

5. 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 a 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 as close as possible to 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 follow (record) 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.

10. Be compatible with and accessible from every CentreComm staff's computer. Extension of this requirement is that it should be accessible from other teams and groups inside TFL.

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

12. Be easy to deploy, configure and maintain.

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 [28]. Our project is addressing an issue which does not have well specified set of requirements and the clients do not have a clear view of what they actually expect. This led us to use evolutionary prototyping [8] as this helps minimise the impact of misunderstanding or miscommunication of the requirements. The risks of which are relatively high as the goals of this project are relatively new and there is very limited 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 more functionality is added and features and/or refine/improve what has already been implemented. Simply stated, this means that with each increment the system moves 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 allows us to engage with the key stakeholders very early in the project life-cycle. This will 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 [8].
- 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 I need to maintain some balance as this project has a fixed deadline and limited resources.
- Increased chances of fully understanding and meeting the user's 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. Evaluation

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 I have defined in Chapter 1 of this report. After a prototype has been implemented and tested by the user, I will evaluate the progress and make any changes in the requirements, design and project plan accordingly. Thus after a number of iterations, the developed prototype will (should) 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 [32] 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 they can then easily be transformed into platform specific models (PSM) using the desired technologies.

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 4.1 below. The use case diagram depicts the way users (actors) interact with the tool (system). There are 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 also have 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.

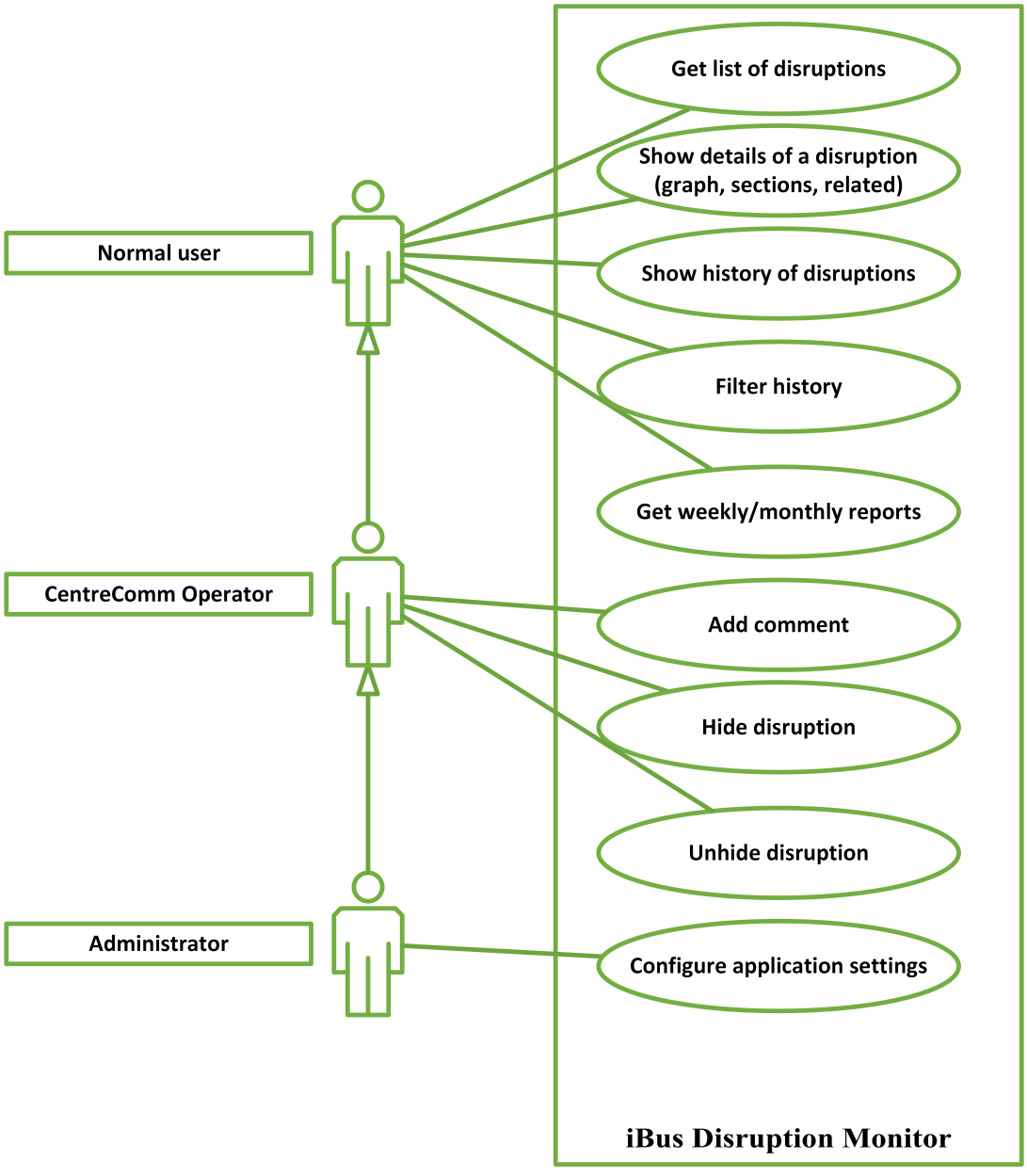

Figure 4.1: Use Case Diagram

4.2 System architecture

In figure 4.2 the architecture diagram of the system could be seen. The overall architecture is following a four-tier architecture with a model view controller (MVC) pattern for the user interface. This structure 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** - It consists of a number of user views and is responsible for visualisation of the user interface.
- **Representation Control Layer** - responsible for the control/transition between user interface windows. In this layer I have made use of the front controller pattern [15] 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 is the data repository layer responsible for storing and maintaining the underlying system data. 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).

The architecture diagram shown in figure 4.2 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.

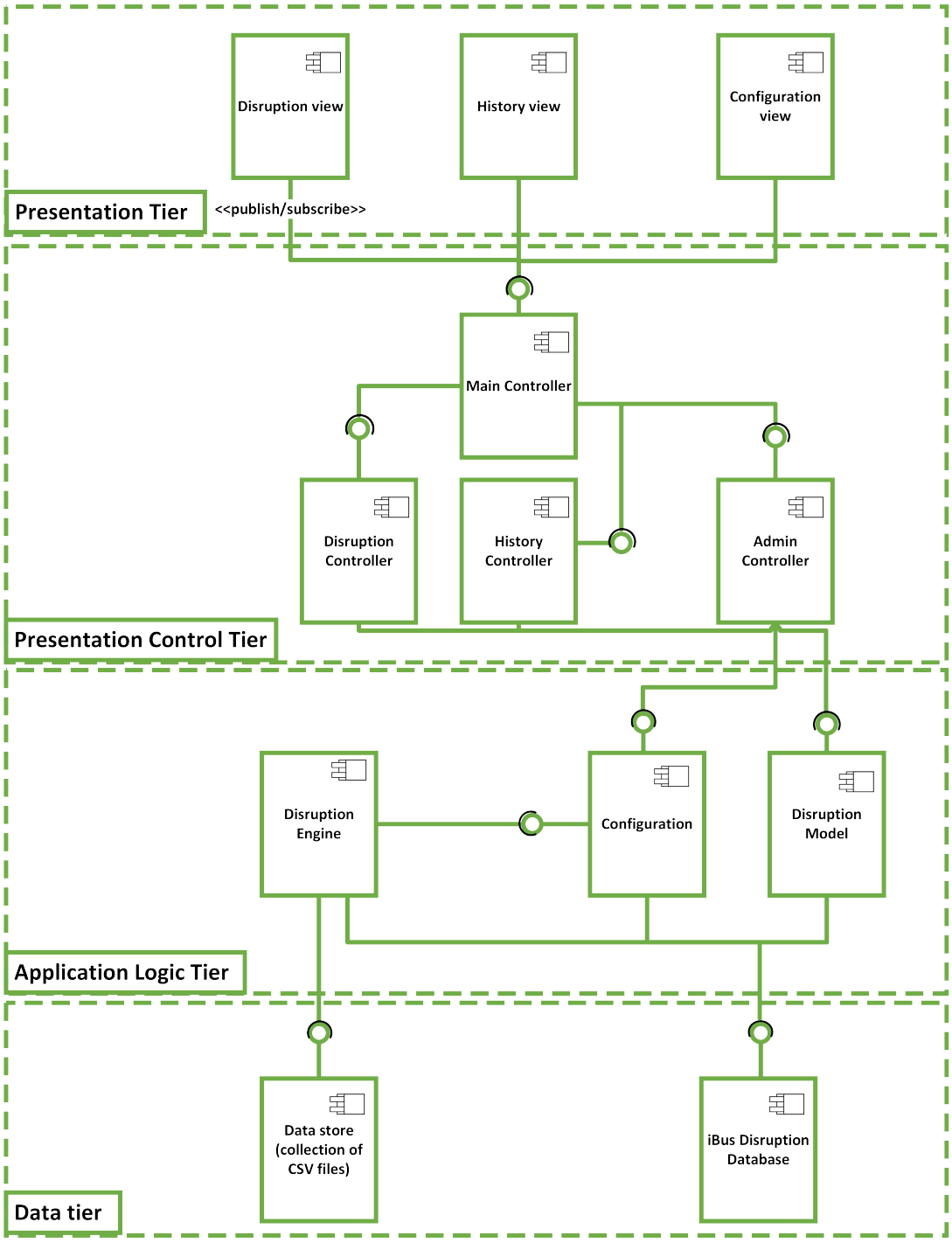

Figure 4.2: System Architecture Diagram

4.3 State Machine

State machine diagrams are useful in explaining in what states the system could reside and how it transitions from one state to the other. The state machine diagram of the disruption engine component of our system is presented in figure 4.3 below. This diagram represents 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 this update results in no changes in the state of the bus network compared to the previous observed state, 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 terminates producing the appropriate error alerting the maintenance personnel. This behaviour is appropriate for this project as its scope is to produce a proof of concept working prototype rather than a fully functional ready to deploy production tool.

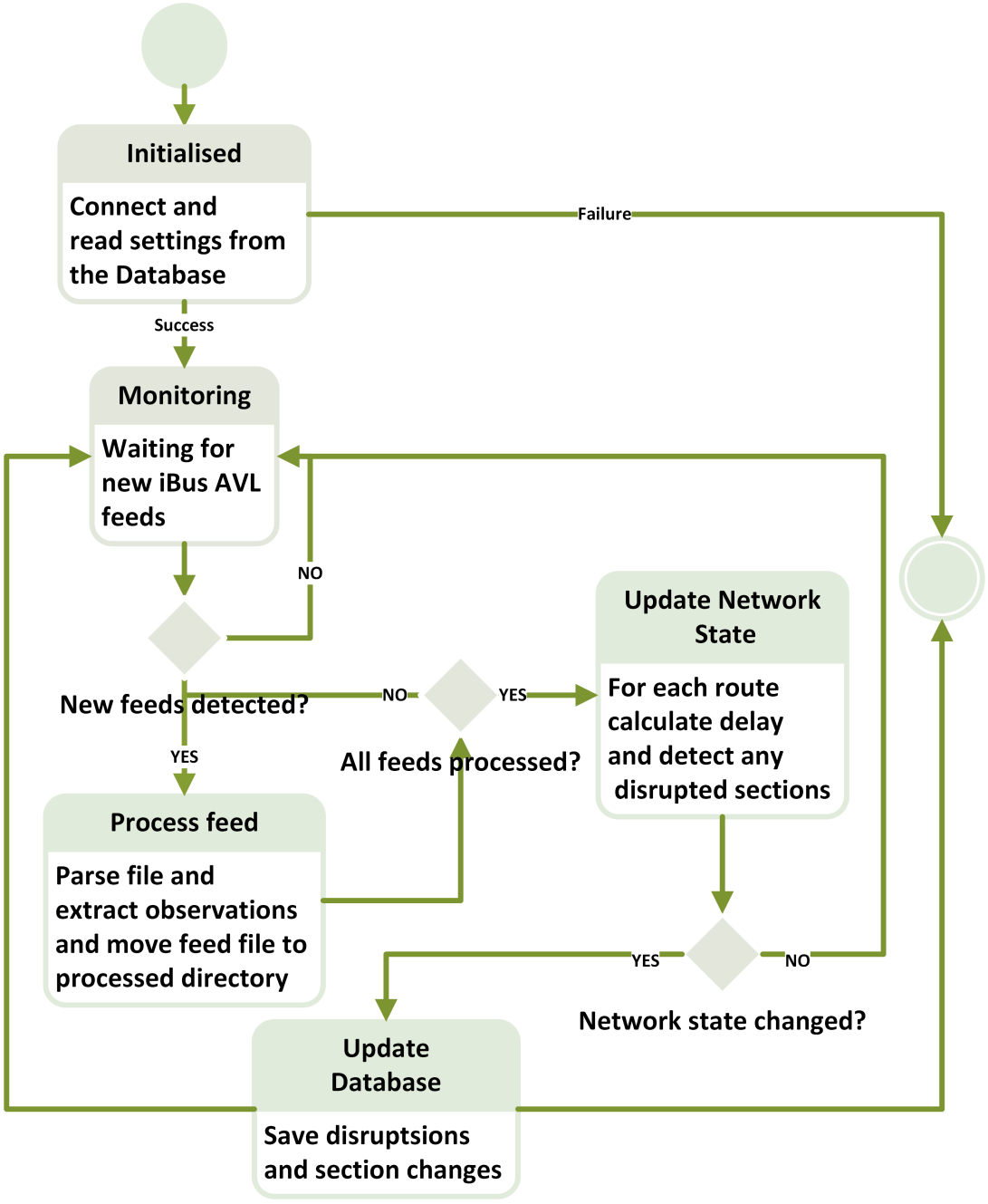

Figure 4.3: State Machine Diagram

4.4 Class organisation

Class decomposition diagrams are the main building block of object oriented programming paradigm. They are a 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. In figure 4.1 below I have depicted the class diagram of the disruption engine component from the architecture diagram. I have only presented the class diagram of this component as this is the component which captures the main business logic with regards to detecting the disruptions in the bus network. In the following 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. The most notable and important attribute of the iBusMonitor class is the link to the configuration file specifying the database connection properties. It can have a number of bus networks 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 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.

4.4.4 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.

4.4.5 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).

4.4.6 BusStop

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

4.4.7 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.

4.4.8 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.

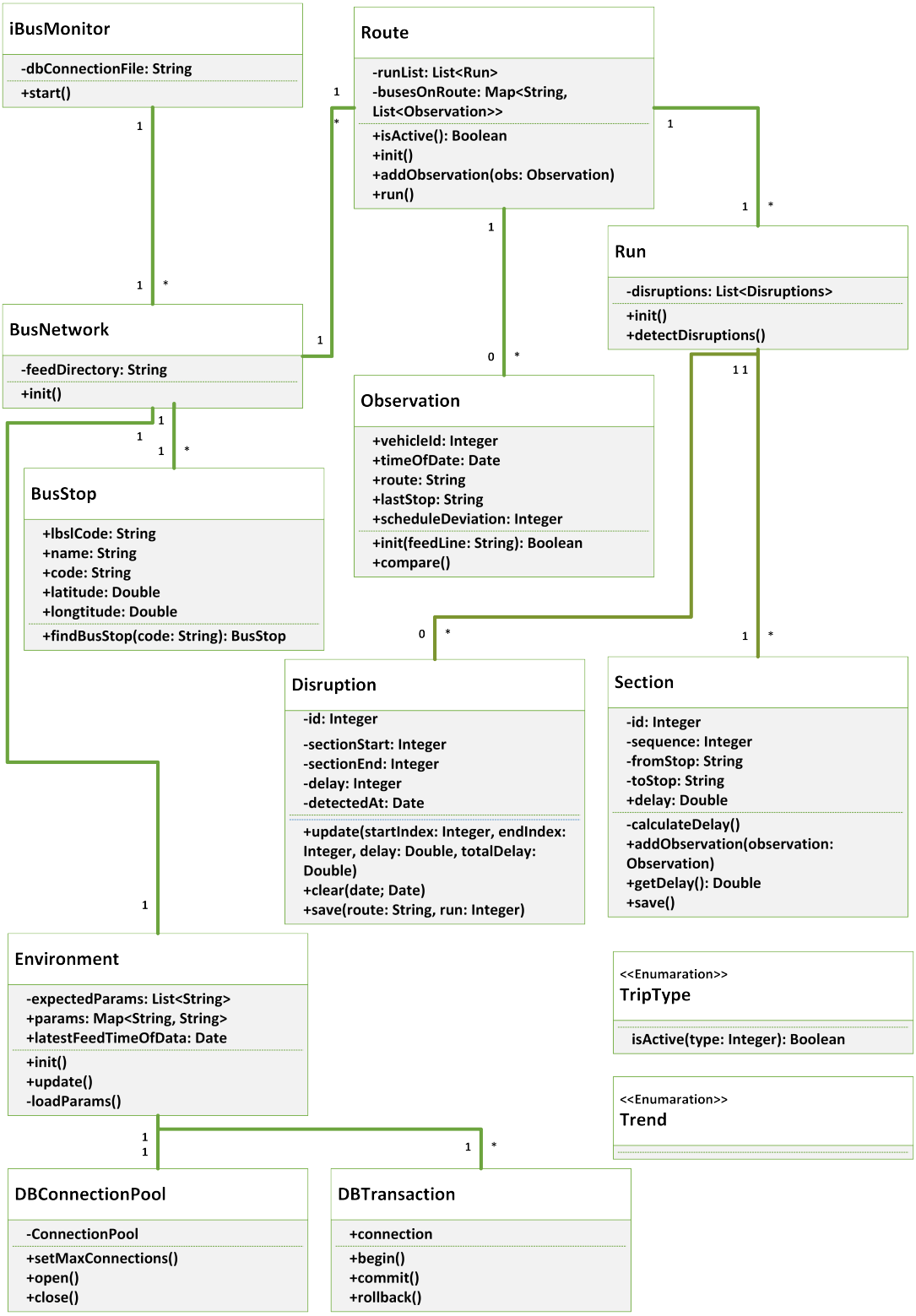

Figure 4.4: Class Diagram

4.5 User Interface

The user interface of our system is addressing the second main aim of this project the one of visualising the calculated list of delays. The design and rational behind the user interface has been developed throughout discussions and meetings with CentreComm staff. The main requirement for the design is to be easy to identify the most important issues in the network. One of the uses of the user interface would be on a large monitoring multi-screen video walls which to be used not only by CentreComm, but by Traffic Management and even the Metropolitan Police. Another usage would be by the individual operators to access it through their personal computers.

The above use cases and requirements have led to the decision of using a web-based application for the purpose of satisfying them fully. Using web rather than a standalone application, we allow for our system to be accessible from any device capable of running a browser (e.g. computer, laptop, smart TV, smart-phone, tablet etc.). Another advantage is that we only need to deploy the web application once and it can be universally accessed through the local network or even through the internet.

In order to improve separations of concerns we have also decided to have separate application for the disruption engine and the user interface. This means that we can change each one without affecting the other (considering we maintain the correct interfaces). This also allows us to implement and add more user interfaces apart from the web application if needed. For example we may even later want to create a dedicated mobile (tablet or smart-phone) application using the output from the disruption engine. This separation also allows to have different dedicated specialised people for maintaining each of the applications.

We have decided to use tabular approach for visualising the prioritised list of disruption. Each entry in the list would give detail of the route and section which are delayed. It also provides the time when the disruption was first detected. Any additional information is only provided on request from the user.

The overall architecture of the structure of the user interface can be seen in the architecture diagram in figure 4.2. An early mock-up of the graphic user interface can be seen in figure 4.5. This has however evolved a lot throughout the project. In Chapter 5 below we will give detailed description of the implemented visualisation and show the end user interface.

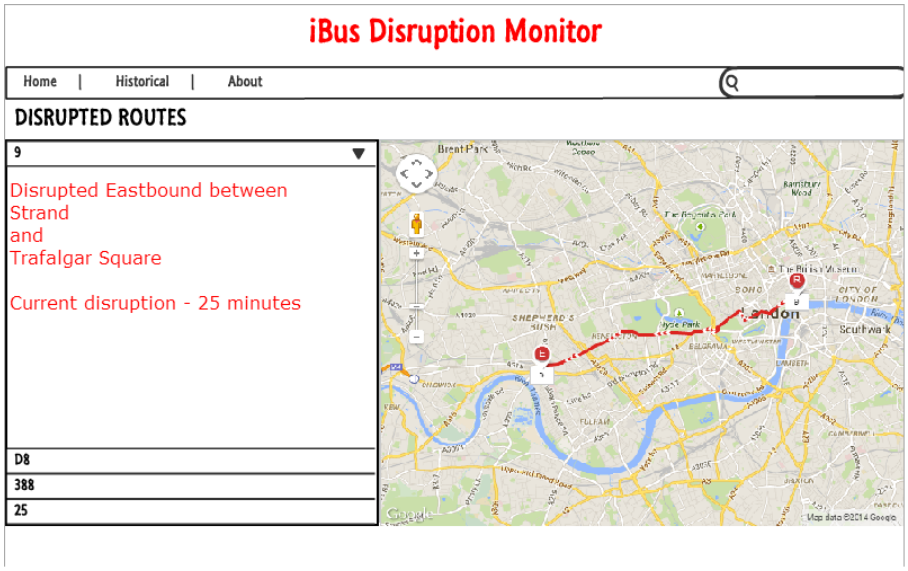

Figure 4.5: Initial GUI mock-up

Chapter 5

Implementation

This chapter aims to present the reader with explanation of the key implementation aspects. These include major challenges, problems that have been encountered and decisions that have been 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 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 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 first describing the data that has been provided by TFL for this project. Afterwards we go into discussion of the implementation of the disruption engine which captures the core functionality and business logic of the system. Then implementation of the visualisation part is discussed which can be viewed as an extension of the disruption engine.

5.1 iBus AVL Data

The data that is used for this project is provided by the Technical Service Group (TSG) at TFL. This data consists of comma separated value (CSV) 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 [23]. 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, 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 figures 5.1 and 5.2 respectively . Below I have provided a detailed explanation of each field in these files [39].

- **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 the end of its current run or somewhere along the run 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 [45]

- **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 for vehicles to have started the route run with some deviation from the schedule already.

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

```
VEHICLE_ID;BONNET_CODE;REGISTRATION_NUMBER;TIME_OF_DATA;BASE_VERSION;BLOCK_NUMBER;TRIP_ID;LBSL_TRIP_NUMBER;TRIP_TYPE;CONTRACT_ROUTE;LAST_STOP_SHORT_DESC;SCHEDULE_DEVIATION;LONGITUDE;LATITUDE;EVENT_ID;DURATION;
18351;AE11;CJ61SUL;2015/02/19 14:01:52;20150213;85004;457551;49;3;298;17775;-60;-0.12806;51.63210;0;;
18356;AE16;SB61SUL;2015/02/19 14:01:47;20150213;85002;451147;45;3;298;OC04;-1080;-0.19187;51.69676;0;;
18366;TPL927;EY03FNL;2015/02/19 14:01:00;20150213;-2147483645;-2147483645;-2147483645;7;UL8;SN;-2147483645;-0.20822;51.49595;0;;
18357;ELV2;PN02XCR;2015/02/19 14:01:39;20150213;-2147483645;-2147483645;-2147483645;7;UL8;RR15;-2147483645;-0.20710;51.49124;0;;
18354;AE14;KS61SUL;2015/02/19 14:01:51;20150213;85001;451047;43;3;298;OC04;-2280;-0.19187;51.69676;0;;
18355;AE15;TW61SUL;2015/02/19 14:01:07;20150213;85005;455779;41;3;298;OC06;-3300;-0.19859;51.69806;0;;
18352;AE12;DS61SUL;2015/02/19 14:01:40;20150213;85005;455780;48;3;298;16915;-1050;-0.15202;51.65436;0;;
18353;AE13;KR61SUL;2015/02/19 14:01:12;20150213;85003;455689;47;3;298;994;-1200;-0.12851;51.63257;0;;
```

Figure 5.1: Sample Raw iBus Data

VEHICLE_ID	BONNET_CODE	REGISTRATION_NUMBER	TIME_OF_DATA	BASE_VERSION	BLOCK_NUMBER	TRIP_ID	LBSL_TRIP_NUMBER	TRIP_TYPE	CONTRACT_ROUTE	LAST_STOP_SHORT_DESC	SCHEDULE_DEVIATION	LONGITUDE	LATITUDE	EVENT_ID	DURATION
18351	AE11	CJ61SUL	19/02/2015 14:01	20150213	85004	457551	49	3	298	17775	-60	-0.12806	51.6321	0	
18356	AE16	SB61SUL	19/02/2015 14:01	20150213	85002	451147	45	3	298	OC04	-1080	-0.19187	51.69676	0	
18366	TPL927	EY03FNL	19/02/2015 14:01	20150213	-2147483645	-2.15E+09	-2147483645	7	UL8	SN	-2147483645	-0.20822	51.49595	0	
18357	ELV2	PN02XCR	19/02/2015 14:01	20150213	-2147483645	-2.15E+09	-2147483645	7	UL8	RR15	-2147483645	-0.2071	51.49124	0	
18354	AE14	KS61SUL	19/02/2015 14:01	20150213	85001	451047	43	3	298	OC04	-2280	-0.19187	51.69676	0	
18355	AE15	TW61SUL	19/02/2015 14:01	20150213	85005	455779	41	3	298	OC06	-3300	-0.19859	51.69806	0	
18352	AE12	DS61SUL	19/02/2015 14:01	20150213	85005	455780	48	3	298	16915	-1050	-0.15202	51.65436	0	
18353	AE13	KR61SUL	19/02/2015 14:01	20150213	85003	455689	47	3	298	994	-1200	-0.12851	51.63257	0	

Figure 5.2: Formatted Sample iBus Data

5.2 Disruption Engine

The disruption engine is implemented based on the design given in the previous chapter. The main implementation language used for the implementation is Scala. Scala is both functional and object oriented language [31]. It is a type-safe Java Virtual Machine (JVM) language [31]. This means that it is compatible with existing Java[7] 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 one can achieve in 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 pick up and learn Scala and put it into use.

The disruption engine needs 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.

[7] `https://www.java.com/en/download/faq/whatis_java.xml`

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.

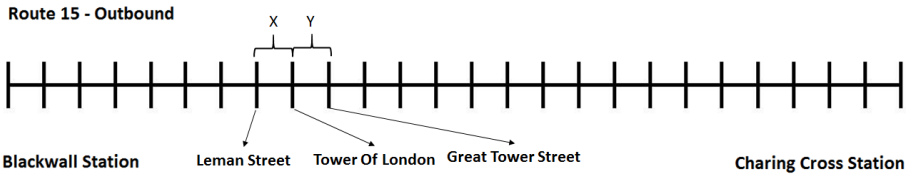

Figure 5.3: Example of a section

Our tool makes use of a PostgreSQL[8] database for storing all configuration parameters and also for storing the required information for the bus network representation. In figure 5.4 below, 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 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 easy to maintain 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 is a big advantage of any technology. Another advantage of

[8] `http://www.postgresql.org/about/`

using PostgreSQL for our implementation is that it ensures reliability and data integrity [27]. Also it supports Listen[9] and Notify[10] functionality which could be used for real-time updating of the web application along with Server Sent Events (SSE) [48].

[9] http://www.postgresql.org/docs/9.1/static/sql-listen.html
[10] http://www.postgresql.org/docs/9.1/static/sql-notify.html

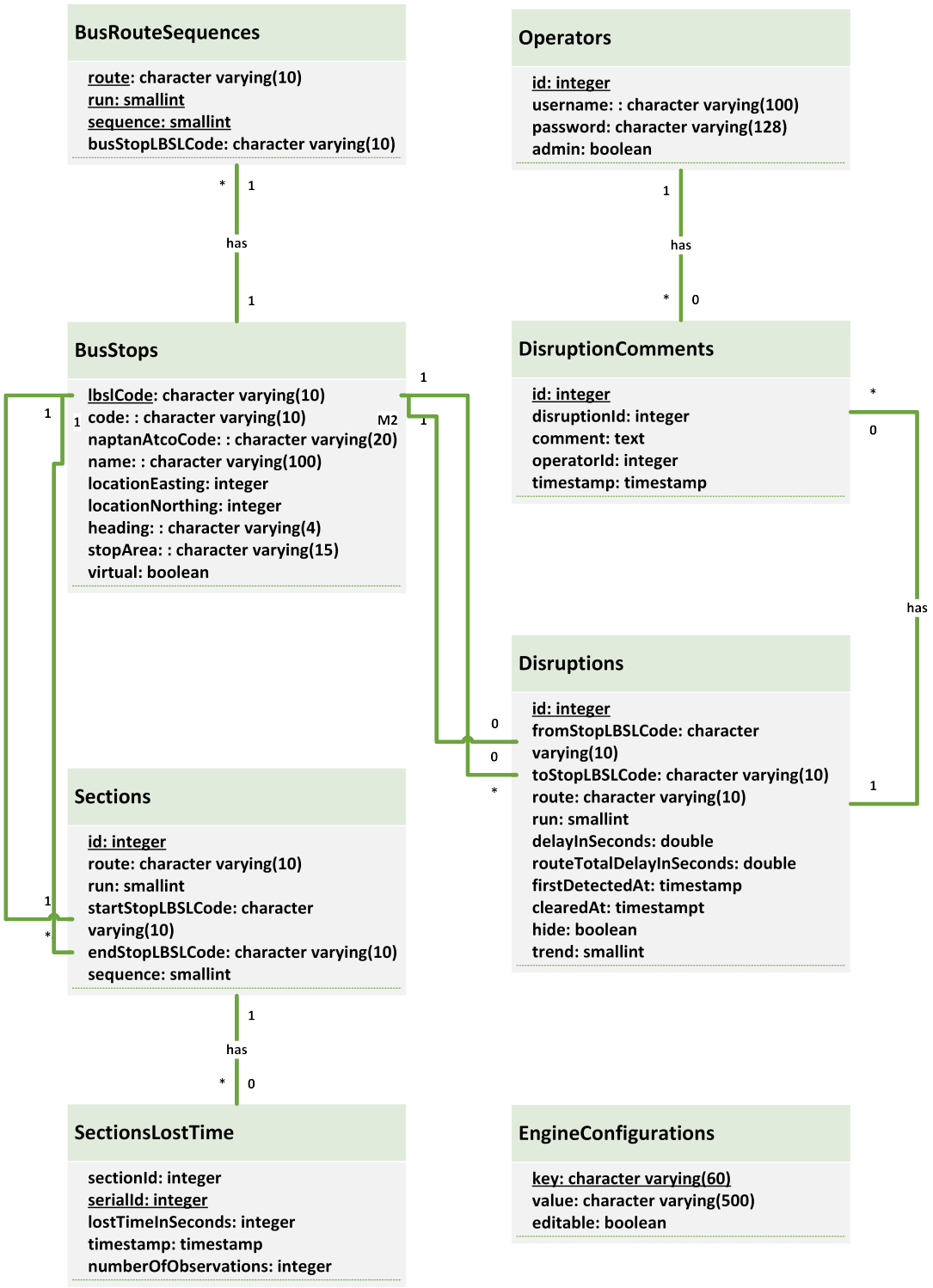

Figure 5.4: Database Model Diagram

5.2.1 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 the TFL website[11]. 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, each route respectively has one or more runs which consists of sequence of bus stops. The implementation assumes that this information is preloaded into the database. This preload consists of simply extracting the information from the CSV files obtained from TFL website into the respective tables (BusStops and BusRouteSequences). In addition to this we also need to preload the sections which will allow the storage of information for individual sections. Currently sections are being generated manually, however, this could be easily automated, but this is not in the scope of this project as it is most likely that TFL already has an internal database with this information in which case the tool will only require pointers to the respective tables. Then, once the engine is started, it would read and load from the database all bus routes and their 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.

5.2.2 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 en-

[11] `http://www.tfl.gov.uk/info-for/open-data-users/our-feeds`

gine. 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 4.3 in Chapter 4). This means that once the engine detects new feed(s) in the directory it is currently monitoring, it will process all new feed files.

5.2.3 Bus network state update

Once feed processing has finished, the system needs 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 5.5). At least two or more observations are required in order to calculate the time loss for a section. In the example below (figure 5.5) we can take the first reading x_1 and the second reading x_2. The difference in the

Figure 5.5: Example of observations

schedule deviation between x_2 and x_1 is then calculated (see table figure 5.6).

Once we have calculated the schedule deviation change between two consecutive stops, we need to assign it to the respective sections. From this example we can see that reading x_1 has been sent somewhere from section A and that

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 5.6: Example schedule deviation calculations for the example in figure 5.5

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 sections or it could have been distributed among 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 observations, 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 that means 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 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 will result in a list of values representing the delay (time lost) and the time (time of the data of the latter observation, i.e. 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 on 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 5.7). 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 along the route run.

However if for example we assume that the minimal threshold is 20 minutes and take the example from figure 5.7 we can clearly see that the cumulative lost time is greater than or equals 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 are always some slight variations and we are only interested in major disruptions which are beyond the control of individual bus operator companies. If we take the example presented in figure 5.7, 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 these sections.

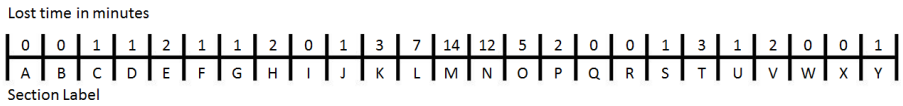

Figure 5.7: Example of a disruption

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 5.8). We can have a long route run with very small loss of time per section (as in the example below, 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 specific disruption until it finds/reaches a section with value less than the minimal threshold for section time loss value or the end of the run. 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 these 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.

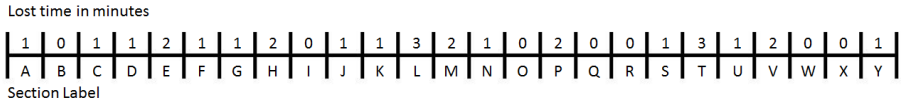

Figure 5.8: Example of no disruption

Every detected disruption or one such that it has changes in 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.

5.3 Graphical User Interface

The engine implementation described above addresses 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 we need to produce some kind of user interface which will be able to meet the requirements for visualisation of the detected bus delays. As described in section 4.5 of this report, I have decided to have individual application for the back-end disruption engine and the user interface. For the implementation of the user interface I have chosen to create a web application using Ruby[12] as the programming language running on Rails[13] framework.

The Rails framework is full stack open source web framework implemented in Ruby which allows for quick development and deployment [19]. It is relatively easy to use and maintain and it supports 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 [15]. Ruby is object-oriented scripting language which is famous for its conciseness. It enables the programmer to express his intentions quickly in very few lines code and it is also easy to read this code later (in future). Another advantage that Rails framework provides is the

[12] `https://www.ruby-lang.org/en/`
[13] `http://rubyonrails.org/`

automatic code generators which provide code skeletons which save effort and time. Rails also allows for agile software development as this is in its roots [19]. It is also important to note that Rails has a strong support community and documentations, as well as expansive list of add-ons and third-party libraries.

Other languages and framework have also been considered to be used instead of Ruby and Rails for this project. However, the chosen ones provide us with technologies which are to be used for quick prototyping and agile development which our project is following. In addition to the above this was also an opportunity for me to learn a new language and framework which I have not used before.

I have also made use of Foundation [14] Cascading Style Sheets[15] (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, enables 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 on small screen and 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.

