

Real-Time Display of Dublin Traffic Information on the Web

Mark Dineen
B.Eng.(Elec.)

A dissertation submitted to the University of Dublin,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

September 2000

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed: _____
Mark Dineen
September 2000

Permission to lend and/or copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Signed: _____
Mark Dineen
September 2000

Acknowledgements

I would like to acknowledge the help of all of the following people without whose assistance this project would never have succeeded.

First of all, I would like to thank my supervisor Dr. Vinny Cahill whose excellent guidance kept me on the straight and narrow. Brendan O'Brien and Dave Traynor from the SCATS Development Unit in Dublin Corporation have my eternal gratitude for willingly sharing their expertise in SCATS and all things traffic and canteen related. A special word of thanks also goes to Ms. Orla Powers, a Corporation summer student, who got stuck with the unenviable task of filling out the project's two thousand-line configuration file.

I would also like to thank the M.Sc. class of 2000 for their great friendship and help throughout the year.

Finally I would like to thank my family for their constant love and support.

Abstract

Many large-scale and highly specialised distributed systems such as Urban Traffic Control (UTC) systems produce considerable quantities of data on a real-time basis. This data is then frequently locked in a proprietary format that makes it difficult to utilise for external purposes such as historical data analysis or additional end-user applications. The purpose of this dissertation is to examine the techniques and issues involved in extracting, processing and presenting such large quantities of data to a variety of end-user applications in a meaningful and scalable fashion.

In order to gain further insights into this subject it was decided to develop an end-user interface for a real-life example of a large-scale distributed system that produces high volumes of data. The example chosen was SCATS, the city of Dublin's UTC system. This system measures the traffic flows passing through several hundred junctions around the city and automatically adjusts the timing of the each intersection's traffic lights on a minute-by-minute basis.

Three central issues present themselves in this project. The first of these is the question of how to extract data from a proprietary system and convert it into a format that can be universally read and understood. The second issue is how an algorithm can be created that takes the available data and generates a meaningful measure of road traffic congestion at any given point in time. The third problem involves presenting such large quantities of data to a wide variety of end-users in a manner that is clear, meaningful and intuitive.

The three tier application resulting from the research carried out during the course of the project is evaluated under several headings. Concurrency issues along with the performance, scalability and usability of each part of the implementation are discussed and evaluated. The congestion algorithm is compared against other measures that are available in SCATS, and formulae from the existing literature.

Table of Contents

CHAPTER 1 INTRODUCTION	1
1.1 BACKGROUND	1
1.2 URBAN TRAFFIC CONTROL SYSTEMS	2
1.3 GOALS	2
1.4 ACHIEVEMENTS	3
1.5 ROAD-MAP	4
1.5.1 <i>Background</i>	4
1.5.2 <i>Urban Traffic Control Systems</i>	4
1.5.3 <i>Architecture</i>	5
1.5.4 <i>Implementation</i>	5
1.5.5 <i>Evaluation</i>	5
1.5.6 <i>Conclusions</i>	5
CHAPTER 2 BACKGROUND	6
2.1 EXISTING TRAFFIC INFORMATION WEBSITES	6
2.2 QUANTIFYING URBAN ROAD TRAFFIC CONGESTION	10
2.2.1 <i>Queue-Based Measures</i>	10
2.2.2 <i>Image Processing</i>	12
2.2.3 <i>Intersection Delay</i>	13
2.2.4 <i>Travel Time</i>	14
2.3 WEBSITE USABILITY	15
CHAPTER 3 URBAN TRAFFIC CONTROL SYSTEMS	18
3.1 GENERAL TRAFFIC TERMINOLOGY	18
3.2 UTC ARCHITECTURES	20
3.2.1 <i>Fully Centralised</i>	20
3.2.2 <i>Fully Distributed</i>	21
3.2.3 <i>Centralised Hierarchical</i>	22
3.2.4 <i>Distributed Hierarchical</i>	23
3.3 SCATS	23
3.3.1 <i>Physical Layer</i>	24
3.3.2 <i>System Operation</i>	25
3.3.3 <i>SCATS Traffic Measurements and Calculations</i>	26
3.3.4 <i>System Monitor Display</i>	30
3.4 ALTERNATIVE UTC SYSTEMS	33
3.4.1 <i>SCOOT</i>	33
CHAPTER 4 ARCHITECTURE	35
4.1 THREE TIER APPROACH	36
4.1.1 <i>Tier 1</i>	37
4.1.2 <i>Tier 2</i>	37
4.1.3 <i>Tier 3</i>	38
4.2 OBJECT MODEL	39
4.3 XML	40
4.3.1 <i>DTD Design</i>	41
4.4 END-USER INTERFACE	45
CHAPTER 5 IMPLEMENTATION	46
5.1 TIER 1	46
5.2 TIER 2	49
5.3 TIER 3	51

5.3.1 <i>Dynamic Congestion Map</i>	51
5.3.2 <i>Static Map Producer</i>	56
5.4 PHYSICAL IMPLEMENTATION	58
CHAPTER 6 EVALUATION.....	60
6.1 CONGESTION ALGORITHM.....	60
6.2 PERFORMANCE	62
6.3 SCALABILITY.....	63
6.4 CONCURRENCY	64
CHAPTER 7 CONCLUSIONS.....	66
7.1 ACHIEVEMENTS.....	66
7.2 POTENTIAL USERS.....	67
7.3 FUTURE WORK.....	68
REFERENCES.....	70

Table of Figures

FIGURE 2.1: GEORGIA NAVIGATOR SPEED AND INCIDENT MAP	6
FIGURE 2.2: GEORGIA NAVIGATOR LIST OF TRAVEL TIMES.....	7
FIGURE 2.3: ONLINE SIMULATION, DOWNTOWN AREA DUISBURG.....	7
FIGURE 2.4: ATHENS REAL-TIME TRAFFIC MAP	8
FIGURE 2.5: ATHENS – HOW FAR CAN YOU TRAVEL IN 15 MINUTES?	9
FIGURE 2.6: MAP SHOWING POSITION OF BUSES ON SUPERROUTE 66	9
FIGURE 2.7: OVERVIEW OF THE TRENDS DEMONSTRATOR	10
FIGURE 2.8: RELATIVE PROPORTION OF NETSCAPE USERS USING DIFFERENT VERSIONS OF THE BROWSER OVER TIME.....	16
FIGURE 3.1: EXAMPLE PHASE SCHEME AT A RIGHT-ANGLE INTERSECTION	19
FIGURE 3.2: FULLY CENTRALISED CONTROL	20
FIGURE 3.3: FULLY DISTRIBUTED CONTROL	21
FIGURE 3.4: CENTRALISED HIERARCHICAL CONTROL.....	22
FIGURE 3.5: DISTRIBUTED HIERARCHICAL CONTROL.....	23
FIGURE 3.6: SCATS PHYSICAL ARCHITECTURE	24
FIGURE 3.7: SYSTEMS AND SUBSYSTEMS [SIMS AND DOBINSON, 1980].....	25
FIGURE 3.8: ILLUSTRATION OF THE STRETCH EFFECT	30
FIGURE 3.9: SAMPLE SM DISPLAY	30
FIGURE 4.1: THREE-TIER ARCHITECTURE	36
FIGURE 4.2: OBJECT MODEL ARCHITECTURE.....	40
FIGURE 4.3: ROUTESDATA.DTD.....	42
FIGURE 4.4: SYSLX.DTD	43
FIGURE 4.5: SMDATA.DTD.....	44
FIGURE 5.1: SAMPLE TELNET LOGIN CODE (GETSM.PL)	47
FIGURE 5.2: AUTOMATED SM DISPLAY RETRIEVAL (GETSM.PL)	48
FIGURE 5.3: CONGESTION ALGORITHM (BLACKBOX.JAVA)	50
FIGURE 5.4: SCREENSHOT OF THE DYNAMIC CONGESTION MAP.....	52
FIGURE 5.5: POP-UP DIALOG BOX SHOWING DETAILED INTERSECTION INFORMATION ..	53
FIGURE 5.6: POP-UP LEGEND DIALOG BOX.....	53
FIGURE 5.7: CONGESTION LINES USING BUFFEREDIMAGE (MAPPANEL.JAVA).....	54
FIGURE 5.8: SCROLLBARS (MAPGUI.JAVA).....	55
FIGURE 5.9: STATIC IMAGE OF CITY CENTRE	56
FIGURE 5.10: STILLORGAN STATIC ROUTE	56
FIGURE 5.11: CREATING A STATIC JPEG FILE (STATICMAPPRODUCER.JAVA).....	57
FIGURE 5.12: STATICMAPSETUP.DTD.....	58
FIGURE 5.13: PHYSICAL IMPLEMENTATION	59
FIGURE 6.1: GRAPH OF DEGREE OF SATURATION OVER A PERIOD OF ONE DAY	60
FIGURE 6.2: GRAPH OF CONGESTION MEASURE OVER A PERIOD OF ONE DAY	61

Chapter 1 Introduction

Many large-scale and highly specialised distributed systems such as Urban Traffic Control (UTC) systems produce considerable quantities of data on a real-time basis. This data is then frequently locked in a proprietary format that makes it difficult to utilise for external purposes such as historical data analysis or additional end-user applications. After allocating the considerable resources necessary to create systems that are capable of collecting such vast quantities of data, it would appear shortsighted to limit the data to one narrow (albeit highly useful) function. The purpose of this dissertation is to examine the techniques and issues involved in extracting, processing and presenting such large quantities of data to a sizable group of end-users in a meaningful and scalable fashion.

1.1 Background

In order to gain further insights into this subject it was decided to develop an end-user interface for a real-life example of a large-scale distributed system that produces high volumes of data. The example chosen was the city of Dublin's UTC system. This system, which is called the Sydney Coordinated Adaptive Traffic System (SCATS), works by measuring real-time traffic flows through several hundred junctions around the city. Using this information SCATS then automatically adjusts the timing of the traffic lights at each intersection on a minute-by-minute basis. In 1999, the Office of the Director of Traffic in Dublin Corporation contacted the Department of Computer Science in Trinity College with a view to investigating the possibility of providing a World-Wide Web based interface to the SCATS system. It was envisaged that this interface could then be used to provide the public with real-time information on the current traffic situation around Dublin city.

At the time of writing there are several websites available on the web that provide road traffic congestion information. Unfortunately most of these sites are American orientated and deal almost exclusively with freeways. In fact, the only urban traffic congestion map that is currently available is the Athens (Greece) Real-Time Traffic Map, which unfortunately provides very little information on how it is implemented.

The concept of measuring or quantifying road traffic congestion is one on which much general research but very little specific information is available. Most of the

existing literature is based upon specific studies that collect customised measurements that are unavailable on a UTC system such as SCATS. This means that a specialised congestion algorithm had to be developed which only uses data that is available from SCATS.

1.2 Urban Traffic Control Systems

Since the first road traffic signals were installed in London circa 1870 there have been many stages in the evolution of traffic signalling systems. Many different physical and system architectures are currently in use. All of the physical architectures can be grouped into one of four categories: fully centralised, fully distributed, centralised hierarchical and distributed hierarchical. The latest system architectures are called adaptive UTC's and these work by combining the two the philosophies of fixed-time UTC and vehicle actuation. The principle of fixed-time UTC is based on the idea that where traffic signals are in close physical proximity good vehicle throughput can be obtained by linking the lights together on a fixed-time plan controlled by computer. This time plan can then change according to different times of the day (e.g. there can be separate "morning peak", "evening-peak", "off-peak" and "night-time" plans). The principle of vehicle actuation relies on detectors that are capable of detecting the difference between traffic volumes on each approach to the intersection. Using this data the amount of "green-time" given to each approach on an intersection is extended or reduced depending on the relative traffic volumes at that approach. SCATS uses inductive loop detectors in order to measure the flows of traffic passing through an intersection. From these inductive loops SCATS can calculate three fundamental values (original volume, reconstituted volume and degree of saturation) for each lane that has a detector.

1.3 Goals

There were three main objectives that this dissertation set out to fulfil. The first of these was to extract the data from SCATS and convert it into a format that could be universally read and understood. The second objective was to create an algorithm that could take the available data and generate a meaningful measure of road traffic congestion at any given point in time. The third goal was to be able to present this large mass of data to a wide variety of end-users in a manner that would be clear, meaningful and intuitive. As well as these three objectives a number of constraints

were placed on any proposed solution to ensure its future viability. These constraints specified that the system had to be scalable, modular, easily configurable and could not interfere with the running of SCATS. Being scalable meant that there could be no constraints placed upon the number of intersections, end-users, or end-user applications that the system could cope with. Any proposed solution had to be modular to ensure that the congestion algorithm could be changed at will, that a new version of SCATS could be used without requiring a complete rewrite of all the code, and that extra applications could be added with no difficulty. The solution had to include user-friendly configuration files to allow changes to be made to the intersections and routes that were to be displayed. Finally, for obvious reasons, any proposed solution could in no way interfere with the running of SCATS itself, which meant that only existing interfaces to SCATS could be used.

1.4 Achievements

In order to achieve the main objectives of this project four major tasks were carried out. The first task was a comprehensive literature survey that looked at the congestion measures and travel time algorithms developed in the existing literature, examined any existing traffic websites, and studied the issues that surround website usability. The second step was to develop a suitable architecture that would attempt to meet all of the objectives and constraints mentioned in the previous section. The third task was to actually implement this architecture and produce a working application. Finally the resulting implementation was compared and evaluated against the original objectives and any features or issues that arose during its construction were discussed.

The architecture that resulted from the design phase of the project is a three-tier model that utilises independent applications on each level in order to maintain as much modularity as possible. Extensible Mark-up Language (XML) was chosen as the interface between the tiers so as to ensure that future applications and modules could be easily integrated. The first tier of the model is responsible for the SCATS data extraction and uses an automated telnet session that is controlled by an Object Oriented (OO) Perl script. The second tier is responsible for applying a suitable congestion algorithm and passing on the data to the third tier. This tier is implemented using a Simple API to XML (SAX) parser that is contained within a Java application. The third and final tier contains the Java based applications that present the data to the

end-users. These applications present the data in three different formats in order to meet the needs of as wide a user base as possible and utilise Java features such as applets, swing, SAX, media codecs and two-dimensional graphics.

During the evaluation phase of the project the results from the congestion algorithm were compared against results from other measures that are available in SCATS. It was found that the congestion algorithm did indeed produce a more meaningful and useful measure of congestion than anything currently available on SCATS. The overall performance of the implementation was examined and the maximum possible inherent latency in getting data from SCATS to the end-user interface was found to be 190 seconds. If it were deemed necessary it was found that this figure could be reduced to approximately 100 seconds by making the second and third tiers work at an increased rate. The very nature of the three-tier architecture ensured that the goals of scalability and modularity were achieved. Unfortunately, the design also introduced some issues of concurrency into the system for which possible solutions were then presented and discussed.

1.5 Road-Map

This section contains a brief introduction and summary to each of the remaining chapters in this dissertation.

1.5.1 Background

This chapter presents all the research that was carried out on existing road traffic websites, current measures of congestion and travel time, and the issues surrounding website usability.

1.5.2 Urban Traffic Control Systems

The purpose of this chapter is to introduce the reader to the world of road traffic engineering and equip them with sufficient knowledge to understand some of the traffic related issues that had to be faced while carrying out this project. The chapter starts with an introduction to some general traffic engineering terminology. After this some of the general physical architectures that UTC's are based upon is discussed. A more in-depth review of the SCATS UTC, it's physical and system architecture, the data it is capable of collecting and the interfaces it uses to convey this information is

then carried out. Finally, some of the alternative UTC's currently in use around the world are introduced and compared against SCATS.