The Ruby on rails web application is implemented following the MVC pattern. MVC allows the programmer to structure their application code better with clear separation of concerns. In the case of this project it consists of few simple models representing the corresponding database tables, controllers and a number of views. The models are implemented as Active Records[16] 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, cre-

[14] `http://foundation.zurb.com/`
[15] `http://www.w3.org/Style/CSS/Overview.en.html`
[16] `http://guides.rubyonrails.org/active_record_basics.html`

dentials where necessary and responding with the requested information. The responses are, in most cases rendered views, containing the requested information.

The user interface can be seen in figures A.5 to A.14. It consists of three main views which are the disruption view (figure A.5), the history view (figure A.7) and the settings view (figure A.12). The main view is the disruption one which is responsible for visualising the disruptions in the network at any point in time. The list is displayed in tabular form (figures A.5). This provides instant awareness of what the network state is to the CentreComm operators. Disruptions are prioritised by their severity and are colour coded.

The application also has a basic authentication in place which enables to distinguish between the three type of users as described in section 4.1. Guest users do not have to authenticate, but they have limited access meaning they have read-only view of the system. The system also allows users to login (figure A.8) and depending on their status they could either be operators or administrator users. The administrator users have full access to the functionalities which includes view of the settings and the ability to change them, this is not possible if you are guest or operator user. Logged in users are allowed to hide/show (the result is that guest users would not see hidden disruptions) disruptions (figure A.11) and also to add comments (figure A.14). All users including guest are enabled to request further information for a given disruption which includes a graph of the lost time on the disrupted route (see figure A.6). This graph is a combo chart[17] (combination of line and bar chart) and is created using Google Charts[18]. The bars of the graph denote the WMA delay per section while the line depict the cumulative lost time along the route (this treats all negative values as 0). The Google Charts API[19] allows us to make the graph interactive such that on hoover we can display more detailed information for the selected section which allows us to create a clear and organised layout with all the information available. This graph is very useful as it provides the

[17] `https://developers.google.com/chart/interactive/docs/gallery/combochart`
[18] `https://developers.google.com/chart/`
[19] API - Application Program Interface

CentreComm operator or other users with a detailed view of what is the state of the given route. The real-time disruption list and the history table are by default sorted by severity of the section delay and the total delay. However users are able to sort by any other column. This is achieved using Ajax [20] in order to minimise network load by only reloading the changed part of the web page (we do not need to reload the navigation menus, footers etc.). I have also added a data filter for the history view. The history view has not been part of the initial requirements or aims of the project, but it has been identified as great feature the tool could have. Because of this it has been added during the later stages of the project just as a proof of concept.

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 [5]. 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 WebSockets[21], Ajax Long Polling [5], Server Sent Events (SSE)[48]. The main advantage of using Ajax short polling over WebSockets and SSE are that Ajax Short Polling is supported by all major web browsers natively, unlike SSE which lacks Internet Explorer support. Also, it does not require the set-up of any additional tools or infrastructure, unlike WebSockets. 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 bandwidth and server resources. SSE is 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 is relatively new standard and it has been standardized only as part of HTML5[22]. As mentioned above Internet Explorer does not support SSE and this is a major drawback in our scenario as this is one of the main web browsers CentreComm staff use. In addition to this, SSE require the use of a concurrency enabled server.

[20] Short for asynchronous JavaScript and XML
[21] `https://tools.ietf.org/html/rfc6455`
[22] `http://www.w3.org/TR/html5/`

WebSockets 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 I have excluded this approach as a viable one.

5.4 Problems

During the implementation of the project a number of obstacles and problems have arisen. In this section we discuss the major ones. All that are not mentioned below are assumed to be solved by our implementation and not of enough significance for the reader.

The main challenge during the implementation has been how to assign the lost time between two consecutive observations to the respective sections that the bus has travelled through. This is because the data provided by TFL is sparse (currently every 5 minutes) which means that during the time between these two observations the bus could have passed a number of sections and bus stops and we will only know the last bust stop it has attended. To solve this problem the disruption engine keeps an ordered list of observations for each unique vehicle which is active on a given route. To assign the delays accordingly, the engine takes two observations at a time and does a number of steps. The first step is to check if the observations that have been made along the same run (e.g. both readings come from the bus when it was travelling outbound along the respective route). To do this we need to check if the last stop from the earlier observation and the last stop from the latter reading are both from the same run and the first is preceding the latter last stop.

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 (few or many stops) and duration (e.g. it can last half an hour or few weeks/months) of the implemented diversion. What happens in such cases is that the bus would still transmit its position, but the central iBus server would not be able to calculate the schedule deviation. This results in

readings in the feed file which have abnormal values for the schedule deviation and other fields. These abnormal values however, are always the same and are equal to the negative integer -2147483645. However, this does not happen only when the bus is on diversion, it can also happen if there is problem with the GPS (e.g. weak signal due to high buildings) of the respective bus or if the bus is not logged on properly in the iBus AVL system. This means that we are unable to know for sure what such readings mean. For these reasons, when such readings are observed by the engine, they would simply be ignored. The implications of this are that the implementation may lose some accuracy and it may lag alerting of a given delay. If we consider the example shown on figure 5.9 where buses on this route are on diversion from section F to section P, what will happen is that we will get correct data before and after the diversion which would be processed. However, during the diversion, the data we will get from the buses currently travelling on this diverted path will have meaningless values and thus the disruption engine will ignore it. This means that if buses experience delay during the diversion this would only be picked up by the system once they return on the normal route. However, the delay that the system will observe would be distributed along the whole diversion (as described above in section 5.1). Meaning that in case we have a long diversion, some significant delays might remain undetected by the system. This can also happen if there is problem during part of the diversion, but during the rest of the diverted route the buses actually manage to get back on schedule.

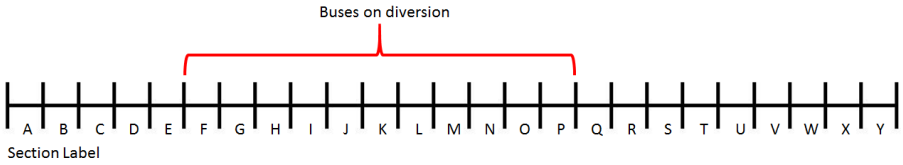

Figure 5.9: Example of diversion

Another problem with the available data is that we do lose some accuracy (some readings are ignored) 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[23] 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 exceeding the allowed by law work hours and more.

Our system relies on monitoring for changes in the schedule deviation value as indicator for the delays in the network. However, there are some scenarios where the engine can detect delays where in fact there are no real-life disruptions taking place. To understand why this can happen, consider the case when a bus is initially ahead of schedule. This can result in the bus driver or operator company to deliberately force this service to lose time in order to be right on time. This is especially a problem for low frequency buses as they run according to fixed schedule and not based on headways. Thus, we can claim that our implementation provides an upper bound of the disruption delays in the bus network.

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 of 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.

Some other problems with the implementation include inconsistencies in the bus route sequence data. This however, is affecting small number of routes and does not prevent the system from serving as a proof of concept. These issues have been discussed with members of the TSG and have been considered irrelevant for the purpose of this project. However, it has been agreed that they would need to be addressed in case this tool is put into real-life use (production environment).

[23] To cut short.

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 the software developed 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. Throughout the development of the product a number of unit tests have been carried out in order to ensure that each of the classes functions 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 system, I have also made great use of the debugging tool provided by IntelliJ IDEA[1] integrated

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

development environment (IDE). This was the main IDE used for the development of this project (both the disruption engine and the user interface web application) and has provided many useful tools and support. The built-in debugger allowed me to carry out inspection of the code during its execution while still developing the application, in order to make sure that desired outcome is produced. The debugging tool allows us to halt the execution (in real time) at specific point during the calculations and examine the state of the program (the values of all variables). This has proved very useful as it allowed me to quickly fix and rectify any issues early in the implementation, which otherwise could have remained hidden.

6.3 Functional Testing

Functional testing is form of black box testing. It is called black box, because we test the software product by feeding it some input and examining the generated output without having knowledge internal states of the software. Functional testing does not mean we are carrying out tests on individual methods/functions or classes/components, it means that we are testing parts of the functionalities stated in the requirements. Since the proposed system consists of two sub-systems I was able to perform separate functional testing of each of the systems.

The disruption engine was tested by having semi-automated functional tests which feed the system with a feed file and then examine the produced results. Some of the example test cases include:

- Correct, but empty feed file is pushed to the system.
- Single correct feed file is pushed, containing a single bus reading.
- Two correct feed files are pushed, each containing a single reading from the same bus.
- A malformed feed file is pushed to the system.
- A batch of feed files is pushed to the system.

The web application part of the system does not contain much complex logic. The most complicated part is retrieving the right data. This required some more complex SQL queries which were tested manually using pgAdmin[2] as a SQL editor program. However, the rest of the functionalities of the user interface have been tested by me throughout the implementation. As the user interface is basically visualising the generated output from the disruption engine, two main scenarios have been considered during its testing. The first is testing the functionalities while the disruption engine is either not running or it does not produce any output. The other test is performed while new output is being generated by the back end (disruption engine) and the user interface is continuously updating itself. In addition to these scenarios, due to the fact the system distinguishes between three types of user roles, I have assumed each one of them and carried out manual tests to verify the correct behaviour of the system.

Some feedback has been received regarding the user interface during demonstration and discussions with CentreComm and my supervisor. All remarks and suggestions from the received feedback have been addressed. 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 respective 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 following the received feedback.

6.4 Stress Testing

One of the main requirements for detecting the disruptions in the bus network is for it to be calculated in real time. Because of this I needed to make sure that the implementation is able to cope with large amounts of data quickly. Each bus reading that is received as part of the AVL feed files is on average 100 bytes. There could be roughly up to 7000 buses active in the bus network

[2] `http://www.pgadmin.org/`

at each single point in time. This means that the system should be capable of processing 1MB of data every 30 seconds as close to real time as possible. Also, as the system will run continuously with more data being pushed to the system throughout its execution, it needs to be ensured that any memory that is occupied by irrelevant (old) data is discarded appropriately without causing any memory leaks.

The approach that I have taken in order to test the system for such performance issues consisted of 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 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 in future if the tool goes into production. For this reason I need to make sure that the proposed prototype is capable of dealing with all of the data that is being pushed to the engine 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 hours there are less active buses thus less data generated compared to daytime) and also on how many disruptions are detected in the system. It is obvious that if there is more data to be processed, the system would naturally take longer. However, the second statement is because the system only updates the database information for a route, run and section only if there are 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 (writes) as one single transaction, after the bus network representation is updated. 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 and using a database instead of CSV files is that it allows us more easily to analysis and query the state of the bus network in the past. Having a database with the historical information, enables us to write simple to complex SQL queries which to produce insightful reports, analysis and statistics. These are all features that have received a lot of positive feedback from TFL during presentations and demonstrations.

Simulations were run to make sure the system is capable of processing and updating itself as close as possible to real time. These 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's 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). I ran a number of simulations with this data, keeping all parameters the same and re-initialising the system before each run. The only parameter I 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 can be seen on figure 6.1 shown below. From the results, I 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 can also be seen in the results shown on figure 6.1 below. 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 on when the Java Virtual Machine (JVM[3]) garbage collector executes. The latter is because the system might already have removed all references to some objects. 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

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

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 (multiple threads) in place, the system is capable of generating output in less than a second. This means that the bottleneck for achieving real-time updates to the users is the user interface implementation (e.g. how often the client web browser will poll the server for updates).

Push Rate*(seconds)*	Avg. Time for Network Update *(seconds)*	Max. Time for Network Update *(seconds)*	Avg. Memory Usage *(MB)*	Max. Memory Usage*(MB)*
2.5	0.13	0.48	102	202
5	0.13	0.51	98	199
10	0.13	0.47	97	196
30	0.12	0.49	96	167

Figure 6.1: Stress test results

Chapter 7

Evaluation & Results

Evaluation is an important step of every software project, yet it is sometimes neglected. This chapter aims to discuss how and what evaluation has been done of the prototypical tool that I have proposed and implemented. This is followed by discussing the results that were obtained during the evaluation.

7.1 Evaluation

7.1.1 Disruption Engine

In order to evaluate and verify that our tool is able to detect disruptions, we need to run the system with some data for which the expected output is known. This allows the calculated by the system values to be analysed once the tool has processed the sample input data. Analysing the data needs to focus on whether the system is able to:

- Detect all disruptions present in the data it is fed with.
- Detect and alert of those disruptions in real time (with no or very minor lag).
- Calculating accurate and reliable estimates of what is the actual (real-life) state of the bus network.

However, achieving this is problematic in our case as the data provided by TFL is only the real AVL feed files. There is no formal information or list of all delays, their severity and duration for a given period of time. I have been provided with some links to web blogs which contain some of the diversions that are being implemented on daily basis in response to disruptions in the network. However, I cannot formally use this data to evaluate the proposed prototype as this is neither an exhaustive list, nor precise and accurate source of information (the delays stated there vary greatly and are somewhat subjective). It is possible to manually analyse the provided bus feeds for a given period, in order to extract what the actual state of the network was during that time. However, if I want to cover a number of scenarios this approach becomes infeasible as it will require great effort and time which is limited. And if more cases for evaluation are needed, it will require us to go over and repeat the same process again and again.

In order to overcome this problem I have decided to generate some test data on my own. This was achieved by implementing a simple program for generating iBus AVL-like feed files. This allowed me to compile test data with different scenarios and test if the system achieves the expected results. I 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 scenario 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 situation (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 I have generated the data for the above scenarios, I need to run the system and feed it with the artificially generated input. Each of the above scenarios is ran separately and the system is re-initialised before each run. This allows for the system to be in the exactly same state at the beginning for each test.

These 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. These 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. I also altered the data validity parameter which represents when are any observations discarded.

In order to calibrate the prototype and its output I ran a numerous simulations with the describer scenarios and different values for our parameters. Using our test-generated data, I have managed to obtained best results with having the data validity set to 90 minutes and the moving average window size equal to 10. This means that it only considers the last 10 observations for a given section which have occurred in the last hour and a half. Also, the system uses weights of 1 to 10 (1 for the oldest and 10 for the most recent as shown in section 2.5.1 above). This allowed for a balanced output, as the system tended to overreact if I used an exponentially growing weights or it lagged behind if bigger moving average window is used. Also, keeping data for the last 90 minutes in memory should work well for both **low**- and **high**-frequency buses, as the data provided does not give any indication of what the type of the service is. This however, needs to be further tested and evaluated by either generating further test data, or using real AVL data for which the bus network state at the respective points in time is known.

7.1.2 Visualisation

The web application part of our system was evaluated by simply using the list of requirements. Using the defined requirements from Chapter 3 of this report

I was able to verify that all expected functionality is in place and produces the correct results. This included testing that the application is behaving as expected and displaying consistently across the most widely used browsers (Internet Explorer, Firefox and Chrome), as well as on some mobile devices (Android tablet, Ipad and Iphone). All of the performed tests verified that the system is visualised consistently across the various devices and no functional issues have been identified.

7.2 Results

During the testing and evaluation of the proposed prototype I have ensured that all user and functional requirements have been met. The simulations ran with the artificially generated data, proved that the proposed system is capable of effectively monitoring changes in the schedule deviation value. However, further evaluations and analysis is required into whether using these values can provide an accurate and reliable measure of the actual delays in the bus network. As it has been discussed in the background chapter of this report, there are very few or no studies which address the problems this particular project is trying to solve. This means that I was unable to compare the results obtained from the proposed system with those of others, simply because I could not find any.

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 [3]. These are rules and professional standards that govern the individual decisions and behaviour. The main rules that apply almost to every software development project state the individual should:

- "have due regard for public health, privacy, security and wellbeing of others and the environment."[3]
- "have due regard for the legitimate rights of Third Parties"[3].

The whole project was planned, designed and developed with both of these rules, as well as other guidelines and standards, in mind. The system makes use of a number of third party libraries and framework. However, I have explicitly stated 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 ideas that I have made use of or have been inspired by throughout the project.

I have also tried to make sure that the applications that were developed as part of this project do not pose 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 I need to make sure that it 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 [43] 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. In case the system is ready to be deployed in a real-life production environment further review and testing of the security aspects of the system need to be undertaken.

The web application displays the last time and date when the disruption engine has updated its state. This value represent the latest time of data value from the iBud AVL feeds. This allows the users to be aware of how old the data they are seeing actually is.

In case of failure during the execution of the disruption engine, an alert system can easily be set. Simple example of such alerting could be done via emails being sent out in case the system fails. This will allow for the responsible maintenance personnel to be alerted in time. However, if the system malfunctions in manner such that it continues its execution and continues to generate output, it can lead to confusion among its users. The tool relies on the input data, thus if data supplied is formatted correctly, but the values are wrong, the proposed system cannot be responsible for any misleading (incorrect) results. However, if the input is correct and the calculations produced by the tool are erroneous it can possibly lead to loss of trust in the application. In order to minimise such risks thorough testing has been performed as part of this project as discussed in the previous chapter. It is however, recommended that the system undergoes trial runs with real data being pushed in real time. This would be another test building on the rest of the testing that has been done, to prove the correctness of the system and its behaviour in real-life scenarios.

Chapter 9

Conclusion

During the course of this exciting project I have 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. This has naturally let to a in-depth literature review of the related work and approaches that could be undertaken in order to solve the problem posed by our project. The background research resulted in the design and implementation of a prototypical tool for detecting bus delays in real time using iBus AVL data.

The 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 the chosen approach lies in its simplicity.

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 such as London's bus network. 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 can be applied not only in the domain of this project, but in a broad range of problems 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. Significant amount of this project was devoted to gathering and refining the user requirements, which included numerous meetings, discussions and even spending a day shadowing CentreComm staff. Another major part of this project was carrying out proper background research which even needed to be revisited during the late stages of this project.

One thing that did not work so well is the evaluation of the software system. Partly this is owed to the lack of readily available information which could be used, as well as due to delay in planning and carrying out this evaluation. However, I have now learnt from these mistakes, which would help me better plan any future software projects I undertake.

9.1 Future Work

The proposed system 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 directions for taking the work done as part of this project further.

One simple extension that could take the project one step ahead is to try to implement a peak/valley detection algorithm which could distinguish if given detected delay/disruption is incident related (very sudden spike) or it is general traffic jam (gradual delay build-up) due to rush hour for instance. This was briefly discussed in Chapter 2 of this report, but due to limited time of the project it has not been explored 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 simple to complex historical reports of the state of the bus network at given points in the past. According to

TFL such feature would provide them with more objective view of what has happened in the transport network and who should take the responsibility for any disruptions.

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 application of various AVL systems being employed by different fleets could be used along with the data present in this report. For example such sources could include taxi fleets [36], delivery service fleet, emergency services and many more. Having more information including different sources could result in one central congestion monitoring system which could calculate and predict traffic conditions in the arterial road network in the city more accurately and for shorter-terms. There is the potential for employing data from GPS-enabled devices that most people of the general travelling public own these days. This is something that is already being investigated [44] and I can foresee that it will receive a lot more attention in the near future.

In addition to everything mentioned above, further research and evaluation of the available techniques and data is recommended in order to verify that this is the optimal solution to the problem being addressed by this project. Some alternative time series analysis tools and methods could be analysed including Kalman filtering [26] [20], Markov Chains [33] [37], Machine learning [21] and Bayesian networks [49].

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

[4] 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.

[5] Engin Bozdag, Ali Mesbah, and Arie Van Deursen. A comparison of push and pull techniques for ajax. In *Web Site Evolution, 2007. WSE 2007. 9th IEEE International Workshop on*, pages 15–22. IEEE, 2007.

[6] Peter J Brockwell. *Introduction to time series and forecasting*, volume 1. Taylor & Francis, 2002.

[7] Peter J Brockwell and Richard A Davis. *Time series: theory and methods*. Springer Science & Business Media, 2009.

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

[9] Mark S. Dougherty and Mark R. Cobbett. Short-term inter-urban traffic forecasts using neural networks. *International Journal of Forecasting*, 13(1):21 – 31, 1997.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

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

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

[18] John F Gilmore and Naohiko Abe. Neural network models for traffic control and congestion prediction. *Journal of Intelligent Transportation Systems*, 2(3):231–252, 1995.

[19] A Pragmatic Guide. Agile web development with rails. 2006.

[20] 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.

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

[22] 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.

[23] 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.

[24] Ran Hee Jeong. *The prediction of bus arrival time using automatic vehicle location systems data*. PhD thesis, Texas A&M University, 2005.

[25] Thanavat Junchaya, Gang-Len Chang, and Alberto Santiago. Advanced traffic management system: real-time network traffic simulation methodology with a massively parallel computing architecture. *Transportation Research Record*, (1358), 1992.

[26] 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.

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

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

[29] Hani S Mahmassani, Srinivas Peeta, Gang-Len Chang, and Thanavat Junchaya. A review of dynamic assignment and traffic simulation models for adis/atms applications, 1991.

[30] 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.

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

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

[33] 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.

[34] "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.

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

[36] 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.

[37] 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.

[38] Stephen Riter and Jan McCoy. Automatic vehicle location–an overview. *IEEE Transactions on Vehicular Technology*, 26(1), 1977.

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

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

[41] Carl P Simon and Lawrence Blume. *Mathematics for economists*, volume 7. Norton New York, 1994.

[42] Brian L Smith and Michael J Demetsky. Traffic flow forecasting: comparison of modeling approaches. *Journal of transportation engineering*, 123(4):261–266, 1997.

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

[44] 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.

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

[46] 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.

[47] 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.

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

[49] 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.

[50] Christopher J Wild and George AF Seber. Chance encounters: A first course in data analysis and inference reviewed by flavia jolliffe.

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

[52] Jinsoo You and Tschangho John Kim. Towards developing a travel time forecasting model for location-based services: A review. In *Methods and Models in Transport and Telecommunications*, pages 45–61. Springer, 2005.

Appendix A

Extra Information

A.1 Figures

Table 1
Literature for the period between 2004 and 2006.

Author(s) and Date[1]	Area	Traffic	Prediction		Data	Methodology							
		Parameter	Step (min)	Horizon (steps)	Collection	Approach	Problem[2]	Model[3]	Comparison	Inputs	State-Space	Optimization[4]	
Cetin and Comert (2006)	Motorway	Speed	1	1	Detectors	Univariate	TS	Statistical	■	Single	■		
Dion and Rakha (2006)	Motorway	Travel Time	1	2	AVI	Univariate	FA	Statistical		Single			
Innamaa (2006)	Motorway	Travel Time	1	1	Detectors	Univariate	FA	NN		Multiple	✓		
Lam et al. (2006)	Motorway	Volume	day	day	Detectors	Univariate	FA	Statistical	✓	Single			
Liu et al. (2006)	Arterial	Volume	1	1	Simulation	Univariate	FA	Statistical		Single			
Quek et al. (2006)	Motorway	Density	1	60	Detectors	Univariate	FA	NN	✓	Multiple		Fuzzy	M
Shekhar and Williams (2008)	Motorway	Volume	15	1	Detectors	Univariate	TS	Statistical	✓	Multiple			
Tsekeris and Stathopoulos (2010)	Arterial	Volume	3	1	Detectors	Univariate	TS	Statistical	✓	Single			
Turochy (2006)	Motorway	Volume	15	1	Detectors	Univariate	PR	Statistical		Single			
van Lint (2006)	Motorway	Travel Time	1	1	Detectors	Univariate	TS	NN	✓	Multiple	✓		
Wang et al. (2006a)	Motorway(U)	Travel Time	1	20	Simulation	Univariate	FA	Statistical	✓	Multiple	✓		
Xie and Zhang (2006)	Motorway(U)	Volume	5	1	Detectors	Univariate	PR	Hybrid**	✓	Multiple	✓	Wavelets	M
Zheng et al. (2006)	Motorway(U)	Volume	15	1	Detectors	Univariate	FA	Hybrid**/C	✓	Multiple		Bayesian	O
Innamaa (2005)	Motorway	Travel Time	1	1	Detectors	Univariate	FA	NN		Multiple	✓		
Jiang and Adeli (2005)	Motorway(U)	Volume	60	1	Detectors	Univariate	TS	Hybrid**		Multiple		Wavelets	M
Kamarianakis et al. (2005)	Arterial	State	7.5	1	Detectors	Univariate	TS	Statistical	✓	Single			
Kwon and Petty (2005)	Motorway(U)	Travel Time	15	1	Detectors	Univariate	FA	Statistical		Single			
Oh et al. (2005)	Motorway	Travel Time	1	1	Detectors	Univariate	TS	NN	✓	Single		Genetic	M
Shang et al. (2005)	Motorway	Speed	2	1	Detectors	Univariate	TS	Statistical		Multiple			
van Lint and van Zulen (2005)	Motorway	Travel Time	1	1	Detectors	Univariate	FA	NN		Multiple			
van Lint et al. (2005)	Motorway	Travel Time	1	1	Detectors	Univariate	TS	NN		Multiple	✓		
Vlahogianni et al. (2005)	Arterial	Volume	3	5	Detectors	Multivariate	FA	NN	✓	Multiple	✓	Genetic	I
Zhong et al. (2005)	Motorway	Volume	60	60	Detectors	Univariate	FA	NN	✓	Multiple		Genetic	I
Alecsandru and Ishak (2004)	Motorway(U)	Speed	5	4	Detectors	Univariate	FA	Hybrid**	✓	Multiple	✓	Genetic	I
Chrobok et al. (2004)	Arterial	Volume	1	60	Detectors	Univariate	PR	Statistical/C	✓	Single			
Ishak and Alecsandru (2004)	Motorway(U)	Speed	5	4	Detectors	Univariate	TS	Hybrid**	✓	Multiple	✓	Fuzzy	I
Lin et al. (2004)	Arterial	Travel Time	1	1	Simulation	Univariate	FA	Bayesian		Multiple			
Rice and van Zwet (2004)	Motorway	Speed	5	1	Detectors	Univariate	PR	Statistical		Single			
Wu et al. (2004)	Motorway	Travel Time	3	1	Detectors	Univariate	FA	Statistical	✓	Multiple			
Yang et al. (2004)	Motorway	Speed	5	10	Detectors	Univariate	TS	Statistical	✓	Single			
Zhong et al. (2004)	Motorway	Volume	60	60	Detectors	Univariate	FA	NN	✓	Multiple	■	Genetic	M

[1] U: urban.
[2] TS: time series, FA: function approximation, O: optimization, PR: pattern recognition, CL: clustering.
[3] NN: neural network, Hybrid*/**: statistical/computational intelligence model as the basis, /C: combined forecasts.
[4] Optimization of M: model parameters, I: input space, S: smoothing.

Figure A.1: Short-term traffic forecasting research (from [47]
)

Table 2
Literature for the period between 2007 and 2009.

Author(s) and Date	Area[1]	Traffic	Prediction		Data	Methodology							
		Parameter	Step (min)	Horizon (steps)	Collection	Approach	Problem[2]	Model[3]	Comparison	Inputs	State-Space	Optimization[4]	
Castro-Neto et al. (2009)	Motorway(U)	Volume	5	1	Detectors	Univariate	FA	Statistical	✓	Single			
Chandra and Al-Deek (2009)	Motorway	Speed	5	1	Detectors	Multivariate	TS	Statistical	✓	Multiple	✓		
Ghosh et al. (2009)	Arterial	Volume	15	50	Detectors	Multivariate	TS	Statistical	✓	Multiple	✓		
Hamad et al. (2009)	Motorway	Speed	5	5	Detectors	Univariate	FA	NN		Multiple	✓	Spectral	I/S
Huang and Sadek (2009)	Arterial	Volume	5	1	Detectors	Univariate	PR	NN	✓	Single			
Innamaa (2009)	Motorway	Travel Time	5	4	Detectors	Multivariate	PR	NN		Multiple			
Jintanakul et al. (2009)	Motorway	Travel Time	5	1	Simulation	Univariate	FA	Bayesian		Single			
Karlaftis and Vlahogianni (2009)	Arterial	Volume/Occupancy	1.5	1	Detectors	Univariate	TS	Statistical	✓	Single			
Sheu et al. (2009)	Motorway	Volume	1	1	Detectors	Multivariate	TS	NN	✓	Multiple			
Srinivasan et al. (2009)	Arterial	Volume	15	1	Detectors	Univariate	FA	NN	✓	Multiple		Fuzzy	I
Szeto et al. (2009)	Arterial	Volume	15	1	Detectors	Univariate	TS	Statistical		Single			
Tan et al. (2009)	Motorway	Volume	60	3	Detectors	Univariate	FA	Hybrid*/C	✓	Multiple		NN	O
van Hinsbergen et al. (2009)	Motorway	Travel Time	5	3	Detectors	Univariate	TS	NN	✓	Multiple	✓	Bayesian	M
Vlahogianni (2009)	Arterial	Volume	1.5	1	Detectors	Univariate	TS	NN	✓	Multiple	✓	Genetic	M
Wang and Shi (2012)	Motorway	Travel Time	1	1	Detectors	Univariate	FA	Bayesian		Multiple			
Zou et al. (2009)	Motorway	Travel Time	5	1	Detectors	Univariate	FA	NN	✓	Multiple	✓		
Chandra and Al-Deek (2008)	Motorway	Speed	5	1	Detectors	Multivariate	TS	Statistical	✓	Multiple	✓		
Dimitriou et al. (2008)	Arterial	Volume	1.5	1	Detectors	Univariate	FA	Fuzzy/C	✓	Multiple	✓	Genetic	M
Guo et al. (2008)	Motorway(U)	Volume	1	1	Detectors	Univariate	TS	Statistical		Single			
Li (2008)	Motorway	Travel Time	5	1	Detectors/AVI	Univariate	FA	Bayesian		Multiple			
Stathopoulos et al. (2008)	Arterial	Volume	3	1	Detectors	Univariate	FA	Fuzzy/C	✓	Multiple			
van Lint (2008)	Motorway	Travel Time	1	1	Detectors	Univariate	TS	Hybrid**		Multiple	✓	Kalman	O
Vlahogianni (2008)	Arterial	State	1.5	1	Detectors	Multivariate	FA	NN	✓	Multiple			
Zhang and Ye (2008)	Motorway	Volume	15	1	Detectors	Univariate	FA	Hybrid**	✓	Single		Fuzzy	M
Ghosh et al. (2007)	Arterial	Volume	15	1	Detectors	Univariate	TS	Bayesian		Single			
Innamaa (2009)	Motorway	Travel Time	1	1	Detectors	Univariate	FA	NN		Multiple	✓		
Juri et al. (2007)	Motorway	Travel Time	1	1	Simulation	Univariate	TS	Statistical		Single			
Sun and Zhang (2007)	Network	Volume	15	1	Detectors	Univariate	FA	Statistical		Multiple	✓		
Vlahogianni (2007)	Arterial	State	1.5	1	Detectors	Univariate	FA	NN	✓	Multiple		Genetic	M
Vlahogianni et al. (2007)	Arterial	Volume	1.5	1	Detectors	Multivariate	TS	Hybrid**	✓	Multiple	✓	Genetic	I
Xie et al. (2007)	Motorway(U)	Volume	5	1	Detectors	Univariate	FA	Statistical	✓	Multiple	✓	Wavelets	M
Zhang and Xie (2007)	Motorway	Volume	15	1	Detectors	Univariate	FA	Statistical	✓	Single			

[1] U: urban.
[2] TS: time series, FA: function approximation, O: optimization, PR: pattern recognition, CL: clustering.
[3] NN: neural network, Hybrid*/**: statistical/computational intelligence model as the basis, /C: combined forecasts.
[4] Optimization of M: model parameters, I: input space, S: smoothing.

Figure A.2: Short-term traffic forecasting research (from [47]
)

Table 3
Literature for the period between 2010 and 2011.

Author(s) and Date	Area[1]	Traffic	Prediction		Data	Methodology							
		Parameter	Step(min)	Horizon (Steps)	Collection	Approach	Problem[2]	Model[3]	Comparison	Inputs	State-Space	Optimization[4]	
Abu-Lebdeh and Singh (2011)	Arterial	Travel Time	5	1	Simulation	Multivariate	FA	Hybrid**		Multiple	✓	Bayesian	I
Bustillos and Chiu (2011)	Motorway	Travel Time	15	1	Simulation	Univariate	PR	Statistical	✓	Single			
Chang et al. (2011)	Arterial	Volume	5	1	Detectors	Univariate	TS	Statistical		Multiple			
Chen et al. (2011)	Arterial	Volume	0.1	1	Detectors	Univariate	PR	Hybrid*	✓	Single		Nature Inspired	M
Djuric et al. (2011)	Motorway	Speed	5	6	Detectors	Univariate	FA	Statistical	✓	Multiple	✓		
Fei et al. (2011)	Motorway(U)	Travel Time	1	1	Detectors	Univariate	TS	Bayesian	✓	Single			
Heilmann et al. (2011)	Motorway(U)	Speed	15	8	ETC.	Univariate	FA	Statistical		Single			
Hong (2011)	Arterial	Volume	60	1	Detectors	Univariate	TS	Hybrid**	✓	Single		Sim. Annealing	M
Hong et al. (2011a)	Arterial	Volume	60	1	Detectors	Univariate	FA	Hybrid*	✓	Single		Nature Inspired	M
Hong et al. (2011b)	Arterial	Volume	60	1	Detectors	Univariate	FA	Hybrid*	✓	Single		Genetic	M
Ishak et al. (2010)	Motorway	Speed	0.5	1	Detectors	Univariate	FA	Statistical		Single			0
Khosravi et al. (2011)	Motorway(U)	Travel Time	5	1	Detectors	Univariate	FA	Hybrid**	✓	Multiple	✓	Bayesian	M
Kuhn and Nicholson (2011)	Motorway	Volume	1	1	Detectors	Univariate	TS	Statistical	✓	Single			
Li and Rose (2011)	Motorway	Travel Time	10	6	AVI	Univariate	FA	NN		Multiple			
Min and Wynter (2011)	Motorway(U)	Volume/Speed	5	12	Detectors	Multivariate	TS	Statistical		Multiple	✓		
Myung et al. (2011)	Motorway	Travel Time	5	1	Detectors/ AVI	Univariate	PR	Statistical		Multiple	✓		
Oh and Park (2011)	Motorway	Travel Time	1	1	AVI	Univariate	TS	NN	✓	Single		Genetic / Wavelets	M/I/ S
Simroth and Zähle (2011)	Motorway	Travel Time	1	1	GPS	Univariate	FA	Statistical		Single			
Soriguera and Robusté (2011)	Motorway(U)	Travel Time	5	1	Detectors / AVI	Univariate	FA	Statistical/ C		Single			
Vlahogianni and Karlaftis (2011)	Arterial	Volume/Occupancy	1.5	1	Detectors	Univariate	TS	Statistical	✓	Single			
Wang et al. (2011)	Motorway	Speed	5	1	Detectors	Univariate	PR	Hybrid**	✓	Single		Bayesian	I
Xia et al. (2011)	Motorway	Travel Time	15	1	Detectors	Multivariate	TS	Statistical		Multiple		Kalman	O
Zhang et al. (2011a)	Motorway	Volume	5	1	Detectors	Univariate	FA	Statistical	✓	Single		Nature Inspired	M
Sun and Xu (2011)	Arterial	Volume	15	1	Detectors	Univariate	FA	Statistical	✓	Single			
Boto-Giralda et al. (2010)	Motorway	Volume	5	2	Detectors	Multivariate	PR	Hybrid**	✓	Multiple	✓	Fuzzy/wavelets	M/I/ S
Ghosh et al. (2010)	Arterial	Volume	5	1	Detectors	Univariate	TS	Hubrid**		Single		Wavelets	I/S
Guo and Williams (2010)	Motorway(U)	Speed	5	1	Detectors	Univariate	TS	Hybrid*		Single		Kalman	O
Kamarianakis et al. (2010)	Arterial	Volume/Speed/ Occupancy	1.5	1	Detectors	Univariate	TS	Statistical	✓	Single			
McCrea and Moutari (2010)	Motorway	Volume	15	1	Detectors	Univariate	FA	Hybrid**		Multiple	✓		
Stathopoulos et al. (2010)	Arterial	Volume	3	1	Detectors	Univariate	FA	Fuzzy/C	✓	Multiple			
Stathopoulos et al. (2010)	Arterial	volume	3	1	Detectors	Univariate	FA	Hybrid*/C		Single		Fuzzy	O
Thomas et al. (2008)	Arterial	Volume	5	2	Detectors	Univariate	TS	Statistical		Single			
Tsekeris and Stathopoulos (2010)	Arterial	Volume	3	1	Detectors	Multivariate	TS	Statistical	✓	Multiple			
Xie and Zhao (2010)	Motorway(U)	Volume	15	2	Detectors	Univariate	FA	Statistical	✓	Single			
Yang et al. (2010)	Motorway	Travel Time	15	1	AVI	Univariate	TS	Statistical		Single			
Zargari et al. (2010)	Motorway	Volume	5	1	Detectors	Univariate	FA	NN	✓	Single		Fuzzy/ Genetic	M/I

[1] U: urban.
[2] TS: time series, FA: function approximation, O: optimization, PR: pattern recognition, CL: clustering.
[3] NN: neural network, Hybrid*/**: statistical/computational intelligence model as the basis, /C: combined forecasts.
[4] Optimization of M: model parameters, I: input space, S: smoothing.

Figure A.3: Short-term traffic forecasting research (from [47]
)

Table 4
Literature for the period between 2012 and 2013.

Author(s) and Date	Area[1]	Traffic	Prediction		Data	Methodology							
		Parameter	Step(min)	Horizon (Steps)	Collection	Approach	Problem[2]	Model[3]	Comparison	Inputs	State-Space	Optimization[4]	
Celikoglu (2013)	Motorway(U)	Density	2	1	Detectors	Univariate	FA	Hybrid**		Multiple	✓		
Wang and Shi (2012)	Motorway(U)	Speed	5	1	Detectors	Univariate	FA	NN	✓	Single		Chaos/Wavelets	I/M
Mu et al. (2012)	Motorway	Travel Time	1	1	Simulation	Univariate	FA	Statistical		Single			
Chan et al. (2013)	Motorway(U)	Speed	1	5	Detectors	Univariate	FA	Hybrid**	✓	Multiple	✓	Nature Inspired	M
Guo et al. (2013)	Arterial	Volume	15	1	Detectors	Univariate	TS	Statistical	✓	Single		Singular Value Decomp.	S
Vlahogianni and Karlaftis (2013)	Motorway(U)	Speed	1	1	Detectors	Multivariate	TS	Hybrid**	✓	Multiple		Genetic	M
Abdi et al. (2012)	Motorway	Volume	1	1	Detectors	Univariate	TS	Hybrid**	✓	Multiple		Fuzzy/Wavelets	M/I/S
Chan et al. (2012a)	Motorway(U)	Speed	1	1	Detectors	Univariate	FA	Statistical	✓	Multiple	✓	Exponential	I/S
Chan et al. (2012b)	Motorway(U)	Speed	1	1	Detectors	Univariate	FA	NN	✓	Multiple	✓	n/a	
Chang et al. (2012a)	Motorway	Volume	15	4	Detectors	Univariate	PR	Hybrid*		Multiple		k-nearest neighbors	I
Chen et al. (2012)	Motorway(U)	Volume	3	1	Detectors	Univariate	FA	Statistical	✓	Single		Principal Comp.	I
Cheng et al. (2012)	Arterial	Volume	5	1	Detectors	Univariate	TS	Statistical		Multiple	✓		
Du et al. (2012)	Network	Travel Time	2	1	Simulation	Univariate	O	Statistical		Multiple	✓		
Dunne and Ghosh (2012)	Motorway	Volume/ Speed	1	1	Detectors	Multivariate	FA	NN	✓	Multiple			
Guo et al. (2012)	Motorway	Volume	15	1	Detectors	Univariate	TS	Statistical		Single			
Haworth and Cheng (2012)	Network	Travel Time	5	1	AVI	Univariate	FA	Statistical/ C	✓	Multiple	✓		
Hong (2012)	Arterial	Volume	60	1	Detectors	Univariate	FA	Statistical	✓	Single		Sim. Annealing	M
Kamarianakis et al. (2012)	Motorway(U)	Speed	5	5	Detectors	Univariate	TS	Statistical	✓	Multiple	✓	penalized estimation	M
Khan (2012)	Motorway	Travel Time	5	1	GPS	Univariate	FA	Bayesian		Single			
Li and Chen (2013)	Motorway	Travel Time	5	1	Detectors/ AVI	Univariate	FA	NN		Multiple			
Lu (2012)	Motorway	Travel Time	5	1	Detectors	Univariate	FA	Bayesian	✓	Single			
Ma et al. (2012)	Motorway	Travel Time	1	1	Simulation	Univariate	PR	Statistical	✓	Single			
Qiao et al. (2012)	Motorway	Travel Time	5	1	Bluetooth	Univariate	PR	Statistical	✓	Single			
Sun et al. (2012)	Network	Volume	15	1	Detectors	Multivariate	PR	NN	✓	Multiple	✓	penalized estimation	I
Tchrakian et al. (2012)	Motorway	Volume	15	5	Detectors	Univariate	TS	Statistical	✓	Multiple			
Tsirigotis et al. (2012)	Motorway(U)	Speed	10	1	Detectors	multivariate	TS	Statistical	✓	Multiple			
	Arterial	Volume	5	1	Detectors	Univariate	FA	Statistical		Single			
Xia et al. (2012)	Motorway	State	5	1	Detectors	Multivariate	CL	Statistical		Multiple			
Ye et al. (2012)	Motorway(U)	Speed	0.01	60	GPS	Univariate	FA	NN	✓	Multiple			
Zheng and Van Zuylen (2012)	Arterial	Travel Time	1	1	GPS	Univariate	FA	NN		Multiple	✓		

[1] U: urban.
[2] TS: time series, FA: function approximation, O: optimization, PR: pattern recognition, CL: clustering.
[3] NN: neural network, Hybrid*/**: statistical/computational intelligence model as the basis, /C: combined forecasts.
[4] Optimization of M: model parameters, I: input space, S: smoothing.

Figure A.4: Short-term traffic forecasting research (from [47]
)