1.5.3 Architecture

This section details the design phase of the project and shows the decision making process that lies behind four key areas of the project. The areas that are discussed are the three-tier architecture, the object model that holds the traffic data, the format of the XML files and ways in which congestion information is presented to the end-user.

1.5.4 Implementation

In this chapter the issues that arose during the construction phase of the project are revealed and discussed. Each layer of the three-tier architecture is individually examined as well as the XML files that act as an interface between the tiers. The physical locations of each of the applications that go to make up the entire system are also shown.

1.5.5 Evaluation

This is the chapter that compares the actual implementation against the original objectives and constraints. The overall performance of the system along with any outstanding issues or features is also examined.

1.5.6 Conclusions

This is the final chapter of the dissertation. It looks at the variety of potential users who could be expected to benefit from the work carried out during this project. Some avenues of future work that were revealed during the course of the project are also discussed and conclusions are drawn from the project's overall results.

Chapter 2 Background

2.1 Existing Traffic Information Websites

There are several websites already on the web that provide road traffic congestion information. The (American) National Associations Working Group for Intelligent Traffic Systems (ITS) provide a table listing most of the real-time traffic websites that are currently available. Many of these sites are American orientated and deal almost exclusively with freeways. A typical example of such a site is the Georgia Department of Transportation's NAVIGATOR website.



Figure 2.1: Georgia NAVIGATOR speed and incident map

This map shows information on the current traffic speeds on the freeways around Georgia as well as information on incidents and road works. Users can also click on the sign symbols to see what information is being shown by a particular variable message sign or on the camera symbols to see a still image of the freeway from a particular traffic camera. There is a choice between six different regions that can be displayed. The user can also toggle (on or off) the type of information (speeds, incidents, etc.) that appears on the map at any given time. A link is provided to a separate page (shown in Figure 2.2) that displays current travel times between various points on the freeways.

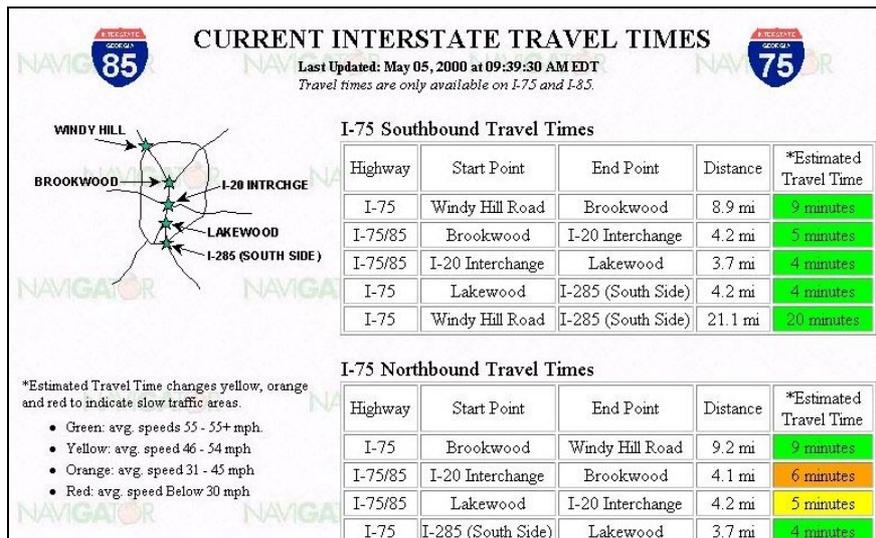


Figure 2.2: Georgia NAVIGATOR list of travel times

The Physics of Transport and Traffic Group at the University of Duisburg, Germany publish a map of a *simulated* traffic flow in downtown Duisburg. The state of the map is calculated from a microscopic simulation that is based on real traffic data from Duisburg.

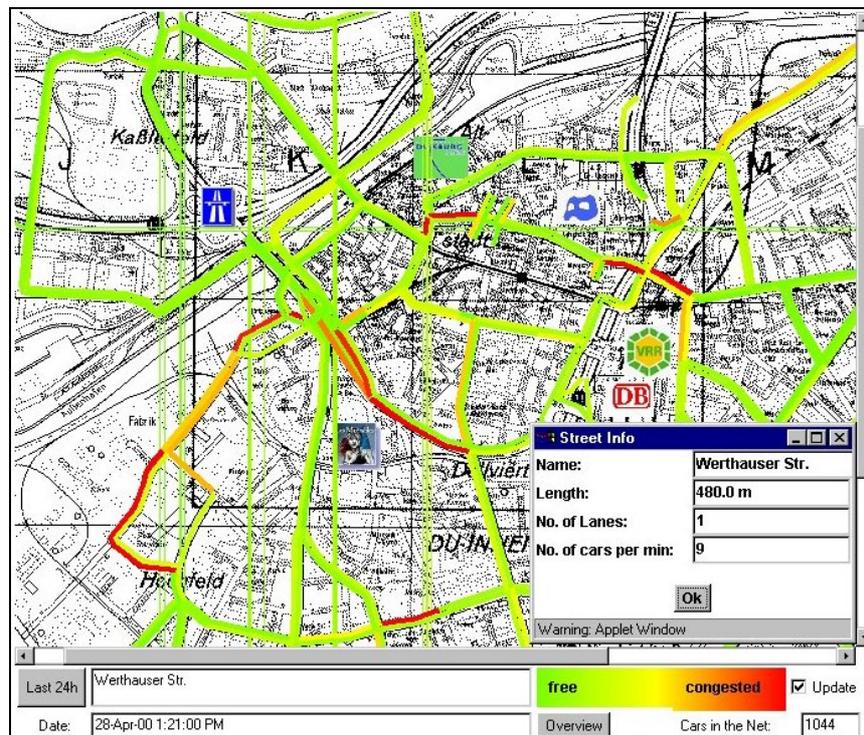


Figure 2.3: Online simulation, downtown area Duisburg

An interesting feature of this map is the way in which an applet is used to display the congestion information. Clicking on any of the simulated roads will bring up a dialogue box (shown in Figure 2.3) containing information on the name of the road,

estimates the user can travel in fifteen minutes from that point. The travel time is estimated solely from the available flow and occupancy data.

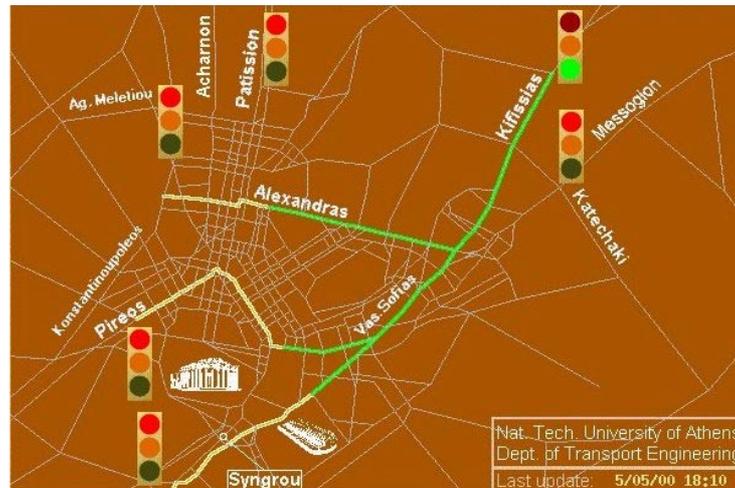


Figure 2.5: Athens – how far can you travel in 15 minutes?

Superoute 66 from British Telecom's Advanced Communications Technology Centre is another example of travel time that is published on the Internet. This particular site only monitors the travel times of buses on a particular route in Ipswich, England. Such information could also be of use to other road users, as the buses often have to operate in the same traffic flow as most of the other traffic vehicles.

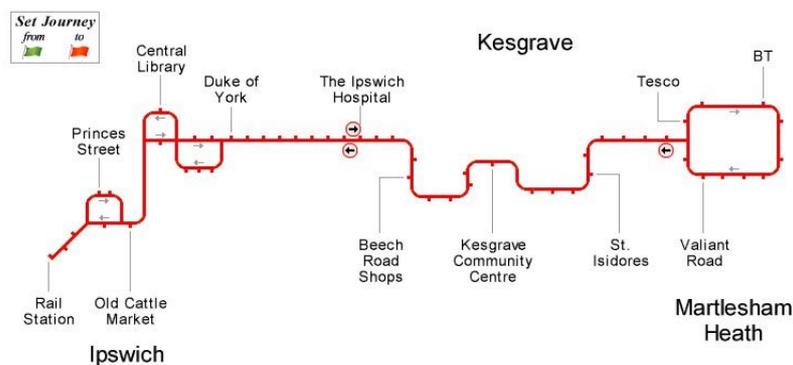


Figure 2.6: Map showing position of buses on Superoute 66

The principle behind the Superoute 66 system is that buses on the route are fitted with GPS receivers. Information from these receivers is broadcast via band 3 radio to a server where bus locations and predicted arrival times are calculated. This information is then published in three different forms on the Internet, i.e. Java applets, HTML pages and WML pages (for use with WAP enabled mobile phones).

A demonstrator from the Traffic Engineering Network Data Service (TRENDS) project shows how traffic information can be provided by a service implemented

using the Common Object Request Broker Architecture (CORBA). The purpose of this project (which ended in 1997 and was developed as part of a European Commission funded ESPRIT programme) was to allow data to be collected in one or more places, moved across the Internet in real-time for processing at another site (or sites) that are location independent. Finally, the processed information would then be published on the Internet. A project demonstrator was created that collected low-level information from Gothenburg's traffic system and transferred the data to a processing centre in Bristol and Oxfordshire. After processing, this data was then published on the Internet.

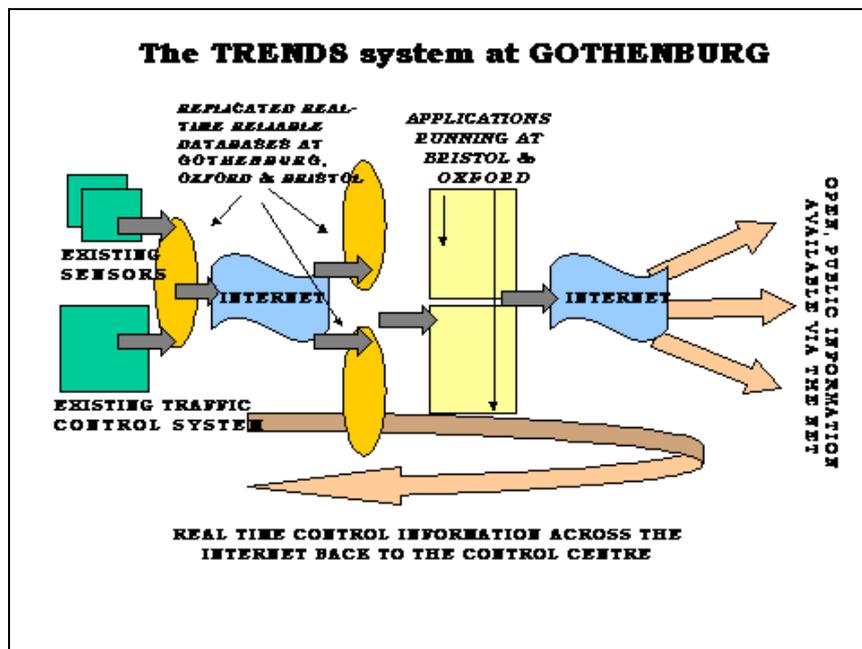


Figure 2.7: Overview of the TRENDS demonstrator

2.2 Quantifying Urban Road Traffic Congestion

Most of the definitions of congestion that appear in the literature use the idea that congestion is a state which occurs when the capacity of a network and its various links or nodes is exceeded in some way [Evans and Bell, 1996]. One definition [Pignatoro et al., 1975] defines “congested operations” as:

“...the entire range of operations which may be experienced when traffic demand approaches or exceeds, or both, the capacity of the signal.”

2.2.1 Queue-Based Measures

In order to measure or detect the presence of urban road traffic congestion, an attempt must first be made to define it. Various field studies [Pignatoro et al., 1975] found that

“queue-based measures were the most meaningful indicators of congestion”. In a similar manner to computer networks it was found that the congestion problem could be focused on one or more critical intersections whose capacity had been exceeded. These critical intersections would then affect several upstream intersections thus resulting in the appearance (and fact) of area-wide congestion. Table 2.1 [Pignatoro et al., 1975] provides some finer definitions of congested operations:

UNCONGESTED	CONGESTED		
	SATURATED		OVERSATURATED
	STABLE	UNSTABLE	
No queue formation	Queue formation, but not growing Delay effects are local	Queue formation and growing Delay effects are still local (This is a transient state and may be of short duration)	Queue formation and growing to a point where upstream intersection performance is adversely affected

Table 2.1: Differing forms of congested operations

From Table 2.1 it can be seen that saturated operations are defined to be those congested operations that are characterised by the formation of local queues. Here a local queue is one that only affects the performance of the junction at which it forms. Oversaturated operations are defined to be congested operations that are characterised by the formation of queues that adversely affect the performance of at least one major upstream junction. Several other authors [Longley, 1968] have also drawn a distinction between local and non-local congestion. One paper [Evans and Bell, 1996] uses data obtained from the Split Cycle Offset Optimisation Technique (SCOOT) to create congestion measures that help with the identification of recurrent (frequently occurring) congestion. Unfortunately, some of the data used in that study, such as queue length and a special measure called “SCOOT Congestion”, is not available in the SCATS UTC. In fact, because SCATS only uses one detector at the stop-line it does not have any method of directly detecting queue length. However, one paper [Shibata and Yamamoto, 1984] shows that there is a relationship between vehicle speed and queue length. This paper states that it was “experimentally confirmed that for each detector there exists a constant relationship between the space-mean speed and the queue length”. A formula for measuring the space-mean speed is given as:

$$\text{"space - mean speed"} = \frac{\text{traffic volume} \times e}{\text{vehicle presence time}} \text{ (m/s)}$$

where:

e = mean vehicle detection length (metres) obtained by adding mean vehicle length (determined by traffic survey) and effective detection area

This formula is more sensitive to congestion than just pure detector occupancy data as the space-mean speed includes both traffic volume and vehicle presence time. The paper notes that one detector can only effectively estimate queue lengths within a 200-metre range on either side. If the queues could potentially be longer than this, more than one detector should be used.

2.2.2 Image Processing

One common element between all the congestion measurement techniques mentioned so far is the fact that they all attempt to infer much global information from one or two local detectors. Such methods are inevitably time consuming and in many cases inaccurate. It would be preferable if congestion measures could be obtained from a sensor that has a wider, and therefore more accurate, “view” of the scene. Traffic cameras coupled with image processing could provide such a sensor. One paper [Aubert, Bouzar, Lenoir and Blosseville, 1996] provides an example of how to make queue measurements using such a system. A drawback of such systems is that they need to be capable of providing accurate information regardless of weather and other environmental conditions such as lighting. Another paper [Bouzar, Lenoir, Blosseville, and Glachet, 1996] show how this problem can be overcome by the use of suitable road markings that are painted onto the road. These markings can be easily detected by the camera and thus aid in reducing the effects that environmental conditions have on the image processing. Another problem with image processing systems was discovered [Diaz, Ferris, Caverro, Martinez, Guillen, and Fuertes, 1995] in the context of vision based incident and congestion detection systems. The problem encountered is that the image processing systems have to be separately calibrated for each implementation and that the configuration of such systems is not straightforward. The overall impression from papers such as those mentioned above is that image-processing systems will have a very important role to play in future road traffic detection systems but at present they are still in the experimental stage.