E.I. Vlahogianni et al./Transportation Research Part C 43 (2014) 3–19

A.1.1 User Interface

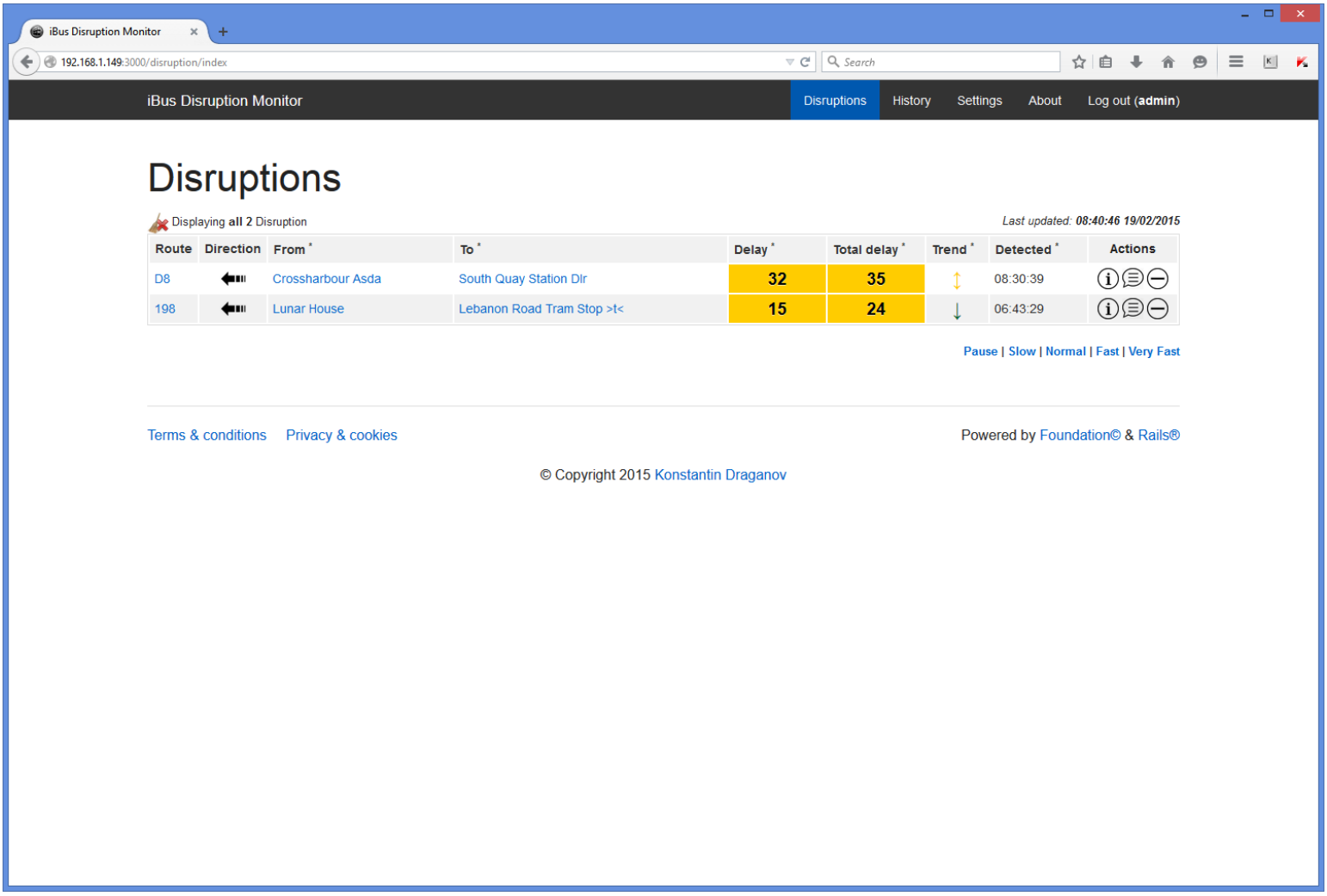

Figure A.5: Disruptions list page view

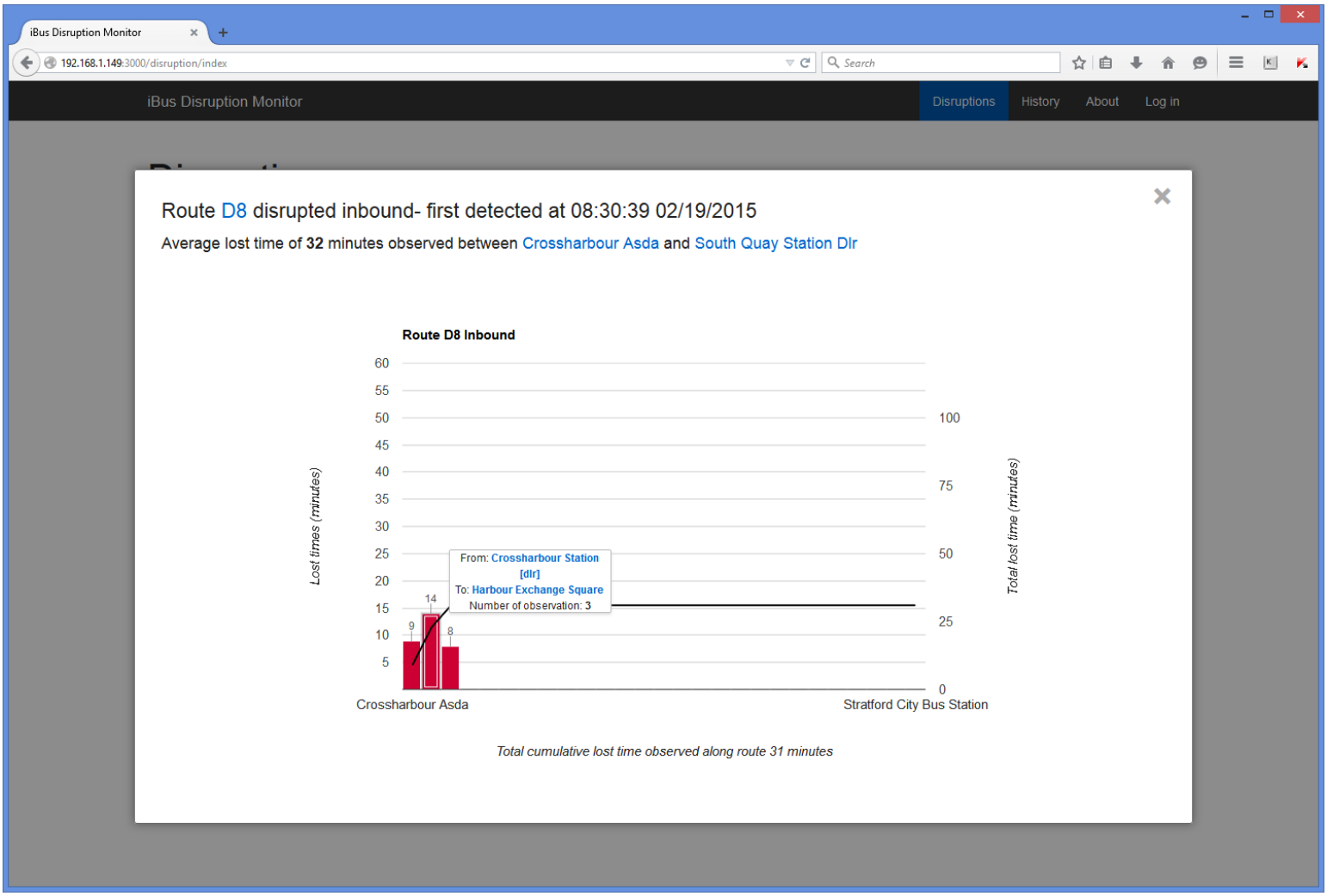

Figure A.6: Disruption details modal window view

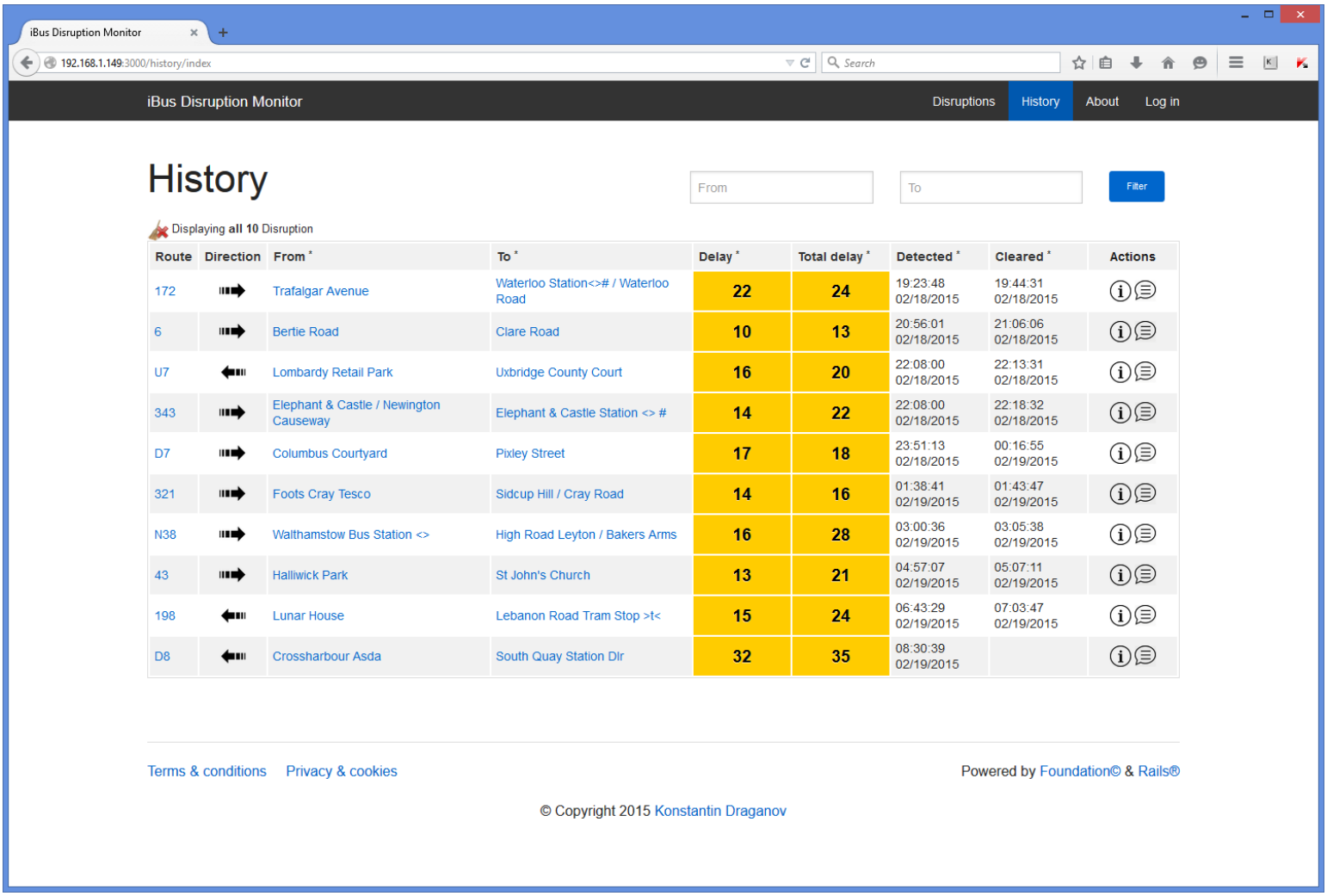

Figure A.7: History page view

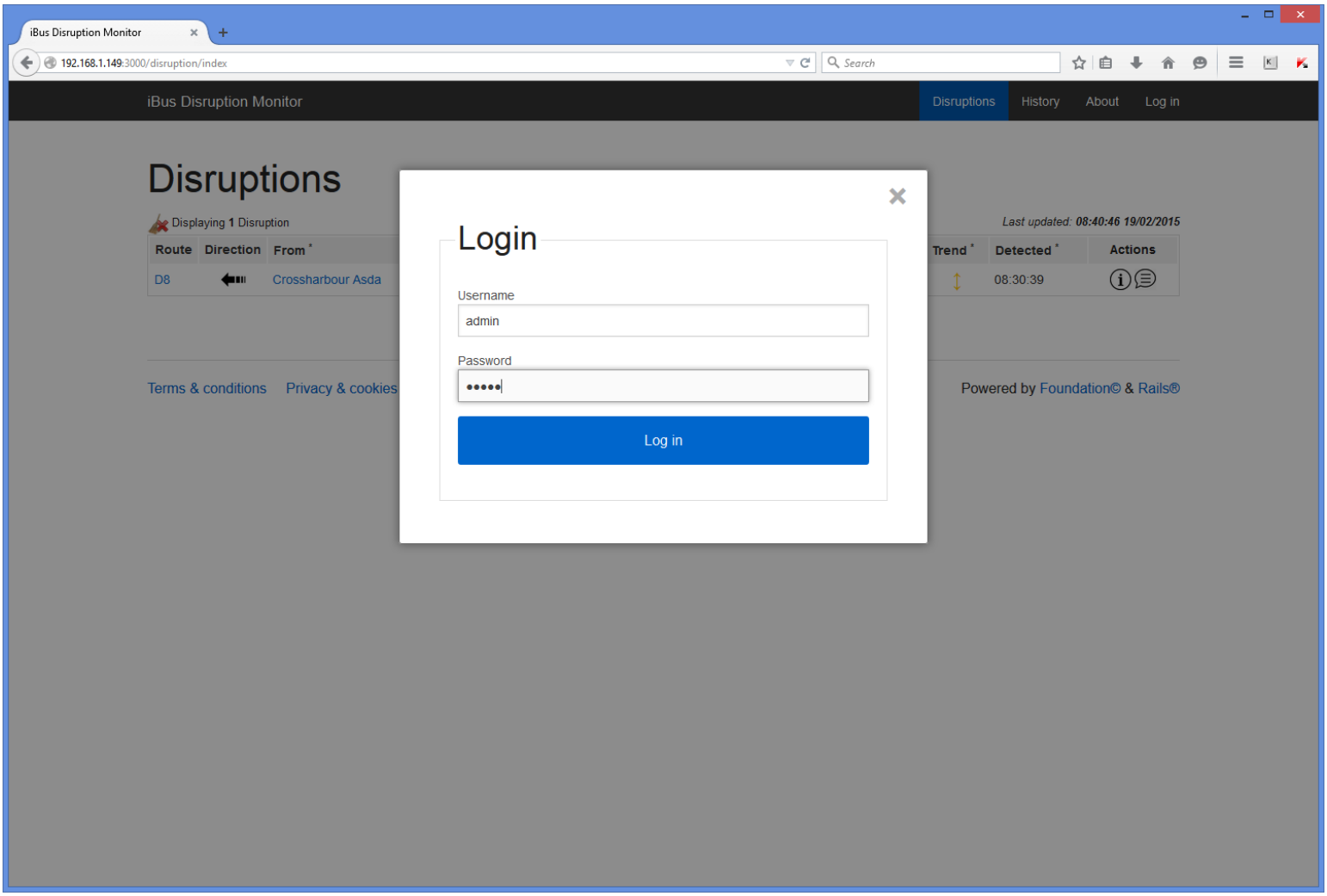

Figure A.8: Login window view

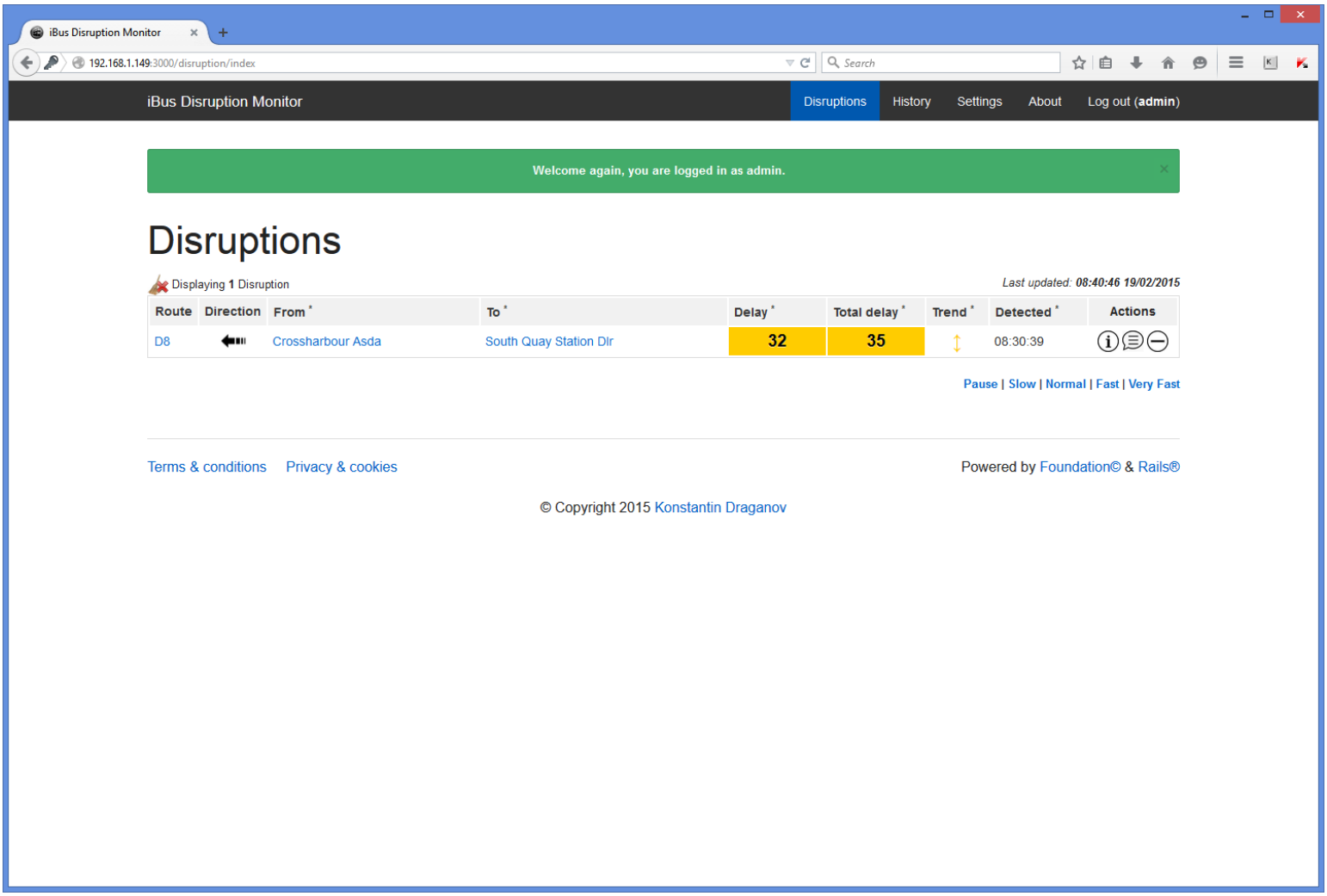

Figure A.9: View after successful log in

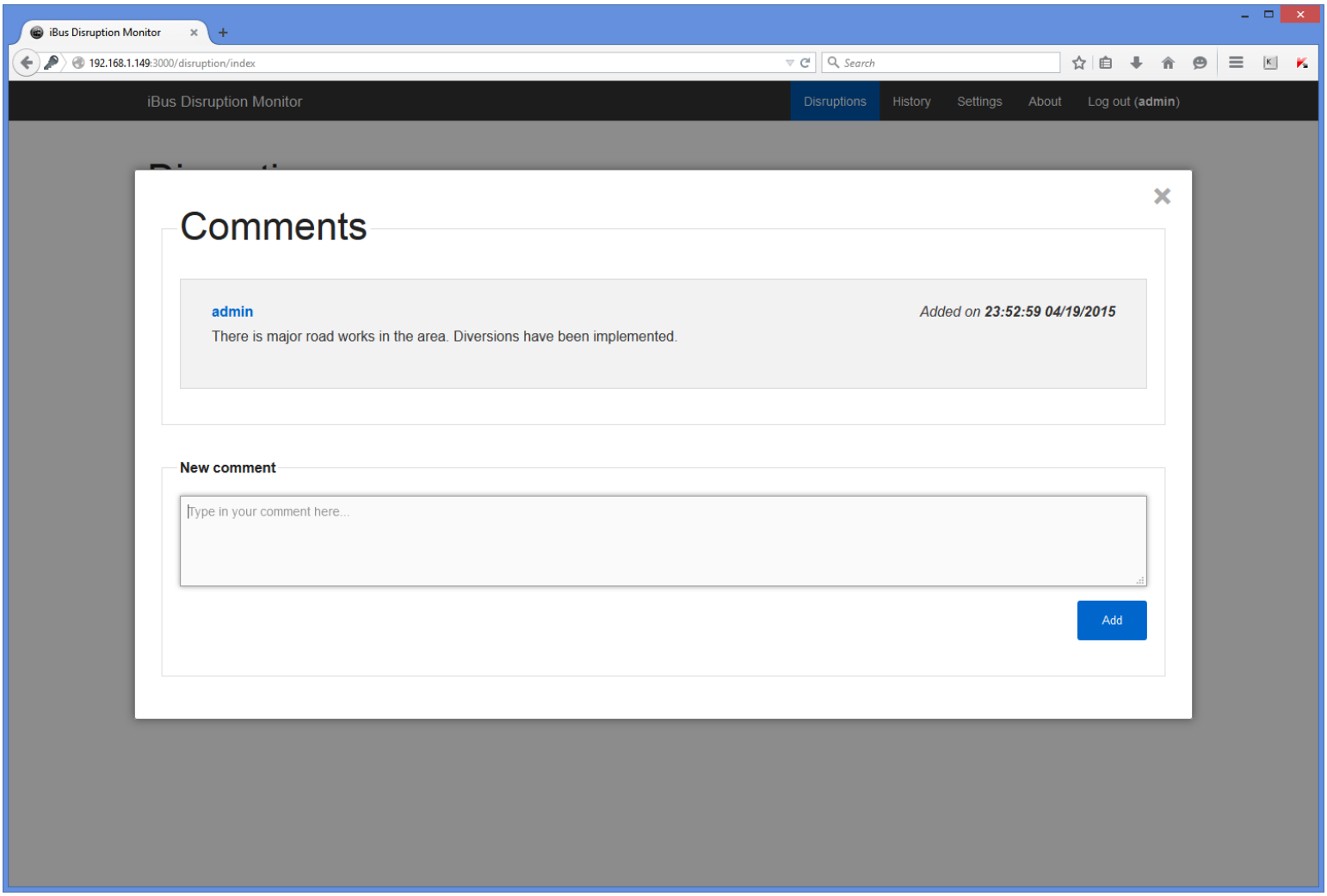

Figure A.10: Comments modal window

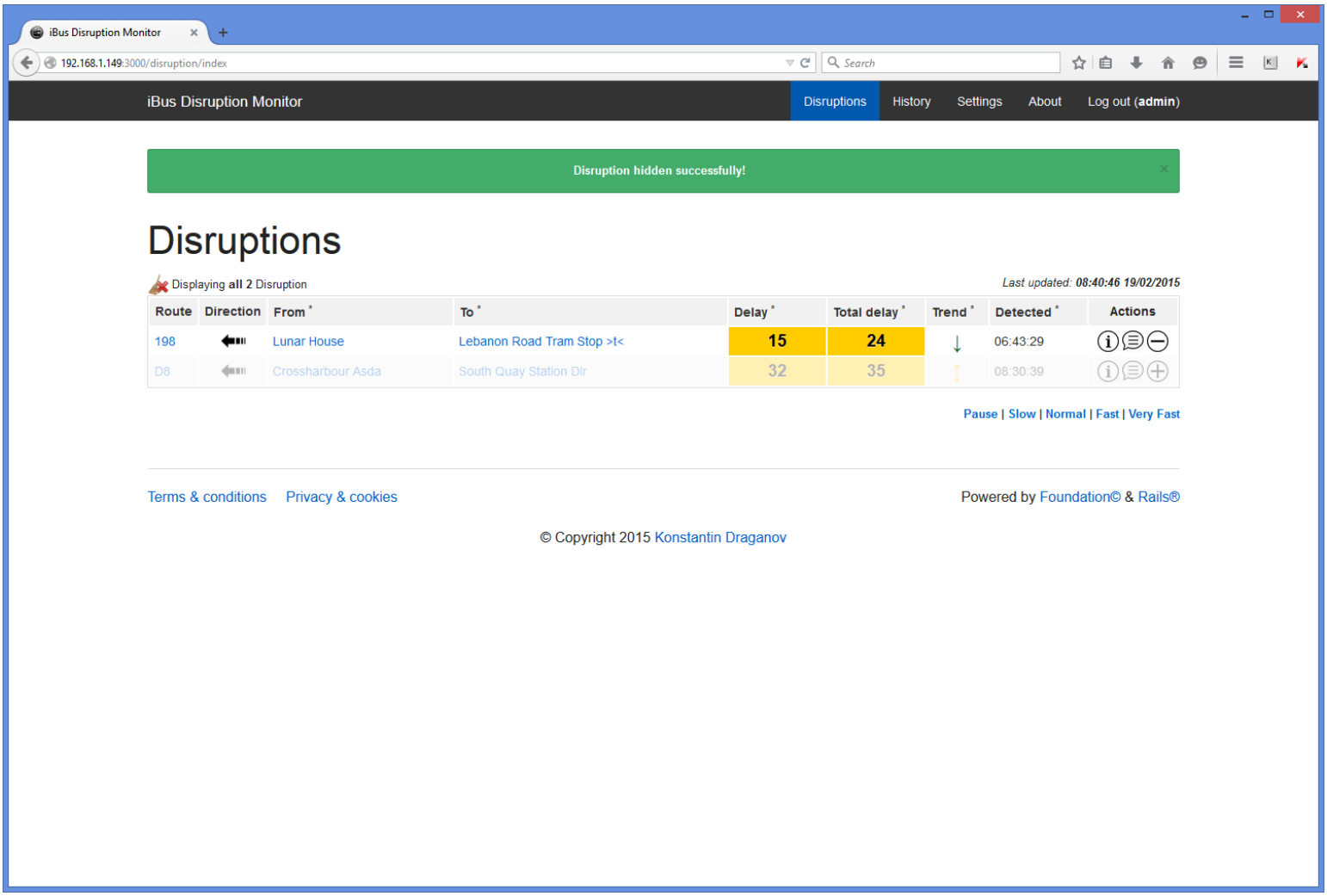

Figure A.11: Disruption view after successful hide/show of a disruption

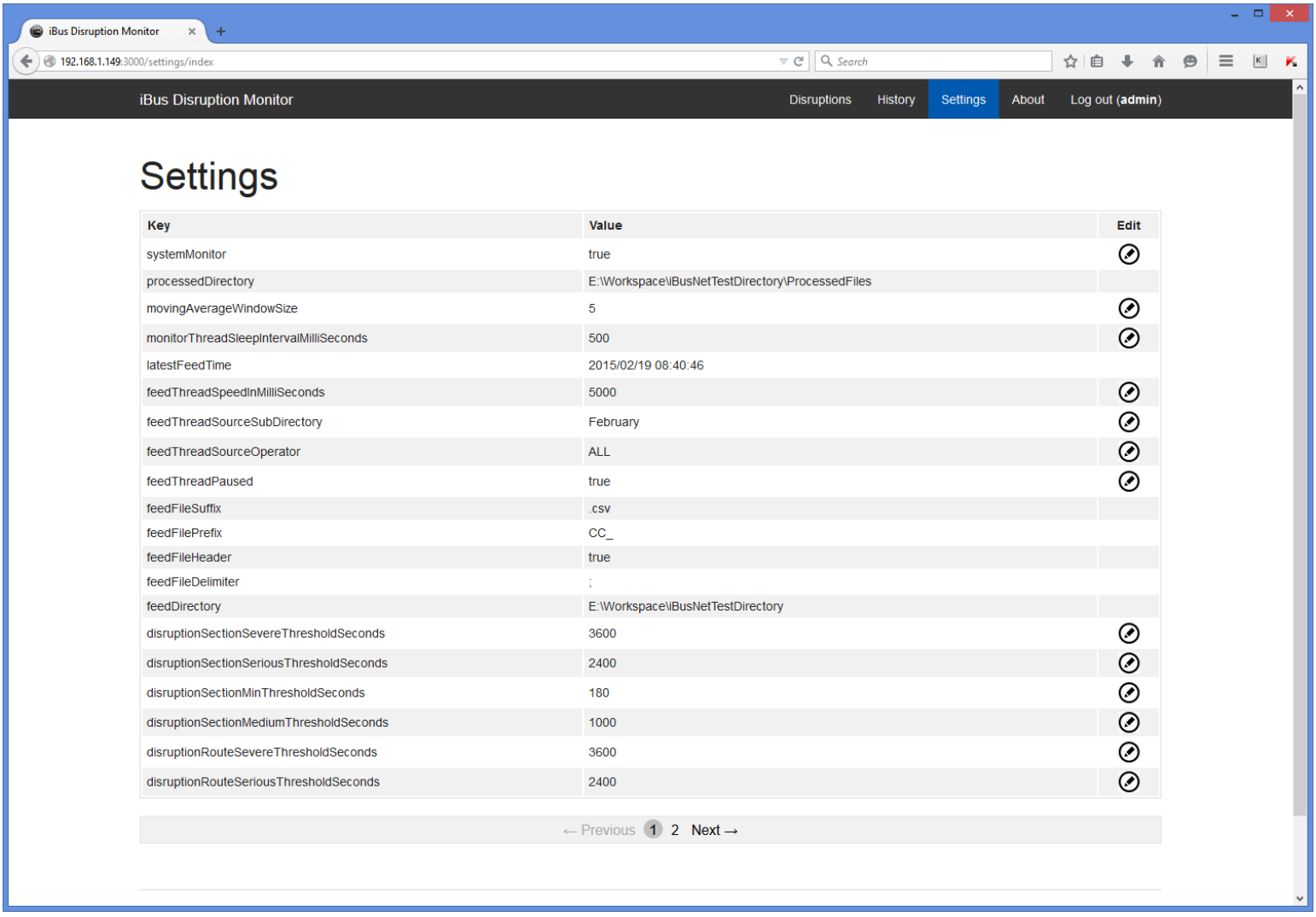

Figure A.12: Settings page view

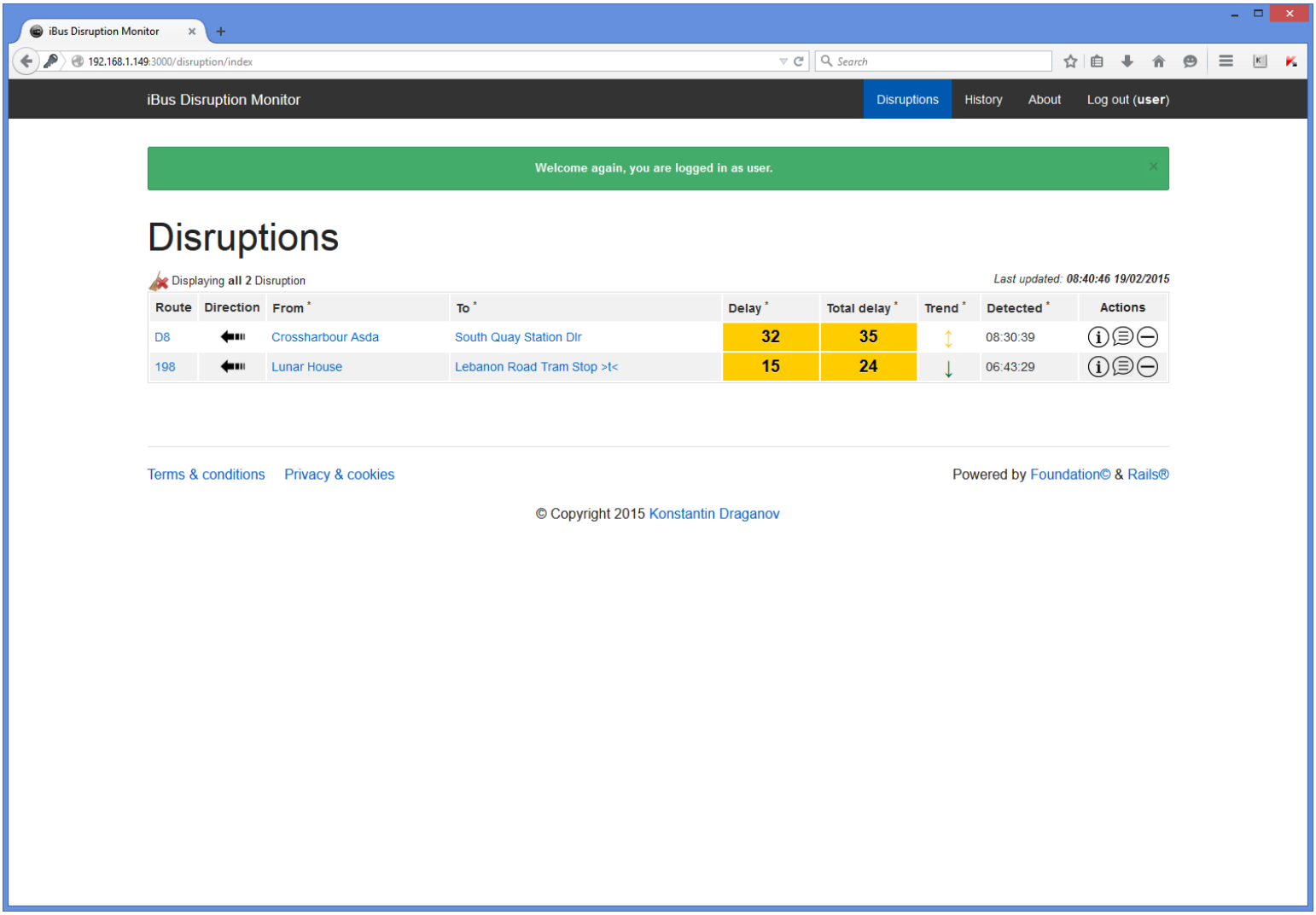

Figure A.13: View after successful log in

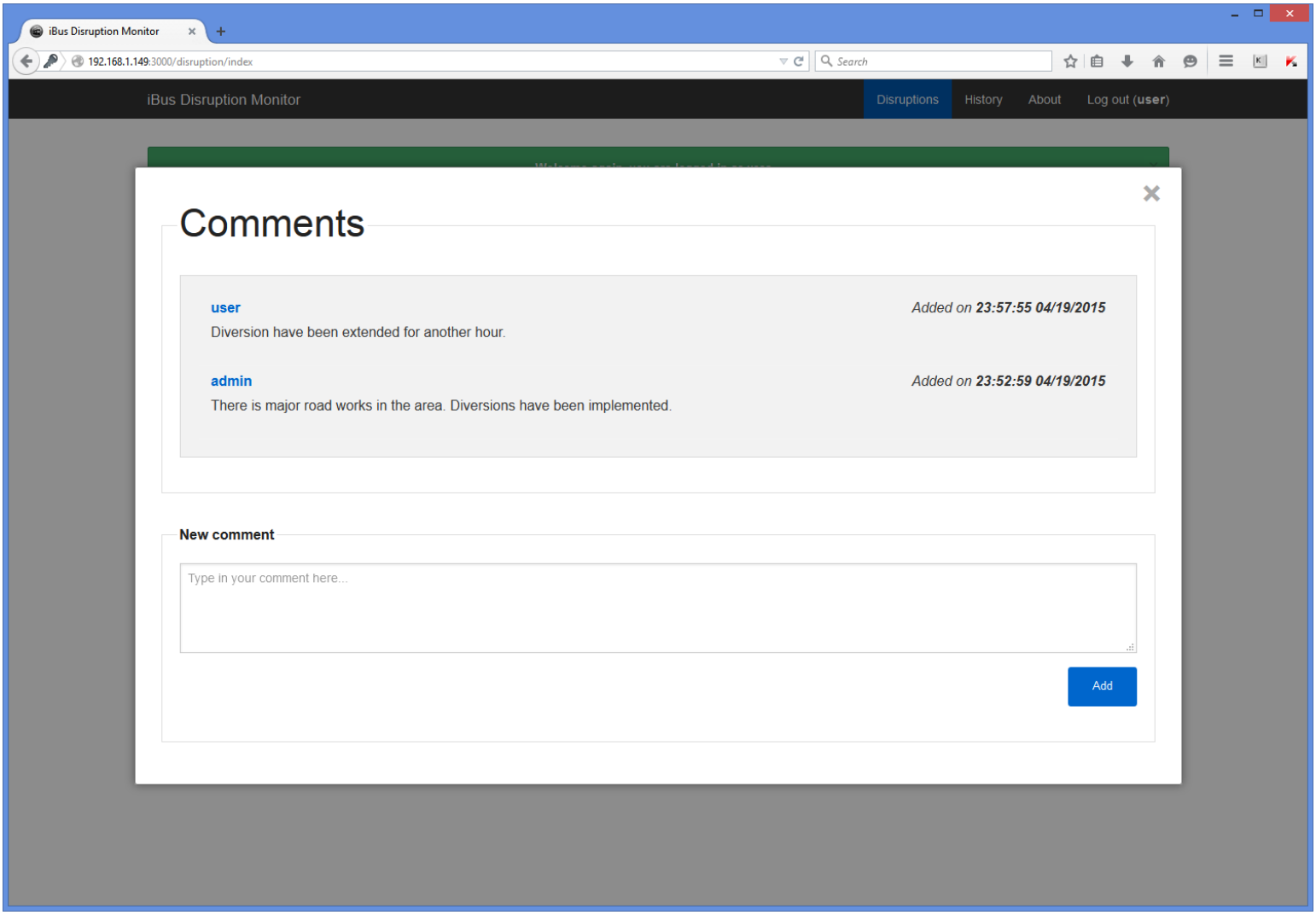

Figure A.14: Comments view

Appendix B

User Guide

B.1 Disruption Engine

Below are the instructions how to set up and run the disruption engine tool.

B.1.1 Installation

The disruption engine installation consists of the following steps:

1. Obtain the jar package of the tool.
2. Obtain and have all of the above dependencies in the CLASSPATH of the system.
3. Set up a database using the provided database script at the end of the source code listing.
4. Create XML file which to contain the connection settings to the respective database.

Below is the list of the required libraries and dependencies:

- JRE 8 - the library and documentation can be found on `http://www.oracle.com/technetwork/java/javase/documentation/index.html`.
- Scala Library 2.11.5 - `http://www.scala-lang.org/news/2.11.5`.

- Scala XML 2.11-1.0.2 - is used for working with XML files.
- JDBC PostgreSQL - PostgreSQL 9.0 JDBC3 and PostgreSQL 9.4 JDBC4 are both used which can be found in `https://jdbc.postgresql.org/download.html`.
- SLF4J 1.6.4 - is used for logging. Documentation and the library can be found here `http://www.slf4j.org/`. It also requires the bellow two libraries in order to work.
- Logback 1.0.1 - is used for logging `http://logback.qos.ch/`. The engine makes use of the classic and core packages both version 1.0.1.
- JCoord-1.0 - is used for converting easting/northing locations to the respective longitude/latitude values. The library was obtained from `http://www.jstott.me.uk/jcoord/`.
- ScalaTest 2.2.4 - library was used for testing. Instructions and the library files can be found in `http://www.scalatest.org/download`.

B.1.2 Execution

In order to run the application you simply need to execute the iBusDisruptionMonitor.jar with the following command from the command line:

```
java -jar iBusDisruptionMonitor.jar [path]
```

In the above command you need to substitute [path] with the path to an XML file containing the connection settings to a database. The XML file should have the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<connection>
    <host>[HOST]</host>
    <port>[PORT]</port>
    <database>[Database name]</database>
    <user>[USERNAME]</user>
```

```
    <password>[PASSWORD]</password>
    <maxPoolSize>5</maxPoolSize>
</connection>
```

In this XML file you need to substitute everything between the square brackets with the respective values for your configuration. The database creation script has been provided in the source code listing.

Executing the above command will start the tool, but do make sure you have set up the right configuration settings in the database before running the application.

If required or in case this moves to production it will be best to run the engine as a service. This can be achieved by using Java Service Wrapper `http://wrapper.tanukisoftware.com/doc/english/download.jsp`.

B.2 Web Application

The web application is build using Ruby 2.1.5 (`https://www.ruby-lang.org/en/`) on Rails 4.2.0 framework (`http://rubyonrails.org/`). The testing and simulations have been carried out on WEBrick 1.3.1 server (`http://ruby-doc.org/stdlib-2.0/libdoc/webrick/rdoc/WEBrick.html`) within the IntelliJ IDE. The application makes use of the Foundation framework (`http://foundation.zurb.com/`) and some other third-part libraries developed for Ruby on Rails. Full list of the dependencies and how to set up the web application can be found in the read-me file inside the archive with the source code submitted as part of this project.

B.3 Database

The database used for this project is PostgreSQL 9.4 `https://wiki.postgresql.org/wiki/What\%27s_new_in_PostgreSQL_9.4`. The database creation scripts can be found in the source code listing for this project. In the source code archive that is submitted along this report I have included a dump of the

database which includes all bus stops and bus routes from TFL's bus network. This data is from March 2015 and is not up to date as TFL publishes updated information every few weeks.

Appendix C

Source Code

Listings

Listing C.1: Engine Source - App Object (Main Class)

```
1  package main
2
3  import java.io.FileNotFoundException
4
5  import org.slf4j.LoggerFactory
6  import org.xml.sax.SAXParseException
7  import utility._
8
9  /**
10  * Created by Konstantin on 20/01/2015.
11  *
12  * Main object which acts as the entry point of the system.
13  * The system expects a single to be provided with a single argument
14  * which is the path to the XML file containing the details for
        connecting
15  * to the database
16  */
17 object app {
18
19   private val logger = LoggerFactory.getLogger("MainApp")
20
21   def main(args: Array[String]) {
22     if (args.isEmpty || (args(0) == null || args(0) == None ||
           args(0).length <= 0)) {
23       logger.error("Missing arguments: Unspecified configuration file.")
24       logger.error("Program will terminate!")
25       System.exit(-1)
26     }
27
28     try {
```

```
29        DBConnectionPool.createPool(args(0))
30      } catch {
31        case e: FileNotFoundException =>
32          logger.info("Configuration file not found or cannot be
                accessed.")
33          logger.error("Exception:", e)
34          logger.info("Program will terminate!")
35          System.exit(-1)
36        case e: NumberFormatException =>
37          logger.info("Incorrect connection settings.")
38          logger.error("Exception:", e)
39          logger.info("Program will terminate!")
40          System.exit(-1)
41        case e: SAXParseException =>
42          logger.info("Incorrect connection settings.")
43          logger.error("Exception:", e)
44          logger.info("Program will terminate!")
45          System.exit(-1)
46      }
47      Environment.init()
48  //    Environment.test()
49
50      val iBusMonitor = new iBusMonitor()
51      iBusMonitor.start()
52
53      //This is used for performance measurement
54      if (Environment.isSystemMonitorActive()) {
55        val systemMonitor = new SystemMonitor()
56        systemMonitor.start()
57      }
58    }
59
60  }
```

Listing C.2: Engine Source - BusStop Class

```
1  package lbsl
2
3  import java.sql.{Connection, PreparedStatement, SQLException}
4
5  import org.slf4j.LoggerFactory
6  import uk.me.jstott.jcoord.OSRef
7  import utility.DBConnectionPool
8
9  /**
10  * Created by Konstantin on 26/01/2015.
11  *
12  * Class representing a BusStop in TFL bus
13  * network. Currently this is unused, however if
14  * required the functionality is ready to be used.
15  *
16  */
17 class BusStop(
18               private val lbslCode: String,
19               private val name: String,
20               private val code: String,
21               private val NaptanAtco: String,
22               private val latitude: Double,
23               private val longitude: Double) {
24
25
26   def getLBSLCode(): String = lbslCode
27
28   def getName(): String = name
29
30   def getNaptanAtco(): String = NaptanAtco
31
32   def getCode(): String = code
```

```
33
34    def getLongitude(): Double = longitude
35
36    def getLatitude(): Double = latitude
37
38  }
39
40  object BusStop {
41
42    final val LBSLCode: Integer = 0
43    final val Code: Integer = 1
44    final val NaptanAtco: Integer = 2
45    final val StopName: Integer = 3
46    final val LocationEasting: Integer = 4
47    final val LocationNorthing: Integer = 5
48    final val Heading: Integer = 6
49    final val StopArea: Integer = 7
50    final val VirtualBusStop: Integer = 8
51
52    /**
53     * Static method for initialising a bus stop
54     * from the database.
55     * @param lbslCode String - the LBSL code of the bus stop
56     * @return - initialised BusStop
57     */
58    def getBusStop(lbslCode: String): BusStop = {
59      var busStop: BusStop = null
60      var connection: Connection = null
61      var preparedStatement: PreparedStatement = null
62      val selectSQL = "SELECT code, \"naptanAtcoCode\", name,
          \"locationEasting\", \"locationNorthing\" FROM \"BusStops\"
          WHERE \"lbslCode\" = ?"
63      try {
64        connection = DBConnectionPool.getConnection()
```

```
65        preparedStatement = connection.prepareStatement(selectSQL)
66        preparedStatement.setString(1, lbslCode)
67        val rs = preparedStatement.executeQuery()
68        if (rs.next()) {
69          val latLng = new OSRef(rs.getDouble("locationEasting"),
              rs.getDouble("locationNorthing")).toLatLng()
70          latLng.toWGS84()
71          busStop = new BusStop(lbslCode, rs.getString("name"),
              rs.getString("code"), rs.getString("naptanAtcoCode"),
              latLng.getLat, latLng.getLng)
72        }
73      }
74      catch {
75        case e: SQLException =>
            LoggerFactory.getLogger(getClass().getSimpleName).error("Exception:",
            e)
76      } finally {
77        if (preparedStatement != null) {
78          preparedStatement.close()
79        }
80        if (connection != null) {
81          DBConnectionPool.returnConnection(connection)
82        }
83      }
84      return busStop
85    }
86  }
```

Listing C.3: Engine Source - CustomFilenameFilter Class

```
1  package utility
2
3  import java.io.{File, FilenameFilter}
4
```

```
/**
 * Created by Konstantin on 17/02/2015.
 *
 * Custom filename filter used for filtering files in a directory
     according to specific prefix and suffix.
 */
class CustomFilenameFilter(private val prefix: String, private val
    suffix: String) extends FilenameFilter {

  def accept(dir: File, name: String): Boolean = {
    if (name.startsWith(prefix) && name.endsWith(suffix)) {
      return true
    }
    return false
  }
}
```

Listing C.4: Engine Source - DBConnectionPool Class

```
package utility

import java.io.{File, FileNotFoundException}
import java.sql.Connection

import org.postgresql.jdbc3.Jdbc3PoolingDataSource

import scala.xml.XML



/**
 * Created by Konstantin on 17/03/2015.
 *
 * Class representing a Database connection pool.
 */
```

```
16 class DBConnectionPool(private val host: String,
17                       private val port: Integer,
18                       private val db: String,
19                       private val user: String,
20                       private val password: String,
21                       private val name: String = "DBConnectionPool") {
22
23   private val pool = new Jdbc3PoolingDataSource()
24   pool.setDataSourceName(name)
25   pool.setServerName(host)
26   pool.setDatabaseName(db)
27   pool.setUser(user)
28   pool.setPassword(password)
29   pool.setMaxConnections(5)
30   pool.setInitialConnections(2)
31   pool.setPortNumber(port)
32
33   /**
34    *
35    * @return Connection - a connection from the pool.
36    */
37   def getConnection(): Connection = {
38     return pool.getConnection()
39   }
40
41   /**
42    *
43    * @param maxConnections Integer - the max number of simultaneous
       connections
44    */
45   def setMaxConnections(maxConnections: Integer): Unit = {
46     pool.setMaxConnections(maxConnections)
47   }
48
```

```
49    /**
50     *
51     * @return Integer - the max number of simultaneous connections
52     */
53    def getMaxConnections(): Integer = pool.getMaxConnections
54
55    /**
56     * Closes the pool.
57     */
58    def close(): Unit = {
59      pool.close()
60    }
61  }
62
63  /**
64   * Static class representing a Database pool used accross the system.
65   */
66  object DBConnectionPool {
67
68    private var sourcePool: DBConnectionPool = null
69    var host = ""
70    var port = 0
71    var db = ""
72    var user = ""
73    var password = ""
74    var maxPoolSize = 5
75
76    /**
77     * Creates the pool
78     * @param host String - the host address
79     * @param port Integer - the port used to connect to the database
80     * @param db String - the database name
81     * @param user String - the username
82     * @param password String - the password
```

```
83    * @param name String - name for the pool
84    */
85   def createPool(host: String,
86                  port: Integer,
87                  db: String,
88                  user: String,
89                  password: String,
90                  name: String = "DBConnectionPool"
91                   ): Unit = {
92     sourcePool = new DBConnectionPool(host, port, db, user, password)
93   }
94
95   /**
96    *
97    * @param connectionSettingsPath String - the path to the file with
          the settings for connecting to the database
98    */
99   def createPool(connectionSettingsPath: String): Unit = {
100    val file = new File(connectionSettingsPath)
101    if (!file.exists() || !file.isFile || !file.canRead) {
102      throw new FileNotFoundException("Settings file [" +
           connectionSettingsPath + "] is missing or cannot be
           accessed.")
103    }
104    val settingsXML = XML.loadFile(file)
105    host = (settingsXML \\ "connection" \\ "host").text
106    port = Integer.parseInt((settingsXML \\ "connection" \\
         "port").text)
107    db = (settingsXML \\ "connection" \\ "database").text
108    user = (settingsXML \\ "connection" \\ "user").text
109    password = (settingsXML \\ "connection" \\ "password").text
110    maxPoolSize = Integer.parseInt((settingsXML \\ "connection" \\
         "password").text)
111    sourcePool = new DBConnectionPool(host, port, db, user, password)
```

```
112   }
113
114   def setPool(pool: DBConnectionPool): Unit = {
115     if (sourcePool != null) {
116       sourcePool.close
117     }
118     sourcePool = pool
119   }
120
121   def setMaxPoolSize(poolSize: Integer): Unit = {
122     sourcePool.setMaxConnections(poolSize)
123   }
124
125   def getMaxPoolSize(): Integer = {
126     sourcePool.getMaxConnections()
127   }
128
129   def getConnection(): Connection = {
130     sourcePool.getConnection()
131   }
132
133   def returnConnection(connection: Connection): Unit = {
134     connection.close()
135   }
136
137   def close(): Unit = {
138     sourcePool.close()
139
140   }
141 }
```

Listing C.5: Engine Source - DBTransaction Class

```
1 package scala.utility
```

```
2
3  import java.sql.{Connection, SQLException}
4
5  import _root_.utility.DBConnectionPool
6  import org.slf4j.LoggerFactory
7
8  /**
9   * Created by Konstantin on 01/04/2015.
10  *
11  * Class representing a Database transaction.
12  */
13 class DBTransaction {
14
15   var connection: Connection = null
16   val logger = LoggerFactory.getLogger(getClass().getSimpleName)
17
18   /**
19    *
20    * @return Connection - a connection for this transaction
21    */
22   def getConnection = connection
23
24   /**
25    * Method for opening/beginning the transaction.
26    */
27   def begin(): Unit = {
28     try {
29       connection = DBConnectionPool.getConnection()
30       connection.setAutoCommit(false)
31     } catch {
32       case e: SQLException =>
             LoggerFactory.getLogger(getClass().getSimpleName).error("Exception:",
             e)
33         logger.error("Terminating application.")
```

```
34      }
35    }
36
37    /**
38     * Method used to commit the transaction.
39     */
40    def commit(): Unit = {
41      try {
42        connection.commit()
43      } catch {
44        case e: SQLException => logger.error("Exception:", e)
45          logger.error("Terminating application.")
46          connection.rollback()
47      } finally {
48        if (connection != null) {
49          connection.close();
50        }
51      }
52      connection = null
53    }
54
55    /**
56     * Method used to rollback the transaction.
57     */
58    def rollback(): Unit = {
59      try {
60        connection.rollback()
61      } catch {
62        case e: SQLException => logger.error("Exception:", e)
63          logger.error("Terminating application.")
64      } finally {
65        if (connection != null) {
66          connection.close();
67        }
```

```
68      }
69      connection = null
70    }
71
72  }
```

Listing C.6: Engine Source - Disruption Class

```
1   package lbsl
2
3   import java.sql._
4   import java.util.Date
5
6   import _root_.utility.{DBConnectionPool, Environment}
7   import org.slf4j.LoggerFactory
8
9   /**
10   * Created by Konstantin on 04/02/2015.
11   *
12   * Class which represent a disruption in the bus network.
13   */
14  class Disruption(private var sectionStartIndex: Integer,
15                   private var sectionEndIndex: Integer,
16                   private var sectionStart: String,
17                   private var sectionEnd: String,
18                   private var delaySeconds: Double,
19                   private var totalDelaySeconds: Double,
20                   private val timeFirstDetected: Date) {
21
22    private var id: Integer = null
23    private val logger = LoggerFactory.getLogger(getClass().getSimpleName)
24    private var trend: Integer = Disruption.TrendWorsening
25
26    def getSectionStartIndex: Integer = sectionStartIndex
```

```
27
28  def getSectionEndIndex: Integer = sectionEndIndex
29
30  def getSectionStartBusStop: String = sectionStart
31
32  def getSectionEndBusStop: String = sectionEnd
33
34  /**
35   *
36   * @return Integer - representing the total
37   *         cumulative delay along a route run in seconds
38   */
39  def getTotalDelay: Integer = {
40    return totalDelaySeconds.toInt
41  }
42
43  /**
44   *
45   * @return Integer - representing the total
46   *         cumulative delay along a route run in minutes
47   */
48  def getTotalDelayInMinutes: Integer = {
49    return getTotalDelay / 60
50  }
51
52  /**
53   *
54   * @return Integer - representing the delay in seconds
55   */
56  def getDelay: Integer = {
57    return delaySeconds.toInt
58  }
59
60  /**
```

```
61      *
62      * @return Integer - representing the delay in minutes.
63      */
64     def getDelayInMinutes: Integer = {
65       return getDelay / 60
66     }
67
68     /**
69      *
70      * @return Date - the time when the disruption was first detected
71      */
72     def getTimeFirstDetected: Date = timeFirstDetected
73
74     /**
75      *
76      * @return Integer - representing the trend direction of the
           disruption
77      */
78     def getTrend: Integer = trend
79
80     /**
81      * Method which updates the disruption parameters.
82      * @param newDelaySeconds
83      * @param newTotalDelaySeconds
84      */
85     def update(newDelaySeconds: Double, newTotalDelaySeconds: Double):
           Unit = {
86       update(sectionStartIndex, sectionEndIndex, sectionStart,
             sectionEnd, newDelaySeconds, newTotalDelaySeconds)
87     }
88
89     /**
90      * Method which updates the disruption parameters.
91      * @param newSectionStartIndex
```

```
92    * @param newSectionEndIndex
93    * @param newSectionStart
94    * @param newSectionEnd
95    * @param newDelaySeconds
96    * @param newTotalDelaySeconds
97    */
98   def update(newSectionStartIndex: Integer, newSectionEndIndex:
          Integer, newSectionStart: String, newSectionEnd: String,
          newDelaySeconds: Double, newTotalDelaySeconds: Double): Unit = {
99     val oldSectionSize = this.sectionEndIndex - this.sectionStartIndex
100    this.sectionStartIndex = newSectionStartIndex
101    this.sectionEndIndex = newSectionEndIndex
102    this.sectionStart = newSectionStart
103    this.sectionEnd = newSectionEnd
104    this.totalDelaySeconds = newTotalDelaySeconds
105    if (newDelaySeconds > delaySeconds) {
106      trend = Disruption.TrendWorsening
107    } else if (newDelaySeconds < delaySeconds) {
108      trend = Disruption.TrendImproving
109    } else if (oldSectionSize < this.sectionEndIndex -
           this.sectionStartIndex) {
110      trend = Disruption.TrendWorsening
111    } else {
112      trend = Disruption.TrendStable
113    }
114    this.delaySeconds = newDelaySeconds
115  }
116
117  /**
118   *
119   * @param that Disruption - to which to compare this disruption
120   * @return Boolean - true if the disruptions are considered as the
         same
121   *         false otherwise
```

```
122     */
123   def equals(that: Disruption): Boolean = {
124     if (this.sectionStartIndex == that.sectionStartIndex ||
125       this.sectionEndIndex == that.sectionEndIndex) {
126       return true
127     } else if ((this.sectionStartIndex >= that.sectionStartIndex &&
          this.sectionStartIndex <= that.sectionEndIndex) ||
128       (that.sectionStartIndex >= this.sectionStartIndex &&
            that.sectionStartIndex <= this.sectionEndIndex)) {
129       return true
130     } else if ((this.sectionEndIndex <= that.sectionEndIndex &&
          this.sectionEndIndex >= that.sectionStartIndex) ||
131       (that.sectionEndIndex <= this.sectionEndIndex &&
            that.sectionEndIndex >= this.sectionStartIndex)) {
132       return true
133     }
134     return false
135   }
136
137   /**
138    * It updates the disruption and marks it as cleared
139    *
140    * @param date Date - the date and time when this disruption has been
         cleared
141    */
142   def clear(date: Date): Unit = {
143     updateDBEntry(date)
144   }
145
146   /**
147    *
148    * @param route String - the route on which the disruption occurred
149    * @param run Integer - the run on which the disruption occurred
150    */
```

```
151   def save(route: String, run: Integer): Unit = {
152     if (id == null) {
153       id = Disruption.getNextId()
154       newEntry(route, run)
155     } else {
156       updateDBEntry()
157     }
158   }
159
160   private def updateDBEntry(clearedAt: Date = null): Unit = {
161     var preparedStatement: PreparedStatement = null
162     val query = "UPDATE \"Disruptions\" SET \"fromStopLBSLCode\" = ?,
          \"toStopLBSLCode\" = ?, \"delayInSeconds\" = ?, trend = ?,
          \"routeTotalDelayInSeconds\" = ?, \"clearedAt\" = ? WHERE id =
          ? ;"
163     try {
164       preparedStatement =
            Environment.getDBTransaction.connection.prepareStatement(query)
165       preparedStatement.setString(1, sectionStart)
166       preparedStatement.setString(2, sectionEnd)
167       preparedStatement.setDouble(3, delaySeconds)
168       preparedStatement.setInt(4, trend)
169       preparedStatement.setDouble(5, totalDelaySeconds)
170       if (clearedAt != null) {
171         preparedStatement.setTimestamp(6, new
              Timestamp(clearedAt.getTime))
172       } else {
173         preparedStatement.setNull(6, Types.NULL)
174       }
175       preparedStatement.setInt(7, id)
176       preparedStatement.executeUpdate()
177     }
178     catch {
```

```
179       case e: SQLException => logger.error("Exception: with query ({})
              ", preparedStatement.toString, e)
180     } finally {
181       if (preparedStatement != null) {
182         preparedStatement.close()
183       }
184     }
185   }
186
187   private def newEntry(route: String, run: Integer): Unit = {
188     //Used for testing only
189     if (delaySeconds > 2400 || totalDelaySeconds > 2400) {
190       logger.debug("New disruption added on route {} run {} with delay
              {} mins and total delay {} mins.",
191         scala.Array[Object](route, run.toString, (delaySeconds /
                60).toString, (totalDelaySeconds / 60).toString))
192     }
193     var preparedStatement: PreparedStatement = null
194     val query = "INSERT INTO \"Disruptions\" (id, \"fromStopLBSLCode\",
              \"toStopLBSLCode\", route, run, \"delayInSeconds\",
              \"firstDetectedAt\", trend, \"routeTotalDelayInSeconds\")
              VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?);"
195     try {
196       preparedStatement =
              Environment.getDBTransaction.connection.prepareStatement(query)
197       preparedStatement.setInt(1, id)
198       preparedStatement.setString(2, sectionStart)
199       preparedStatement.setString(3, sectionEnd)
200       preparedStatement.setString(4, route)
201       preparedStatement.setInt(5, run)
202       preparedStatement.setDouble(6, delaySeconds)
203       preparedStatement.setTimestamp(7, new
              Timestamp(timeFirstDetected.getTime))
204       preparedStatement.setInt(8, trend)
```

```
205       preparedStatement.setDouble(9, totalDelaySeconds)
206       preparedStatement.executeUpdate()
207     }
208     catch {
209       case e: SQLException => logger.error("Exception: with query ({})
              ", preparedStatement.toString, e)
210     } finally {
211       if (preparedStatement != null) {
212         preparedStatement.close()
213       }
214     }
215   }
216 }
217
218 /**
219  * Contains some static methods
220  */
221 object Disruption {
222
223   // NRT delay classifications:
224   // Moderate - 0 - 20 min
225   // Serious - 21 - 40 min
226   // Severe - 41 - 60 min
227   private var id: Integer = null
228
229   final val TrendImproving = 1
230   final val TrendStable = 0
231   final val TrendWorsening = -1
232
233   /**
234    *
235    * @return Integer - the next id to be used for a disruption
236    */
237   def getNextId(): Integer = {
```

```
238     this.synchronized {
239       if (id == null) {
240         getMaxId()
241       }
242       id += 1
243       return id
244     }
245   }
246
247   private def getMaxId() {
248     var connection: Connection = null
249     var statement: Statement = null
250     try {
251       connection = DBConnectionPool.getConnection()
252       statement = connection.createStatement()
253       val rs = statement.executeQuery("SELECT max(id) FROM
              \"Disruptions\"")
254       while (rs.next()) {
255         id = rs.getInt(1)
256       }
257     }
258     catch {
259       case e: SQLException =>
              LoggerFactory.getLogger(getClass().getSimpleName).error("Exception:",
              e)
260     } finally {
261       if (statement != null) {
262         statement.close()
263       }
264       if (connection != null) {
265         DBConnectionPool.returnConnection(connection)
266       }
267     }
268   }
```

```
}
```

Listing C.7: Engine Source - Environment Class

```
package utility

import java.io.File
import java.sql.{Connection, PreparedStatement, SQLException}
import java.text.SimpleDateFormat
import java.util.Date

import org.slf4j.LoggerFactory

import scala.collection.mutable
import scala.utility.DBTransaction

/**
 * Created by Konstantin on 22/03/2015.
 *
 * Static class representing the Environment variables
 * and configurations. Which are all loaded from the database.
 */
object Environment {

  private val logger = LoggerFactory.getLogger(getClass().getSimpleName)
  private val expectedParams = Array[String](
    "dateFormat",
    "disruptionRouteSeriousThresholdSeconds",
    "disruptionRouteSevereThresholdSeconds",
    "disruptionSectionMediumThresholdSeconds",
    "disruptionSectionMinThresholdSeconds",
    "disruptionSectionSeriousThresholdSeconds",
    "disruptionSectionSevereThresholdSeconds",
    "feedDirectory",
```

```
    "feedFileDelimiter",
    "feedFileHeader",
    "feedFilePrefix",
    "feedFileSuffix",
    "monitorThreadSleepIntervalMilliSeconds",
    "processedDirectory",
    "dataValidityTimeInMinutes",
    "movingAverageWindowSize",
    "systemMonitor"
  )

  private var latestFeedTimeOfData: Date = new Date(0)
  private val paramsMap = new mutable.HashMap[String, String]()
  private val quoteRegex: String =
      "(?=([^\\\"]*\\\"[^\\\"]*\\\")*[^\\\"]*$)"
  private val dbTransaction: DBTransaction = new DBTransaction

  def isSystemMonitorActive(): Boolean =
      getValue("systemMonitor").toBoolean

  def getMovingAverageWindowSize(): Integer = {
    return Integer.parseInt(getValue("movingAverageWindowSize"))
  }

  def getDBTransaction = dbTransaction

  def getLatestFeedTimeOfData = latestFeedTimeOfData

  def setLatestFeedTimeOfData(date: Date): Unit = {
    latestFeedTimeOfData = date
  }

  def getFeedDirectory(): File = {
    val feedDirectory = new File(getValue("feedDirectory"))
```

```
63    if (!feedDirectory.exists()) {
64      logger.warn("Processed directory missing. Trying to create
          directory [{}].", feedDirectory.getAbsolutePath)
65      if (!feedDirectory.mkdir()) {
66        logger.error("Failed to create directory [{}].Terminating
            application.", feedDirectory.getAbsolutePath)
67        System.exit(1)
68      }
69    }
70    return feedDirectory
71  }
72
73  def getFeedFilePrefix: String = getValue("feedFilePrefix")
74
75  def getFeedFileSuffix: String = getValue("feedFileSuffix")
76
77  def getFeedFileDelimiter: String = getValue("feedFileDelimiter")
78
79  def getFeedFileRegex: String = getFeedFileDelimiter + quoteRegex
80
81  def getFeedFileHeader: Boolean = {
82    getValue("feedFileHeader").toBoolean
83  }
84
85  def getDataValidityTimeInMinutes: Integer = {
86    return Integer.parseInt(getValue("dataValidityTimeInMinutes"))
87  }
88
89  def getProcessedDirectory(): File = {
90    val processedDirectory = new File(getValue("processedDirectory"))
91    if (!processedDirectory.exists()) {
92      logger.warn("Processed directory missing. Trying to create
          directory [{}].", processedDirectory.getAbsolutePath)
93      if (!processedDirectory.mkdir()) {
```

```
94        logger.error("Failed to create directory [{}].Terminating
              application.", processedDirectory.getAbsolutePath)
95        System.exit(1)
96      }
97    }
98    return processedDirectory
99  }
100
101 def getMonitorThreadSleepInterval(): Long = {
102   return getValue("monitorThreadSleepIntervalMilliSeconds").toLong
103 }
104
105 def getSectionMediumThreshold: Integer = {
106   return
        Integer.parseInt(getValue("disruptionSectionMediumThresholdSeconds"))
107 }
108
109 def getSectionSeriousThreshold: Integer = {
110   return
        Integer.parseInt(getValue("disruptionSectionSeriousThresholdSeconds"))
111 }
112
113 def getSectionSevereThreshold: Integer = {
114   return
        Integer.parseInt(getValue("disruptionSectionSevereThresholdSeconds"))
115 }
116
117 def getSectionMinThreshold: Integer = {
118   return
        Integer.parseInt(getValue("disruptionSectionMinThresholdSeconds"))
119 }
120
121 def getRouteSevereThreshold: Integer = {
```

```
122     return
            Integer.parseInt(getValue("disruptionRouteSevereThresholdSeconds"))
123   }
124
125   def getRouteSeriousThreshold(): Integer = {
126     return
            Integer.parseInt(getValue("disruptionRouteSeriousThresholdSeconds"))
127   }
128
129   def getDateFormat(): SimpleDateFormat = {
130     return new SimpleDateFormat(getValue("dateFormat"))
131   }
132
133   def init(): Unit = {
134     loadParams()
135     checkParams()
136   }
137
138   def update(): Unit = {
139     loadParams()
140     checkParams()
141   }
142
143   private def getValue(key: String): String = {
144     return paramsMap.getOrElse(key, null)
145   }
146
147   private def loadParams(): Unit = {
148     var connection: Connection = null
149     var preparedStatement: PreparedStatement = null
150     val query = "SELECT key, value FROM \"EngineConfigurations\""
151     try {
152       connection = DBConnectionPool.getConnection()
153       preparedStatement = connection.prepareStatement(query)
```

```
154       val rs = preparedStatement.executeQuery()
155       while (rs.next()) {
156         paramsMap.put(rs.getString("key"), rs.getString("value"))
157       }
158     }
159     catch {
160       case e: SQLException => logger.error("Exception:", e)
161     } finally {
162       if (preparedStatement != null) {
163         preparedStatement.close()
164       }
165       if (connection != null) {
166         DBConnectionPool.returnConnection(connection)
167       }
168     }
169   }
170
171   private def checkParams(): Unit = {
172     for (key <- expectedParams) {
173       if (paramsMap.get(key) == None) {
174         logger.warn("Missing configuration [{}] parameter. Terminating
              application.", key)
175         System.exit(-1)
176       }
177       if (paramsMap.get(key) == null) {
178         logger.warn("Missing configuration value for [{}]. Terminating
              application.", key)
179         System.exit(-1)
180       }
181     }
182   }
183
184   def test(): Unit = {
```

```
185     logger.trace("MonitorThreadSleepInterval - [{}]",
            getMonitorThreadSleepInterval().toString)
186     logger.trace("FeedsDirectory - [{}]",
            getFeedDirectory().getAbsolutePath)
187     logger.trace("ProcessedDirectory - [{}]",
            getProcessedDirectory().getAbsolutePath)
188     logger.trace("DateFormat - [{}]", getDateFormat().toPattern)
189     logger.trace("FeedFilePrefix - [{}]", getFeedFilePrefix)
190     logger.trace("FeedFileSuffix - [{}]", getFeedFileSuffix)
191     logger.trace("FeedFileDelimiter - [{}]", getFeedFileDelimiter)
192     logger.trace("FeedFileHeader - [{}]", getFeedFileHeader.toString)
193     logger.trace("FeedFileRegex - [{}]", getFeedFileRegex)
194     logger.trace("SectionMediumThreshold - [{}]",
            getSectionMediumThreshold.toString)
195     logger.trace("SectionSeriousThreshold - [{}]",
            getSectionSeriousThreshold.toString)
196     logger.trace("SectionSevereThreshold - [{}]",
            getSectionSevereThreshold.toString)
197     logger.trace("RouteSeriousThreshold - [{}]",
            getRouteSeriousThreshold.toString)
198     logger.trace("RouteSevereThreshold - [{}]",
            getRouteSevereThreshold.toString)
199     logger.trace("SectionMinThreshold - [{}]",
            getSectionMinThreshold.toString)
200     logger.trace("Data validity in minutes - [{}]",
            getDataValidityTimeInMinutes.toString)
201     logger.trace("Moving Average Window Size - [{}]",
            getMovingAverageWindowSize.toString)
202     logger.trace("System Monitor - [{}]",
            isSystemMonitorActive.toString)
203   }
204 }
```

Listing C.8: Engine Source - FeedThread Class

```
1  package utility
2
3  import java.io.{File, FileNotFoundException}
4  import java.nio.file.{FileSystems, Files, StandardCopyOption}
5  import java.sql.{Connection, PreparedStatement, SQLException}
6
7  import org.slf4j.LoggerFactory
8
9  import scala.collection.mutable.ArrayBuffer
10
11 /**
12  * Created by Konstantin on 17/02/2015.
13  *
14  * Used for simulation, however it has been moved as a separate
       application.
15  */
16 class FeedThread(private val subDir: String, private val operator:
      String, private var sleepInterval: Long = 5000) extends Thread {
17
18   private val logger = LoggerFactory.getLogger(getClass().getSimpleName)
19   private val feedDirectory: File = new
         File("E:\\Workspace\\iBusNetTestDirectory\\Feeds\\" + subDir)
20   private val feedFilenameFilter = new CustomFilenameFilter("CC_",
         ".csv")
21   private val operatorFilenameFilter = new CustomFilenameFilter("CC_",
         operator + "_YYYYMMDD_NNNNN")
22   private val feedsBuffer: ArrayBuffer[File] = new ArrayBuffer[File]()
23
24   def init(): Unit = {
25     val tempBuffer = new ArrayBuffer[File]()
26     for (operatorDir: File <-
           feedDirectory.listFiles(operatorFilenameFilter) if
```

```
      operatorDir.isDirectory) {
    tempBuffer.appendAll(operatorDir.listFiles(feedFilenameFilter))
  }
  feedsBuffer.appendAll(tempBuffer.sortBy(f =>
      (f.getName.substring(f.getName.indexOf("_", 3) + 1))))
  logger.debug("{} feed files loaded in buffer.", feedsBuffer.size)
}

override
def run(): Unit = {
  init()
  var terminate = false
  val seen: ArrayBuffer[String] = new ArrayBuffer[String]()
  while (!terminate) {
    seen.clear()
    speedControl()
    var seenAll = false
    try {
      while (!seenAll && !terminate) {
        val fileOperator = feedsBuffer(0).getName.substring(3,
            feedsBuffer(0).getName.indexOf("_", 3))
        if (seen.contains(fileOperator)) {
          seenAll = true
        } else {
          seen.append(fileOperator)
          copy(feedsBuffer.remove(0))
        }
        terminate = feedsBuffer.isEmpty
      }
    } catch {
      case e: FileNotFoundException => logger.error("File {}
          generated FileNotFoundException. File is being ignored.",
          feedsBuffer.remove(0).getAbsolutePath)
```

```
55          case e: Exception => logger.error("TERMINATING - FeedsThread
                interrupted:", e)
56            System.exit(-1)
57        }
58
59        try {
60          Thread.sleep(sleepInterval)
61        } catch {
62          case e: InterruptedException => logger.error("Feed thread
                interrupted:", e)
63        }
64      }
65      logger.debug("All feed files have been copied.")
66      logger.debug("Feed thread completed!")
67    }
68
69    @throws(classOf[FileNotFoundException])
70    private def copy(file: File): Unit = {
71      logger.trace("Copying file [{}] to {}.", file.getName,
            Environment.getFeedDirectory().getName)
72      val sourceFile =
            FileSystems.getDefault.getPath(file.getAbsolutePath)
73      val destinationFile =
            FileSystems.getDefault.getPath(Environment.getFeedDirectory().getAbsolutePath,
            file.getName)
74      Files.copy(sourceFile, destinationFile,
            StandardCopyOption.REPLACE_EXISTING)
75    }
76
77    private def speedControl(): Unit = {
78      var pause = true
79      while (pause) {
80        var connection: Connection = null
81        var preparedStatement: PreparedStatement = null
```

```
82      val selectSQL = "SELECT * FROM \"EngineConfigurations\" WHERE key
            like 'feedThread%'"
83      try {
84        connection = DBConnectionPool.getConnection()
85        preparedStatement = connection.prepareStatement(selectSQL)
86        val rs = preparedStatement.executeQuery()
87        while (rs.next()) {
88          if (rs.getString("key") == "feedThreadPaused") {
89            pause = rs.getBoolean("value")
90          } else if (rs.getString("key") ==
                "feedThreadSpeedInMilliSeconds") {
91            sleepInterval = rs.getLong("value")
92          }
93        }
94      }
95      catch {
96        case e: SQLException => logger.error("Exception:", e)
97      } finally {
98        if (preparedStatement != null) {
99          preparedStatement.close()
100       }
101       if (connection != null) {
102         DBConnectionPool.returnConnection(connection)
103       }
104     }
105     if (pause) {
106       logger.debug("Feeds thread is paused.")
107       Thread.sleep(5000)
108     } else {
109       logger.debug("Feeds thread resuming ({}ms sleep interval).",
              sleepInterval)
110     }
111   }
112 }
```

```
113
114 }
```

Listing C.9: Engine Source - iBusMonitor Class

```
1  package main
2
3  import java.io.{File, FileNotFoundException, FilenameFilter}
4  import java.nio.file._
5
6  import lbsl.{Network, Observation}
7  import org.slf4j.LoggerFactory
8  import utility.{CustomFilenameFilter, Environment}
9
10 import scala.collection.JavaConversions._
11 import scala.io.Source
12
13 /**
14  * Created by Konstantin on 20/01/2015.
15  * Class which extends Thread and provides monitoring
16  * functionality. It is responsible for monitoring a predefined
17  * directory for new feed files.
18  *
19  */
20 class iBusMonitor() extends Thread {
21
22   private val busNetwork: Network = new Network
23   private val logger = LoggerFactory.getLogger(getClass().getSimpleName)
24   private var updateNetwork: Boolean = false;
25   private val feedFilenameFilter: FilenameFilter = new
       CustomFilenameFilter(Environment.getFeedFilePrefix,
       Environment.getFeedFileSuffix)
26
27   /**
```

```
28    * The run method of this class is expected to run continuously.
29    * It listens for new files being created/updated in a predefine
30    * directory. Once it detects a new file it checks if the file
31    * is following the expected naming conventions and format. If
32    * all holds the files is processed by extracting the observations.
33    * After processing the file is moved to processed feed files
          directory.
34    */
35   override
36   def run() {
37     init()
38     val watchService = FileSystems.getDefault.newWatchService()
39     Paths.get(Environment.getFeedDirectory().getAbsolutePath).register(watchService,
           StandardWatchEventKinds.ENTRY_CREATE,
           StandardWatchEventKinds.ENTRY_MODIFY)
40     while (true) {
41       val key = watchService.take()
42       val events = key.pollEvents()
43       for (event <- events) {
44         val event_path = event.context().asInstanceOf[Path]
45         if (event_path != null) {
46           val fileName = event_path.toString()
47           if (event.kind() == StandardWatchEventKinds.ENTRY_CREATE) {
48             if (fileName.startsWith(Environment.getFeedFilePrefix) &&
                   fileName.endsWith(Environment.getFeedFileSuffix)) {
49               //             logger.info("New file detected: {}",
                       fileName)
50               processFile(new
                       File(Environment.getFeedDirectory().getAbsolutePath +
                       "\\" + fileName))
51             }
52           }
53         }
54       }
```

```
55      key.reset()
56
57      //checks for any unprocessed files
58      for (file <-
            Environment.getFeedDirectory().listFiles(feedFilenameFilter)
            if file.isFile) {
59        processFile(file)
60      }
61      //Only update network if valid observations have been processed
62      if (updateNetwork) {
63        try {
64          busNetwork.update()
65        } catch {
66          case e: Exception => logger.error("TERMINATING -
                iBusMonitorThread interrupted:", e)
67            System.exit(-1)
68        }
69        updateNetwork = false
70      }
71      nap()
72    }
73  }
74
75  private def processFile(file: File): Unit = {
76    if (file.isFile && file.exists() && file.canRead() &&
          file.canExecute) {
77      logger.info("Processing file [{}].", file.getAbsolutePath)
78      while (file.exists()) {
79        try {
80          processFeed(file)
81        } catch {
82          case e: FileNotFoundException => nap(1000)
83            logger.info("File {} not accessible - sleeping for 1s." ,
                  file.getAbsolutePath) //logger.warn("Exception:", e)
```

```
84          case e: Exception => logger.error("TERMINATING -
                iBusMonitorThread interrupted:", e)
85            System.exit(-1)
86        }
87      }
88      updateNetwork = true
89    }
90  }
91
92  private def nap(timeMilliSeconds: Long =
        Environment.getMonitorThreadSleepInterval()): Unit = {
93    try {
94      Thread.sleep(timeMilliSeconds)
95    } catch {
96      case e: InterruptedException => logger.error("iBusMonitorThread
            interrupted:", e)
97    }
98  }
99
100 @throws(classOf[FileNotFoundException])
101 private def processFeed(file: File): Unit = {
102   val source = Source.fromFile(file.getAbsolutePath)
103   //Check whether to drop header
104   for (line <- source.getLines().drop(if
          (Environment.getFeedFileHeader) 1 else 0)) {
105     val observation = new Observation()
106     if (observation.init(line, file.getName)) {
107       busNetwork.addObservation(observation)
108     }
109   }
110   source.close
111   val sourceFile =
          FileSystems.getDefault.getPath(file.getAbsolutePath)
```

```
112     val destinationFile =
            FileSystems.getDefault.getPath(Environment.getProcessedDirectory().getAbsolutePath,
            file.getName)
113     Files.move(sourceFile, destinationFile,
            StandardCopyOption.REPLACE_EXISTING)
114   }
115
116   private def init(): Unit = {
117     logger.trace("Starting iBusMonitor initialisation.")
118     logger.trace("**************************************************")
119     val start = System.nanoTime()
120     busNetwork.init()
121     val elapsedTime = System.nanoTime() - start
122     logger.trace("Finished iBusMonitor initialisation in {} seconds.",
            (elapsedTime / 1000000000.0))
123     logger.trace("**************************************************")
124     logger.trace("Start monitoring folder [{}] for new file feeds.",
            Environment.getFeedDirectory().getAbsolutePath)
125   }
126
127 }
```

Listing C.10: Engine Source - MissingData Class

```
1  package utility
2
3  import org.slf4j.LoggerFactory
4
5  import scala.collection.mutable.HashMap
6
7  /**
8   * Created by Konstantin on 18/02/2015.
9   *
10  * Static class used for recording any missing
```

```
 * routes or bus stops which are observed in
 * the feed files, but are missing the bus
 * network data available on TFL Open Data website.
 */

object MissingData {
  private val logger = LoggerFactory.getLogger(getClass().getSimpleName)
  private val missingRoutes: HashMap[String, String] = new
      HashMap[String, String]()
  private val missingBusStops = new HashMap[String, String]()

  def addMissingRoute(route: String, operator: String): Unit = {
    if (missingRoutes.getOrElse(route, null) == null) {
      missingRoutes.put(route, operator)
      logRoutes()
    }
  }

  def addMissingStop(stop: String, route: String, file: String): Unit =
      {
    if (missingBusStops.getOrElse(stop, null) == null) {
      missingBusStops.put(stop, route + "_" + file)
      logStops()
    }
  }

  def logRoutes(): Unit = {
    var output = ""
    for ((route, operator) <- missingRoutes) {
      output += " | " + route + " => " + operator
    }
    logger.trace("MISSING BUS ROUTES: {}", output)
  }

```

```
43    def logStops(): Unit = {
44      var output = ""
45      for ((stop, route) <- missingBusStops) {
46        output += " | " + stop + " => " + route
47      }
48      logger.trace("MISSING BUS STOPS: {}", output)
49    }
50
51  }
```

Listing C.11: Engine Source - Network Class

```
1   package lbsl
2
3   import java.sql.{Connection, PreparedStatement, SQLException}
4
5   import org.slf4j.LoggerFactory
6   import utility.{DBConnectionPool, Environment, MissingData}
7
8   import scala.collection.mutable
9
10  /**
11   * Created by Konstantin
12   *
13   * This class represents a bus network.
14   */
15  class Network {
16
17    private val logger = LoggerFactory.getLogger(getClass().getSimpleName)
18    private val routeMap: mutable.HashMap[String, Route] = new
          mutable.HashMap[String, Route]()
19    private var maxExecutionTime: Double = 0
20
21    /**
```

```
22    * Method which updates the bus network state.
23    */
24   def update(): Unit = {
25     logger.info("BEGIN:Calculating disruptions...")
26     val start = System.nanoTime()
27     calculateDisruptions()
28     val elapsedTime = (System.nanoTime() - start) / 1000000000.0
29     maxExecutionTime = Math.max(maxExecutionTime, elapsedTime)
30     logger.info("FINISH:Calculating disruptions. Calculation time {}
           seconds (Max calculation time {}).", elapsedTime,
           maxExecutionTime)
31   }
32
33   def getRouteCount(): Integer = {
34     return routeMap.size
35   }
36
37   /**
38    *
39    * @param number the bus route number
40    * @return the bus route if exists,
41    *         otherwise null
42    */
43   def getRoute(number: String): Route = routeMap.getOrElse(number, null)
44
45   /**
46    *
47    * @param observation Observation to be added to network
48    * @return true if bus route exists and observation has been added
           successfully,
49    *         otherwise false
50    */
51   def addObservation(observation: Observation): Boolean = {
52     var route = routeMap.getOrElse(observation.getContractRoute, null)
```

```
53    //Check if it is a 24h service bus
54    if (route == null && observation.getContractRoute.startsWith("N")) {
55      route =
            routeMap.getOrElse(observation.getContractRoute.substring(1),
            null)
56    }
57    if (route != null) {
58      val tempDate = observation.getTimeOfData
59      if (tempDate.getTime >
            Environment.getLatestFeedTimeOfData.getTime) {
60        Environment.setLatestFeedTimeOfData(tempDate)
61      }
62      route.addObservation(observation)
63      return true
64    }
65    MissingData.addMissingRoute(observation.getContractRoute,
          observation.getOperator)
66    return false
67  }
68
69  /**
70   * Initializes the bus network,
71   * it loads the bus stops and bus routes
72   */
73  def init(): Unit = {
74    logger.info("BEGIN: Loading bus routes.")
75    loadRoutes()
76    logger.info("FINISH: Loaded {} bus routes.", routeMap.size)
77  }
78
79  private def calculateDisruptions(): Unit = {
80    Environment.getDBTransaction.begin()
81    for ((routeNumber, route) <- routeMap) {
82      route.run()
```

```
83      }
84      var preparedStatement: PreparedStatement = null
85      val query = "UPDATE \"EngineConfigurations\" SET \"value\" = ?
            WHERE key = 'latestFeedTime'"
86      try {
87        preparedStatement =
              Environment.getDBTransaction.getConnection.prepareStatement(query)
88        preparedStatement.setString(1,
              Environment.getDateFormat().format(Environment.getLatestFeedTimeOfData))
89        preparedStatement.executeUpdate()
90      } catch {
91        case e: SQLException => logger.error("Exception: with query ({})
              ", preparedStatement.toString, e)
92          logger.error("Terminating application.")
93      } finally {
94        if (preparedStatement != null) {
95          preparedStatement.close()
96        }
97      }
98      Environment.getDBTransaction.commit()
99    }
100
101   private def loadRoutes(): Unit = {
102     var connection: Connection = null
103     var preparedStatement: PreparedStatement = null
104     val query = "SELECT DISTINCT route FROM \"BusRouteSequences\""
105     try {
106       connection = DBConnectionPool.getConnection()
107       preparedStatement = connection.prepareStatement(query)
108       val routesSet = preparedStatement.executeQuery()
109       while (routesSet.next()) {
110         routeMap.put(routesSet.getString("route"), new
                Route(routesSet.getString("route")))
111       }
```

```
112     }
113     catch {
114       case e: SQLException => logger.error("Exception:", e)
115     } finally {
116       if (preparedStatement != null) {
117         preparedStatement.close()
118       }
119       if (connection != null) {
120         DBConnectionPool.returnConnection(connection)
121       }
122     }
123     //TODO: Concurrent execution could be introduced here
124     for ((routeNumber, route) <- routeMap) {
125       route.init()
126     }
127   }
128
129 }
```

Listing C.12: Engine Source - Observation Class

```
1  package lbsl
2
3  import java.util.Date
4
5  import utility.Environment
6
7  /**
8   * Created by Konstantin on 21/01/2015.
9   *
10  * Class representing an observation of the
11  * bus for a given point in time. It is based
12  * on the data from the feed files which in turn
13  * are based on the transmitted AVL data.
```

```
 */
class Observation() extends Ordered[Observation] {

  private var vehicleId: Integer = 0
  private var timeOfData: Date = null
  private var tripType: Integer = 0
  private var tripId: Integer = 0
  private var contractRoute: String = null
  private var lastStopShortDesc: String = null
  private var scheduleDeviation: Integer = 0
  private var longitude: Double = 0
  private var latitude: Double = 0
  private var eventId: Integer = 0
  private var operator: String = null

  def getOperator: String = operator

  def getVehicleId: Integer = vehicleId

  def getTimeOfData: Date = timeOfData

  def getTripId: Integer = tripId

  def getTripType: Integer = tripType

  def getContractRoute: String = contractRoute

  def getLastStopShortDesc: String = lastStopShortDesc

  def getScheduleDeviation: Integer = scheduleDeviation

  def getLongitude: Double = longitude

  def getLatitude: Double = latitude
```

```
48
49  def getEventId: Integer = eventId
50
51  /**
52   *
53   * @return Boolean - true if the observation has valid
54   *         values for the parameters, false otherwise
55   */
56  def isValid: Boolean = {
57    if (scheduleDeviation == Observation.NegativeIntegerError) {
58      return false
59    }
60    return true
61  }
62
63  /**
64   *
65   * @param feed String - the line representing the observation in the
       feed file.
66   * @param companyOperator String - the bus operator company
67   * @return Boolean - true if the observation is valid, meaning it can
68   *         be used by the system, otherwise false
69   */
70  def init(feed: String, companyOperator: String): Boolean = {
71    val tokens: Array[String] = feed.split(Environment.getFeedFileRegex)
72    scheduleDeviation =
         Integer.parseInt(tokens(Observation.ScheduleDeviation))
73    tripType = Integer.parseInt(tokens(Observation.TripType))
74
75    if (scheduleDeviation == Observation.NegativeIntegerError ||
         !lbsl.TripType.isActiveTrip(tripType)) {
76      return false
77    }
78
```

```
79      vehicleId = Integer.parseInt(tokens(Observation.VehicleId))
80      longitude = tokens(Observation.Longitude).toDouble
81      latitude = tokens(Observation.Latitude).toDouble
82      eventId = Integer.parseInt(tokens(Observation.EventId))
83      timeOfData =
            Environment.getDateFormat().parse(tokens(Observation.TimeOfData))
84      contractRoute = tokens(Observation.ContractRoute)
85      lastStopShortDesc = tokens(Observation.LastStopShortDesc)
86      operator = companyOperator
87      tripId = Integer.parseInt(tokens(Observation.TripId))
88      return true
89    }
90
91    def compare(that: Observation) = this.getTimeOfData compareTo
          that.getTimeOfData
92  }
93
94  object Observation {
95
96    final val VehicleId: Integer = 0
97    final val BonnetCode: Integer = 1
98    final val RegistrationNumber: Integer = 2
99    final val TimeOfData: Integer = 3
100   final val BaseVersion: Integer = 4
101   final val BlockNumber: Integer = 5
102   final val TripId: Integer = 6
103   final val LBSLTripNumber: Integer = 7
104   final val TripType: Integer = 8
105   final val ContractRoute: Integer = 9
106   final val LastStopShortDesc: Integer = 10
107   final val ScheduleDeviation: Integer = 11
108   final val Longitude: Integer = 12
109   final val Latitude: Integer = 13
110   final val EventId: Integer = 14
```

```
111   final val Duration: Integer = 15
112
113   final val NegativeIntegerError: Integer = -2147483645
114 }
```

Listing C.13: Engine Source - OutputWriter Class

```
1  package utility
2
3  import java.io.{File, FileNotFoundException, PrintWriter}
4  import java.text.SimpleDateFormat
5  import java.util.{Calendar, Date}
6
7  import lbsl.BusStop
8  import org.slf4j.LoggerFactory
9
10 /**
11  * Created by Konstantin on 10/03/2015.
12  *
13  * Class used for writing the output of the system to CSV files.
14  * This however has become obsolete as the system has moved to use a
        database.
15  */
16 class OutputWriter {
17
18   private val logger = LoggerFactory.getLogger(getClass().getSimpleName)
19
20   private val fileDateFormat = new SimpleDateFormat("yyyyMMdd_HHmmss")
21   private val dateFormat = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss")
22   private val outputDirectory: File = new
         File("E:\\Workspace\\iBusNetTestDirectory\\DisruptionReports")
23   private val outputFilename: String = outputDirectory.getAbsolutePath
         + "\\Report.csv"
24   private val outputFile: File = new File(outputFilename)
```

```
25  private val header: String =
        "Route,Direction,FromStopName,FromStopCode,ToStopName,ToStopCode,DisruptionObserved,RouteTotal,
26
27  private var prevTime: String =
        fileDateFormat.format(Calendar.getInstance().getTime())
28  private var output: String = ""
29
30  //TODO: need to add quotes around the bus stop names in case they
        have commas
31  def write(contractRoute: String, direction: String, stopA: BusStop,
        stopB: BusStop, delayInMinutes: Integer, routeTotalDelayMinutes:
        Integer, trend: Integer, timeFirstDetected: Date): Unit = {
32    output += contractRoute + ","
33    output += direction + ","
34    output += "\"" + stopA.getName() + "\","
35    output += stopA.getCode() + ","
36    output += "\"" + stopB.getName() + "\","
37    output += stopB.getCode() + ","
38    output += delayInMinutes + ","
39    output += routeTotalDelayMinutes + ","
40    output += trend + ","
41    output += dateFormat.format(timeFirstDetected) + "\n"
42    logger.trace("{} - {} disrupted section between stop [{}] and stop
          [{}] of [{}] minutes. ", Array[Object](contractRoute,
          direction, stopA.getCode(), stopB.getCode(),
          delayInMinutes.toString))
43  }
44
45  /**
46   * Save to file if output is not empty
47   */
48  def close(): Unit = {
49    checkOutputDirectory()
50    try {
```

```
51      renameCurrent()
52    } catch {
53      case e: FileNotFoundException =>
54        logger.error("File {} is in use. Unable to access it.",
              outputFile.getAbsolutePath)
55        logger.error("Exception:", e)
56        logger.error("Terminating application.")
57        System.exit(1)
58    }
59    if (output.length > 0) {
60      save()
61    }
62  }
63
64  private def renameCurrent(): Unit = {
65    try {
66      if (outputFile.exists()) {
67        val newFile: File = new File(outputDirectory.getAbsolutePath +
              "\\Report_" + prevTime + ".csv")
68        outputFile.renameTo(newFile)
69      }
70
71    } catch {
72      case e: FileNotFoundException =>
73        logger.error("File {} is in use. Unable to access it.",
              outputFile.getAbsolutePath)
74        logger.error("Exception:", e)
75        logger.error("Terminating application.")
76        System.exit(1)
77    }
78  }
79
80  private def save(): Unit = {
81    val fileWriter = new PrintWriter(new File(outputFilename))
```

```
82      fileWriter.write(header + "\n" + output)
83      fileWriter.close()
84      prevTime = fileDateFormat.format(Calendar.getInstance().getTime())
85    }
86
87    private def checkOutputDirectory() {
88      if (!outputDirectory.exists()) {
89        logger.warn("Processed directory missing. Trying to create
              directory [{}].", outputDirectory.getAbsolutePath)
90        if (!outputDirectory.mkdir()) {
91          logger.error("Failed to create directory [{}].Terminating
                application.", outputDirectory.getAbsolutePath)
92          System.exit(1)
93        }
94      }
95    }
96
97  }
```

Listing C.14: Engine Source - Route Class

```
1   package lbsl
2
3   import java.sql.{Connection, PreparedStatement, SQLException}
4
5   import org.slf4j.LoggerFactory
6   import utility.{DBConnectionPool, Environment}
7
8   import scala.collection.mutable.{ArrayBuffer, HashMap}
9   import scala.concurrent.duration._
10
11  /**
12   * Created by Konstantin on 26/01/2015.
13   *
```

```
14  * Class representing a route in the bus network.
15  */
16
17 class Route(private val contractRoute: String) extends Runnable {
18
19   private val logger = LoggerFactory.getLogger(getClass().getSimpleName)
20   private val runList: ArrayBuffer[Run] = new ArrayBuffer[Run]()
21   private val busesOnRoute: HashMap[Integer, ArrayBuffer[Observation]]
         = new HashMap[Integer, ArrayBuffer[Observation]]()
22
23   def getContractRoute = contractRoute
24
25   /**
26    *
27    * @return boolean true if there are active (e.g. have received
         reading in the past 1h or so) buses on the route
28    *         false otherwise
29    */
30   def isActive(): Boolean = {
31     updateObservations()
32     !busesOnRoute.isEmpty
33   }
34
35   /**
36    *
37    * @param observation Observation - to be added to the route
38    */
39   def addObservation(observation: Observation): Unit = {
40     val observationList =
           busesOnRoute.getOrElse(observation.getVehicleId, new
           ArrayBuffer[Observation]())
41     observationList.append(observation)
42     busesOnRoute.put(observation.getVehicleId, observationList)
43   }
```

```
44
45  /**
46   * Method for initialisation of the route. This includes
47   * the preload of all the information for the route and its
48   * runs and sections from the database.
49   */
50  def init(): Unit = {
51    var connection: Connection = null
52    var preparedStatement: PreparedStatement = null
53    val query = "SELECT DISTINCT run FROM \"BusRouteSequences\" WHERE
          run <= 2 AND route = ? ORDER BY run "
54    try {
55      connection = DBConnectionPool.getConnection()
56      preparedStatement = connection.prepareStatement(query)
57      preparedStatement.setString(1, contractRoute)
58      val rs = preparedStatement.executeQuery()
59      while (rs.next()) {
60        runList.append(new Run(contractRoute, rs.getInt("run")))
61      }
62    }
63    catch {
64      case e: SQLException => logger.error("Exception: with query ({})
            ", preparedStatement.toString, e)
65    } finally {
66      if (preparedStatement != null) {
67        preparedStatement.close()
68      }
69      if (connection != null) {
70        DBConnectionPool.returnConnection(connection)
71      }
72    }
73    for (run <- runList) {
74      run.init()
75    }
```

```
76  }
77
78  override
79  def run(): Unit = {
80    updateObservations()
81    for ((busId, observationList) <- busesOnRoute if
          observationList.size > 1) {
82      for (i <- 1 until observationList.size) {
83        assignLostTimeToSections(observationList(i - 1),
              observationList(i))
84      }
85    }
86    for (run <- runList) {
87      run.detectDisruptions()
88    }
89    for (run <- runList) {
90      run.clearSections()
91    }
92  }
93
94  private def assignLostTimeToSections(prevObservation: Observation,
        observation: Observation): Boolean = {
95    for (run <- runList) {
96      if (run.checkStops(prevObservation, observation)) {
97        return true
98      }
99    }
100   //Used for testing only - logs unassigned observation differences
101   //    logger.debug("Unassigned observations prevLastStop {} [Route
          {} - {}] and lastStop {} [Route {} - {}].",
102   //      Array[Object](prevObservation.getLastStopShortDesc,
          prevObservation.getContractRoute, prevObservation.getOperator,
103   //        observation.getLastStopShortDesc,
          prevObservation.getContractRoute, observation.getOperator))
```

```
104     return false
105   }
106
107   /**
108    * Sort observation and remove old elements and remove observation
          list from map if no observations
109    */
110   private def updateObservations(): Unit = {
111     val latestFeedTimeOfDataTime =
            Environment.getLatestFeedTimeOfData.getTime
112     for ((busId, observationList) <- busesOnRoute) {
113       val sortedObservationList = observationList.sortBy(x =>
              x.getTimeOfData)
114       // difference in MILLISECONDS
115       var timeDiff = Duration(latestFeedTimeOfDataTime -
              sortedObservationList(0).getTimeOfData.getTime, MILLISECONDS)
116       while (timeDiff.toMinutes >
              Environment.getDataValidityTimeInMinutes &&
              sortedObservationList.size > 0) {
117         timeDiff = Duration(latestFeedTimeOfDataTime -
                sortedObservationList.remove(0).getTimeOfData.getTime,
                MILLISECONDS)
118       }
119       if (sortedObservationList.isEmpty) {
120         //Testing purposes
121         //logger.debug("Bus with id {} has not been active on route {}
              in the last hour.", busId, contractRoute)
122         busesOnRoute.remove(busId)
123       } else {
124         busesOnRoute.put(busId, sortedObservationList)
125       }
126     }
127   }
128
```

```
129 }
```

Listing C.15: Engine Source - Run Class

```
1  package lbsl
2
3  import java.sql.{Connection, PreparedStatement, SQLException}
4  import java.util.Date
5
6  import org.slf4j.LoggerFactory
7  import utility.{DBConnectionPool, Environment}
8
9  import scala.collection.mutable.ArrayBuffer
10
11 /**
12  * Created by Konstantin on 22/03/2015.
13  *
14  * Class representing a single run on a given route.
15  */
16 class Run(private val routeNumber: String, private val run: Integer) {
17
18   private val logger = LoggerFactory.getLogger(getClass().getSimpleName)
19   private val busStops: ArrayBuffer[String] = new ArrayBuffer[String]()
20   private val disruptions: ArrayBuffer[Disruption] = new
         ArrayBuffer[Disruption]()
21   private val prevDisruptions: ArrayBuffer[Disruption] = new
         ArrayBuffer[Disruption]()
22   private val sections: ArrayBuffer[Section] = new
         ArrayBuffer[Section]()
23
24   /**
25    * Method which initialises the Run by
26    * loading all information from the database
27    * and generating and initialising its sections.
```