2.2.3 Intersection Delay

A measurement that is complimentary to congestion analysis is that of intersection delay. When referring to adaptive UTC systems, intersection delay is considered to be unrelated to queue length as such a system can adapt (e.g. increase cycle length) at any given time to clear longer queues in less time. One paper [Wolshon and Taylor, 1999] details a study that was carried out to compare intersection delay under SCATS to the delay experienced when signal timings were changed once per hour. To achieve this SCATS output files were generated and fed into a simulation package that used the delay calculations found in the Signal Operations Analysis Package (SOAP). The delay calculation used by SOAP is a modified version of the Webster algorithm [Webster and Cobbe, 1966] that is shown below.

$$d = \frac{C(1-\lambda)^2}{2(1-\lambda X)} + \frac{X^2}{2q(1-X)} - 0.65 \left(\frac{C}{q^2} \right)^{1/3} \cdot X^{(2+5\lambda)}$$

where:

d = average delay per vehicle (seconds /vehicle)

C = cycle time (seconds)

λ = proportion of the cycle that is green for a particular phase = (g/C)

q = flow (vehicles / hour)

X = degree of saturation = ($q / \lambda s$)

s = saturation flow (vehicles per second of green)

The modified version of this formula that is used by SOAP is called the Robertson equation [Robertson and Grower, 1977]:

$$D_2 + D_3 = \left[\left(\frac{2(1-X) + Xz}{4z - z^2} \right)^2 + \frac{X^2}{4z - z^2} \right]^{1/2} - \left(\frac{2(1-X) + Xz}{4z - z^2} \right)$$

where:

$z = ((2X) / v) \cdot (60 / T)$

v = approach volume (vehicles / hour)

T = period length (minutes)

The purpose of this modification was to more accurately assess delay for traffic volumes that approach or exceed the design capacity of the intersection. The Webster equation can analyse approaches with degrees of saturation between approximately 15% and 97%. If the degree of saturation exceeds 97% then the Webster equation

produces negative delay values that do not return to being positive until the degree of saturation exceeds 130%.

2.2.4 Travel Time

Presenting road traffic congestion information to an end-user undoubtedly provides a useful service. Such information is useful because it offers a guide to the user as to which roads are congested and should be avoided if possible. However congestion information could also be considered as secondary information that only *aids* the user in making an informed decision as to the “best” route available. Most users would generally define “best route” to mean the quickest route using a particular mode of transport. The primary information that the public is therefore looking for is travel time, i.e. “how long will it take to reach my destination?”

Two basic methods can be used to estimate vehicle travel time. The first approach is to uniquely identify a vehicle and to either measure the time that it takes to travel a particular route or the time it takes to travel between two set points. The second method used is to measure average vehicle speeds at certain points along a route. From this information the travel time can then be approximated.

Two basic methods of vehicle identification [Papageorgiou, p.410, 1991] are license plate readers and automatic vehicle identification (AVI). License plate readers use CCTV cameras and image processing techniques to automatically identify each vehicle’s license plate as it passes by the camera. AVI systems essentially comprise of three functional elements: a vehicle-mounted transponder, a roadside reader unit with its associated antenna and a system for the transmission, analysis and storage of data. The main methods that AVI uses for the transmission of data are optical, infrared, inductive-loop, radio frequency and microwave systems.

Single loop detectors similar to those of the SCATS system can be used to estimate travel times [Petty, Bickel, Jiang, Ostland, Rice, Ritov and Schoenberg, 1998]. Unfortunately the methodology used is based on a stochastic model of freeway traffic flow and is therefore unsuitable for use on urban roads such as those in Dublin. Another study [Cherrett, Bell and McDonald, 1996] compared the measured journey times by collecting data from both inductive loops and CCTV cameras over a half-mile section of urban carriageway. A three-stage feed-forward neural network was used to train the system to recognise number plates from the video images. It was

found that the neural network approach tended to “slightly over predict journey time in free flow conditions but was accurate in queuing situations in which the mechanistic [inductive loop] approach tended to over predict.”

All of the techniques mentioned above for measuring travel time would require the installation of extra equipment around Dublin. As this was not a viable option for this project, travel times will not be directly utilised in this project. An interesting point to note however is that at the time of writing Dublin Bus are currently trialing a GPS system that will allow estimation of travel times over particular bus routes. It is possible that this information could be made available to future projects.

2.3 Website Usability

According to the New York Times the “guru of web page usability” is Jacob Nielsen. The reason for this accolade is probably more to do with the fact that Nielsen is one of the few people who grasped and articulated the importance of *usable* website design early on rather than any momentous ideas that he has produced. Nielsen [Nielsen, 2000] states that there are “two basic approaches to design: the artistic ideal of expressing yourself and the engineering ideal of solving a problem”. This dissertation attempts to concentrate firmly on the side of engineering.

One book [Nielsen, 1993] shows how the idea of usability has been incorporated into the design of user interfaces for both software and hardware systems for many years. It is well understood that usability design can have large benefits for safety, as well as ease of use and economic benefits through timesaving. For these very reasons large amounts of time and effort are put into the design of aircraft cockpits, operating system GUI’s, monitoring system GUI’s, etc.

Large-scale public use of the Internet is still a relatively new phenomenon. On the web, designers have rushed to explore the new medium with the result that end-user’s usability requirements more often than not got left behind. Many new techniques have been integrated into website design simply because they existed rather than for any sound practical reasons. This attitude is now rapidly changing as the initial “gold rush” mentality fades and the realisation dawns that the public will only use a particular technology or website if it is convenient, easy to use and offers well defined advantages over more traditional approaches.

Nielsen divides web usability into five main areas: page design, content design, site design, accessibility for users with disabilities, and international user requirements. Usability studies were then used to help expand each of these topics.

Page design is concerned with the layout of individual pages within a website. Simplicity should be the key goal of page design. Users rarely access a site to enjoy the design; instead they prefer to focus on the content. It is very important that page designs work across a wide range of platforms and that they can be accessed by people who use older technology. Pages have to be designed so that they will work on many different types and sizes of displays as content negotiation between clients and servers is not yet widely available. Nielsen recommends that all pages should work on two-year-old browsers, plugins and other software. Page response times should never be longer than ten seconds as this is the limit for keeping a users attention focused upon a particular dialogue. These restrictions may appear unduly harsh but as the graph below [Nielsen, 2000] shows, failure to observe them may mean alienating up to ten percent of the sites potential user base before they ever get to see any content.

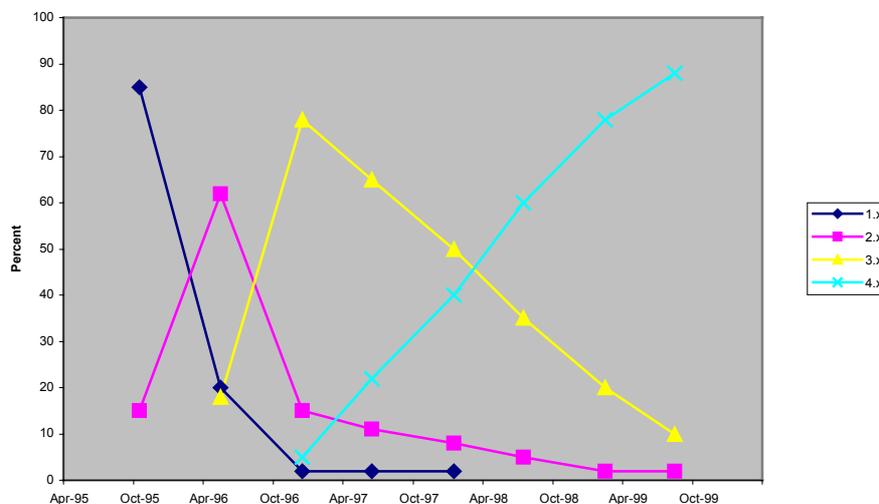


Figure 2.8: Relative proportion of Netscape users using different versions of the browser over time

Content is the focus of the web user's attention and is the first thing that is looked at when a new page is loaded. Nielsen asserts that "content is one of the two most important determinants of web usability, the other being whether users can find the page they want". Content such as beautiful pictures are less important than for traditional media. Web users are very goal driven and impatient. Therefore content needs to be orientated towards providing fast answers and useful information.

The main aim of site design is to allow a new user to quickly navigate through a site so as to find the desired information. This is achieved through the use of search engines, appropriate site navigation tools and a unified website look and feel so that the user does not have to rediscover the navigation tools on each new page. At present the “dominating web user experience is that *on the average, you are on the wrong page.*” Appropriate site design should aim to reduce this wasted time to a minimum.

In the US alone there are over *thirty million* people who have problems that make it difficult for them to use traditional input and output devices in the way they were intended. Assisting these users is not just good citizenship for in many countries there are legal obligations to facilitate intranet use by disabled employees. A simple way to check for many accessibility problems is to access a website in a text-only browser. If a site works well in such a browser then it will probably be reasonably accessible for many users with disabilities. It is also good to look at all graphics in a greyscale version so as to check their accessibility for users who may be colour-blind. International user requirements should also be borne in mind. Website design should at least be conscious of the fact that the rest of the world exists outside of the local audience. The main items of difference are of course language and culture. It could be said that a road traffic congestion monitoring system would have a very limited domestic audience however if it is well designed and implemented then it is certain to be visited by foreign researchers. An example of this is the Athens congestion map which provides an English, as well as Greek, language option.

Nielsen concludes that the four main reasons why users return to some websites and not to others can be summarized by the following acronym H.O.M.E.

- High-quality content
- Often updated
- Minimal download time
- Ease of use

Chapter 3 Urban Traffic Control Systems

A rapid increase in urban traffic during the last quarter of the 19th century generated the need to impose order on the movement of vehicles. This resulted in the first gas-powered traffic signals being installed in London circa 1870. Large growth in the use of traffic signals did not occur however until the first electrically powered signals appeared in 1926 in Wolverhampton, England. All of these early signals operated using a method known as fixed-time operation where each intersection was allocated a fixed period of “green time”. It was soon realised that a system capable of adjusting to different traffic volumes would be far superior to the fixed-time approach. This realisation led to the development of the so-called vehicle-actuated mode of operation that is still widely used today, particularly in suburban areas. This system relies on detectors that are capable of detecting the difference between traffic volumes on each approach to the intersection. Using this data the green-time given to each approach on an intersection is extended or reduced depending on the relative traffic volumes at that approach. In the 1950’s and 1960’s pads and pneumatic tubes were used to detect the presence of vehicles. Today the technology used is based on inductive loops that were first introduced in the 1970’s. The next phase in the development of traffic signalling occurred with the introduction of UTC systems. This idea was based on the fact that where traffic signals were in close physical proximity better vehicle throughput could be obtained by linking the lights together on a fixed-time plan controlled by computer. This time plan would then change according to different times of the day (e.g. there would be separate “morning peak”, “evening-peak”, “off-peak” and “night-time” plans). The next and most current approach is the adaptive UTC. This is the system used by SCATS and it works by combining the benefits of both fixed-time UTC and vehicle actuation.

3.1 General Traffic Terminology

Figure 3.1 provides an illustration of many common traffic terms that are used throughout this dissertation. This example shows how the sequence of light changes at a particular intersection can be broken up into two distinct phases.

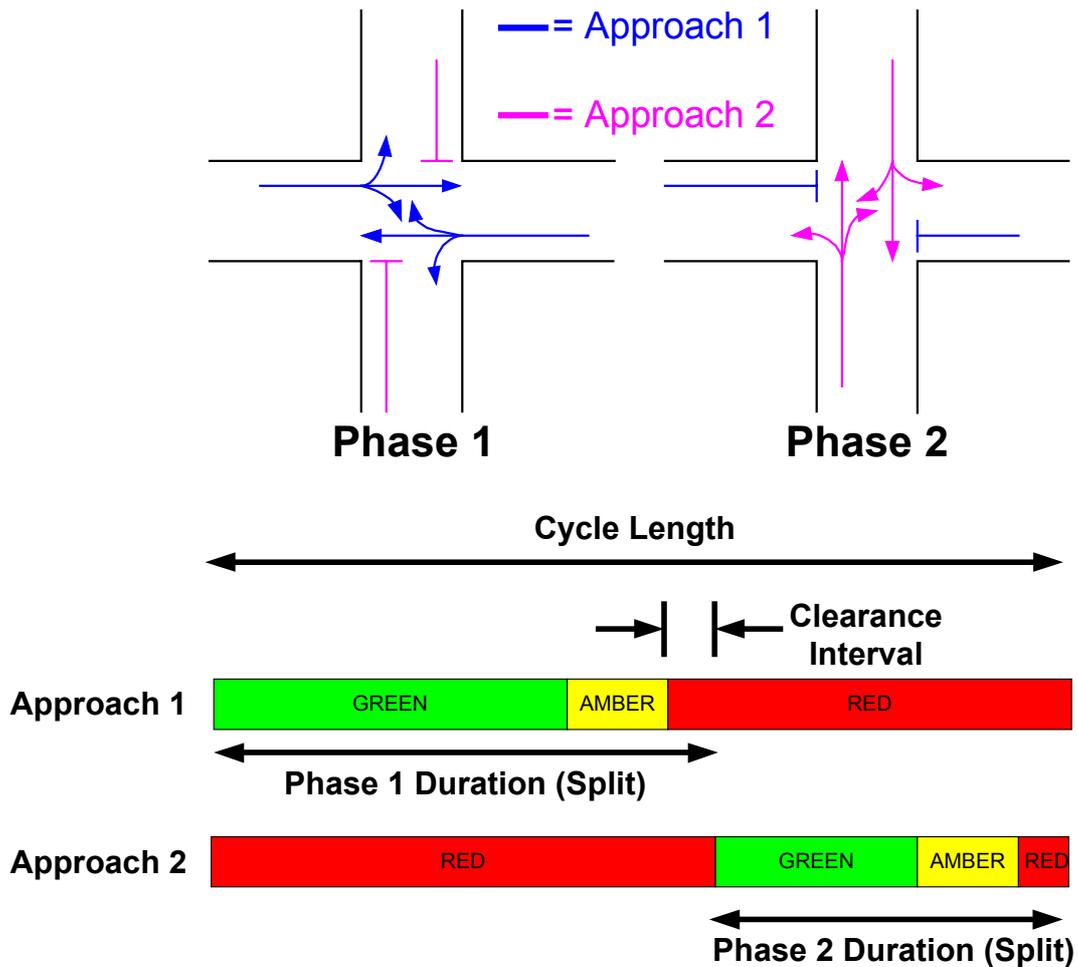


Figure 3.1: Example phase scheme at a right-angle intersection

Using Figure 3.1 as a guide the following terms [Shibata and Yamamoto, 1984] are now defined as they will be used for the remainder of this dissertation:

- **Approach:** is a single direction of traffic flow that leads to an intersection.
- **Phase:** is one sequence of traffic movements that are allowed during one portion of the cycle length.
- **Cycle Length:** is the time duration that is required to complete one cycle of all the required phases for an intersection.
- **Split:** is the proportion of the cycle length that is assigned to one particular phase. The split is represented as a percentage of the cycle length.
- **Offset:** is the time difference between the points at which the green indications are initiated between specified (usually adjacent) intersections.

3.2 UTC Architectures

Many different methods have been applied to the basic layout and structures of UTC systems throughout the world. All of the architectures used can be grouped into one of the following four categories [Dion and Yagar, 1996]:

- Fully Centralised
- Fully Distributed
- Centralised Hierarchical
- Distributed Hierarchical

Each of these different techniques will now be examined in further detail.

3.2.1 Fully Centralised

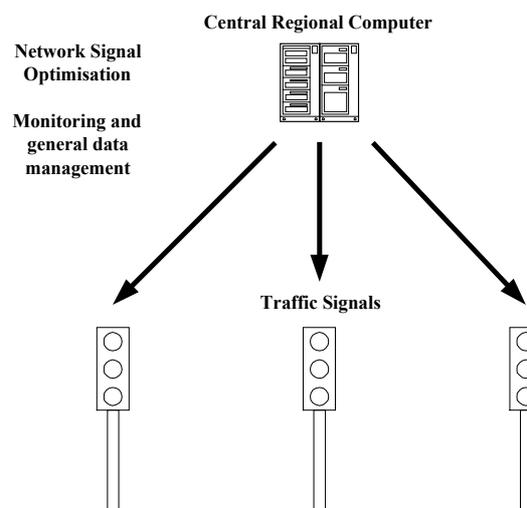


Figure 3.2: Fully centralised control

In the centralised approach a central computer performs all the timing calculations. Centralised control focuses on network coordination. This approach has the advantage that it enables a system-wide optimisation of all timing parameters due to the fact that all the data required for such calculations is made available at a single location (i.e. the central computer). It also facilitates network-wide flow management strategies such as traffic re-routing, gating, etc., to combat large-scale congestion. A drawback of fully centralised systems is that the central computer has too many input variables to be able to simulate and thus optimise the entire system in real-time. This type of system deliberately neglects random and cycle-to-cycle fluctuations to ensure a certain stability in the overall signal timing process. It can therefore be said that the

main control philosophy of the system is to react to time-of-day and long-term changes in traffic demand. An example of a UTC that uses the fully centralised approach is SCOOT. A more in-depth review of SCOOT is carried out further on in this chapter.

3.2.2 Fully Distributed

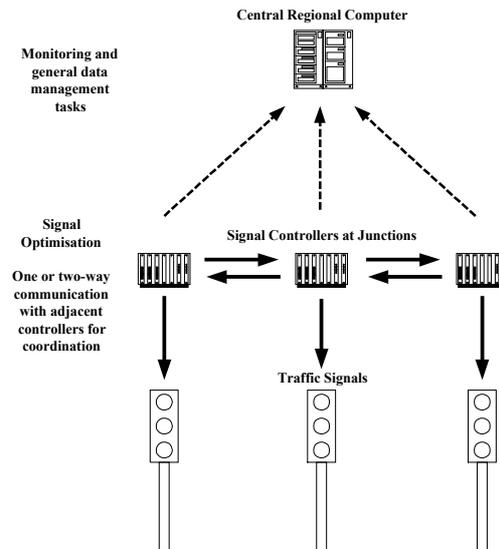


Figure 3.3: Fully distributed control

In fully distributed systems the signal controllers carry out all the timing calculations. The purpose of the central computer is to carry out data monitoring and data management so that human operatives have some idea of what the system is doing at any given moment in time. The central computer does not perform any network coordination calculations. The advantage of this approach is that it resolves the large-scale optimisation problems that are faced by the fully centralised systems that were described in the previous section. This means that the signal optimisation problem becomes independent of the size of the controlled network and allows each controller to spend more time on signal timing calculations. Although each intersection is considered as an independent optimisation problem, inputs from adjacent signals can be used to improve network-wide performance. Systems that implement the fully distributed approach include PRODYN [Henry and Farges, 1989], as well as the OPAC [Gartner, 1983] and SPPORT [Yagar and Han, 1984] models.

3.2.3 Centralised Hierarchical

Hierarchical systems try to combine centralised and distributed principles. An attempt is made to perform network-wide coordination whilst retaining some form of local adaptability to improve the overall control strategy. Depending on which elements (network coordination or local intersection adaptability) dominate the signal optimisation, hierarchical systems can be classified as being either centralised or distributed.

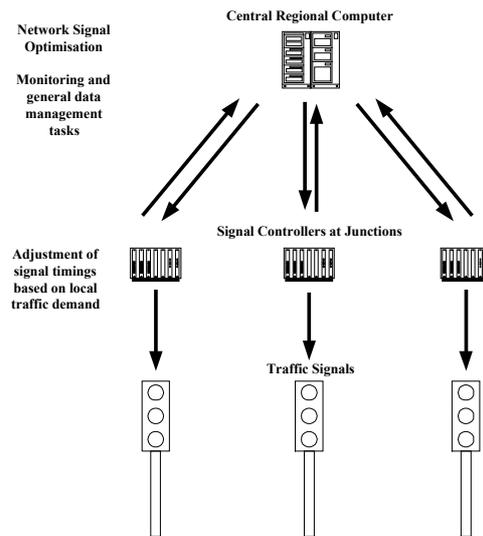


Figure 3.4: Centralised hierarchical control

SCATS is considered to be an example of a two-level centralised hierarchical system as it is the central computer that plays the most important role in the generation of signal timings. It should be noted that the regional computers in SCATS are the equivalent of the central computer described here. The central (or regional) computer determines the timings for each intersection based on the prevailing traffic conditions inside each predefined subsystem. A subsystem (see Chapter 3) is a group of intersections that have some of the same timing attributes (e.g. cycle length). Subsystems tend to be created so as to allow traffic to progress more quickly through a series of intersections. Within the centralised hierarchical system, local controllers at each intersection can make some modifications to the timing plans sent from the central computer. These modifications are limited to actions such as decreasing green time or skipping a phase altogether on non-major major approaches where there is reduced demand or no demand.

3.2.4 Distributed Hierarchical

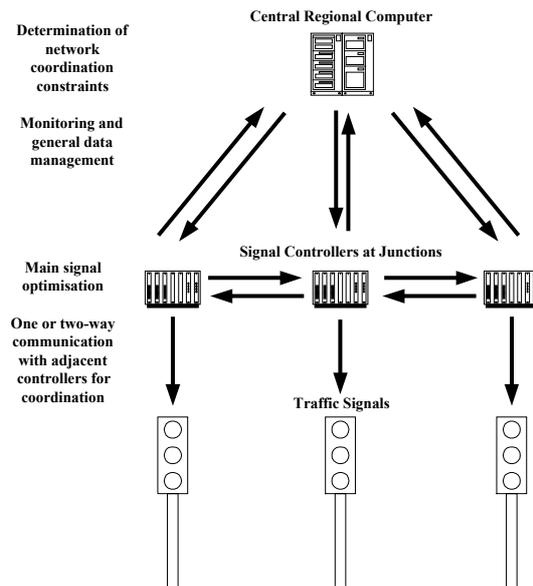


Figure 3.5: Distributed hierarchical control

An example of a system that utilises a distributed hierarchical structure is UTOPIA [Mauro, 1991]. Like SCATS, UTOPIA is a two level distributed system. The central or regional computer still produces the signal timings but unlike SCATS the local controllers can completely override these timings if they so wish. Each controller calculates its own optimum timing plan using and only uses the results generated by the central computer as an initial solution. The reasoning behind such a system is that UTOPIA was designed with the explicit objective to give priority to detected transit vehicles that are assumed to travel on separate lanes (e.g. bus lanes). The system also has to be capable of responding to quickly changing traffic conditions. Direct communication between adjacent controllers is used to detect and act upon local oversaturation.

3.3 SCATS

SCATS was first developed in the early 1970's [Lowrie, 1982] by the Department of Main Roads, New South Wales, Australia and was first installed in Dublin, Ireland in 1989. SCATS is an adaptive UTC system. This means that it operates in real-time to adjust traffic signal timings throughout the system. SCATS tries to optimise pedestrian, public transport and general traffic flows throughout the network in response to any variations in traffic demand. At the time of writing this dissertation there were a total of five hundred and seventeen signalised intersections throughout

the Dublin area of which more than two hundred and sixty are controlled by SCATS. Between the years 2000 and 2001, Dublin Corporation hope to have extended the number of SCATS controlled intersections to nearly four hundred.

3.3.1 Physical Layer

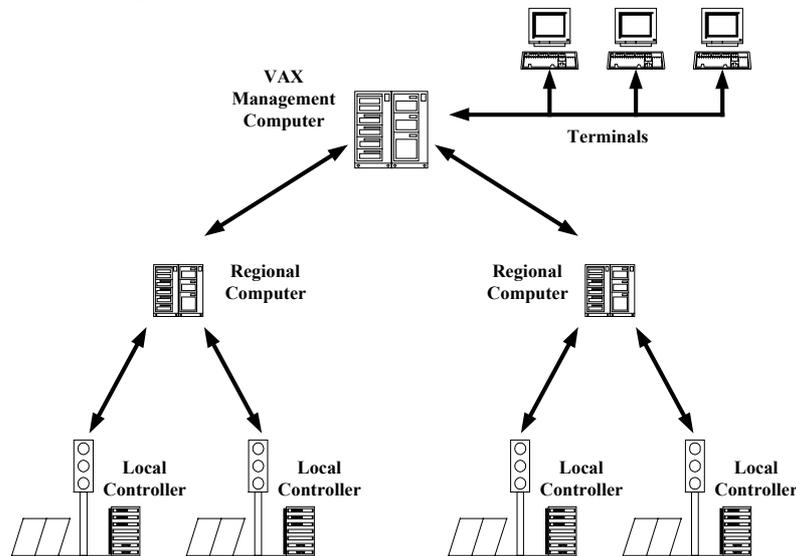


Figure 3.6: SCATS physical architecture

Figure 3.6 [Sims and Dobinson, 1980] illustrates the basic hierarchy of the SCATS system. At each intersection there is an inductive loop buried in the ground immediately before the stop-line of each of the main traffic lanes. These inductive loops are connected to the local controller that is situated at the intersection. The local controller is responsible for:

1. Sending the data collected by the loops to the regional computer.
2. Receiving the timing data for the lights back from the regional computer.
3. Implementing the light changes at the intersection.

In Dublin there are five regional computers. The purpose of each of these regional computers is to analyse the traffic data from up to one hundred and twenty local controllers that are geographically related. The regional computer assesses this data and calculates the most appropriate cycle lengths, splits and offsets for each of the local controllers that are within its jurisdiction. The results of these calculations are then sent back to be implemented by each of the local controllers. The purpose of the management computer is to provide centralised monitoring of system performance and equipment status. It also provides for remote access to regional computers by

workstation operatives who can manually adjust the settings that are sent to the local controllers.

3.3.2 System Operation

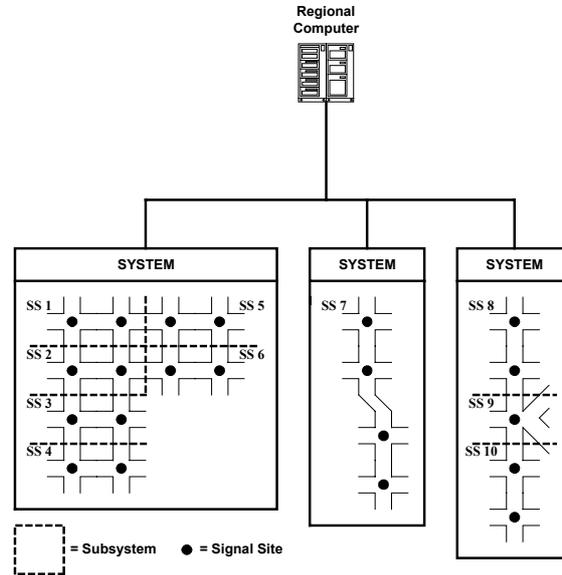


Figure 3.7: Systems and subsystems [Sims and Dobinson, 1980]

The hierarchy of control that SCATS uses to calculate the timings of the traffic lights is separate and distinct from the hardware hierarchy that is described in the previous section. For control purposes all of the intersections under SCATS control are grouped into what are called systems and subsystems. Subsystems normally consist of intersections that are physically located very close to one another and as such it is easier to treat them as single “traffic entities”. Another example of a subsystem might be an intersection that has a pedestrian crossing located nearby. For many traffic conditions these subsystems will run independently of each other (i.e. their timing plans will not be in any way linked). As traffic conditions demand, the subsystems “marry” with adjacent subsystems to form a number of larger systems or one large system.

The data for each subsystem specifies the minimum, maximum and optimum cycle lengths. Four background plans are also stored in for each subsystem (e.g. morning peak, evening peak, etc). Cycle length and the appropriate plan are selected independently of each other to meet the current traffic demand. For this purpose a number of detectors within each subsystem are defined as strategic detectors; these are detectors at key approaches to the intersections within the subsystem. Important directional traffic flows are defined by groups of strategic detectors that are called

Strategic Approaches (SA's). The numbering convention for strategic approaches is odd numbers for those that experience heavy morning traffic and even numbers for those that experience heavy evening traffic.

There are up to four linking plans for each subsystem. The purpose of these linking plans is to define the conditions for marriage with other subsystems. While a number of subsystems are linked together they each have the same cycle length. This cycle length is equal to the longest cycle length of any of the subsystems within the system. When a married subsystem no longer meets the conditions of the linking plans then the subsystem "divorces" itself from the system and reverts back to operating independently.

3.3.3 SCATS Traffic Measurements and Calculations

Two basic forms of traffic data are sensed by each of the detectors and sent by the local controller to the regional computer at the end of every cycle. This data takes the form of:

- The number of gaps (where nothing is detected) that occurred between the vehicles and the total non-occupancy (or space) time that occurred during the lane's green time. Non-occupancy or space-time is the amount of time (in seconds) during a lane's green time that the detector has no vehicles travelling over it.
- The phase time for the lane plus any remaining or unused phase time. Remaining or unused phase time can occur if a local controller decides to end a phase prematurely, due for example, to a lack of available vehicles which wish to pass through the green light.

From this raw data three fundamental values that are needed by SCATS for each lane can be calculated. These values are:

1. Original Volume (VO).
2. Degree of Saturation (DS).
3. Reconstituted Volume (VK).

The algorithms [Lowrie, 1982] used to create these values are described in further detail within the next sections.

Original Volume (VO)

This is the actual vehicle count for one lane over the period of one cycle. It is calculated by adding 1 to the number of spaces occurring during the green time. One is added to account for the vehicle sitting on the detector when the light turns green. VO is needed to calculate the values for DS and VK.

$$VO = n + 1$$

where:

VO = Original Volume (measured in vehicles)

n = number of spaces counted during green time

Degree of Saturation (DS)

In essence, degree of saturation is the “ratio of the effectively used green time to the total available green time”.

$$DS = \frac{NF[g - (T - t.n)]}{g + r} \equiv DS = \frac{NF(g')}{g + r}$$

where:

DS = Degree of Saturation (percent)

NF = DS bias factor (weighting factor)

g = phase time actually given to the lane (seconds)

T = Total non-occupancy (space) time (seconds)

t = space time which is unavoidably associated with each vehicle (seconds)

r = remaining (or unused) phase time (seconds)

g' = effectively used green time (in seconds) = $g - (T - t.n)$

In order to determine the effectively used green time, the time while the light is green during which no vehicles cross the stop-line ($T - t.n$) must be subtracted from the total available green time (g). It should be noted however, that not all the space crossing the stop-line is usable as each vehicle has associated with it a space, which is a function primarily of speed, and which cannot be zero. This fact explains the use of ($T - t.n$).