```
28    */
29   def init(): Unit = {
30     var connection: Connection = null
31     var preparedStatement: PreparedStatement = null
32     val query = "SELECT \"busStopLBSLCode\" FROM \"BusRouteSequences\"
           WHERE route = ? AND run = ? ORDER BY \"sequence\""
33     try {
34       connection = DBConnectionPool.getConnection()
35       preparedStatement = connection.prepareStatement(query)
36       preparedStatement.setString(1, routeNumber)
37       preparedStatement.setInt(2, run)
38       val rs = preparedStatement.executeQuery()
39       while (rs.next()) {
40         busStops.append(rs.getString("busStopLBSLCode"))
41       }
42     }
43     catch {
44       case e: SQLException => logger.error("Exception:", e)
45     } finally {
46       if (preparedStatement != null) {
47         preparedStatement.close()
48       }
49       if (connection != null) {
50         DBConnectionPool.returnConnection(connection)
51       }
52     }
53     generateSections()
54     if (busStops.length - sections.length != 1) {
55       logger.debug("ERROR: Number of bus stop {} and number of sections
             {}.", busStops.length, sections.length)
56       logger.debug("Terminating application.")
57       System.exit(1)
58     }
59   }
```

```
60
61  /**
62   * Method for detecting any disruptions along the run.
63   */
64  def detectDisruptions(): Unit = {
65    prevDisruptions.clear()
66    disruptions.copyToBuffer(prevDisruptions)
67    disruptions.clear()
68    findDisruptions()
69    updateDB()
70  }
71
72  private def updateDB(): Unit = {
73    val latestObservationDate: Date =
          Environment.getLatestFeedTimeOfData
74    for (disruption <- prevDisruptions) {
75      disruption.update(getLostTime(disruption.getSectionStartIndex,
            disruption.getSectionEndIndex), getCumulativeLostTime())
76      disruption.clear(latestObservationDate)
77    }
78
79    for (disruption <- disruptions) {
80      disruption.save(routeNumber, run)
81    }
82
83    if (!disruptions.isEmpty || !prevDisruptions.isEmpty) {
84      for (section <- sections) {
85        section.save(latestObservationDate)
86      }
87    }
88  }
89
90  private def findDisruptions(): Unit = {
91    val cumulativeLostTime = getCumulativeLostTime()
```

```
92    if (cumulativeLostTime >= Environment.getSectionMediumThreshold) {
93      findDisruptedSections(Environment.getSectionMinThreshold)
94
95      if (disruptions.isEmpty && cumulativeLostTime >
            Environment.getRouteSeriousThreshold) {
96        findDisruptedSections(Environment.getSectionMinThreshold / 2)
97      }
98
99      //TODO: BEGIN Remove - just for testing purposes
100     if (disruptions.isEmpty && cumulativeLostTime >
            Environment.getRouteSeriousThreshold) {
101       var max: Double = 0
102       for (section <- sections) {
103         if (section.getDelay() > max) {
104           max = section.getDelay()
105         }
106       }
107       logger.debug("Route {} direction {} disrupted by {} minutes
              [max section disruption is {}].",
108         Array[Object](routeNumber, Run.getRunString(run),
              (cumulativeLostTime / 60).toString, max.toString))
109     }
110     //TODO: END
111   }
112 }
113
114 /**
115  * Method which checks the run for any disrupted sections.
116  * @param sectionMinThreshold Integer - the minimum section time loss
        threshold
117  */
118 private def findDisruptedSections(sectionMinThreshold: Integer): Unit
        = {
119   var sectionStartStopIndex: Integer = null
```

```
120     var disruptionSeconds: Double = 0
121     for (i <- 0 until sections.length) {
122
123       if (sections(i).getDelay() > sectionMinThreshold) {
124         if (sectionStartStopIndex == null) {
125           sectionStartStopIndex = i
126         }
127         disruptionSeconds += sections(i).getDelay()
128       } else {
129         if (sectionStartStopIndex != null) {
130           // end of sectionDisruption
131           if (disruptionSeconds >=
                Environment.getSectionMediumThreshold) {
132             addDisruption(sectionStartStopIndex, i, disruptionSeconds)
133           }
134           sectionStartStopIndex = null
135           disruptionSeconds = 0
136         }
137       }
138
139     }
140     if (sectionStartStopIndex != null) {
141       // end of sectionDisruption
142       if (disruptionSeconds >= Environment.getSectionMediumThreshold) {
143         addDisruption(sectionStartStopIndex, sections.length - 1,
              disruptionSeconds)
144       }
145     }
146   }
147
148   private def addDisruption(sectionStartStopIndex: Integer,
        sectionEndStopIndex: Integer, delaySeconds: Double): Unit = {
149     var disruption = new Disruption(sectionStartStopIndex,
          sectionEndStopIndex, busStops(sectionStartStopIndex),
```

```
      busStops(sectionEndStopIndex), delaySeconds,
      getCumulativeLostTime(), Environment.getLatestFeedTimeOfData)
150   val index = prevDisruptions.indexWhere(disruption.equals(_))
151   if (index > -1) {
152     disruption = prevDisruptions.remove(index)
153     disruption.update(sectionStartStopIndex, sectionEndStopIndex,
          busStops(sectionStartStopIndex),
          busStops(sectionEndStopIndex), delaySeconds,
          getCumulativeLostTime())
154   }
155   disruptions.append(disruption)
156 }
157
158 /**
159  *
160  * @param prevObservation Observation - the previous observation
161  * @param observation Observation - the subsequent observation
162  * @return Boolean - tru if the observation are made from the same
       run of a given route
163  */
164 def checkStops(prevObservation: Observation, observation:
      Observation): Boolean = {
165   //This controls whether to include or exclude curtailments (in case
       when included we assume that the lost time has happened on
       previous trip)
166   if (prevObservation.getTripId == observation.getTripId ||
       (prevObservation.getTripType == 3 && observation.getTripType ==
       2)) {
167     val prevLastStopIndex =
         busStops.indexOf(prevObservation.getLastStopShortDesc)
168     val lastStopIndex =
         busStops.indexOf(observation.getLastStopShortDesc)
169     if (lastStopIndex >= prevLastStopIndex && prevLastStopIndex > -1
         && lastStopIndex <= busStops.size - 1) {
```

```
170         val numberOfSections = (lastStopIndex - prevLastStopIndex) + 1
171         val lostTimePerSection = (observation.getScheduleDeviation -
                prevObservation.getScheduleDeviation) / numberOfSections
172         for (i <- prevLastStopIndex to Math.min(lastStopIndex,
                sections.size - 1)) {
173           sections(i).addObservation(observation.getVehicleId,
                  lostTimePerSection, observation.getTimeOfData)
174           //         sections(i).addObservation(new
                  Tuple2(lostTimePerSection, observation.getTimeOfData))
175         }
176         return true
177       }
178     }
179     return false
180   }
181
182   /**
183    * Method for clearing the data stored for each section.
184    */
185   def clearSections(): Unit = {
186     for (section <- sections) {
187       section.clear()
188     }
189   }
190
191   private def getLostTime(start: Int, end: Int): Double = {
192     var totalDelay: Double = 0
193     for (i <- start until end) {
194       totalDelay += Math.max(sections(i).getDelay(), 0)
195     }
196     return totalDelay
197   }
198
```

```
199   private def getCumulativeLostTime(considerNegatives: Boolean =
          false): Double = {
200     var totalDelay: Double = 0
201     for (section <- sections) {
202       if (considerNegatives) {
203         totalDelay += section.getDelay()
204       } else {
205         totalDelay += Math.max(section.getDelay(), 0)
206       }
207     }
208     return totalDelay
209   }
210
211   private def generateSections(): Unit = {
212     var connection: Connection = null
213     var preparedStatement: PreparedStatement = null
214     val query = "SELECT id, \"startStopLBSLCode\", \"endStopLBSLCode\",
          \"sequence\" FROM \"Sections\" WHERE route = ? AND run = ?
          ORDER BY \"sequence\""
215     try {
216       connection = DBConnectionPool.getConnection()
217       preparedStatement = connection.prepareStatement(query)
218       preparedStatement.setString(1, routeNumber)
219       preparedStatement.setInt(2, run)
220       val rs = preparedStatement.executeQuery()
221       while (rs.next()) {
222         sections.append(new Section(rs.getInt("id"),
              rs.getInt("sequence"), rs.getString("startStopLBSLCode"),
              rs.getString("endStopLBSLCode")))
223       }
224     }
225     catch {
226       case e: SQLException => logger.error("Exception:", e)
227     } finally {
```

```
228       if (preparedStatement != null) {
229         preparedStatement.close()
230       }
231       if (connection != null) {
232         DBConnectionPool.returnConnection(connection)
233       }
234     }
235   }
236 }
237
238 object Run {
239   /**
240    * Static method which returns the string for a run
241    * @param run Integer - the intger representing the run type
242    * @return String - the string for the provided run
243    */
244   def getRunString(run: Int): String = {
245     run match {
246       case 1 => return "Outbound"
247       case 2 => return "Inbound"
248       case default => return "Undefined"
249     }
250   }
251 }
```

Listing C.16: Engine Source - Section Class

```
1 package utility
2
3 import java.io.{File, FilenameFilter}
4
5 /**
6  * Created by Konstantin on 17/02/2015.
7  *
```

```
 * Custom filename filter used for filtering files in a directory
     according to specific prefix and suffix.
 */
class CustomFilenameFilter(private val prefix: String, private val
    suffix: String) extends FilenameFilter {

  def accept(dir: File, name: String): Boolean = {
    if (name.startsWith(prefix) && name.endsWith(suffix)) {
      return true
    }
    return false
  }
}
```

Listing C.17: Engine Source - SystemMonitor Class

```
package utility

import org.slf4j.LoggerFactory

/**
 * Created by Konstantin on 10/02/2015.
 *
 * Class used for gathering memory usage information,
 * throughout the execution of the system. Used for
 * stress testing measurements.
 */
class SystemMonitor extends Thread {

  private val runtime: Runtime = Runtime.getRuntime()
  private val logger = LoggerFactory.getLogger(getClass().getSimpleName)
  //in milliseconds
  private val sleepInterval: Long = 1000 * 30
  private val kb = 1024
```

```
19  private val mb = kb * kb
20  private var maxUsedMemory: Long = 0
21
22  override
23  def run(): Unit = {
24    while (true) {
25      val usedMemory = getUsedMemory()
26      if (usedMemory > maxUsedMemory) {
27        maxUsedMemory = usedMemory
28      }
29      logger.info("Used memory - [{}] Max used memory - [{}] Total
            memory - [{}] Max memory - [{}] Free memory - [{}] Available
            processors (cores) - [{}]",
30        Array[Object](
31          getMBString(usedMemory),
32          getMBString(maxUsedMemory),
33          getMBString(runtime.totalMemory()),
34          if (runtime.maxMemory() == Long.MaxValue) "No Limit" else
                getMBString(runtime.maxMemory()),
35          getMBString(runtime.freeMemory()),
36          runtime.availableProcessors().toString))
37
38      try {
39        Thread.sleep(sleepInterval)
40      } catch {
41        case e: InterruptedException => logger.error("iBusMonitorThread
              interrupted:", e)
42      }
43    }
44  }
45
46  private def getUsedMemory(): Long = {
47    runtime.totalMemory() - runtime.freeMemory()
48  }
```

```
49
50    private def getMBString(value: Long): String = {
51      val temp = value / mb
52      return temp.toString + "MB"
53    }
54  }
```

Listing C.18: Engine Source - TripType Class

```
1  package lbsl
2
3  /**
4   * 1: Trip from the depot to the start stop of the block.
5   * 2: Trip to a new starting point.
6   * 3: Normal trip with passengers.
7   * 4: Trip from the last stop of the block to the depot.
8   * 5: Trip without passengers.
9   * 6: Route Variant
10  * 7: Vehicle is not logged in to either block or route (e.x sending
        from garage)
11  *
12  * Created by Konstantin on 26/01/2015.
13  */
14 object TripType extends Enumeration {
15
16   val FromDepotToStartPoint = 1
17   val ToNewStartingPoint = 2
18   val Normal = 3
19   val ToDepot = 4
20   val WithoutPassengers = 5
21   val RouteVariant = 6
22   val NotLogged = 7
23
24   /**
```

```
25    *
26    * @param tripType Integer - the trip type to be checked if is active
          one
27    * @return Boolean - true if the provided trip type is active, false
          otherwise
28    */
29   def isActiveTrip(tripType: Integer): Boolean = {
30     if (tripType == 3 || tripType == 2) {
31       return true
32     }
33     return false
34   }
35
36 }
```

Listing C.19: Engine Source - BusNetwork Test

```
1  package scala.lbsl
2
3  import java.sql.PreparedStatement
4
5  import _root_.lbsl.Network
6  import _root_.utility.{DBConnectionPool, Environment}
7
8  import scala.main.UnitSpec
9
10 /**
11  * Created by Konstantin on 12/03/2015.
12  */
13 class BusNetworkTest extends UnitSpec {
14
15   before {
16     DBConnectionPool.createPool(dbConnectionSettingsPath)
17     Environment.init()
```

```
18    cleanDBTables()
19  }
20
21  after {
22    cleanDBTables()
23    DBConnectionPool.close()
24  }
25
26  test("BusNetworkTest") {
27    val busNetwork: Network = new Network()
28    busNetwork.init()
29    assert(busNetwork.getRouteCount() == 680)
30    check()
31    busNetwork.update()
32    check()
33  }
34
35  private def check() {
36    var preparedStatement: PreparedStatement = null
37    val connection = DBConnectionPool.getConnection()
38    var query = "SELECT * FROM \"Disruptions\""
39    preparedStatement = connection.prepareStatement(query)
40    var rs = preparedStatement.executeQuery()
41    assert(!rs.next())
42    preparedStatement.close()
43
44    query = "SELECT * FROM \"SectionsLostTime\" "
45    preparedStatement = connection.prepareStatement(query)
46    rs = preparedStatement.executeQuery()
47    assert(!rs.next())
48    preparedStatement.close()
49    DBConnectionPool.returnConnection(connection)
50  }
51
```

```
}
```

Listing C.20: Engine Source - BusStop Test

```
package scala.lbsl

import _root_.lbsl.BusStop
import _root_.utility.{DBConnectionPool, Environment}
import uk.me.jstott.jcoord.OSRef

import scala.main.UnitSpec

/**
 * Created by Konstantin on 12/03/2015.
 */
class BusStopTest extends UnitSpec {

  private val stopList: Array[String] = Array[String](
    "10024;55564;490012751W;ST JOSEPH'S CHURCH;521017;168169",
    "10258;47901;490010101S;THIRLMERE GARDENS;507950;191964",
    "1037;53342;490012521E;STAMFORD BROOK BUS GARAGE;521636;178587",
    "10343;75778;490010345W;NORTH COUNTESS ROAD;536782;190809",
    "10439;55159;490007093W1;WOOLWICH ROAD / GALLIONS
        ROAD;540897;178446",
    "1040;76767;490005423FF;CLIFTON GARDENS;520773;178494",
    "1041;77336;490012434HH;ST ANN'S ROAD;531763;188697",
    "58;53292;490013046N;SUSSEX GARDENS;527216;181547",
    "4797;76585;490013046M;EDGWARE ROAD / PRAED STREET;527173;181593",
    "28747;77640;490013046T;SUSSEX GARDENS;527213;181529",
    "29802;58110;490006135T;DORSET SQUARE / MARYLEBONE STATION
        <>;527719;182082",
    "4773;51725;490010713E;PADDINGTON GREEN POLICE
        STATION;526955;181719",
    "33761;51848;490010567H;OLD MARYLEBONE ROAD;527306;181770"
```

```
28  )
29  before {
30    DBConnectionPool.createPool(dbConnectionSettingsPath)
31    Environment.init()
32  }
33
34  after {
35    DBConnectionPool.close()
36  }
37
38  test("GetBusStop") {
39    for (line <- stopList) {
40      val testStop =
          line.split(";(?=([^\\\"]*\\\"[^\\\"]*\\\")*[^\\\"]*$)")
41      val stop = BusStop.getBusStop(testStop(0))
42      assert(stop.getLBSLCode() == testStop(0))
43      assert(stop.getCode() == testStop(1))
44      assert(stop.getNaptanAtco() == testStop(2))
45      assert(stop.getName() == testStop(3))
46      val latLng = new OSRef(testStop(4).toDouble,
          testStop(5).toDouble).toLatLng()
47      latLng.toWGS84()
48      assert(stop.getLongitude() == latLng.getLng)
49      assert(stop.getLatitude() == latLng.getLat)
50    }
51  }
52
53  test("NewBusStop") {
54    for (line <- stopList) {
55      val testStop =
          line.split(";(?=([^\\\"]*\\\"[^\\\"]*\\\")*[^\\\"]*$)")
56      val latLng = new OSRef(testStop(4).toDouble,
          testStop(5).toDouble).toLatLng()
57      latLng.toWGS84()
```

```
58      val stop = new BusStop(testStop(0), testStop(3), testStop(1),
            testStop(2), latLng.getLat, latLng.getLng)
59      assert(stop.getLBSLCode() == testStop(0))
60      assert(stop.getCode() == testStop(1))
61      assert(stop.getNaptanAtco() == testStop(2))
62      assert(stop.getName() == testStop(3))
63      assert(stop.getLongitude() == latLng.getLng)
64      assert(stop.getLatitude() == latLng.getLat)
65    }
66  }
67
68 }
```

Listing C.21: Engine Source - Disruption Test

```
1  package scala.lbsl
2
3  import java.util.Calendar
4
5  import _root_.lbsl.Disruption
6  import _root_.utility.{DBConnectionPool, Environment}
7
8  import scala.main.UnitSpec
9
10 /**
11  * Created by Konstantin on 12/03/2015.
12  */
13 class DisruptionTest extends UnitSpec {
14
15   private val disruptionTestList: Array[String] = Array[String](
16     "1;6;29844;2593;360;600;0",
17     "23;27;34554;BP2028;1200;1600;0"
18   )
19   before {
```

```
20    DBConnectionPool.createPool(dbConnectionSettingsPath)
21    Environment.init()
22  }
23
24  after {
25    DBConnectionPool.close()
26  }
27
28  test("DisruptionTest") {
29    for (line <- disruptionTestList) {
30      val tokens = line.split(";")
31      val calendar = Calendar.getInstance()
32      calendar.add(Calendar.MINUTE, tokens(6).toInt)
33      val disruption = new Disruption(tokens(0).toInt, tokens(1).toInt,
            tokens(2), tokens(3), tokens(4).toDouble, tokens(5).toDouble,
            calendar.getTime)
34
35      assert(disruption.getSectionStartIndex == tokens(0).toInt)
36      assert(disruption.getSectionEndIndex == tokens(1).toInt)
37      assert(disruption.getSectionStartBusStop == tokens(2))
38      assert(disruption.getSectionEndBusStop == tokens(3))
39      assert(disruption.getDelay == tokens(4).toDouble)
40      assert(disruption.getDelayInMinutes == (tokens(4).toDouble /
            60).toInt)
41      assert(disruption.getTotalDelay == tokens(5).toDouble)
42      assert(disruption.getTotalDelayInMinutes == (tokens(5).toDouble /
            60).toInt)
43      assert(disruption.getTimeFirstDetected == calendar.getTime)
44      assert(disruption.getTrend == -1)
45
46      //Trend stable
47      disruption.update(tokens(4).toDouble, tokens(5).toDouble)
48      assert(disruption.getDelay == tokens(4).toDouble)
```

```
49      assert(disruption.getDelayInMinutes == (tokens(4).toDouble /
            60).toInt)
50      assert(disruption.getTotalDelay == tokens(5).toDouble)
51      assert(disruption.getTotalDelayInMinutes == (tokens(5).toDouble /
            60).toInt)
52      assert(disruption.getTrend == 0)
53
54      for (i <- 0 to 1) {
55        val newDelay = 10000 * i
56        val newTotalDelay = 15000 * i
57        disruption.update(newDelay, newTotalDelay)
58        assert(disruption.getDelay == newDelay)
59        assert(disruption.getDelayInMinutes == (newDelay / 60))
60        assert(disruption.getTotalDelay == newTotalDelay)
61        assert(disruption.getTotalDelayInMinutes == (newTotalDelay /
              60))
62        if (i == 1) {
63          assert(disruption.getTrend == -1)
64        } else {
65          assert(disruption.getTrend == 1)
66        }
67      }
68
69      disruption.update(tokens(0).toInt, tokens(1).toInt - 2,
            tokens(2), tokens(3), tokens(4).toDouble, tokens(5).toDouble)
70      assert(disruption.getTrend == 1)
71
72      disruption.update(tokens(0).toInt, tokens(1).toInt + 2,
            tokens(2), tokens(3), tokens(4).toDouble, tokens(5).toDouble)
73      assert(disruption.getTrend == -1)
74    }
75  }
76
77  test("Equality") {
```

```
78      val disruption1 = new Disruption(2, 5, "34554", "BP2028", 650, 980,
            Calendar.getInstance().getTime)
79      val disruption2 = new Disruption(2, 8, "34554", "BP2028", 650, 980,
            Calendar.getInstance().getTime)
80      val disruption3 = new Disruption(3, 5, "34554", "BP2028", 650, 980,
            Calendar.getInstance().getTime)
81      val disruption4 = new Disruption(0, 3, "34554", "BP2028", 650, 980,
            Calendar.getInstance().getTime)
82
83      val disruption5 = new Disruption(8, 9, "34554", "BP2028", 650, 980,
            Calendar.getInstance().getTime)
84      val disruption6 = new Disruption(9, 11, "34554", "BP2028", 650,
            980, Calendar.getInstance().getTime)
85
86      assert(disruption1.equals(disruption2))
87      assert(disruption1.equals(disruption3))
88      assert(disruption1.equals(disruption4))
89      assert(!disruption1.equals(disruption5))
90
91      assert(disruption2.equals(disruption3))
92      assert(disruption2.equals(disruption4))
93      assert(disruption2.equals(disruption5))
94      assert(!disruption2.equals(disruption6))
95
96      assert(disruption3.equals(disruption4))
97      assert(!disruption3.equals(disruption5))
98      assert(!disruption3.equals(disruption6))
99      assert(!disruption3.equals(disruption6))
100
101     assert(!disruption4.equals(disruption5))
102     assert(!disruption4.equals(disruption6))
103
104     assert(disruption5.equals(disruption6))
105 }
```

```
106
107 }
```

Listing C.22: Engine Source - Environment Test

```
1  package scala.utility
2
3  import java.io.File
4  import java.sql.PreparedStatement
5  import java.text.SimpleDateFormat
6  import java.util.{Calendar, Date}
7
8  import _root_.utility.{DBConnectionPool, Environment}
9
10 import scala.main.UnitSpec
11
12
13 /**
14  * Created by Konstantin on 12/03/2015.
15  */
16 class EnvironmentTest extends UnitSpec {
17
18   before {
19     DBConnectionPool.createPool(dbConnectionSettingsPath)
20     Environment.init()
21     Environment.setLatestFeedTimeOfData(new Date(0))
22   }
23
24   after {
25     DBConnectionPool.close()
26   }
27
28   //Below values may change - check db
29   test("Getters") {
```

```
30    assert(Environment.getMovingAverageWindowSize == 5)
31    assert(Environment.isSystemMonitorActive == true)
32    assert(Environment.getLatestFeedTimeOfData == new Date(0))
33    assert(Environment.getFeedDirectory == new
          File("E:\\Workspace\\iBusNetTestDirectory"))
34    assert(Environment.getFeedFilePrefix == "CC_")
35    assert(Environment.getFeedFileSuffix == ".csv")
36    assert(Environment.getFeedFileDelimiter == ";")
37    assert(Environment.getFeedFileHeader == true)
38    assert(Environment.getDataValidityTimeInMinutes == 90)
39    assert(Environment.getProcessedDirectory == new
          File("E:\\Workspace\\iBusNetTestDirectory\\ProcessedFiles"))
40    assert(Environment.getMonitorThreadSleepInterval == 500)
41    assert(Environment.getSectionMediumThreshold == 1000)
42    assert(Environment.getSectionSeriousThreshold == 2400)
43    assert(Environment.getSectionSevereThreshold == 3600)
44    assert(Environment.getSectionMinThreshold == 180)
45    assert(Environment.getRouteSevereThreshold == 3600)
46    assert(Environment.getRouteSeriousThreshold == 2400)
47    assert(Environment.getDateFormat == new
          SimpleDateFormat("yyyy/MM/dd HH:mm:ss"))
48  }
49
50  test("LatestFeedTimeOfData") {
51    assert(Environment.getLatestFeedTimeOfData == new Date(0))
52    val calendar = Calendar.getInstance()
53
54    calendar.add(Calendar.DATE, -1)
55    Environment.setLatestFeedTimeOfData(calendar.getTime)
56    assert(Environment.getLatestFeedTimeOfData == calendar.getTime)
57
58    calendar.add(Calendar.DATE, +1)
59    Environment.setLatestFeedTimeOfData(calendar.getTime)
60    assert(Environment.getLatestFeedTimeOfData == calendar.getTime)
```

```
61  }
62
63  test("RegEx") {
64    val testColumnValuesArrays: Array[Array[String]] =
          Array[Array[String]](
65      Array[String]("ColumnA", "ColumnB", "ColumnC", "ColumnD",
            "ColumnE", "ColumnF", "ColumnG"),
66      Array[String]("ColumnA", "ColumnB", "ColumnC", "\"ColumnD;\"",
            "ColumnE", "ColumnF", "ColumnG"),
67      Array[String]("\";ColumnA;\"", "ColumnB", "\"Col;umnC\"",
            "\";ColumnD;\"", "ColumnE", ",ColumnF,", "\"Co;lumn;G\""),
68      Array[String]("ColumnA,", "ColumnB", "Column,C", "ColumnD4",
            "ColumnE", "ColumnF", "ColumnG2"),
69      Array[String]("ColumnA32,", "ColumnB!", "ColumnC?", "ColumnD4",
            "ColumnE", "ColumnF", "ColumnG#")
70    )
71    for (columnValues <- testColumnValuesArrays) {
72      val testString =
            columnValues.mkString(Environment.getFeedFileDelimiter)
73      val testColumnValues =
            testString.split(Environment.getFeedFileRegex)
74      assert(columnValues.length == testColumnValues.length)
75      for (i <- 0 until columnValues.length) {
76        assert(columnValues(i) == testColumnValues(i))
77      }
78    }
79  }
80
81  test("Update") {
82    val oldDataValidityTimeInMinutes =
          Environment.getDataValidityTimeInMinutes
83    val newDataValidityTimeInMinutes = 120
84    var preparedStatement: PreparedStatement = null
85    val connection = DBConnectionPool.getConnection()
```

```
86      val query = "UPDATE \"EngineConfigurations\" SET value=? WHERE
            key=?"
87      preparedStatement = connection.prepareStatement(query)
88      preparedStatement.setInt(1, newDataValidityTimeInMinutes)
89      preparedStatement.setString(2, "dataValidityTimeInMinutes")
90      preparedStatement.executeUpdate()
91      preparedStatement.close()
92
93      assert(Environment.getDataValidityTimeInMinutes ==
            oldDataValidityTimeInMinutes)
94      Environment.update()
95      assert(Environment.getDataValidityTimeInMinutes ==
            newDataValidityTimeInMinutes)
96
97      preparedStatement = connection.prepareStatement(query)
98      preparedStatement.setInt(1, oldDataValidityTimeInMinutes)
99      preparedStatement.setString(2, "dataValidityTimeInMinutes")
100     preparedStatement.executeUpdate()
101     preparedStatement.close()
102     DBConnectionPool.returnConnection(connection)
103
104     assert(Environment.getDataValidityTimeInMinutes ==
            newDataValidityTimeInMinutes)
105     Environment.update()
106     assert(Environment.getDataValidityTimeInMinutes ==
            oldDataValidityTimeInMinutes)
107   }
108
109 }
```

Listing C.23: Engine Source - Observation Test

```
1  package scala.lbsl
2
```

```
3  import _root_.lbsl.Observation
4  import _root_.utility.{DBConnectionPool, Environment}
5
6  import scala.main.UnitSpec
7
8  /**
9   * Created by Konstantin on 12/03/2015.
10  */
11 class ObservationTest extends UnitSpec {
12
13   private val trueLines: Array[String] = Array[String](
14     "9534;SE93;YX11CPZ;2015/02/18
           20:29:54;20150213;122076;508640;207;3;100;34190;260;-0.07433;51.51468;0;;",
15     "8977;E181;SN61BHU;2015/02/18
           20:30:29;20150213;34327;1013459;239;3;468;R0430;0;-0.11082;51.48500;0;;",
16     "7892;LT292;LTZ1292;2015/02/18
           20:30:26;20150213;122034;848054;309;3;453;1582;230;-0.15023;51.52316;0;;",
17     "9795;WVL360;LX60DWL;2015/02/18
           20:30:40;20150213;10056;212183;171;2;229;BP5396;-510;
           0.09941;51.42034;0;;",
18     "8978;E182;SN61BHV;2015/02/18
           20:30:23;20150213;34061;1017904;224;3;68;34686;0;-0.09694;51.47346;0;;",
19     "8990;EH18;SN61DCE;2015/02/18
           20:30:34;20150213;80305;1009030;255;3;436;532;-1500;-0.11194;51.48170;0;;",
20     "8446;E166;SN61BGU;2015/02/18
           20:29:53;20150213;34123;1011339;221;3;185;13380;-270;-0.07903;51.46144;0;;"
21   )
22
23   private val falseLines: Array[String] = Array[String](
24     "7624;WVN19;LK59FDM;2015/02/18
           20:30:06;20150213;-2147483645;-2147483645;-2147483645;7;259;29899;-2147483645;-0.08194;51.578
25     "7955;WVL190;LX05EZV;2015/02/18
           20:29:48;20150213;-2147483645;-2147483645;-2147483645;7;191;26975;-2147483645;-0.05149;51.600
```

```
    "7958;WVL193;LX05EZK;2015/02/18
        20:30:10;20150213;-2147483645;-2147483645;-2147483645;7;257;BP4267;-2147483645;-0.04075;51.58
    "8806;WVL389;LX11CVO;2015/02/18
        20:27:22;20150213;-2147483645;-2147483645;-2147483645;7;171;NX;-2147483645;-0.04388;51.47298;
    "9345;PVL397;LX54GZH;2015/02/18
        20:30:32;20150213;-2147483645;-2147483645;-2147483645;7;280;4529;-2147483645;-0.19030;51.4157
    "7626;WVN21;LK59FDO;2015/02/18
        20:28:26;20150213;-2147483645;-2147483645;-2147483645;7;259;2780;-2147483645;-0.05060;51.6003
  )

  before {
    DBConnectionPool.createPool(dbConnectionSettingsPath)
    Environment.init()
  }

  after {
    DBConnectionPool.close()
  }

  test("ValidObservations") {
    for (line <- trueLines) {
      val tokens: Array[String] =
          line.split(";(?=([^\\\"]*\\\"[^\\\"]*\\\")*[^\\\"]*$)")
      val observation = new Observation()
      assert(observation.init(line, "TEST"))
      assert(observation.getOperator == "TEST")
      assert(observation.getVehicleId == tokens(0).toInt)
      assert(observation.getTimeOfData ==
          Environment.getDateFormat().parse(tokens(3)))
      assert(observation.getTripId == tokens(6).toInt)
      assert(observation.getTripType == tokens(8).toInt)
      assert(observation.getContractRoute == tokens(9))
      assert(observation.getLastStopShortDesc == tokens(10))
      assert(observation.getScheduleDeviation == tokens(11).toInt)
```

```
54      assert(observation.getLongitude == tokens(12).toDouble)
55      assert(observation.getLatitude == tokens(13).toDouble)
56      assert(observation.getEventId == tokens(14).toInt)
57      assert(observation.isValid)
58    }
59  }
60
61  test("InvalidObservations") {
62    for (line <- falseLines) {
63      val tokens: Array[String] =
            line.split(";(?=([^\\\"]*\\\"[^\\\"]*\\\")*[^\\\"]*$)")
64      val observation = new Observation()
65      assert(!observation.init(line, "TEST"))
66      assert(observation.getTripType == tokens(8).toInt)
67      assert(observation.getScheduleDeviation == tokens(11).toInt)
68      assert(!observation.isValid)
69      assert(observation.getOperator == null)
70      assert(observation.getVehicleId == 0)
71      assert(observation.getTimeOfData == null)
72      assert(observation.getTripId == 0)
73      assert(observation.getContractRoute == null)
74      assert(observation.getLastStopShortDesc == null)
75      assert(observation.getLongitude == 0)
76      assert(observation.getLatitude == 0)
77      assert(observation.getEventId == 0)
78    }
79  }
80
81  test("Comparison") {
82    val line1 = "9534;SE93;YX11CPZ;2015/02/18
          20:29:54;20150213;122076;508640;207;3;100;34190;260;-0.07433;51.51468;0;;"
83    val line2 = "9534;SE93;YX11CPZ;2015/02/18
          22:29:54;20150213;122076;508640;207;3;100;34190;260;-0.07433;51.51468;0;;"
84    val observation1 = new Observation()
```

```
85      assert(observation1.init(line1, "TEST"))
86      val observation2 = new Observation()
87      assert(observation2.init(line1, "TEST"))
88      val observation3 = new Observation()
89      assert(observation3.init(line2, "TEST"))
90
91      assert(observation1.compareTo(observation2) == 0)
92
93      assert(observation1.compareTo(observation3) < 0)
94      assert(observation2.compareTo(observation3) < 0)
95
96      assert(observation3.compareTo(observation1) > 0)
97      assert(observation3.compareTo(observation2) > 0)
98    }
99
100 }
```

Listing C.24: Engine Source - Run Test

```
1  package scala.lbsl
2
3  import java.sql.PreparedStatement
4  import java.util.Calendar
5
6  import _root_.lbsl.{Observation, Run}
7  import _root_.utility.{DBConnectionPool, Environment}
8
9  import scala.main.UnitSpec
10
11 /**
12  * Created by Konstantin on 12/03/2015.
13  */
14 class RunTest extends UnitSpec {
15
```

```
16  before {
17    DBConnectionPool.createPool(dbConnectionSettingsPath)
18    Environment.init()
19    Environment.setLatestFeedTimeOfData(Calendar.getInstance().getTime)
20  }
21
22  after {
23    DBConnectionPool.close()
24  }
25
26  test("Run") {
27    cleanDBTables()
28    val input = Array[String](
29      "1234;1234;1234;2015/04/15
          08:00:00;20150419;55;1;1;3;RV1;BP3385;0;0;0;0;0",
30      "1234;1234;1234;2015/04/15
          08:02:00;20150419;55;1;1;3;RV1;29985;0;0;0;0;0",
31      "1234;1234;1234;2015/04/15
          08:04:00;20150419;55;1;1;3;RV1;1835;0;0;0;0;0",
32      "1234;1234;1234;2015/04/15
          08:06:00;20150419;55;1;1;3;RV1;33432;0;0;0;0;0",
33      "1234;1234;1234;2015/04/15
          08:08:00;20150419;55;1;1;3;RV1;BP3400;0;0;0;0;0",
34      "1234;1234;1234;2015/04/15
          08:10:00;20150419;55;1;1;3;RV1;BP3394;0;0;0;0;0",
35      "1234;1234;1234;2015/04/15
          08:12:00;20150419;55;1;1;3;RV1;BP3395;0;0;0;0;0",
36      "1234;1234;1234;2015/04/15
          08:14:00;20150419;55;1;1;3;RV1;226;0;300;0;0;0",
37      "1234;1234;1234;2015/04/15
          08:16:00;20150419;55;1;1;3;RV1;25940;900;0;0;0;0", //Disrupted
38      "1234;1234;1234;2015/04/15
          08:09:00;20150419;55;1;1;3;RV1;25938;1500;0;0;0;0",
          //Disrupted
```