(t) is the standard space-time at Maximum Flow (MF) which can also be stated as the extra space that is unavoidably associated with each vehicle. Maximum flow is the greatest number of vehicles that was ever recorded passing over the detector during its green time and is measured in vehicles per hour. Standard space-time at MF is self-calibrated daily by the formula:

$$t = \frac{3600}{MF} - \frac{KP}{100}$$

where:

MF = Maximum flow ever recorded for the lane (vehicles per hour)

KP = Average occupancy recorded per lane during max flow (percent)

When viewing the result of a DS calculation the following conclusions can be drawn:

1. If $DS < 1$ then the traffic flow is said to be less than saturated.
2. If $DS > 1$ then the traffic flow is oversaturated. This is caused as a result of the flow across the stop-line being restricted by congestion beyond the stop-line and hence vehicles are closer than the standard space-time at maximum flow (t).

Reconstituted Volume (VK)

Reconstituted volume is a measure of how many cars *should* have passed over the stop-line for a lane's degree of saturation at that point in time.

$$VK = \frac{DS.g.MF}{3600}$$

where:

VK = Reconstituted Volume (measured in vehicles)

A verbal version of this formula is:

"VK = degree of saturation x phase time x vehicles per second at maximum flow"

VK is useful when compared to VO. The ratio of VK / VO can be written as the ratio of expected throughput to actual throughput for a lane. In normal operation this ratio should be approximately equal to one. Whenever this ratio is significantly greater than one it indicates that fewer vehicles got through the intersection than would normally be expected. This can therefore help to provide an indication of congestion.

SCATS Data Smoothing and Damping

The values of VO, DS and VK can occasionally fluctuate quite significantly (e.g. due to emergency vehicles passing through an intersection). It is therefore important that some form of averaging (or hysteresis) is performed on the data before it is used by

SCATS. This process takes the form of weighted averages over a period of three cycles as described below:

$$AVO = 0.45(VO') + 0.33(VO'') + 0.22(VO''')$$

$$ADS = 0.45(DS') + 0.33(DS'') + 0.22(DS''')$$

$$AVK = 0.45(VK') + 0.33(VK'') + 0.22(VK''')$$

where:

AVO = Average Original Volume (vehicles per cycle)

ADS = Average Degree of Saturation (percent)

AVK = Average Reconstituted Volume (vehicles per cycle)

VO' = Original Volume for this cycle

VO'' = Original Volume for the previous cycle

VO''' = Original Volume for second-last cycle

etc.

SCATS Congestion

Table 3.1 shows a range of congestion conditions that are defined by SCATS [RTA, 1999].

Conditions	Meaning
$CL < SCL$ AND $RL < XCL - 5$	Low CL, light traffic
$XCL > CL \geq SCL$	Medium CL, light traffic
$CL \geq XCL$ AND $RL < XCL - 5$	High CL, medium traffic
$CL < SCL$ AND $RL \geq XCL - 5$	Low CL, heavy traffic
$CL \geq XCL$ AND $RL \geq XCL - 5$ AND $VK / VO \leq 2.4$	High CL, heavy traffic
$CL \geq XCL$ AND $RL \geq XCL - 5$ AND $VK / VO > 2.4$	High CL, congested traffic

Table 3.1: SCATS congestion conditions

where:

SCL = stopper cycle length (a minimum cycle length that is triggered by specified vehicle volumes)

XCL = stretch cycle length.

RL = recommended cycle length

Stretch cycle length (XCL) is the cycle length up to which all stages gain additional cycle length. Above XCL only the most important phase (which is designated the stretch phase) gets the benefit. This is illustrated by the “stretch effect” shown in Figure 3.8.

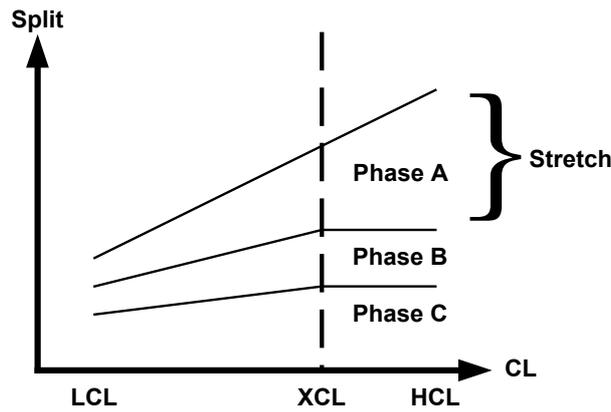


Figure 3.8: Illustration of the stretch effect

where:

LCL = Low Cycle Length (minimum CL possible)

HCL = High Cycle Length (maximum CL possible)

3.3.4 System Monitor Display

The System Monitor (SM) display is the basic method of viewing the data from a subsystem (SS) within SCATS. An SM display consists of a header, column title, strategic approach (SA) data and link (LK) data. An example of a typical output from this display is given below along with a breakdown and explanation of its various components. Data that is not relevant to this project is shaded in grey and is included for informational purposes only.

08:02	SS24M+	PL7#3	PV	6.2	CL^	98-03	RL	93'	SA	12	DS118							
INT	SA/LK	PH	PT!	DS	VO	VK!	DS	VO	VK!	DS	VO	VK!	DS	VO	VK!	ADS		
637	S	3	AB	69!	22	6	7!	52	17	17!	25	9	9!			-	-!	52
11224	S	6	A	104!	3	1	1!	47	25	26!	34	15	17!			-	-!	67
637	S	7^	4	60!	34	4	7!	58	12	14!	49	11	13!			-	-!	53
637	S	8#	C	39!	58	9	10!									-	-!	16
637	S	54*	B	28>	105	14	16!									-	-!	105

Figure 3.9: Sample SM display

System Monitor Header

The first line of the display is the SM Header, i.e.

08:02 SS24M+ PL7#3 PV 6.2 CL^ 98-03 RL 93' SA 12 DS118

Data Displayed	Options	Details
08:02		Time of day - 24hr clock
SS24M+		The number of the displayed Subsystem (SS 24)
	M	The 'M' indicates that this subsystem is married to another subsystem.
	#	The 'M' will be replaced by a '#' if the SS has been prevented from marrying with a divorce lock command
	F	Both will be replaced by an 'F' if the SS has been forced to fallback mode (e.g. SS24F). Fallback mode is where an intersection operates using the plans stored on the local controller.
	+	The '+' signifies that a marriage vote occurred.
	-	A '-' shows a divorce vote
		A blank indicates an unmarried SS (either never marries or does not marry at this cycle length)
PL7#3		Shows the operating plans for the cycle.
	PL7	Is the split plan
	3	Is the link plan.
	#	The cross hatch (#) following the split plan indicates that the split plan is locked. Similarly, if a hash follows the link plan, the link plan is locked.
PV 6.2		Shows the plans voted for.
	6	The first plan (plan 6) is the split plan vote
	.	The full stop is a separator
	2	The last plan (plan 2) is the link plan vote
CL^ 98-03		The cycle length for the Subsystem (seconds). The -03 is the modification to the cycle length when married, to bring the cycle generator of this sub system to the correct offset to the cycle generator of the subsystem to which it is married.
	^	The up arrow symbol preceding the CL value indicates that the CL is locked.
RL 93'		The recommended cycle length in seconds.
SA 12		The number of the SA which produced the RL for the subsystem.
DS118		is highest lane degree of saturation from the above SA.

Table 3.2: Details of system monitor header

System Monitor Title Line

The second line of the display is the title line. It identifies the data that follows on successive lines, i.e.

INT SA/LK PH PT!DS VO VK!DS VO VK!DS VO VK!DS VO VK!ADS

The first 4 items in this line are fixed, and are:

INT SA/LK PH PT!

where:

INT is the intersection number

SA/LK is the strategic approach or link identifier
 PH is the phase number
 PT is phase time (seconds)
 ! is a separator from remaining titles

Then follows four columns identifying the lane data. An exclamation mark separates each lane. Each lane consists of a degree of saturation, original volume and reconstituted volume.

```
! lane 1 ! lane 2 ! lane 3 ! lane 4 !
! DS VO VK ! DS VO VK ! DS VO VK ! DS VO VK !ADS
```

The last item on the line refers to a SA only. It can consist of either ADS, AVO or AVK.

Strategic Approach Display in a System Monitor

These lines show all the data for each of the designated strategic approaches within a subsystem:

```
637 S 3 AB 69! 22 6 7! 52 17 17! 25 9 9! - -! 52
```

Data Displayed	Options	Details
637		The intersection number which supplied the volume data for the SA
S		Indicates SA data
3		Is the SA number. If followed by:
	Blank	SA controls cycle length and split plan selection
	#	SA controls only cycle length
	^	SA controls only split plan selection
	*	SA controls nothing (included for monitoring purposes only)
AB		Indicates the phases specified at intersection 637.
69		Phase time in seconds.
!		Lane separator
22		Percentage DS of the detector. The exclamation mark normally preceding this value will be replaced by:
	>	If the DS is between 100-199
	*	If the DS is 200 or greater.
6		Indicates an actual count (VO) of 6 vehicles
7		Reconstituted volume.
!		Lane separator.

Table 3.3: Details of strategic approach line in an SM display

The remaining figures show the DS, VO and VK values for the next 3 lanes. The presence of dashes indicates that a lane is not active. The number 52 (at end of the line) shows the ADS for the strategic approach.

3.4 Alternative UTC Systems

Although the main focus of this dissertation is on the data that can be derived from SCATS it is a useful exercise to look briefly at the some of the alternative UTC systems currently in use around the world. Using SCATS as a comparison the following section provides a brief introduction to SCOOT, which is one of the most popular alternative UTC systems today.

3.4.1 SCOOT

SCOOT stands for Split Cycle Offset Optimisation Technique and was developed in the England during the 1970's. In the absence of any field evaluation directly comparing SCATS with SCOOT, it is only possible to look at the differences in the philosophies adopted by the two methods. SCOOT makes use of the fully centralised physical architecture whereas SCATS uses a centralised hierarchical structure. Another significant difference [Hunt, Robertson, Bretherton and Royle, 1982] between SCOOT and other UTC's such as SCATS is that SCOOT uses a second set of detectors that are usually situated 50 - 300 metres before the stop-line. These additional detectors provide a count of the vehicles that are approaching the stop-line so that extra information such as queue length can be determined. The fact that the second set of detectors are located before the stop-line also means that the presence of a vehicle is detected several seconds *before* it actually reaches the stop-line. As a result of this it could be said that the SCOOT central computer has a more current snapshot of the traffic conditions than a system such as SCATS, which is only aware of the traffic *after* it has passed over the stop-line. One paper [Dion and Yagar, 1996] observes that "whereas SCATS is reactive to short-term traffic fluctuations it does not have SCOOT's predictive capability and is therefore less proactive." This statement is true insofar as SCATS only uses stop-line detectors as its primary source of data and so has no information about near-future arrivals. Both SCOOT and SCATS vary the cycle time according to the level of congestion, however the two methods differ [Luk, 1984] in the following aspects:

- SCATS can vary the cycle length after every cycle whereas SCOOT can only change cycle length at most once every 2½ minutes.
- The level of congestion in SCATS is indicated by the degree of saturation measures at the stop-line. SCOOT only *estimates* degree of saturation by

measuring the flow upstream of the stop-line and using predetermined values of saturation flow.

It could also be said that SCATS is more reactive than SCOOT to demands in the current cycle because the local controllers can fine-tune the timings that are produced by the regional computer, whereas SCOOT does not have such a facility.

Chapter 4 Architecture

Before the actual implementation phase of this dissertation commenced it was of vital importance that a robust architecture be designed. Due to the relatively short time span available for the implementation (four months) it would not be possible to start from the beginning if any major design flaws were encountered. Before any specific architecture could be considered the main objectives and constraints of the proposed system had to be clearly defined. The three primary objectives of the proposed system were as follows:

1. The extraction and conversion of SCATS data into a universally usable and extendable format.
2. The creation of an algorithm that would be capable of using the extracted data to generate a meaningful measure of road traffic congestion.
3. The presentation of all this data to a wide variety of users in a clear and meaningful format.

As in most real-world computing problems a number of constraints were present that the proposed solution had to adhere to. These constraints were:

1. The solution could not under any circumstances interfere with the running of SCATS. This meant in practise that only existing interfaces to the system could be utilised.
2. All data extraction from SCATS had to be fully automated, require the minimum of human intervention and be capable of coping with any output or situation that might be reasonably expected to arise during the normal running of SCATS.
3. The proposed architecture had to be as scalable as possible. This meant that no software limits could be placed on the number of intersections, end-users or end-user applications that could be added to the system.
4. The system configuration (i.e. the required lanes, intersections, routes, etc.) had to be “soft” (i.e. minimum use of hard-coding). The manner in which the system could be configured had to allow for straightforward set-up, configuration and maintenance.

5. The architecture had to be as modular as possible. This was specified so that some potential situations could be dealt with by swapping an old module of code for a new module without the need for a redesign or rewrite of the remaining code. The situations envisaged included:
 - a. Insertion of a new congestion algorithm module.
 - b. SCATS upgrades. This would potentially mean that the data extraction module would have to be altered.
 - c. Addition of extra end-user application modules that can utilise the data from SCATS and the congestion algorithm.

4.1 Three Tier Approach

In order to cater for all the issues mentioned in the previous section the architecture shown in Figure 4.1 was devised.

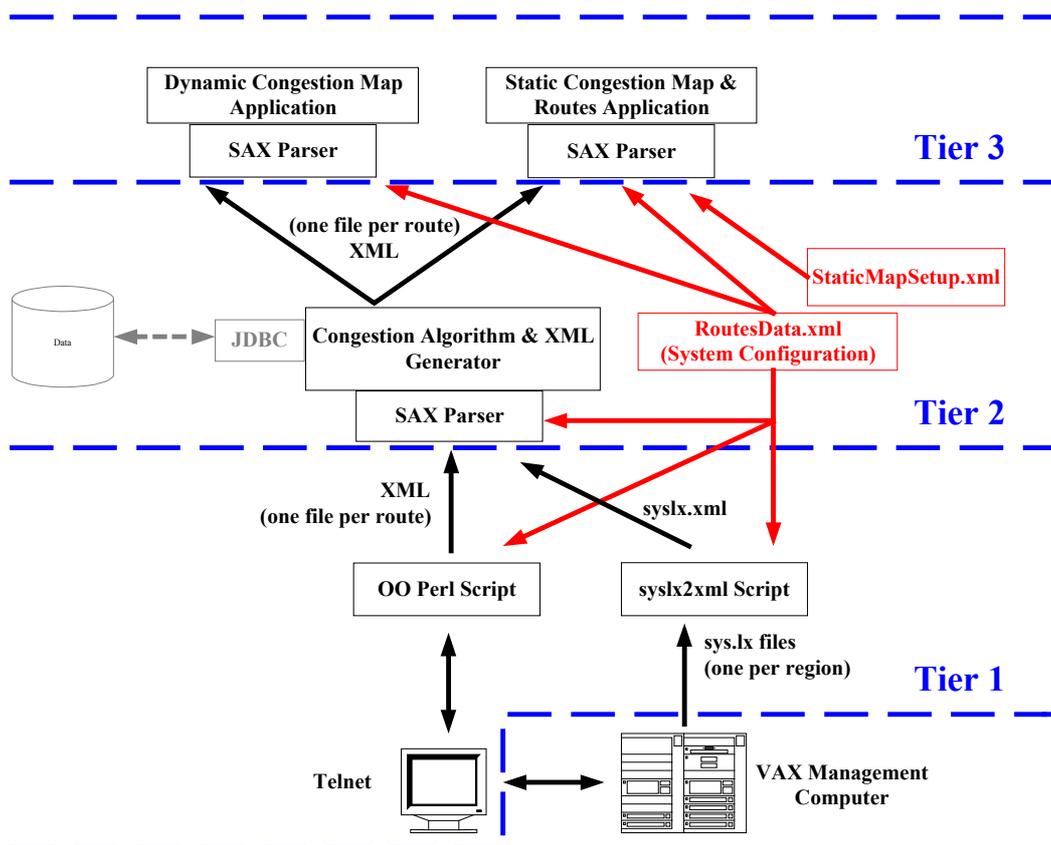


Figure 4.1: Three-tier architecture

4.1.1 Tier 1

The purpose of the first tier is to achieve the first objective, i.e. to extract the data from SCATS and to export it into a usable format. After discussions with the SCATS system administrators it was decided that the best method of interfacing with SCATS would be through the use of an automated telnet session. The advantages of this were threefold:

1. A telnet session would appear to SCATS as just another user so the potential extra load from polling the intersections would be within the SCATS design parameters.
2. Tier 1 would be treated as a standard user on SCATS therefore making it possible to be assigned a very low level of SCATS privileges (i.e. read access only). This greatly decreases any potential security risks.
3. The data is extracted from SCATS in plain text using the SM display (discussed in Chapter 3). This fact that the data is in plain text simplifies any subsequent data manipulation that has to take place.