```
39      "1234;1234;1234;2015/04/15
            08:09:00;20150419;55;1;1;3;RV1;25938;2000;0;0;0;0",
            //Disrupted
40      "1234;1234;1234;2015/04/15
            08:0:00;20150419;55;1;1;3;RV1;1520;2300;0;0;0;0",
41      "1234;1234;1234;2015/04/15
            08:00:00;20150419;55;1;1;3;RV1;8316;2300;0;0;0;0",
42      "1234;1234;1234;2015/04/15
            08:00:00;20150419;55;1;1;3;RV1;2198;2300;0;0;0;0",
43      "1234;1234;1234;2015/04/15
            08:00:00;20150419;55;1;1;3;RV1;33599;2300;0;0;0;0",
44      "1234;1234;1234;2015/04/15
            08:00:00;20150419;55;1;1;3;RV1;R0049;2300;0;0;0;0",
45      "1234;1234;1234;2015/04/15
            08:00:00;20150419;55;1;1;3;RV1;BP4909;2300;0;0;0;0",
46      "1234;1234;1234;2015/04/15
            08:00:00;20150419;55;1;1;3;RV1;BP3446;2300;0;0;0;0",
47      "1234;1234;1234;2015/04/15
            08:00:00;20150419;55;1;1;3;RV1;BP3693;2300;0;0;0;0",
48      "1234;1234;1234;2015/04/15
            08:00:00;20150419;55;1;1;3;RV1;BP3453;2300;0;0;0;0"
49    )
50    val observationList: Array[Observation] = new Array[Observation](20)
51    for (i <- 0 until input.length) {
52      val observation = new Observation
53      observation.init(input(i), "TestOperator")
54      observationList(i) = observation
55    }
56    val route = "RV1"
57    val run = new Run(route, 1)
58    run.init()
59    for (i <- 1 until observationList.length) {
60      run.checkStops(observationList(i - 1), observationList(i))
61    }
```

```
62
63    Environment.getDBTransaction.begin()
64    run.detectDisruptions()
65    Environment.getDBTransaction.commit()
66
67    var preparedStatement: PreparedStatement = null
68    val connection = DBConnectionPool.getConnection()
69    var query = "SELECT * FROM \"Disruptions\" WHERE route = ?"
70    preparedStatement = connection.prepareStatement(query)
71    preparedStatement.setString(1, route)
72    var rs = preparedStatement.executeQuery()
73    if (rs.next()) {
74      assert(rs.getString("fromStopLBSLCode") == "226")
75      assert(rs.getString("toStopLBSLCode") == "1520")
76      assert(rs.getString("route") == "RV1")
77      assert(rs.getInt("run") == 1)
78      assert(rs.getInt("delayInSeconds") == 2150)
79      assert(rs.getInt("routeTotalDelayInSeconds") == 2300)
80      assert(rs.getInt("trend") == -1)
81    }
82    preparedStatement.close()
83
84    query = "SELECT \"SectionsLostTime\".* FROM \"SectionsLostTime\"
          LEFT JOIN \"Sections\" ON \"SectionsLostTime\".\"sectionId\" =
          \"Sections\".id WHERE route = ?"
85    preparedStatement = connection.prepareStatement(query)
86    preparedStatement.setString(1, route)
87    rs = preparedStatement.executeQuery()
88    while (rs.next()) {
89      if (rs.getInt("sectionId") < 51382 || rs.getInt("sectionId") >
            51385) {
90        assert(rs.getInt("lostTimeInSeconds") == 0)
91      } else {
92        rs.getInt("sectionId") match {
```

```
93            case 51382 => assert(rs.getInt("lostTimeInSeconds") == 450)
94            case 51383 => assert(rs.getInt("lostTimeInSeconds") == 750)
95            case 51384 => assert(rs.getInt("lostTimeInSeconds") == 950)
96            case 51385 => assert(rs.getInt("lostTimeInSeconds") == 150)
97          }
98        }
99        assert(rs.getInt("numberOfObservations") == 1)
100     }
101     preparedStatement.close()
102     DBConnectionPool.returnConnection(connection)
103
104     cleanDBTables()
105   }
106
107
108 }
```

Listing C.25: Engine Source - Section Test

```
1  package scala.lbsl
2
3  import java.util.{Calendar, Date}
4
5  import _root_.lbsl.Section
6  import _root_.utility.{DBConnectionPool, Environment}
7
8  import scala.main.UnitSpec
9
10 /**
11  * Created by Konstantin on 12/03/2015.
12  */
13 class SectionTest extends UnitSpec {
14
```

```
15  //Need to update the data if window size and weights change -
        currently assumed window size = 5 and normal sequential weights
16  private val observations: Array[Tuple3[Integer, Double, Date]] = new
        Array[Tuple3[Integer, Double, Date]](5)
17  private val id = 1
18  private val sequence = 1
19  private val fromStop = "14456"
20  private val toStop = "29844"
21  //    1;"1";1;"14456";"29844";1
22  private var section: Section = null
23
24  before {
25    DBConnectionPool.createPool(dbConnectionSettingsPath)
26    Environment.init()
27    val observationValues: Array[Double] = Array[Double](89, 856, 1349,
          533, 579) //expected WMA 725
28    val calendar = Calendar.getInstance()
29    for (i <- 0 until observations.length) {
30      observations(i) = new Tuple3(i, observationValues(i),
            calendar.getTime)
31      calendar.add(Calendar.MINUTE, +5)
32    }
33
34    val temp = observations(3)
35    observations(3) = observations(0)
36    observations(0) = temp
37    section = new Section(id, sequence, fromStop, toStop)
38  }
39
40  after {
41    DBConnectionPool.close()
42  }
43
44  test("GetDelay") {
```

```
45    assert(section.getDelay() == 0)
46    for (observation <- observations) {
47      section.addObservation(observation._1, observation._2,
          observation._3)
48    }
49    assert(section.getDelay() == 725)
50  }
51
52  test("LatestObservationTime") {
53    assert(section.getLatestObservationTime() == null)
54    for (observation <- observations) {
55      section.addObservation(observation._1, observation._2,
          observation._3)
56    }
57    assert(section.getLatestObservationTime() == observations(4)._3)
58  }
59
60  test("Clear") {
61    assert(section.getLatestObservationTime() == null)
62    for (observation <- observations) {
63      section.addObservation(observation._1, observation._2,
          observation._3)
64    }
65    section.clear()
66    assert(section.getLatestObservationTime() == null)
67  }
68
69  // test("Save") {
70  //   assert(section.getLatestObservationTime() == null)
71  //   assert(section.getDelay() == 0)
72  //   for (observation <- observations) {
73  //     section.addObservation(observation)
74  //   }
75  //   assert(section.getDelay() == 725)
```

```
76    //    section.save(Calendar.getInstance().getTime)
77    // }
78  }
```

Listing C.26: Engine Source - UnitSpec

```
1
2   package scala.main
3
4   import _root_.utility.DBConnectionPool
5   import org.scalatest.{BeforeAndAfter, FunSuite}
6
7   /**
8    * Created by Konstantin on 18/03/2015.
9    */
10  class UnitSpec extends FunSuite with BeforeAndAfter {
11
12    val dbConnectionSettingsPath: String = "settings-test.xml"
13
14    def cleanDBTables(): Unit = {
15      val connection = DBConnectionPool.getConnection()
16      var updateStatement = connection.createStatement()
17      updateStatement.execute("DELETE FROM \"SectionsLostTime\"")
18      updateStatement.close()
19
20      updateStatement = connection.createStatement()
21      updateStatement.execute("DELETE FROM \"DisruptionComments\"")
22      updateStatement.close()
23
24      updateStatement = connection.createStatement()
25      updateStatement.execute("DELETE FROM \"Disruptions\"")
26      updateStatement.close()
27      DBConnectionPool.returnConnection(connection)
28    }
```

```
}
```

Listing C.27: Web Application - Disruption Comments Partial View

```
<fieldset>
    <legend><h1>Comments</h1></legend>
    <div class="panel">
        <i><%= @disruption.comments.empty? ? "No comments to display" :
            "" %></i>
        <% @disruption.comments.order(timestamp: :desc).each do
            |comment| %>
            <div class="row" style="margin-top: 0.3em;">
                <div class="small-7 medium-7 columns">
                    <span style="font-weight: bold; color: #0066CC;"><%=
                        comment.operator.username %></span></div>
                <div class="small-5 medium-5 columns">
                    <div style="text-align: right;">
                        <i>Added on <b><%=
                            comment.timestamp.strftime("%H:%M:%S
                            %m/%d/%Y") %></b></i></div>
                </div>
                <div class="large-12 columns" style="margin-top:
                    0.2em;border-bottom: 1px inset #DDDDDD;
                    border-bottom-radius: 25px; padding-bottom: 0.3em;">
                    <p class="text-justify"> <%=
                        comment.comment.gsub(/\n/, '<br>').html_safe %>
                        </p>
                </div>
            </div>
        <% end %>
    </div>
</fieldset>
<% if (session[:operatorId] != nil) %>
    <div class="row" style="margin-top: 0.3em;">
```

```
        <form>
            <fieldset>
                <legend>New comment</legend>
                <textarea id="commentText" name="comment"
                    placeholder="Type in your comment here..." rows="4"/>
                <input id="commentId" type="hidden" value="<%=
                    @disruption.id %>"/>
                <a class="button radius small right"
                    onclick="addNewComment()">Add</a>
            </fieldset>
        </form>
    </div>
<% end %>
<a class="close-reveal-modal">&#215;</a>
```

Listing C.28: Web Application - Disruption Details Partial View

```
<h4>Route <%= getLinkToBusRoute(@disruption.route,
    @disruption.getRunString).html_safe %>
    disrupted <%= @disruption.getRunString.downcase %>- first detected
        at <%= @disruption.getDetectedAt(true) %>
</h4>
<h5>Average lost time of <strong><%= @disruption.getDelay %></strong>
    minutes observed between
    <%= getLinkToBusStop(@disruption.fromStop).html_safe %> and
    <%= getLinkToBusStop(@disruption.toStop).html_safe %></h5>

<div class="row">
    <div class="large-12 columns">
        <div style="text-align: center; overflow-x: auto; overflow-y:
            hidden; ">
            <div id="lineGraph" style="display:inline-block; margin: 0
                auto;"></div>
        </div>
```

```
    </div>
</div>
<a class="close-reveal-modal">&#215;</a>
```

Listing C.29: Web Application - Disruption List Partial View

```
<div class="row">
    <div class="large-6 medium-6 small-6 columns">
        <span style="font-size:0.8em;float:left;">
            <a onclick="sort(null, null)"> <%= image_tag "broom.png",
                size: "24", alt: "Clear filters", title: "Clear
                filters" %></a>
            <%= page_entries_info @disruptions %>
        </span>
    </div>
    <div class="large-6 medium-6 small-6 columns">
        <span style="font-size:0.8em;float:right;">
            <i>Last updated: <span id="lastUpdateTime"
                style="font-weight:bold;"><%= @lastUpdateTime
                %></span></i>
        </span>
    </div>
</div>

<div class="row">
    <div class="large-12 columns">
        <table id="disruptionsTable" width="100%">
            <thead>
            <tr>
                <th width="20"><%= sortable("route", "Route").html_safe
                    %></th>
                <th width="20"><%= sortable("run",
                    "Direction").html_safe %></th>
                <th>
```

```
23                  <%= sortable("fromStopLBSLCode", "From").html_safe %>
24                  <sup data-tooltip aria-haspopup="true"
                        class="has-tip" title="Stop
                        <strong>from</strong> which disruption is
                        observed.">*</sup>
25              </th>
26              <th>
27                  <%= sortable("toStopLBSLCode", "To").html_safe %>
28                  <sup data-tooltip aria-haspopup="true"
                        class="has-tip" title="Stop <strong>to</strong>
                        which disruption is observed.">*</sup>
29              </th>
30              <th width="110">
31                  <%= sortable("delayInSeconds", "Delay").html_safe %>
32                  <sup data-tooltip aria-haspopup="true"
                        class="has-tip" title="WMA (Weighted moving
                        average) delay observed across the given
                        section(in minutes).">*</sup>
33              </th>
34              <th width="110">
35                  <%= sortable("routeTotalDelayInSeconds", "Total
                        delay").html_safe %>
36                  <sup data-tooltip aria-haspopup="true"
                        class="has-tip" title="Total delay observed
                        across the route in the given direction(in
                        minutes).">*</sup>
37              </th>
38              <th width="70">
39                  <%= sortable("trend", "Trend").html_safe %>
40                  <sup data-tooltip aria-haspopup="true"
                        class="has-tip" title="If disruption is
                        worsening (<span
                        style=color:#FF5468;>&#8593;</span>) and if it
                        is improving (<span
```

```
                    style=color:#4DFA90;>&#8595;</span>) else (<span
                    style=color:#FABE4D;>&#8597;</span>)."><sup>*</sup>
41              </th>
42              <th width="110">
43                  <%= sortable("firstDetectedAt",
                        "Detected").html_safe %>
44                      <sup data-tooltip aria-haspopup="true"
                            class="has-tip" title="Time when the system
                            has first detected the disruption."><sup>*</sup>
45              </th>
46              <th width="100" style="text-align: center;">Actions</th>
47          </tr>
48          </thead>
49          <tbody>
50          <% @disruptions.each do |disruption| %>
51              <tr class="<%= "hidden" if disruption.hide %>">
52                  <td>
53                      <%= getLinkToBusRoute(disruption.route,
                            disruption.getRunString).html_safe %>
54                  </td>
55                  <td style="text-align: center;"
                        sorttable_customkey="<%= disruption.run %>">
56                      <%= image_tag
                            disruption.getRunString.downcase+".png",
                            size: "32", title: disruption.getRunString,
                            class: "has-tip", title:
                            disruption.getRunString, "data-tooltip" =>
                            "", "aria-haspopup" => true %>
57                  </td>
58                  <td>
59                      <%=
                            getLinkToBusStop(disruption.fromStop).html_safe
                            %>
60                  </td>
```

```
61                  <td>
62                      <%= getLinkToBusStop(disruption.toStop).html_safe
                            %>
63                  </td>
64                  <td class="delayCell <%= disruption.delayColor
                        %>"><%= disruption.getDelay %></td>
65                  <td class="delayCell <%= disruption.totalDelayColor
                        %>"><%= disruption.getTotalDelay %></td>
66                  <td class="trendCell <%= disruption.trendColor
                        %>"><%= raw(disruption.trendSymbol) %>
67                  </td>
68                  <td> <%= disruption.getDetectedAt %></td>
69                  <td style="text-align: center;">
70                      <a onclick="details(<%=disruption.id%>)">
71                          <!--<a href="<= disruption_details_url(id:
                                disruption.id) %>"
                                data-reveal-id="revealModal"
                                data-reveal-ajax="true">-->
72                          <%= image_tag "details.png", size: "24", alt:
                                "Show more details", title: "Show more
                                details about disruption", class:
                                "has-tip", "data-tooltip" => "",
                                "aria-haspopup" => true %>
73                      </a>
74                      <a href="<%= disruption_comments_url(id:
                            disruption.id) %>"
                            data-reveal-id="revealModal"
                            data-reveal-ajax="true">
75                          <%= image_tag "comments.png", size: "24",
                                alt: "Show comments", title: "Show
                                comments about disruption", class:
                                "has-tip", "data-tooltip" => "",
                                "aria-haspopup" => true %>
76                      </a>
```

```
77                      <% if (session[:operatorId] != nil) %>
78                          <% if disruption.hide %>
79                              <a
                                    onclick="hideDisruption(<%=disruption.id%>,
                                    true)"> <%= image_tag "unhide.png",
                                    size: "24", alt: "Show", title: "Show
                                    disruption", class: "has-tip",
                                    "data-tooltip" => "", "aria-haspopup"
                                    => true %></a>
80                          <% else %>
81                              <a
                                    onclick="hideDisruption(<%=disruption.id%>,
                                    false)"> <%= image_tag "hide.png",
                                    size: "24", alt: "Hide", title: "Hide
                                    disruption", class: "has-tip",
                                    "data-tooltip" => "", "aria-haspopup"
                                    => true %></a>
82                          <% end %>
83                      <% end %>
84                  </td>
85              </tr>
86          <% end %>
87          </tbody>
88      </table>
89      <% if @disruptions.total_entries > 20 %>
90          <div class="apple_pagination">
91              <%= will_paginate @disruptions, :container => false,
                    :params => {:controller => "disruption", :action =>
                    "index"} %>
92          </div>
93      <% end %>
94  </div>
95 </div>
```

Listing C.30: Web Application - Disruption Index View

```
<div class="row">
    <div class="large-12 columns">
        <h1>Disruptions</h1>
    </div>
</div>

<div id="disruptionList">
    <%= render 'list' %>
</div>

<%= render 'main/engineSpeedControl' %>

<div id="revealModal" class="reveal-modal xlarge" data-reveal>
</div>

<script type="text/javascript">
    function poll(timeout) {
        setTimeout(function () {
            if ($("#disruptionList") != null) {
                ajaxPollCall();
            }
        }, timeout);
    }

    function ajaxPollCall() {
        if (document.URL == "<%= request.protocol + request.host%>" +
            ":3000/disruption/index" || document.URL == "<%=
            request.protocol + request.host%>" + ":3000/") {
            new $.ajax('/disruption/list', {
                method: 'get',
                success: function (result) {
                    if (result.update) {
```

```
                    $("#disruptionList").html(result.partial);
                }
                poll(result.timeout)
            }
        });
    }
}

$(document).ready(function () {
    poll("<%= @lastUpdateTime %>", <%= @timeout %>);
});

function sort(column, direction){
    new $.ajax('/disruption/list', {
        method: 'get',
        data: {sort: column, direction: direction},
        success: function (result) {
            if (result.update) {
                $("#disruptionList").html(result.partial);
            } else {
                $("#lastUpdateTime").html(result.lastUpdateTime)
            }
            $(document).foundation('tooltip', 'reflow');
        }
    });
}
</script>
```

Listing C.31: Web Application - History List Partial View

```
<div class="row">
    <div class="large-6 medium-6 small-6 columns">
        <span style="font-size:0.8em;float:left;">
```

```
            <a onclick="clearFilters()"> <%= image_tag "broom.png",
                size: "24", alt: "Clear filters", title: "Clear
                filters" %></a>
            <%= page_entries_info @disruptions %>
        </span>
    </div>
</div>

<div class="row">
    <div class="large-12 columns">
        <table id="disruptionsTable" width="100%">
            <thead>
            <tr>
                <th width="20"><%= sortable("route", "Route").html_safe
                    %></th>
                <th width="20"><%= sortable("run",
                    "Direction").html_safe %></th>
                <th>
                    <%= sortable("fromStopLBSLCode", "From").html_safe %>
                    <sup data-tooltip aria-haspopup="true"
                        class="has-tip" title="Stop
                        <strong>from</strong> which disruption is
                        observed.">*</sup>
                </th>
                <th>
                    <%= sortable("toStopLBSLCode", "To").html_safe %>
                    <sup data-tooltip aria-haspopup="true"
                        class="has-tip" title="Stop <strong>to</strong>
                        which disruption is observed.">*</sup>
                </th>
                <th width="110">
                    <%= sortable("delayInSeconds", "Delay").html_safe %>
                    <sup data-tooltip aria-haspopup="true"
                        class="has-tip" title="WMA (Weighted moving
```

```
                    average) delay observed across the given
                    section(in minutes).">*</sup>
28              </th>
29              <th width="110">
30                  <%= sortable("routeTotalDelayInSeconds", "Total
                        delay").html_safe %>
31                  <sup data-tooltip aria-haspopup="true"
                        class="has-tip" title="Total delay observed
                        across the route in the given direction(in
                        minutes).">*</sup>
32              </th>
33              <th width="110">
34                  <%= sortable("firstDetectedAt",
                        "Detected").html_safe %>
35                  <sup data-tooltip aria-haspopup="true"
                        class="has-tip" title="Time when the system has
                        first detected the disruption.">*</sup>
36              </th>
37              <th width="110">
38                  <%= sortable("clearedAt", "Cleared").html_safe %>
39                  <sup data-tooltip aria-haspopup="true"
                        class="has-tip" title="Time when disruption was
                        no longer detected by the system.">*</sup>
40              </th>
41              <th width="100" style="text-align: center;"
                    width="80">Actions</th>
42          </tr>
43          </thead>
44          <tbody>
45          <% @disruptions.each do |disruption| %>
46              <tr>
47                  <td>
48                      <%= getLinkToBusRoute(disruption.route,
                            disruption.getRunString).html_safe %>
```

```
49                  </td>
50                  <td style="text-align: center;"
                        sorttable_customkey="<%= disruption.run %>">
51                      <%= image_tag
                            disruption.getRunString.downcase+".png",
                            size: "32", title: disruption.getRunString,
                            class: "has-tip", "data-tooltip" => "",
                            "aria-haspopup" => true %>
52                  </td>
53                  <td>
54                      <%=
                            getLinkToBusStop(disruption.fromStop).html_safe
                            %>
55                  </td>
56                  <td>
57                      <%= getLinkToBusStop(disruption.toStop).html_safe
                            %>
58                  </td>
59                  <td class="delayCell <%= disruption.delayColor
                        %>"><%= disruption.getDelay %></td>
60                  <td class="delayCell <%= disruption.totalDelayColor
                        %>"><%= disruption.getTotalDelay %></td>
61                  </td>
62                  <td> <%= disruption.getDetectedAt(true) %></td>
63                  <td> <%= disruption.getClearedAt(true) %></td>
64                  <td style="text-align: center;">
65                      <a onclick="details(<%=disruption.id%>)">
66                          <!--<a href="<= disruption_details_url(id:
                                disruption.id) %>"
                                data-reveal-id="revealModal"
                                data-reveal-ajax="true">-->
67                          <%= image_tag "details.png", size: "24", alt:
                                "Show more details", title: "Show more
                                details about disruption", class:
```

```
                                "has-tip", "data-tooltip" => "",
                                "aria-haspopup" => true %>
                        </a>
                        <a href="<%= disruption_comments_url(id:
                            disruption.id) %>"
                            data-reveal-id="revealModal"
                            data-reveal-ajax="true">
                            <%= image_tag "comments.png", size: "24",
                                alt: "Show comments", title: "Show
                                comments about disruption", class:
                                "has-tip", "data-tooltip" => "",
                                "aria-haspopup" => true %>
                        </a>
                    </td>
                </tr>
            <% end %>
            </tbody>
        </table>
        <% if @disruptions.total_entries > 20 %>
            <div class="apple_pagination">
                <%= will_paginate @disruptions, :container => false,
                    :params => {:controller => "history", :action =>
                    "index"} %>
            </div>
        <% end %>
    </div>
</div>
```

Listing C.32: Web Application - History Index View

```
<div class="row">
    <div class="large-6 medium-4 columns">
        <h1>History</h1>
    </div>
```

```
    <div class="large-6 medium-8 columns end">
        <div class="large-5 medium-5 columns" style="padding-top:
            25px;">
            <input id="fromDate" name="from" type="text"
                placeholder="From" value="<%= session[:fromFilter] %>"/>
        </div>
        <div class="large-5 medium-5 columns" style="padding-top:
            25px;">
            <input id="toDate" name="to" type="text" placeholder="To"
                value="<%= session[:toFilter] %>"/>
        </div>
        <div class="large-2 medium-2 columns end" style="padding-top:
            25px;">
            <button class="button radius tiny" type="button"
                onclick="update()">Filter</button>
        </div>
    </div>
</div>

<div class="row">
    <div class="large-12 columns">
        <div id="disruptionList">
            <%= render 'list' %>
        </div>
    </div>
</div>



<div id="revealModal" class="reveal-modal xlarge" data-reveal>
</div>

<script type="text/javascript">
    function clearFilters() {
        new $.ajax('/history/filter', {
```

```
            method: 'get',
            data: {sort: "", direction: "", from: "", to: ""},
            success: function (result) {
                $("#disruptionList").html(result);
                $('#fromDate').val("");
                $('#toDate').val("");
                $(document).foundation('tooltip', 'reflow');
            }
        });
    }

    function update() {
        new $.ajax('/history/filter', {
            method: 'get',
            data: {from: $('#fromDate').val(), to: $('#toDate').val()},
            success: function (result) {
                $("#disruptionList").html(result);
                $(document).foundation('tooltip', 'reflow');
            }
        });
    }

    $('#fromDate').datetimepicker({
        format: 'd/m/Y H:i:s',
        dayOfWeekStart: 1,
        lang: 'en-GB',
        closeOnTimeSelect: true,
//       closeOnDateSelect: true,
        roundTime: 'floor',
        onShow: function (ct) {
            date = $('#toDate').val().split(" ")[0].split("/")
            date = date[2] + "/" + date[1] + "/" + date[0]
            this.setOptions({
                maxDate: $('#toDate').val() ? date : '0'
```

```
            })
        }
    })
    $('#toDate').datetimepicker({
        format: 'd/m/Y H:i:s',
        dayOfWeekStart: 1,
        lang: 'en-GB',
        closeOnTimeSelect: true,
//        closeOnDateSelect: true,
        roundTime: 'ceil',
        onShow: function (ct) {
            date = $('#fromDate').val().split(" ")[0].split("/")
            date = date[2] + "/" + date[1] + "/" + date[0]
            this.setOptions({
                minDate: $('#fromDate').val() ? date : false
            })
        }
    })

    function sort(column, direction) {
        new $.ajax('/history/filter', {
            method: 'get',
            data: {sort: column, direction: direction},
            success: function (result) {
                $("#disruptionList").html(result);
                $(document).foundation('tooltip', 'reflow');
            }
        });
    }
</script>
```

Listing C.33: Web Application - Layout Flash Partial View

```
<div class="row">
```

```
    <div id="flashDiv" class="small-12 large-12 columns">
        <% flash.each do |key, value| %>
            <div data-alert class="alert-box <%=key%> radius
                text-center">
                <strong><%= value %></strong>
                <a href="#" class="close">&times;</a>
            </div>
        <% end %>
    </div>
</div>
```

Listing C.34: Web Application - Layout Login Modal Partial View

```
<div id="loginModal" class="reveal-modal large" data-reveal
    style="max-width: 600px;">
    <div class="row">
        <div class="small-12 medium-12 large-12 small-centered
            medium-centered large-centered columns">
            <fieldset>
                <legend><h2>Login</h2></legend>
                <%= form_tag(:action => 'login', :controller => 'main')
                    do %>
                    <label>Username<%= text_field_tag(:username) %>
                    </label>
                    <label>Password<%= password_field_tag :password %>
                    </label>
                    <%= submit_tag("Log in", class: "button radius
                        expand") %>
                <% end %>
            </fieldset>
        </div>
    </div>
    <a class="close-reveal-modal">&#215;</a>
</div>
```

Listing C.35: Web Application - Layout Privacy Modal Partial View

```
<div id="privacyModal" class="reveal-modal" data-reveal
    style="text-align: justify">

    <h1 class="title">Privacy & cookies</h1>

    <p><strong>Privacy Statement:<span>  </span>How we use
        Cookies</strong></p>

    <p>Cookies are very small text files that are stored on your
        computer when you visit some
        websites.<span>  </span></p>

    <p>We use cookies to help identify your computer so we can tailor
        your user experience, track
        shopping basket contents and remember where you are in the
            order process.</p>

    <p>You can disable any cookies already stored on your computer, but
        these may stop our website
        from functioning properly.</p>

    <p><strong>The following is strictly necessary in the operation of
        our website.</strong></p>

    <p>This Website Will:</p>
    <ul>
        <li><span><span><span></span></span></span>Remember what is in
            your shopping basket</li>
        <li><span><span><span></span></span></span>Remember where you
            are in the order process</li>
        <li><span><span><span></span></span></span>Remember that you
            are logged in and that your session is
```

```
23          secure.<span>  </span>You need to be logged in to
                complete an order.
24      </li>
25  </ul>
26  <p>The following are not Strictly Necessary, but are required to
        provide you with the best user
27      experience and also to tell us which pages you find most
            interesting (anonymously).</p>
28
29  <p><strong>Functional Cookies</strong></p>
30
31  <p>This Website Will:</p>
32  <ul>
33      <li><span><span><span></span></span></span>Offer Live Chat
            Support (If available)</li>
34      <li><span><span><span></span></span></span>Track the pages you
            visits via Google Analytics</li>
35  </ul>
36  <p><strong>Targeting Cookies</strong></p>
37
38  <p>This Website Will:</p>
39  <ul>
40      <li><span><span><span></span></span></span>Allow you to share
            pages with social networks such as Facebook (If
41          available)
42      </li>
43      <li><span><span></span></span>Allow you to share pages via Add
            This (If available)</li>
44  </ul>
45  <p>To view the âĂŸAdd ThisâĂŹ Privacy Policy or to opt out of any
        online behavioural advertising,
46      please visit
47      <a
            href="http://www.addthis.com/privacy?utm_source=mf&amp;utm_medium=link&amp;utm_content=AT_
```

```
            title="Add This">Add
48          This</a> and click on the âĂŸOpt OutâĂŹ button.</p>
49
50      <p><strong>This website will not</strong></p>
51      <ul>
52          <li><span><span></span></span>Share any personal information
                with third parties.</li>
53      </ul>
54      <a class="close-reveal-modal">&#215;</a>
55  </div>
```

Listing C.36: Web Application - Layout Terms Modal Partial View

```
1   <div id="termsModal" class="reveal-modal" data-reveal
        style="text-align: justify">
2       <h2>Terms & conditions</h2>
3
4       <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec
            eget accumsan diam. Vivamus neque libero, rhoncus
5           vitae elit et, tincidunt sollicitudin justo. Quisque non mauris
                quis lorem fringilla congue sit amet vitae
6           risus. Sed tortor enim, auctor a tincidunt vehicula,
                pellentesque consequat justo. Ut dictum est sit amet sapien
7           suscipit semper. Mauris sit amet lacus augue. Etiam nisl nibh,
                auctor in elit eget, vulputate iaculis enim.
8           Suspendisse quis luctus metus. Quisque et lacinia sem. Fusce ut
                faucibus lacus. Quisque quis erat metus. Mauris
9           scelerisque rutrum quam nec mollis. Fusce nec metus vitae diam
                facilisis porta. Nunc rutrum, enim eget efficitur
10          faucibus, urna odio placerat nunc, vel rhoncus mi urna ac enim.
11      </p>
12
13      <p> Vivamus sit amet bibendum nunc. Duis et diam malesuada elit
            vulputate dictum. Curabitur ut scelerisque elit.
```

```
        Interdum et malesuada fames ac ante ipsum primis in faucibus.
            In elementum ultricies mi sit amet egestas.
        Quisque in mi convallis, vestibulum tellus vitae, vehicula
            nisl. Fusce auctor risus eu massa blandit maximus.
        Praesent bibendum eget arcu ac volutpat.
    </p>

    <p> Vestibulum sollicitudin arcu felis, vitae sodales nisi finibus
         a. Quisque neque est, pulvinar ut elementum
        vitae, pellentesque in arcu. Lorem ipsum dolor sit amet,
            consectetur adipiscing elit. Curabitur accumsan laoreet
        augue. Vestibulum nisi neque, malesuada non est porttitor,
            tincidunt dictum lacus. Proin ac odio id leo ornare
        ornare. Class aptent taciti sociosqu ad litora torquent per
            conubia nostra, per inceptos himenaeos.
    </p>

    <p> Nam sollicitudin turpis enim, sit amet euismod tellus fringilla
         consectetur. Duis id accumsan augue. Sed
        fermentum nulla nisi, et blandit ipsum maximus eget. Proin
            pulvinar in nisi et fermentum. Mauris a ullamcorper
        ex. Lorem ipsum dolor sit amet, consectetur adipiscing elit.
            Maecenas non tellus in mi fermentum suscipit. Ut
        consequat egestas dictum. Vestibulum ullamcorper odio lacus,
            quis auctor eros ultrices et.
    </p>

    <p> Duis viverra maximus condimentum. Suspendisse potenti. Cras
         sagittis pretium urna eget eleifend. Sed in rutrum
        metus, vitae faucibus lacus. Suspendisse auctor pellentesque
            odio, at molestie nibh cursus et. Morbi placerat
        purus porttitor tincidunt luctus. Cras at vehicula arcu, eget
            faucibus risus. Mauris leo nisi, efficitur in
```

```
        ligula ut, malesuada commodo nunc. Vestibulum tristique at nibh
            ut ultrices.
    </p>
    <a class="close-reveal-modal">&#215;</a>
</div>
```

Listing C.37: Web Application - Layout Application View

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width,
        initial-scale=1.0"/>
    <%= favicon_link_tag 'favicon.ico' %>
    <title>iBus Disruption Monitor</title>
    <%= javascript_include_tag "vendor/modernizr" %>
    <%= stylesheet_link_tag "application" %>
    <%= javascript_include_tag "application", 'data-turbolinks-track'
        => true %>
    <%= content_for :header %>
    <%= csrf_meta_tags %>
</head>

<body class="f-topbar-fixed" data-no-turbolink>
<!--BEGIN:TopNavMenu-->
<div class="row fullWidth" style="/*background: #333;">
    <div class="large-12 columns">
        <div class="contain-to-grid fixed">
            <nav class="top-bar" data-topbar role="navigation"
                data-options="fixed_on: large">
                <ul class="title-area">
                    <li class="name">
                        <h1>
```

```
                    <%= link_to "iBus Disruption Monitor",
                        controller: "disruption", action: "index"
                        %>
                </h1>
            </li>
            <li class="toggle-topbar menu-icon"><a
                href="#"><span>menu</span></a></li>
        </ul>
        <section class="top-bar-section">
            <ul class="right">
                <li class="<%= "active" if params[:controller] ==
                    "disruption" %>"><%= link_to "Disruptions",
                    controller: "disruption", action: "index"
                    %></li>
                <li class="<%= "active" if params[:controller] ==
                    "history" %>"><%= link_to "History",
                    controller: "history", action: "index" %></li>
                <% if (session[:operatorId] != nil &&
                    session[:operatorAdmin]) %>
                    <li class="<%= "active" if
                        params[:controller] == "settings" %>"><%=
                        link_to "Settings", controller:
                        "settings", action: "index" %></li>
                <% end %>
                <li class="<%= "active" if params[:controller] ==
                    "main" %>"><%= link_to "About", controller:
                    "main", action: "about" %></li>
                <% if (session[:operatorId] != nil) %>
                    <li><a href="<%= main_logout_url %>">Log out
                        (<strong><%= session[:operatorUsername]
                            %></strong>)</a></li>
                <% else %>
                    <li><a href="#"
                        data-reveal-id="loginModal">Log
```

```
                                in</a></li>
42                          <% end %>
43                          <!--
44                          <li class="search">
45                              <form>
46                                  <input type="search">
47                              </form>
48                          </li>
49                          <li class="has-button">
50                              <a class="small button disabled"
                                    href="#">Search</a>
51                          </li>-->
52                      </ul>
53                  </section>
54              </nav>
55          </div>
56      </div>
57  </div>
58  <!--END:TopNavMenu-->
59
60  <!--BEGIN:MainContent-->
61  <div class="mainContent" style="margin-top: 2em; margin-bottom: 2em;">
62      <%= render 'layouts/flash' %>
63      <%= yield %>
64  </div>
65  <!--END:MainContent-->
66  <!--BEGIN:Footer-->
67  <footer class="row">
68      <div class="small-12 large-12 columns">
69          <hr>
70          <div class="row">
71              <div class="small-6 large-8 columns">
72                  <ul class="inline-list left">
```

```
73                      <li><a href="#" data-reveal-id="termsModal">Terms &
                            conditions</a></li>
74                      <li><a href="#"
                            data-reveal-id="privacyModal">Privacy &
                            cookies</a></li>
75                  </ul>
76              </div>
77              <div class="small-6 large-4 columns">
78                  <p class="right">Powered by
79                      <a href="http://foundation.zurb.com/"
                            target="_blank">FoundationÂĭ</a> &
80                      <a href=" http://rubyonrails.org/"
                            target="_blank">RailsÂő</a>
81                  </p>
82              </div>
83          </div>
84          <div class="row text-center">
85              <p>
86                  Ãĭ Copyright <%= Time.now.year %>
87                  <a href="http://uk.linkedin.com/in/kdraganov"
                        target="_blank">Konstantin Draganov</a>
88              </p>
89          </div>
90      </div>
91  </footer>
92  <!--END:Footer-->
93  <% if (session[:operatorId] == nil) %>
94      <%= render 'layouts/loginModal' %>
95  <% end %>
96  <%= render 'layouts/termsModal' %>
97  <%= render 'layouts/privacyModal' %>
98  </body>
99  </html>
```