The initial idea of using a Perl script to translate the SCATS data into XML came from a final year project [Brennan, 2000] where it proved to work extremely well. Perl contains some very powerful string manipulation methods (such as regular expressions) which help to simplify the task of breaking the data contained in an SM display into its constituent parts. Another advantage of using Perl in Tier 1 is that most of the main methods needed to operate an automated telnet session (e.g. login methods) already exist in an add-on module (telnet.pm) that is freely available on the Internet. For details on an unforeseen difficulty that arose with the use of Perl in the implementation phase of the dissertation see Chapter 5. Once all the necessary data for a route is extracted it is then passed onto Tier 2 using XML.

4.1.2 Tier 2

Tier 2 provides a solution for the second major objective (i.e. the implementation of a congestion algorithm). At first glance a simpler and faster running solution might appear to be an amalgamation of Tiers 1 and 2 in the same program. There are three main reasons as to why the second tier was designed on its own as an independent entity:

1. Security – the second tier acts a buffer between the first tier, which has access to SCATS, and the third tier which can be situated on the Internet. From a SCATS security standpoint it is much safer if the third tier cannot directly access the second tier.
2. Modularity – it was a specified constraint that the congestion algorithm and the SCATS data extractor be modular and easily swapped out if required. The existence of a second tier makes this task considerably easier.
3. Scalability – if the load on either the first or the second tier becomes sufficiently large then the two tiers can be placed on separate machines that have greater resources. This point becomes even more pertinent as at some future stage the design envisaged the addition of a database (greyed out in Figure 4.1) that would store and retrieve historical SCATS data. To have the data extraction routine, congestion algorithm and database interface all on the same tier would inevitably lead to problems in scalability.

The purpose of the SAX parser is to take the XML from Tier 1 as an input, verify that it is well formatted (see section 4.3 on DTD's below), and place the data into an appropriate object model. It would have been feasible to do without the extra overhead that is involved in using the SAX parser by implementing a customised parser. This was not done due to the far superior error handling that is already implemented in SAX. For further details on this refer to Chapter 5. Once the parser has translated the XML data into the object model of Tier 2, the congestion algorithm produces the congestion measures for the various intersections and this data is then converted into an XML file which is passed onto the third and final tier by means of placing it onto a web server.

4.1.3 Tier 3

The third tier deals with the last primary objective, which is to provide a clear and meaningful user interface. There were no real constraints placed upon the design of the third tier as the applications on this level could potentially be run on any computer that has access to the Internet. Figure 4.1 shows the two applications that were designed as part of this dissertation. The first of these is the dynamic congestion map application. This client application was designed to allow a user in any location (where Internet access is available) to retrieve all the extracted SCATS data and

congestion measures in a dynamic map format. This application can also be run as an applet without any changes to its code. The other application shown in the architecture is the static congestion map and routes application. The purpose of this is to create static JPEG images of the dynamic map that are regularly refreshed and made available for download on the web server. This application also converts the SCATS data into static graphics that show the congestion information for each route. These graphics are then also placed on the web server for download. The biggest difference between these two applications is that the static congestion map application runs locally within the control of Dublin Corporation whereas the dynamic map can be run on any machine, internal or external, provided an Internet connection is available.

4.2 Object Model

The purpose of the object model is to create a structure capable of holding and easily retrieving all of the current SCATS and congestion data within the memory of the running application. This applies to all the applications within this dissertation. All of the data required to initially set up the object model is held within the routesData.xml configuration file. To provide an indication of the amount and complexity of this file an actual routesData.xml file containing information on just ten routes runs to over 2,300 lines and is nested six layers deep (see section 4.3 for further details). It was therefore a vital task to design a suitable object model that could store and retrieve this data in as efficient a manner as possible. Figure 4.2 provides an illustration showing how the object model was designed.

This first element of the object model is a vector (or variable sized array) of routes. Each of the elements within this vector is an object-oriented class that contains all of the information on a particular route. The vector containing the routes has to be of a variable size due to the fact that the number of routes in the system is unknown until runtime when the routesData.xml file is read. Contained within each route is a further vector of intersections. Each element within this array is a class that contains all the information on a particular intersection. Within each intersection is a vector of strategic approaches. It is necessary to have the strategic approach number in order to be able to locate a particular lane within the SM display discussed in Chapter 3. Each strategic approach contains a vector of lanes and each lane contains a vector of map

coordinates. Each map coordinates class contains the coordinates of a lane relative to the static background map so that the lane can be painted onto the map in its correct position.

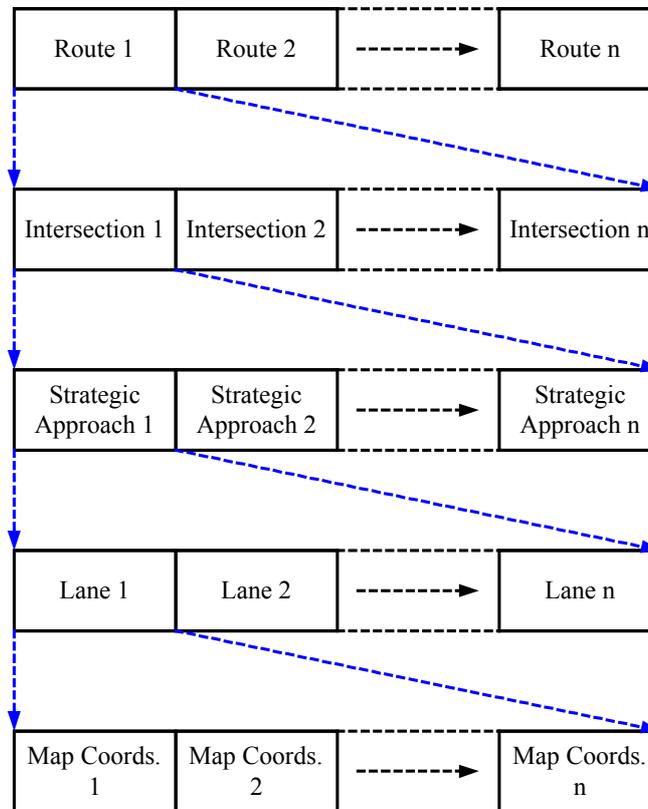


Figure 4.2: Object model architecture

The use of this type of model results in a large pyramid-type structure that can be quickly traversed in order to find any individual information element. If the location of the element is known in advance then it is simply a matter of “drilling” straight down to access it.

4.3 XML

A suitable format for transferring data between the various tiers, applications and scripts was an important design consideration that had to be quickly decided upon. Given the constraint that the overall architecture had to allow for the addition of future third-party applications it was essential that the data format used would be as universal as possible. Two possibilities that presented themselves were standard ASCII plain-text files or XML files. XML was chosen for the following reasons:

1. Portability between operating and hardware systems could have been a problem for plain-text files (e.g. different numbers of characters per line would make parsing difficult).
2. XML inherently describes and places a structure on all the data it contains and the use of the DTD makes finding any errors in the document considerably easier. This is especially important where you have large data files such as routesData.xml.

From Figure 4.1 it can be seen that there are four different XML files used throughout the architecture. Each of these XML files is described in further detail in the next section.

4.3.1 DTD Design

The purpose of a Document Type Definition (DTD) is to provide a set of rules that must be applied if an XML document is to be “well formed”. As an XML document is being parsed it is checked against its corresponding DTD (if there is one). If the format of the XML document does not match that of the DTD then an error is flagged, or in the case of the SAX parser that was used in this project, a Java exception is thrown.

routesData.dtd

The first DTD that had to be designed was routesData.dtd (shown in Figure 4.3). The purpose of this DTD is to define the overall configuration file routesData.xml which is called by every application within the architecture. This configuration file allows every application to create the appropriate sized object model and to then populate this model with some of the basic data required e.g. the identification numbers of the intersections and lanes that have to be interrogated.

```
<?xml version='1.0' encoding='utf-8'?>
<!ELEMENT routesData (details?, lastModified, route*) >
  <!ELEMENT details (#PCDATA) >
  <!ELEMENT lastModified (#PCDATA) >
  <!ELEMENT route (routeName, routeNumber, intersection*) >
    <!ELEMENT routeName (#PCDATA) >
    <!ELEMENT routeNumber (#PCDATA) >
    <!ELEMENT intersection (intersectionNumber, intersectionName,
                           intersectionCoordinates, approach*) >
      <!ELEMENT intersectionNumber (#PCDATA) >
```

```

<!ELEMENT intersectionName (#PCDATA) >
<!ELEMENT intersectionCoordinates (#PCDATA) >
<!ELEMENT approach (strategicApproachNumber, lane*) >
  <!ELEMENT strategicApproachNumber (#PCDATA) >
  <!ELEMENT lane (smColumnNumber, detectorNumber, nextIntersection,
    lineCoordinates*) >
    <!ELEMENT smColumnNumber (#PCDATA) >
    <!ELEMENT detectorNumber (#PCDATA) >
    <!ELEMENT nextIntersection (#PCDATA) >
    <!ELEMENT lineCoordinates (#PCDATA) >

```

Figure 4.3: routesData.dtd

Five of the tags (routesData, route, intersection, approach and lane) are there to encapsulate the other data within the file. They are equivalent to the names of each of the overall vectors shown in Figure 4.2. Each of these encapsulating tags define the information that is to be found within them. If the data definition is followed by * then this signifies that zero or more occurrences can be expected. If followed by + then one or more occurrences can be expected. If the symbol used is ? then this tag can be included or left out at will. A brief description of each of the data tags is contained in Table 4.1.

XML Tag	Description
<details>	Optional tag that allows the user to add comments to the XML file
<lastModified>	Date this file was last modified (dd/mm/yyyy)
<routeName>	The name of the route (e.g. the Quays)
<routeNumber>	Every route must be given a unique number (starting at 1)
<intersectionNumber>	Every SCATS intersection has a unique number
<intersectionName>	The name of the intersection as described on SCATS
<intersectionCoordinates>	The Cartesian coordinates describing the relative position of the intersection on the static map (X,Y)
<strategicApproachNumber>	SCATS specific SA number
<smColumnNumber>	The column number of the SA row that the lane can be found on within the SM display
<detectorNumber>	The detector number that this lane corresponds to (this is not needed by the application but helps to avoid confusion when filling out the configuration file).
<nextIntersection>	The next intersection number that will be encountered by traffic travelling on this route and in the direction allowed by the particular lane. 0 is used to signify the end of a route.
<lineCoordinates>	The Cartesian coordinates describing the map line(s) that traffic follows to get from the previous intersection to the current intersection relative to the static map (X1,Y1,X2,Y2)

Table 4.1: routesData tag descriptions

Syslx.dtd

The second XML document that needed to be defined was syslx.xml. The purpose of this file was to contain all the data that was required from the SCATS configuration

files (which are called sys.lx files). There is a separate sys.lx file for each regional computer and they contain data that is necessary for the congestion algorithm. A separate Perl script was designed to extract the necessary data from these files and translate the data into a file called syslx.xml. As the sys.lx files on SCATS are rarely altered syslx2xml need only be run once every time Tier 1 is initially started. syslx.xml is then read by the Java program in Tier 2 which uses the data in its congestion algorithm.

```

<!ELEMENT syslx (details?, lastModified, region*) >
  <!ELEMENT details (#PCDATA) >
  <!ELEMENT lastModified (#PCDATA) >
  <!ELEMENT region (regionName, SCATSVersion, intersection*) >
    <!ELEMENT regionName (#PCDATA) >
    <!ELEMENT SCATSVersion (#PCDATA) >
    <!ELEMENT intersection (intersectionNumber, subsystem) >
      <!ELEMENT intersectionNumber (#PCDATA) >
      <!ELEMENT subsystem (subsystemNumber, LCL, HCL, SCL1, SCL2, XCL) >
        <!ELEMENT subsystemNumber (#PCDATA) >
        <!ELEMENT LCL (#PCDATA) >
        <!ELEMENT HCL (#PCDATA) >
        <!ELEMENT SCL1 (#PCDATA) >
        <!ELEMENT SCL2 (#PCDATA) >
        <!ELEMENT XCL (#PCDATA) >

```

Figure 4.4: syslx.dtd

It can be seen from Figure 4.4 that there are four encapsulating tags within syslx.dtd: <syslx>, <region>, <intersection> and <subsystem>. The purpose of the region tag is to indicate which regional computer the intersection data has come from. It should be noted that an intersection can only belong to one region. The interesting feature about this DTD is that the subsystem tag is contained within the intersection tag. In SCATS (see chapter 3) intersections are grouped into subsystems, however for the purposes of this DTD, the search of the sys.lx files is done on the basis of all the intersection numbers that are present in routesData.xml so therefore the intersection tag comes first. Table 4.2 provides further details on each of the tags that have not previously been encountered. See Chapter 3 for explanations of some of the SCATS specific terms.

XML Tag	Description
<regionName>	The name of the regional computer (e.g. CCITY)
<SCATSVersion>	This is the version of SCATS that is running on the particular regional computer
<subsystemNumber>	Subsystem number to which the particular intersection is a member
<LCL>	Low Cycle Length
<HCL>	High Cycle Length
<SCL1>	Stopper Cycle Length 1
<SCL2>	Stopper Cycle Length 2
<XCL>	Stretch Cycle Length

Table 4.2: syslx tag descriptions

smData.dtd

The purpose of this DTD is to define each of the routeXsm.xml files (where X is replaced by the route number). After extracting all the necessary data from SCATS for one entire route the OO Perl script transfers this data to the second tier by means of a routeXsm.xml file (e.g. route 1 = route1sm.xml).

```

<!ELEMENT smData (routeNumber*, subsystem*) >
  <!ELEMENT routeNumber (#PCDATA) >
  <!ELEMENT subsystem (intersectionID*, time*, cycleLength*,
    recommendedCycleLength*, strategicApproach*) >
    <!ELEMENT intersectionID (#PCDATA) >
    <!ELEMENT time (#PCDATA) >
    <!ELEMENT cycleLength (#PCDATA) >
    <!ELEMENT recommendedCycleLength (#PCDATA) >
    <!ELEMENT strategicApproach (saNumber*, smColumnData*) >
      <!ELEMENT saNumber (#PCDATA) >
      <!ELEMENT smColumnData (columnNumber, DS, VO, VK) >
        <!ELEMENT columnNumber (#PCDATA) >
        <!ELEMENT DS (#PCDATA) >
        <!ELEMENT VO (#PCDATA) >
        <!ELEMENT VK (#PCDATA) >

```

Figure 4.5: smData.dtd

All of the encapsulating tags used in the DTD (Figure 4.5) have previously been explained except for smColumnData. The purpose of this tag is to contain all the data from one lane within a particular strategic approach (i.e. one “cell” from the SM display).

XML Tag	Description
<intersectionID>	Unique SCATS intersection number
<time>	The time that this data was extracted from SCATS (from SM header line)
<cycleLength>	Cycle Length of this intersection
<recommendedCycleLength>	Recommended Cycle Length for this intersection
<saNumber>	Strategic Approach Number
<columnNumber>	SM display column number (1 - 4)
<DS>	Degree of Saturation
<VO>	Original Volume
<VK>	Reconstituted Volume

Table 4.3: smData tag descriptions

mapData.dtd

This is the final DTD and it is used by the mapRouteX.xml files (one file per route) in order to transfer data between the second and third tiers. This DTD is an exact replica of smData.dtd except for one small addition. This addition is a congestionMeasure tag that is placed at the very end of the file and is encapsulated by smColumnData. The purpose of this tag is to contain the measure of congestion for that lane which has been calculated by the second tier application.

4.4 End-User Interface

The design of end-user interfaces is often an area that is given scant attention by many programmers and designers even though it is the first and probably only thing the user will ever have to interact with. The design of the interface used in the third tier was based largely on other interfaces that were examined during the research section of the dissertation (Chapter 2). An attempt was made to incorporate the best features and to discard the worst features of all the existing websites. The result of this research led to the following wish list of features which were to be designed into the third tier applications:

1. The user should be given choices as to how they would like to view the data e.g. as an application, an applet, or static images. (Figure 2.6)
2. A map that can show congestion in the form of coloured lines along various routes within the city (Figure 2.4)
3. The congestion shown should be directional i.e. if a road is two-way then two lines should be shown. (Figure 2.4)
4. The map should have as many landmarks, street names, etc. as possible to allow the users to orientate themselves (Figure 2.4)
5. The ability to click on intersections to obtain more detailed data on the state of that intersection (Figure 2.3)
6. Static JPEG pictures that are refreshed regularly should be available if a user wants a quick view of one section of the city.
7. Information should also be available on single routes rather than the entire system (Figure 2.6)

Chapter 5 Implementation

This chapter details some of the major technologies and techniques used to translate the design of the previous section into a practical solution. Any problematic issues that were encountered during the implementation and their solutions are also presented. Any code snippets that are displayed in this chapter also show line numbers and the name of the program from which they were taken. This is done to help any future person(s) who may wish to build upon the work completed within this dissertation.

5.1 Tier 1

The first, and possibly most important, part of the implementation was to extract the data from SCATS. The Perl script created to carry out this task was named `getsm.pl`. An important, and slightly unusual, feature of this script is that it is object-oriented. Generally Perl scripts are used for short repetitive tasks where very little thought needs to go into the structure of the internal program data. As has been shown in Chapter 4 the data structure needed throughout this implementation is far from simple and would have been extremely difficult and unwieldy to create and use without the ability to create classes and vectors of classes, i.e. standard features of OO programming. As version five of ActivePerl provides for OO programming it was decided to use an object-oriented Perl script. Although this decision fared well in the long run gaining an initial understanding of how to use OO Perl was not an easy task. This was mainly due to the fact that Perl was not designed from the outset to be an object-oriented language and so the facility for object orientation is supplied as an add-on rather than as an inherent part of the core language. This results in features such as classes being stored as a Perl “hash” data type which makes for a non-intuitive interface.

As has been previously discussed the best interface to SCATS that was available involved the use of telnet. Setting up an automated telnet session using Perl is a well-understood technology and involves using the `telnet.pm` module which is freely available on the Internet.

```
149 sub openTelnetSession{
150     use Net::Telnet ();
151     $t = new Net::Telnet (Timeout => 20,
152                          Prompt => '/\$/');

```

```

153                                     # Uncomment these lines to create debug logs
154                                     # Dump_Log => "dump_log.txt",
155                                     # Input_Log => "input_log.txt",
156                                     # Option_Log => "option_log.txt",
157                                     # Output_Log => "output_log.txt");
158     $t->open ("151.177.220.88");
159     $username = "username";
160     $passwd = "password";
161     $t->login($username, $passwd);
162     @lines = $t->cmd(String => "scats", Prompt => '/SCATS\>');
163 }

```

Figure 5.1: Sample telnet login code (getsm.pl)

As can be seen from Figure 5.1 logging onto the VAX is a relatively straightforward task that did not present any problems. The purpose of line 162 is to issue the SCATS command to start a SCATS terminal on the VAX after the username and password have been accepted. It was this action that caused one of the biggest problems to become apparent. Line 162 illustrates the usual method used to issue an automated telnet command (cmd). This function takes two arguments: the command to be issued and the prompt that is expected to be returned once the command has been completed. Unfortunately, after starting the SCATS terminal no more prompts were returned to the client telnet session. This was not expected as a human user can log into SCATS using a standard telnet client and will receive a prompt after every command. The very fact that the problem lay in a prompt not being returned took some detective work and considerable use of the optional log files (lines 154 – 157). After ascertaining the nature of the problem the next task was to discover why this was occurring in just the automated telnet session. The reason for this apparent anomaly was eventually realised as being a result of SCATS mistaking the automated telnet session for a line printer that is used to log all commands that are issued to SCATS. As a line printer only requires the *results* of any issued commands the automated telnet session was not receiving any prompts. Every conceivable solution to this problem was tried but with no success. In the end an inelegant work-around was implemented that got over the problem of not having a prompt. This solution was to issue the required command and assume that the results will be completely returned within one second. Figure 5.2 shows an example of how this would work if an SM display for intersection 422 was required.

After issuing the SM command line 43 causes the program to wait for one second after which time line 44 converts any data that has been returned into a string that can then be manipulated. This technique was, of course, recognised as being possibly dangerous due to the fact that there is no 100% guarantee that all of the data will be

returned within one second. To cater for this possibility there are traps in the program that deal with just such an eventuality. These traps search for the data that should be present in the current SM display according to routesData.xml, e.g. intersection and strategic approach numbers. If any of these numbers are missing then the program flags an error and makes a second request for the data. Although it is not currently implemented it would be possible for the program to then extend the sleep period of one second before retrieving the data. After two months of operation the one second wait has yet to cause a problem, however it is something that needs to be taken into account should any future problems arise.

```

41  $ok = $t->print("422");
42  $ok = $t->print("sm");
43  sleep(1);
44  $smScreen = $t->get();
```

Figure 5.2: Automated SM display retrieval (getsm.pl)

A second problem that arose when implementing the data extraction script was again caused by a feature within SCATS. The issue this time was due to the fact that a user can globally change the manner in which SM information is displayed on each region. This would not be a problem except that *all* users on the region are affected by such changes. An example of this would be where a user issues the request “SM=HD” while on a specific region. From that point on any SM data that is requested on that particular regional computer will only show the SM header, i.e. no data on any of the strategic approaches or lanes will be shown. To combat this potential problem a second trap is built into the script that resets the SM format configuration any time the layout of the SM data does not match what is expected. When getsm.pl is initially started it also runs checks on every regional computer to ensure that the correct SM format is set.

The second Perl script that goes to make up Tier 1 is called syslx2xml.pl. The purpose of this script is to trawl through each of the SCATS region’s sys.lx files and extract any additional data (e.g. XCL, SCL, etc) that is required by the congestion algorithm in Tier 2. The extracted data is then saved out to an XML file that is read and parsed by the Java program in Tier 2.

5.2 Tier 2

As can be seen from Figure 4.1 there are two main sections in Tier 2. These are the SAX parser and the congestion algorithm itself. At the time of writing there were two main types of XML parsers available – the SAX parser and the Document Object Model (DOM) parser. SAX runs considerably faster than DOM and is used in cases where all that is necessary is to parse a file and translate the contents into a local object model. DOM is more complex and is used in cases where the programmer may want to alter the XML file that is being parsed. Given the requirements of this implementation SAX was the obvious choice. Both the SAX parser and the congestion algorithm are run from one central Java application called Program.java. When Program.java is first started it creates a new object model (RoutesData.java) that initialises the structure shown in Figure 4.2 and then proceeds to start three separate parsers. The first of these parsers (SaxRoutesDataHandler.java) takes routesData.xml as its input and from this it expands the vectors and populates the RoutesData object model with all the necessary configuration information. The second parser (SaxSysLxHandler.java) takes syslx.xml as its input and adds this data to the RoutesData.java. Finally, the last parser (SaxSmDataHandler.java) is called and takes each of the RouteXsm.xml files from Tier 1 and adds their contents to RoutesData.java. This last parser is placed in an infinite loop so that it regularly refreshes the contents of RoutesData.java with the latest traffic flow information from SCATS.

The purpose of the second section of Tier 2 is to calculate a measure of congestion from the data that is now present within the object model. Extracting the data from the object model is a trivial problem, however what is not trivial is to calculate a meaningful measure of congestion from this data. The difficulty in creating a meaningful congestion algorithm lies in the fact that SCATS is an adaptive UTC. This means that the traffic conditions that cause undue delays at an intersection for one moment in time may not cause any delays a different time. This is because as a strategic approach becomes busier SCATS adjusts the timing of the lights to compensate and allows greater numbers of vehicles through. An example of this would be where a group (also called a platoon) of twenty vehicle approach an intersection at a certain moment in time then they may not all get through the intersection in one cycle, thereby creating a queue. After this SCATS then adjusts the

lights to compensate and at the next cycle a platoon of twenty cars *can* get through the intersection in the same cycle. This illustration shows how for any congestion algorithm to be effective it must take into account the current state of SCATS (i.e. its current ability to compensate for congestion) as well as the actual flows of traffic at an intersection.

```

143  if ((CL>=XCL) && (RL>=(XCL-5) && ((VK/VO)>2.4) && (DS>=100))
                                           {congestionValue=6;}
157  else if ((CL>=XCL) && (RL>=(XCL-5)) && (((VK/VO)>2) && ((VK/VO)<=2.4)))
                                           {congestionValue=5;}
169  else if ((CL>=XCL) && (RL>=(XCL-5)) && (((VK/VO)>1.6) && ((VK/VO)<=2)))
                                           {congestionValue=4;}
175  else if ((CL>=XCL) && (RL>=(XCL-5)) && ((VK/VO)<=1.6))
                                           {congestionValue=3;}
182  else if ((CL>=XCL) && (RL<(XCL-5))) {congestionValue=2;}
187  else if ((XCL>CL) && (CL>=SCL)) {congestionValue=1;}
192  else if ((CL<SCL) && (RL<(XCL-5))) {congestionValue=0;}
200  else if ((VO==0) && (VK==0)) {congestionValue=0;}
202  else {congestionValue=0;}

```

Figure 5.3: Congestion algorithm (BlackBox.java)

Figure 5.3 shows the basic congestion algorithm that was in use at the time of writing this dissertation. The basic idea behind this algorithm come from the SCATS definition of congested operations (Chapter 3). As can be seen there are eight separate conditions that can be present and detected. The algorithm starts by trying to satisfy the most congested condition (i.e. congestionValue = 6). If this condition is not satisfied the then the algorithm continues through the conditions until a match is found. If no match is found then the congestionValue is set to zero. An explanation for each of the terms in the maximum congestion level condition (line 143) is as follows:

Condition	Explanation
CL >= XCL	The current cycle length of the intersection is greater than or equal to the stretch cycle length. This means that at least one of the strategic approaches on this subsystem has a very high degree of saturation and is demanding a longer cycle length than is encountered during normal operations.
RL >= XCL - 5	The requested cycle length is greater than or equal to five seconds less than the stretch cycle length. If CL >= XCL is true then this condition means that the demand on the strategic approaches is not significantly decreasing as otherwise the requested cycle length would be lower than XCL - 5.
(VK / VO) > 2.4	The ratio of reconstituted volume to original volume is greater than 2.4. If this ratio is greatly larger than one then the system got fewer vehicles through a junction than it would have expected to for that cycle length operating under normal conditions. This is generally indicative of a situation where a junction or road ahead of the lane is blocked and vehicle cannot easily pass through.
DS >= 100	The degree of saturation is greater than or equal to one hundred percent.

	The purpose of this additional condition is to rule out the rare situations where the three rules above could be true even though a lane is not massively congested. For instance when lanes are very quiet it is possible that the ratio of VK/VO could be quite high due to a lane becoming suddenly quiet even though the intersection has a very high cycle length.
--	---

Table 5.1: Explanation of maximum congestion measure terms

Most of the other rules are variations on the above conditions for lighter levels of congestion. The one exception to this is the rule where VO and VK are both zero. This situation normally occurs under two different conditions. The first situation could be late at night when no vehicles pass through a lane during one whole cycle. The conditions could also be met when a detector is faulty and is no longer detecting the vehicles as they pass by. As there is no reliable way to tell the difference between these two conditions it was decided for the time being to let the human user decide for themselves which situation was resulting in the conditions being met, e.g. if a congestion measure of zero is being flagged on a normally busy road during rush hour then it is probable that the detector is faulty. A method of automating this decision process might be to factor in the time of day as well as the length of time that a congestion measure has been at zero for.

5.3 Tier 3

Within the third tier there are two separate applications that convey the same basic information in different formats. The dynamic congestion map was produced for people who wished to have congestion information available on a constant basis (e.g. Dublin Corporation traffic control room operatives). It was felt that this user group would be willing to take the time to download the application and install it on their local computer. The second group of users that was targeted were those who wished to get the odd quick “snapshot” of the general traffic situation or of a specific route as they are about to set out on a journey. It is unlikely that this group of users would be prepared to wait the time for an application to download or start up. To cope with this group of users the static map producer application was created.

5.3.1 Dynamic Congestion Map

In Chapter 4 the basic method to be used to convey all the information to an end-user was decided. This design involved overlaying coloured lines to represent differing levels of congestion on top of a map showing the main routes around the city.