Listing C.38: Web Application - Main Engine Speed Control Partial View

```
<% if (session[:operatorId] != nil && session[:operatorAdmin]) %>
    <div class="row">
        <div class="large-12 columns" style="font-size:
            0.8em;text-align: right;font-weight: bold;">
            <a id="pause" onclick="setSpeed(0)">Pause</a> |
            <a id="slow" onclick="setSpeed(1)">Slow</a> |
            <a id="normal" onclick="setSpeed(2)">Normal</a> |
            <a id="fast" onclick="setSpeed(3)">Fast</a> |
            <a id="fast" onclick="setSpeed(4)">Very Fast</a>
        </div>
    </div>
    <script type="text/javascript">
        function setSpeed(speed) {
            new $.ajax('/main/speed', {
                method: 'get',
                data: {speed: speed},
                success: function (result) {
                }
            });
        }
    </script>
<% end %>
```

Listing C.39: Web Application - Main Index View

```
<section class="main-section">
    <div class="row">
        <div class="large-12 columns">
            <h1>About</h1>
        </div>
    </div>

```

```
8   <div class="row">
9       <div class="large-12 columns">
10          <div class="panel" style="text-align: justify;">
11              <br>
12
13              <h3>Real-time visualisation of bus delays in London in
                    collaboration with Transport for London</h3>
14              <br>
15
16              <p>
17                  The focus of this project is to develop a tool for
                        automating the data analysis
18                  carried out by CentreCommâĂŹs staff. This tool would
                        increase the productivity
19                  of the control room team as they would be able to
                        focus on solving the problems
20                  rather then trying to find them.
21              </p>
22
23              <p>
24                  London bus network is one of the largest and most
                        reliable network in the
25                  world responsible for more than 2.4 billion
                        passenger journeys a year. The
26                  constant population growth of EnglandâĂŹs capital
                        has been also driving the expansion
27                  and improvement of the transport networks across the
                        city. Transport
28                  for London (TFL) bus network is recognised as one of
                        the top in the world in
29                  terms of reliability, affordability and
                        cost-effectiveness. The capitalâĂŹs bus
30                  network is continually increasing along with the
                        cityâĂŹs population.
```

```
                Maintaining such a large scale network requires
                    careful planning and monitoring.
                Being able to maintain such high reliability service
                    levels requires
                employing new technologies. This also helps keep
                    costs down and thus keeping
                the service more affordable. Each TFL bus in the
                    network has been equipped
                with state of the art GPS enabled Automated Vehicle
                    Location System (AVL).
                This system generates a large set of data which
                    helps CentreComm staff
                and bus operators in managing and maintaining the
                    smooth operation of the
                network. However there are currently problems for
                    which the control room staff
                are required to manually analyse large data sets in
                    order to discover where are
                the problems occurring in the network. This is very
                    impractical and time
                consuming and they currently ultimately rely on bus
                    operators and drivers to
                notify them of possible problem before they go and
                    further investigate it. This
                is where this project comes in place to address this
                    problem by proposing a
                prototype tool which would analyse this data and
                    pro-actively highlight the
                problems in the bus network.
            </p>

            <div class="row" style="text-align: left;">
                <hr size="3" style="padding-top:1em;"/>
                <div class="large-4 medium-4 columns ri">
```

```
51                      <p><a href="http://uk.linkedin.com/in/kdraganov"
                            target="_blank">Konstantin
                            Draganov</a><br/>Author
52                      </p>
53                  </div>
54                  <div class="large-4 medium-4 columns">
55                      <p><a href="http://steffen-zschaler.de/"
                            target="_blank">Dr Steffen
                            Zschaler</a><br/>Project
56                          supervisor</p>
57                  </div>
58                  <div class="large-4 medium-4 columns">
59                      <p><a
                            href="http://tfl.gov.uk/corporate/about-tfl/"
                            target="_blank">TFL</a><br/>Data provider
60                      </p>
61                  </div>
62              </div>
63          </div>
64      </div>
65  </div>
66  </section>
```

Listing C.40: Web Application - Settings Edit Partial View

```
1  <div class="row">
2      <h1>Edit</h1>
3  </div>
4  <div class="row">
5      <div class="large-8 large-offset-2 columns" style="text-align:
           center">
6          <strong><%= @configs.key %></strong>
7      </div>
8  </div>
```

```
<div class="row" style="margin-top: 1em; margin-bottom: 1em;">
    <div class="large-8 large-offset-2 columns">
        <input id="value" type="text" value="<%= @configs.value %>">
        <input id="key" type="hidden" value="<%= @configs.key %>">
    </div>
</div>
<div class="row">
    <div class="large-2 large-offset-4 columns">
        <button class="button radius"
            onclick="$('#revealModal').foundation('reveal',
            'close');">Back</button>
    </div>
    <div class="large-6 columns">
        <button class="button radius" onclick="save()">Save</button>
    </div>
</div>
<a class="close-reveal-modal">&#215;</a>
```

Listing C.41: Web Application - Settings Index View

```
<div class="row">
    <div class="large-12 columns">
        <h1>Settings</h1>
    </div>
</div>



<div class="row">
    <div class="large-12 columns">
        <table id="settings" class="sortable" width="100%">
            <thead>
            <tr>
                <th>Key</th>
                <th>Value</th>
```

```
15                <th class="sorttable_nosort" style="text-align:
                      center;">Edit</th>
16            </tr>
17            </thead>
18            <tbody>
19            <% @configs.each do |config| %>
20                <tr>
21                    <td><%= config.key %></td>
22                    <td id="<%= config.key %>"><%= config.value %></td>
23                    <td style="text-align: center;">
24                        <% if config.editable %>
25                        <a href="<%= settings_edit_path(id: config.key)
                              %>" data-reveal-id="revealModal"
                              data-reveal-ajax="true">
26                            <%= image_tag "edit.png", size: "24", alt:
                                  "Edit", title: "Edit configuration
                                  parameter" %>
27                        </a>
28                        <% end %>
29                    </td>
30                </tr>
31            <% end %>
32            </tbody>
33        </table>
34        <% if @configs.total_entries > 20 %>
35            <div class="apple_pagination">
36                <%= will_paginate @configs, :container => false %>
37            </div>
38        <% end %>
39    </div>
40</div>
41
42
43<div id="revealModal" class="reveal-modal small" data-reveal>
```

```
</div>



<script type="text/javascript">
    function save() {
        new $.ajax('/settings/save', {
            method: 'post',
            data: {key: $('#key').val(), value: $('#value').val()},
            success: function (result) {
                if (!result.error) {
                    $('#revealModal').foundation('reveal', 'close');
                    document.getElementById(result.key).innerHTML =
                        result.newValue;
                    showAlert("Value changed successfully!", "success")
                }
            }
        });
    }
</script>
```

Listing C.42: Web Application - Application Controller

```
class ApplicationController < ActionController::Base
  # Prevent CSRF attacks by raising an exception.
  # For APIs, you may want to use :null_session instead.
  protect_from_forgery with: :exception

  protected
  def authenticateUser
    if session[:operatorId]
      @current_operator = Operator.find(session[:operatorId])
      return true
    else
      flash[:alert] = "You are not authorised to access this page!"
```

```
13      redirect_to(:controller => 'main', :action => 'index')
14      return false
15    end
16  end
17
18  def saveLoginState
19    if session[:operatorId]
20      redirect_to(:controller => 'main', :action => 'index')
21      return false
22    else
23      return true
24    end
25  end
26
27 end
```

Listing C.43: Web Application - Disruption Controller

```
1  class DisruptionController < ApplicationController
2    helper_method :sort_column, :sort_direction
3    before_filter :authenticateUser, :only => [:addComment, :hide]
4    require 'time'
5    $disruptionListLatUpdateTime = nil
6
7    #Method responsible for querying the database for the details of a
         given disruption
8    def details
9      error = "<h1>No disruption specified</h1> <a
           class=\"close-reveal-modal\">&#215;</a>"
10     id = nil
11     begin
12       id = Integer(params[:id])
13     rescue ArgumentError
14       render text: error and return
```

```
15      end
16
17      if (id == nil)
18        render text: error and return
19      end
20      begin
21        @disruption = Disruption.includes(:fromStop, :toStop).find(id)
22      rescue ActiveRecord::RecordNotFound
23        render text: error and return
24      end
25      if (@disruption == nil)
26        render text: error and return
27      end
28      # @sections = Section.includes(:startStop, :endStop,
            :latestLostTime).where("route = ? AND run = ? ",
            @disruption.route, @disruption.run).order(sequence: :asc)
29      @sections = Section.find_by_sql(['SELECT * FROM (
30  SELECT s.*, "SectionsLostTime".*,ROW_NUMBER() OVER(PARTITION BY s.id
        ORDER BY "SectionsLostTime".timestamp DESC) rn
31  FROM "SectionsLostTime" JOIN "Sections" s ON
        "SectionsLostTime"."sectionId" = s.id
32  WHERE s."route" = ? AND s.run = ? AND "SectionsLostTime".timestamp <=
        ? )
33  a WHERE rn = 1 ORDER BY a.sequence', @disruption.route,
        @disruption.run, @disruption.clearedAt == nil ? Time.now :
        @disruption.clearedAt])
34      ActiveRecord::Associations::Preloader.new.preload(@sections,
            [:startStop, :endStop, :latestLostTime])
35
36      startIndex = Section.where("route = ? AND run = ? AND
            \"startStopLBSLCode\" = ?", @disruption.route, @disruption.run,
            @disruption.fromStopLBSLCode)[0].sequence
37      endIndex = Section.where("route = ? AND run = ? AND
            \"startStopLBSLCode\" = ?", @disruption.route, @disruption.run,
```

```
    @disruption.toStopLBSLCode)[0].sequence
38  data = Array.new
39  data.push(['Section', 'Section Lost Time', {type: 'string', role:
        'annotation'},
40              {type: 'string', role: 'tooltip', p: {html: true}},
                    {type: 'boolean', role: 'scope'},
41              'Total Lost Time', {type: 'string', role: 'tooltip', p:
                    {html: true}}, {type: 'boolean', role: 'scope'}])
42  totalLostTime = 0
43  sectionLostTime = 0
44  @sections.each do |section|
45    scope = false
46    if (section.sequence == 1)
47      label = capitalizeAll(section.startStop.name)
48    elsif (section.sequence == @sections.length)
49      label = capitalizeAll(section.endStop.name)
50    else
51      label = ''
52    end
53    lostTime = (section.latestLostTime.lostTimeInSeconds / 60).round
54    lostTime = [lostTime, 0].max
55    totalLostTime += lostTime
56
57    if (section.sequence >= startIndex && section.sequence < endIndex)
58      sectionLostTime += lostTime
59      scope = true
60    end
61
62    tooltip = "From: <strong>"+getLinkToBusStop(section.startStop)+
63        "</strong><br>To: <strong>"
                +getLinkToBusStop(section.endStop)+
64        "</strong><br>Number of observation: <strong>" +
65        section.latestLostTime.numberOfObservations.to_s+ "</strong>"
66
```

```
67      totalTooltip = "Total minutes lost: <strong>"+totalLostTime.to_s+
68          "</strong><br>From: <strong>" +
                getLinkToBusStop(section.startStop) +
69          "</strong><br>To: <strong>" +getLinkToBusStop(section.endStop)
70
71      data.push([label, lostTime, lostTime, tooltip, scope,
             totalLostTime, totalTooltip, true])
72    end
73
74    title = 'Route '+@disruption.route+' '+ @disruption.getRunString
75    hAxisTitle = 'Total cumulative lost time observed along route
          '+totalLostTime.to_s+' minutes'
76    @return = {:error => false, :update => true, :partial =>
          render_to_string(:partial => "details"), :data => data, :title
          => title, :hAxisTitle => hAxisTitle}
77    render :json => ActiveSupport::JSON.encode(@return)
78  end
79
80  # Lists all active disruptions in the network
81  def list
82    lastUpdateTime = getLatUpdateTime
83    hidden = $disruptionListLatUpdateTime != nil &&
          session[:disruptionListLatUpdateTime] <=
          $disruptionListLatUpdateTime
84    dbUpdated = session[:fileVersion] == nil || session[:fileVersion] <
          lastUpdateTime
85    if (hidden || dbUpdated || params[:sort])
86      @lastUpdateTime = formatDatetimeString(lastUpdateTime)
87      @disruptions = getDisruptions
88      @return = {:error => false, :update => true, :partial =>
             render_to_string(:partial => "list"), :timeout => TIMEOUT} #,
             :lastUpdateTime => formatDatetimeString(lastUpdateTime)
89    else
```

```
90      @return = {:error => false, :update => false, :timeout =>
            TIMEOUT} #, :lastUpdateTime =>
            formatDatetimeString(lastUpdateTime)
91    end
92    render :json => ActiveSupport::JSON.encode(@return)
93  end
94
95  def index
96    @lastUpdateTime = formatDatetimeString(getLatUpdateTime)
97    @timeout = TIMEOUT
98    @disruptions = getDisruptions
99  end
100
101 def hide
102   id = nil
103   begin
104     id = Integer(params[:id])
105   rescue ArgumentError
106     render :json => ActiveSupport::JSON.encode({:error => true,
            :errorInfo => "Invalid disruption selected!"}) and return
107   end
108   begin
109     disruption = Disruption.find(id)
110   rescue ActiveRecord::RecordNotFound
111     render :json => ActiveSupport::JSON.encode({:error => true,
            :errorInfo => "Invalid disruption selected!"}) and return
112   end
113   if (disruption == nil)
114     render :json => ActiveSupport::JSON.encode({:error => true,
            :errorInfo => "Invalid disruption selected!"}) and return
115   end
116   disruption.hide = !disruption.hide
117   disruption.save
118   $disruptionListLatUpdateTime = DateTime.now
```

```
119     list and return
120   end
121
122   def addComment
123     id = nil
124     commentText = nil
125     begin
126       id = Integer(params[:id])
127       commentText = params[:comment]
128     rescue ArgumentError
129       render :json => ActiveSupport::JSON.encode({:error => true,
              :message => "Incorrect parameters."}) and return
130     end
131     if (commentText == nil)
132       render :json => ActiveSupport::JSON.encode({:error => true,
              :message => "Comment is empty."}) and return
133     end
134     begin
135       disruption = Disruption.find(id)
136     rescue ActiveRecord::RecordNotFound
137       render :json => ActiveSupport::JSON.encode({:error => true,
              :message => "Invalid disruption."}) and return
138     end
139     if (disruption == nil)
140       render :json => ActiveSupport::JSON.encode({:error => true,
              :message => "Invalid disruption."}) and return
141     end
142     comment = DisruptionComment.new
143     comment.disruptionId = disruption.id
144     comment.comment = commentText
145     comment.operatorId = session[:operatorId]
146     comment.save
147     @disruption = Disruption.includes(:comments).find(id)
```

```
148     @return = {:error => false, :partial => render_to_string(:partial
            => "comments")}
149     render :json => ActiveSupport::JSON.encode(@return)
150   end
151
152   def comments
153     error = "<h1>No disruption specified</h1> <a
            class=\"close-reveal-modal\">&#215;</a>"
154     id = nil
155     begin
156       id = Integer(params[:id])
157     rescue ArgumentError
158       render text: error and return
159     end
160     if (id == nil)
161       render text: error and return
162     end
163     begin
164       @disruption = Disruption.includes(:comments).find(id)
165     rescue ActiveRecord::RecordNotFound
166       render text: error and return
167     end
168     if (@disruption == nil)
169       render text: error and return
170     end
171     render partial: "comments"
172   end
173
174   private
175   #Time between each Ajax update call for the disruption list
176   TIMEOUT = 1000
177   TIME_FORMAT = "%Y/%m/%d %H:%M:%S"
178
179   def getLatUpdateTime
```

```
180     return
            Time.strptime(EngineConfiguration.find("latestFeedTime").value,
            TIME_FORMAT)
181   end
182
183   def getDisruptions
184     session[:disruptionListLatUpdateTime] =
            DateTime.now.in_time_zone('London')
185     session[:fileVersion] = getLatUpdateTime
186     whereClause = "\"clearedAt\" IS NULL"
187     order = {hide: :asc, delayInSeconds: :desc,
            routeTotalDelayInSeconds: :desc, firstDetectedAt: :desc}
188     if session[:operatorId] == nil
189       whereClause += " AND NOT \"hide\""
190       order.drop(1)
191     end
192     if sort_column != false && sort_direction != false
193       order = "\""+sort_column + "\" " + sort_direction
194     end
195     disruptions = Disruption.includes(:fromStop,
            :toStop).where(whereClause).order(order)
196     return disruptions.paginate(:page => params[:page], :per_page => 20)
197   end
198
199   def formatDatetimeString(timeString)
200     return timeString.strftime("%H:%M:%S %d/%m/%Y")
201   end
202
203   def capitalizeAll(string)
204     return string.split.map(&:capitalize).join(' ')
205   end
206
207   def getURLToBusStop(busStop)
208     if (busStop.instance_of?(BusStop))
```

```
209       return 'http://countdown.tfl.gov.uk/#|searchTerm=' + busStop.code
210     end
211     return 'Undefined'
212   end
213
214   def getLinkToBusStop(busStop)
215     if (busStop.instance_of?(BusStop))
216       return '<a href="'+getURLToBusStop(busStop) + '"
            target="_blank">' + capitalizeAll(busStop.name) + '</a>'
217     end
218     return 'Undefined'
219   end
220
221   def sort_column
222     if params[:sort]
223       session[:sort] = params[:sort]
224     end
225     Disruption.column_names.include?(session[:sort]) ? session[:sort] :
          false
226   end
227
228   def sort_direction
229     if params[:direction]
230       session[:direction] = params[:direction]
231     end
232     %w[asc desc].include?(session[:direction]) ? session[:direction] :
          false
233   end
234
235 end
```

Listing C.44: Web Application - History Controller

```
1 class HistoryController < ApplicationController
```

```
2   helper_method :sort_column, :sort_direction
3
4   def filter
5     checkParams
6     @disruptions = getDisruptions(session[:fromFilter],
          session[:toFilter])
7     render partial: 'list'
8   end
9
10  def index
11    checkParams
12    @disruptions = getDisruptions(session[:fromFilter],
          session[:toFilter])
13  end
14
15  private
16
17  def getDisruptions(fromDate, toDate)
18    if (toDate)
19      begin
20        to = Time.strptime(toDate, "%d/%m/%Y %H:%M:%S")
21      rescue ArgumentError
22        to = nil
23      end
24    end
25
26    if (fromDate)
27      begin
28        from = Time.strptime(fromDate, "%d/%m/%Y %H:%M:%S")
29      rescue ArgumentError
30        from = nil
31      end
32    end
33
```

```
34    whereClause = nil
35    if (to != nil && from != nil)
36      whereClause = "\"firstDetectedAt\" >= '"+ getFormatForDB(from)+
          "' AND \"firstDetectedAt\" <= '"+getFormatForDB(to)+"'"
37    elsif (to != nil)
38      whereClause = "\"firstDetectedAt\" <= '"+ getFormatForDB(to)+"'"
39    elsif (from != nil)
40      whereClause = "\"firstDetectedAt\" >= '"+getFormatForDB(from)+"'"
41    end
42
43    order = {firstDetectedAt: :asc, delayInSeconds: :desc,
          routeTotalDelayInSeconds: :desc}
44    if sort_column != false && sort_direction != false
45      order = "\""+sort_column + "\" " + sort_direction
46    end
47
48    if (whereClause != nil)
49      disruptions = Disruption.includes(:fromStop,
            :toStop).where(whereClause).order(order)
50    else
51      disruptions = Disruption.includes(:fromStop, :toStop).order(order)
52    end
53
54    return disruptions.paginate(:page => params[:page], :per_page => 20)
55  end
56
57  def getFormatForDB(time)
58    return time.strftime("%Y-%m-%d %H:%M:%S")
59  end
60
61
62  def sort_column
63    Disruption.column_names.include?(session[:sort]) ? session[:sort] :
          false
```

```
64    end
65
66    def sort_direction
67      %w[asc desc].include?(session[:direction]) ? session[:direction] :
            false
68    end
69
70    def checkParams
71      if params[:sort]
72        session[:sort] = params[:sort]
73      end
74      if params[:direction]
75        session[:direction] = params[:direction]
76      end
77      if params[:from]
78        session[:fromFilter] = params[:from]
79      end
80      if params[:to]
81        session[:toFilter] = params[:to]
82      end
83    end
84  end
```

Listing C.45: Web Application - Main Controller

```
1   class MainController < ApplicationController
2     before_filter :saveLoginState, :only => [:login]
3
4     SPEED_PAUSED = 0
5     SPEED_SLOW = 1
6     SPEED_NORMAL = 2
7     SPEED_FAST = 3
8     SPEED_VERY_FAST = 4
9
```

```
10  def login
11    operator = Operator.authenticate(params[:username],
          params[:password])
12    if operator
13      session[:operatorId] = operator.id
14      session[:operatorUsername] = operator.username
15      session[:operatorAdmin] = operator.admin
16      flash[:success] = "Welcome again, you are logged in as
            #{operator.username}."
17    else
18      flash[:alert] = "Invalid Username or Password!"
19    end
20    redirect_to disruption_index_url, status: 301
21  end
22
23  def logout
24    session[:operatorId] = nil
25    session[:operatorUsername] = nil
26    session[:operatorAdmin] = false
27    flash[:success] = "You logged out successfully."
28    redirect_to disruption_index_url, status: 301
29  end
30
31  def index
32    redirect_to disruption_index_url, status: 301
33  end
34
35  def about
36  end
37
38  #Used for changing the speed for the thread responsible for
        simulations - TODO: Remove before putting into production
39  def speed
40    feedThreadPaused = EngineConfiguration.find("feedThreadPaused")
```

```
41      feedThreadPaused.value = "false"
42      feedThreadSpeedInMilliSeconds =
            EngineConfiguration.find("feedThreadSpeedInMilliSeconds")
43      speed = Integer(params[:speed])
44      if (speed == SPEED_SLOW)
45        feedThreadSpeedInMilliSeconds.value = 10000
46      elsif (speed == SPEED_FAST)
47        feedThreadSpeedInMilliSeconds.value = 2500
48      elsif (speed == SPEED_NORMAL)
49        feedThreadSpeedInMilliSeconds.value = 5000
50      elsif (speed == SPEED_VERY_FAST)
51        feedThreadSpeedInMilliSeconds.value = 1000
52      else
53        feedThreadPaused.value = "true"
54        feedThreadSpeedInMilliSeconds.value = 5000
55      end
56      feedThreadPaused.save
57      feedThreadSpeedInMilliSeconds.save
58      @return = {:success => true}
59      render :json => ActiveSupport::JSON.encode(@return)
60    end
61
62  end
```

Listing C.46: Web Application - Settings Controller

```
1   class SettingsController < ApplicationController
2     before_filter :authenticateUser, :only => [:edit, :save, :index]
3
4     def edit
5       if (params[:id] != nil)
6         @configs = EngineConfiguration.find_by_key(params[:id])
7         render partial: "edit"
8       else
```

```
      render text: "<h1>No configuration parameter specified</h1> <a
          class=\"close-reveal-modal\">&#215;</a>"
    end
  end

  def save
    if (params[:key] != nil)
      config = EngineConfiguration.find_by_key(params[:key])
      if config.editable
        config.value = params[:value]
        config.save
        result = {:error => false, :newValue => config.value, :key =>
            params[:key]}
      else
        result = {:error => true, :errorText => "Field is not
            editable."}
      end
    else
      result ={:error => true, :errorText => "Some error occured."}
    end
    render :json => ActiveSupport::JSON.encode(result)
  end

  def index
    configs = EngineConfiguration.all.order(key: :desc)
    @configs = configs.paginate(:page => params[:page], :per_page => 20)
  end

end
```

Listing C.47: Web Application - Application Helper

```
module ApplicationHelper
  require 'time'
```

```
  TIME_FORMAT = "%m/%d/%Y %H:%M:%S"

  def formatDatetimeString(timeString, format = TIME_FORMAT)
    return Time.strptime(timeString, format).strftime("%H:%M:%S")
  end

  def formatDatetimeStringWithDate(timeString, format = TIME_FORMAT)
    return Time.strptime(timeString, format).strftime("%H:%M:%S
        %m/%d/%Y")
  end

  def capitalizeAll(string)
    return string.split.map(&:capitalize).join(' ')
  end

  def sortable(column, title = nil)
    title ||= column.titleize
    css_class = column == sort_column ? "current #{sort_direction}" : ""
    direction = column == sort_column && sort_direction == "asc" ?
        "desc" : "asc"
    return "<a class=\""+css_class.to_s+"\"
        onclick=\"sort('"+column+"','"+direction+"')\" "">"+title+"</a>"
    # link_to title, disruption_list_path({:sort => column, :direction
        => direction}), :remote => true, :id => "ajax_trigger", :class
        => css_class
  end

end
```

Listing C.48: Web Application - Disruption Helper

```
module DisruptionHelper

  def getURLToBusStop(busStop)
```

```
    if (busStop.instance_of?(BusStop))
      return 'http://countdown.tfl.gov.uk/#|searchTerm=' + busStop.code
    end
    return 'Undefined'
  end

  def getLinkToBusStop(busStop)
    if (busStop.instance_of?(BusStop))
      return '<a href="'+getURLToBusStop(busStop) + '"
          target="_blank">' + capitalizeAll(busStop.name) + '</a>'
    end
    return 'Undefined'
  end

  def getLinkToBusRoute(route, run)
    if (route != nil && run != nil)
      return '<a href="http://www.tfl.gov.uk/bus/route/' + route +
          '/?direction=' + run + '" target="_blank">' + route + '</a>'
    end
    return 'Undefined'
  end

  def secondsToMinutes(seconds)
    return (seconds / 60).round
  end

end
```

Listing C.49: Web Application - Bus Stop Model

```
class BusStop < ActiveRecord::Base
  self.table_name = "BusStops"
  belongs_to :fromStop, :class_name => "Disruption", :foreign_key =>
      "fromStopLBSLCode", :primary_key => "lbslCode"
```

```
  belongs_to :toStop, :class_name => "Disruption", :foreign_key =>
      "toStopLBSLCode", :primary_key => "lbslCode"
  belongs_to :startStop, :class_name => "Section", :foreign_key =>
      "startStopLBSLCode", :primary_key => "lbslCode"
  belongs_to :endStop, :class_name => "Section", :foreign_key =>
      "endStopLBSLCode", :primary_key => "lbslCode"
end
```

Listing C.50: Web Application - Disruption Model

```
class Disruption < ActiveRecord::Base
  self.table_name = "Disruptions"
  has_one :fromStop, :class_name => "BusStop", :foreign_key =>
      "lbslCode", :primary_key => "fromStopLBSLCode"
  has_one :toStop, :class_name => "BusStop", :foreign_key =>
      "lbslCode", :primary_key => "toStopLBSLCode"
  has_many :comments, :class_name => "DisruptionComment", :foreign_key
      => "disruptionId", :primary_key => "id"

  require 'time'
  TIME_FORMAT = "%m/%d/%Y %H:%M:%S"

  MEDIUM_THRESHOLD = 20
  SERIOUS_THRESHOLD = 40
  SEVERE_THRESHOLD = 60

  RED_COLOR = "#CC3333"
  YELLOW_COLOR = "#FFCC00"
  GREEN_COLOR = "#006633"
  ORANGE_COLOR = "#FF6600"

  TREND_SYMBOL_POSITIVE = "&#8593;"
  TREND_SYMBOL_NEGATIVE = "&#8595;"
  TREND_SYMBOL_NEUTRAL = "&#8597;"
```

```
22
23  def getDetectedAt(date = false)
24    if(date)
25      format = "%H:%M:%S %m/%d/%Y"
26    else
27      format = "%H:%M:%S"
28    end
29    return self.firstDetectedAt.strftime(format)
30  end
31
32  def getClearedAt(date = false)
33    if(date)
34      format = "%H:%M:%S %m/%d/%Y"
35    else
36      format = "%H:%M:%S"
37    end
38    if (self.clearedAt != nil)
39      return self.clearedAt.strftime(format)
40    end
41    return ""
42  end
43
44  def getRunString
45    if (self.run == 1)
46      return "Outbound"
47    end
48    return "Inbound"
49  end
50
51  def getDelay
52    return secondsToMinutes(self.delayInSeconds)
53  end
54
55  def getTotalDelay
```

```
    return secondsToMinutes(self.routeTotalDelayInSeconds)
  end

  def totalDelayColor
    if (self.getTotalDelay > SEVERE_THRESHOLD)
      return "severe"
    end
    if (self.getTotalDelay > SERIOUS_THRESHOLD)
      return "serious"
    end
    return "medium"
  end

  def delayColor
    if (self.getDelay > SEVERE_THRESHOLD)
      return "severe"
    end
    if (self.getDelay > SERIOUS_THRESHOLD)
      return "serious"
    end
    return "medium"
  end

  def trendColor
    if (self.trend == 1)
      return "decrease"
    elsif (self.trend == 0)
      return "stable"
    else
      return "increase"
    end
  end

  def trendSymbol
```

```
    if (self.trend == 1)
      return TREND_SYMBOL_NEGATIVE
    elsif (self.trend == 0)
      return TREND_SYMBOL_NEUTRAL
    else
      return TREND_SYMBOL_POSITIVE
    end
  end

  def secondsToMinutes(seconds)
    return (seconds / 60).floor
  end
end
```

Listing C.51: Web Application - Disruption Comment Model

```
class DisruptionComment < ActiveRecord::Base
  self.table_name = "DisruptionComments"
  has_one :operator, :class_name => "Operator", :foreign_key => "id",
      :primary_key => "operatorId"
  belongs_to :disruption, :class_name => "Disruption", :foreign_key =>
      "id", :primary_key => "disruptionId"

end
```

Listing C.52: Web Application - Engine Configuration Model

```
class EngineConfiguration < ActiveRecord::Base
  self.table_name = "EngineConfigurations"
end
```

Listing C.53: Web Application - Operator Model

```
class Operator < ActiveRecord::Base
```

```
  self.table_name = "Operators"
  has_many :disruption_comments, :class_name => "DisruptionComment",
      :foreign_key => "operatorId", :primary_key => "id"

  def self.authenticate(user="", logingPassword="")
    user = Operator.find_by(username: user)
    if user && user.match_password(logingPassword)
      return user
    else
      return false
    end
  end

  def match_password(logingPassword="")
    return logingPassword == self.password
    #TODO:Add encryption before putting into production
    # encrypted_password == BCrypt::Engine.hash_secret(password, salt)
  end

end
```

Listing C.54: Web Application - Section Model

```
class Section < ActiveRecord::Base
  self.table_name = "Sections"
  has_many :lostTimes, :class_name => "SectionsLostTime", :foreign_key
      => "sectionId", :primary_key => "id"
  has_one :latestLostTime, :class_name => "SectionsLostTime",
      :foreign_key => "serialId", :primary_key => "serialId"
  has_one :startStop, :class_name => "BusStop", :foreign_key =>
      "lbslCode", :primary_key => "startStopLBSLCode"
  has_one :endStop, :class_name => "BusStop", :foreign_key =>
      "lbslCode", :primary_key => "endStopLBSLCode"

```

```
  # has_one :latestLostTime, -> { where("\"SectionsLostTime\".timestamp
      = (SELECT \"Sections\".\"latestLostTimeUpdateTime\" FROM
      \"Sections\" WHERE \"SectionsLostTime\".\"sectionId\" =
      \"Sections\".id)") }, :class_name => "SectionsLostTime",
      :foreign_key => "sectionId", :primary_key => "id"

  def getLatestLostTimeMinutes
    return (self.latestLostTime.lostTimeInSeconds / 60).round
  end
end
```

Listing C.55: Web Application - Section Lost Time Model

```
class SectionsLostTime < ActiveRecord::Base
  self.table_name = "SectionsLostTime"
  belongs_to :section, :class_name => "Sections", :foreign_key => "id",
      :primary_key => "sectionId"

end
```

Listing C.56: Database creation script

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


-- Dumped from database version 9.4.1
-- Dumped by pg_dump version 9.4.1
-- Started on 2015-04-26 13:04:12


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: -
--

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



\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: -
--

CREATE SCHEMA public;



--
-- TOC entry 2078 (class 0 OID 0)
-- Dependencies: 5
```

```
-- Name: SCHEMA public; Type: COMMENT; Schema: -; Owner: -
--


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 2079 (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: -;
    Tablespace:
--
```

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



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



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

CREATE TABLE "DisruptionComments" (
```

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



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

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



--
-- TOC entry 2080 (class 0 OID 0)
-- Dependencies: 186
-- Name: DisruptionComments_id_seq; Type: SEQUENCE OWNED BY; Schema:
    public; Owner: -
--

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



--
```

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


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

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


--
```

```
-- TOC entry 2081 (class 0 OID 0)
-- Dependencies: 183
-- Name: Disruptions_id_seq; Type: SEQUENCE OWNED BY; Schema: public;
    Owner: -
--


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



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


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



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


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

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

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



--
-- TOC entry 2082 (class 0 OID 0)
-- Dependencies: 177
-- Name: Operators_id_seq; Type: SEQUENCE OWNED BY; Schema: public;
     Owner: -
--

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



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

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

```
    "endStopLBSLCode" character varying(10),
    sequence smallint
);



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



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

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

```
--
-- TOC entry 2083 (class 0 OID 0)
-- Dependencies: 181
-- Name: SectionsLostTime_sectionId_seq; Type: SEQUENCE OWNED BY;
    Schema: public; Owner: -
--


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: -
--


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



--
-- TOC entry 2084 (class 0 OID 0)
-- Dependencies: 184
-- Name: SectionsLostTime_serialId_seq; Type: SEQUENCE OWNED BY;
    Schema: public; Owner: -
--


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: -
--

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


--
-- TOC entry 2085 (class 0 OID 0)
-- Dependencies: 179
-- Name: Sections_id_seq; Type: SEQUENCE OWNED BY; Schema: public;
    Owner: -
--

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


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

CREATE TABLE schema_migrations (
    version character varying NOT NULL
);
```

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

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: -
--

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: -
--

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: -
--
```

```
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: -
--

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: -
--

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: -;
    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:
    -; 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: -;
    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: -; 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: -; 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: -;
    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: -;
    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: -;
    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:
    -; 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: -
--

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: -
--

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: -
--

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: -
--

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: -
--

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: -
--

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: -
--

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: -
--

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: -
--


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



-- Completed on 2015-04-26 13:04:12


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