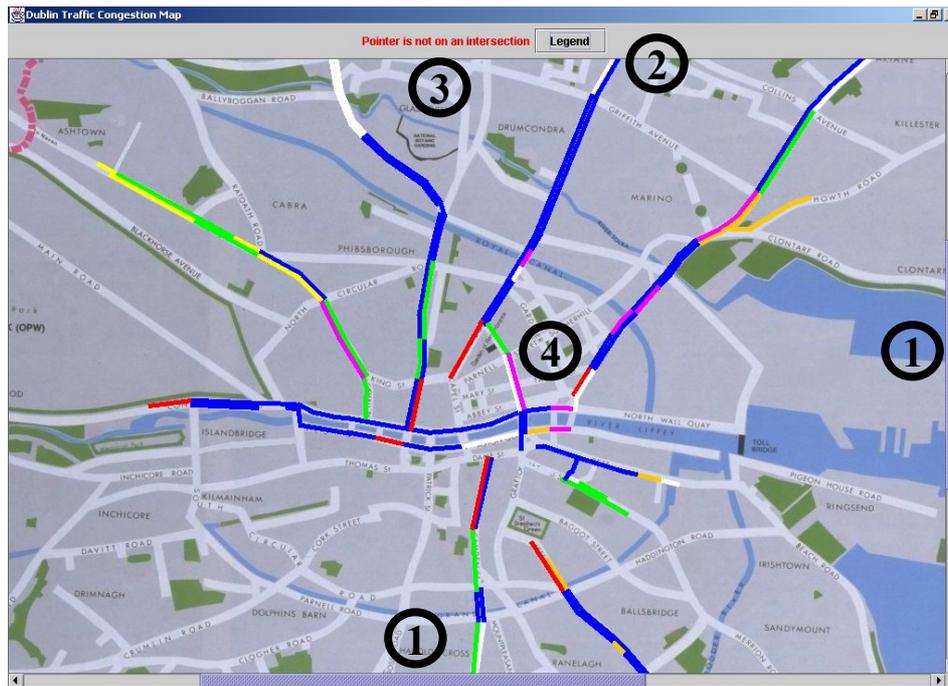
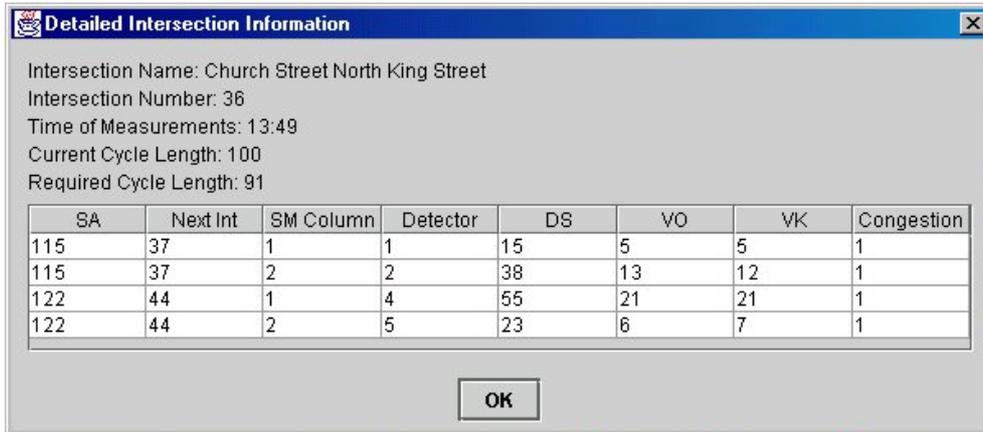


Figure 5.4: Screenshot of the dynamic congestion map

Figure 5.4 shows a screenshot of the dynamic congestion map with the following circled numbers serving to point out some of the features that will be talked about in further detail during the course of this section:

1. Scrollbars
2. Legend button
3. Intersection name and number that the mouse is currently positioned over
4. Coloured lines representing differing levels of congestion at intersections along selected routes

A user can obtain secondary information (e.g. DS, VO, VK, etc) on single lanes by clicking on an intersection. This brings up a dialog box displaying the detailed information as shown in Figure 5.5.



SA	Next Int	SM Column	Detector	DS	VO	VK	Congestion
115	37	1	1	15	5	5	1
115	37	2	2	38	13	12	1
122	44	1	4	55	21	21	1
122	44	2	5	23	6	7	1

Figure 5.5: Pop-up dialog box showing detailed intersection information

The last separate piece of the congestion map was the legend button. When pressed this creates a pop-up box showing the colours relative to their congestion measure.



Figure 5.6: Pop-up legend dialog box

The first difficulty with implementing the dynamic congestion map was to secure a map of Dublin that could be used for the background and did not have any copyright issues that might prevent its publication on the web. Eventually a map was sourced from Dublin Corporation's GIS department. Unfortunately there were several problems with this map but since the only alternative would have been to commission a customised map it was accepted as a suitable compromise. The basic difficulties with the map lay in the fact that it was originally scanned from a paper version. This resulted in poor resolution at high levels of detail and relatively high memory requirements for a web based image (396KB using JPEG compression). The big advantages of the map were that it showed all the major routes around the city and left out the minor roads thus resulting in a less cluttered appearance. The map was also owned by Dublin Corporation and so there were no copyright or licensing issues associated with reproducing it on the Internet.

The second implementation issue that had to be dealt with was how to draw the congestion lines on top of a static image using Java. This was accomplished using the inbuilt Java classes: Graphics, Graphics2D and BufferedImage.

```

private BufferedImage bi;

31  public MapPanel(...) {
    ...
67      setBackground(Color.white);
69      Image img = getToolkit().getImage("map.jpg");
70      Try {
71          MediaTracker tracker = new MediaTracker(this);
72          tracker.addImage(img, 0);
73          tracker.waitForID(0);
74      } catch (Exception e) {}
76      Int iw = img.getWidth(this);
78      Int ih = img.getHeight(this);
80      bi = new BufferedImage(iw, ih, BufferedImage.TYPE_INT_RGB);
81      Graphics2D big = bi.createGraphics();
82      Big.drawImage(img, 0, 0, this);
    }

154 public void paintComponent( Graphics g ) {
    ...
158     Graphics2D g2 = (Graphics2D) g;
164     g2.drawImage(bi, null, 0, 0);
        g2.setColor(Color.white);
        g2.setStroke(new BasicStroke(5.0f));
        g2.drawLine(x1, y1, x2, y2);
    ...
}

```

Figure 5.7: Congestion lines using BufferedImage (MapPanel.java)

By using BufferedImage (shown in Figure 5.7) a copy of the map could be stored in memory where it was then possible to draw custom lines on top of it before displaying the final version on the user's screen. Lines 69 - 74 load the static map and convert it into a Java data-type known as Image. The purpose of the MediaTracker is to keep track of the current state of the image object, i.e. waitForID tells the program to wait until the image is fully loaded before proceeding. This feature becomes important if the image is being downloaded over the Internet as you won't want the map to be displayed until it is fully loaded into memory. The next important step to note is the creation of the BufferedImage in line 80. The three arguments that the BufferedImage constructor takes is the width and height (in pixels) as well as the colour depth that is required. In this case TYPE_INT_RGB means represents an image with 8-bit RGB (Red Blue Green) colour components packed into integer pixels. Line 81 creates a two-dimensional graphics object (Graphics2D), which can be used to draw into the specified BufferedImage. Line 82 then draws as much of the specified image as is currently available which in this case should be all of the image due to the use of the MediaTracker. The method paintComponent (line 154) gets called automatically whenever MapPanel.java is refreshed. Line 158 casts the standard Graphics argument

to a Graphics2D object. This allows us to use the features of the Graphics2D API as well as the more standard Graphics API. Line 164 renders the BufferedImage (i.e. the static background map) that is stored in memory onto the screen and the last three lines of the code snippet show how to set the colour and thickness of a line and then draw it onto the screen.

```

public class MapGUI extends JApplet{
    ...
    public void init(){
        ...
82         Container c = getContentPane();
83         mapArea = new MapPanel( width, height, statusBar, routesData,
                                mapColors );
84         int imageWidth = mapArea.getImageWidth();
85         int imageHeight = mapArea.getImageHeight();
86         MapArea.setPreferredSize(new Dimension(imageWidth, imageHeight));
87         JScrollPane scrollPane = new JScrollPane(mapArea);
88         c.add( topPanel, BorderLayout.NORTH );
            c.add( scrollPane, BorderLayout.CENTER );
        }
    }

```

Figure 5.8: Scrollbars (MapGUI.java)

One of the more problematic implementation features from a Java GUI viewpoint was the use of the scrollbars to view different portions of the map. The use of scrollbars was necessary because the background map size (2046 by 1560 pixels) was far bigger than the standard screen size of 640 by 480 pixels. Getting scrollbars to work on a static image was a straightforward exercise that presented no problems, however getting scroll bars to work on a BufferedImage that consisted of a static map with lines drawn on top was a different matter. Eventually the source of the trouble was located as being due to the order in which the methods used to set up the JScrollPane (the container for the image and its scrollbars) were implemented. Lines 86 and 87 must be called in exactly the same order as they appear in Figure 5.8 (the lines had previously been placed in the reverse order). The purpose of line 86 is to set the preferredSize option of the MapPanel object that contains the BufferedImage and the congestion lines. When the JScrollPane was implemented on MapPanel it had previously been looking for the preferredSize setting and if it did not find it then the process of adding scrollbars failed. After reversing the order of the two lines the problem was solved.

5.3.2 Static Map Producer

The purpose of the StaticMap application is to produce JPEG images of the current traffic situation that Internet users can download quickly and easily. Two different types of images are produced by the methods `StaticMapProducer` and `StaticRoutesProducer` which are called by `StaticMap`. `StaticMapProducer` creates a sectional view of the main congestion map where the user can choose which section of the city they want to look at in further detail. Figure 5.9 shows the static city centre section.



Figure 5.9: Static image of city centre

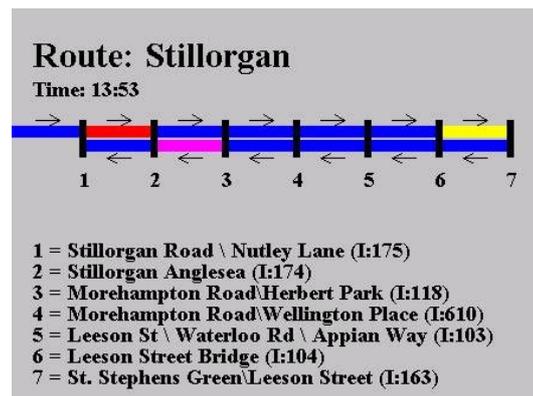


Figure 5.10: Stillorgan static route

It should be noticed that a time stamp is printed onto all the static images to ensure that the user knows whether the images are up-to-date or not. The second type of image that is produced is the static route that is generated by `StaticRoutesProducer`. Frequently, a public user might be only interested in one or two particular routes that they will be using for a specific journey. In this case a user can view a graphic that represents the congestion on one specific route. Figure 5.10 shows the Stillorgan route. Each intersection is represented by a numbered line with the routes being shown as flat lines in a similar fashion to a London Underground map. A key to the intersection names and numbers is also given on the graphic.

StaticMapProducer differs from the dynamic congestion map in two key ways. Firstly it parses an extra XML file called staticMapSetup.xml, the purpose of which is to allow the program administrator to easily select the coordinates and dimensions (relative to the original map) of the static images that are produced. The second main difference is that instead of printing the output to the screen, the StaticMapProducer compresses the output using the JPEG algorithm and then saves the result of this to a file. Figure 5.11 shows the snippet of code that is repeated for every different section of the map that needs to be converted to a JPEG.

```

59 String fileName = staticFileSetup.getImage(i).getFileName();
60 FileOutputStream out = new FileOutputStream(fileName);
61 BufferedImage bi2 = getSplitImage(screen, i);
62 bi2 = addTimeToImage(bi2);
64 if( bi2 != null) {
65     JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(out);
66     JPEGEncodeParam param = encoder.getDefaultJPEGEncodeParam(bi2);
67     param.setQuality(0.7f, false);
68     Encoder.setJPEGEncodeParam(param);
69     Encoder.encode(bi2);
70     System.out.println("Created File " + fileName);
71 }

```

Figure 5.11: Creating a static JPEG file (StaticMapProducer.java)

Line 61 of Figure 5.11 calls `getSplitImage(...)` which is a method that uses the `BufferedImage` function `getSubImage(...)` to return the section of the map we are interested in. After this the time (from SCATS) of the last update made to the map is superimposed upon the sub-image (line 62). Finally the sub-image is then converted to a JPEG using the JPEG coder / decoder (codec) that is supplied as a standard part of the Java Development Kit (JDK). The level of compression performed on the image is set by the first argument supplied to the method `setJPEGEncodeParam(...)` on line 67. This value is of type float and should be between zero and one, where a value of one means maximum output quality, lowest compression. At the time of writing the same JPEG codec is used to compress the graphics for `StaticRoutesProducer`. It would have been preferable however to use a Graphics Interchange Format (GIF) algorithm for these images as this would have resulted in a higher quality image with smaller file sizes. The reason for this is that JPEG is optimised for complex images such as photographs whereas GIF is optimised for computer generated type graphics (i.e. regular shapes with blocks of the same colour). Due to the fact that CompuServe holds a patent on GIF a suitable codec is not supplied as part of the JDK. A suitable codec was obtained from the Internet but there was insufficient time to implement it. Figure 5.12 shows the DTD that is used for the

relatively small XML file that is used to set the sections of the map that are required to be saved out.

```

<!ELEMENT staticMapSetup (staticImage*) >
  <!ELEMENT staticImage (fileName, xCoordinate, yCoordinate, width, height) >
    <!ELEMENT fileName (#PCDATA) >
    <!ELEMENT xCoordinate (#PCDATA) >
    <!ELEMENT yCoordinate (#PCDATA) >
    <!ELEMENT width (#PCDATA) >
    <!ELEMENT height (#PCDATA) >

```

Figure 5.12: staticMapSetup.dtd

A staticImage tag is used to enclose every separate description of an image section that is needed. A description of the other tags is contained in Table 5.2.

XML Tag	Description
<fileName>	The file name that we wish the section to be saved as (e.g. ccity.jpg)
<xCoordinate>	The X coordinate of the top left-hand corner of the section (relative to the overall map)
<yCoordinate>	The Y coordinate of the top left-hand corner of the section (relative to the overall map)
<width>	The width of the required section (in pixels)
<height>	The height of the required section (in pixels)

Table 5.2: staticMapSetup tag descriptions

5.4 Physical Implementation

The purpose of this section is show how all the various software components fit together and where they can be run by different users. Figure 5.13 shows one possible physical implementation of the system. An architecture very similar to the one shown has already been implemented in Dublin Corporation and was proven to work at a project demonstration where a dynamic congestion map was run in Trinity College and took its live data feed from a web server in Dublin Corporation.

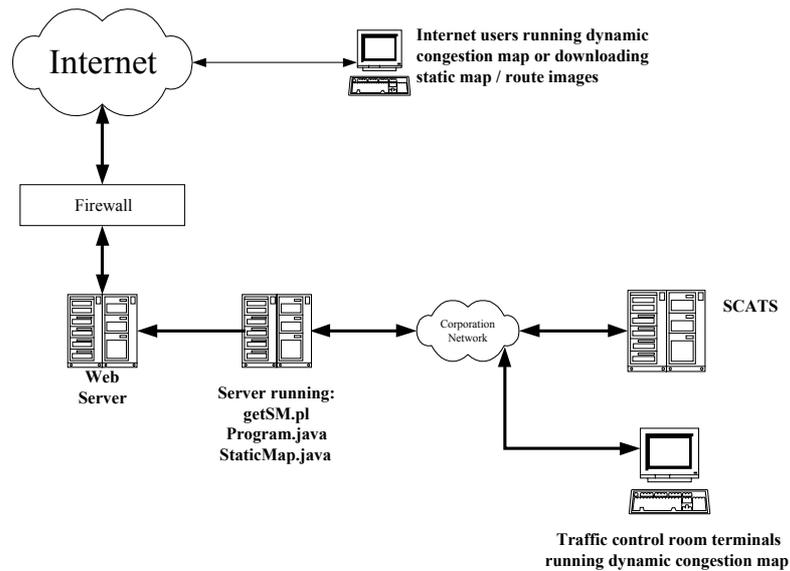


Figure 5.13: Physical implementation

The two distinct types of users shown in the scenario above are people within the Dublin Corporation private network and the general public who are on the Internet. The only major difference between the users is that the internal users directly access XML files from the server running Tiers 1 and 2, whereas public users on the Internet can read the XML files from the web server. The reason why both sets of users do not use the web server is that at present there is no access to the Internet via Dublin Corporation's private network.

Chapter 6 Evaluation

The purpose of this chapter is to evaluate and compare the work carried out in the implementation chapter against the original architecture that was designed to meet the primary objectives of the dissertation.

6.1 Congestion Algorithm

Finding an objective way of evaluating the implemented congestion measurement algorithm is very difficult due to the fact that there are no other congestion measures available to compare it against. At present degree of saturation is the nearest thing that SCATS operators have to a congestion measurement and is one of the first measures that is looked at when the SM display is called up. Due to the nature of an adaptive UTC the DS reading will tend to fluctuate quite wildly and rapidly over the course of a day as SCATS adjusts the system to cope with the varying demands. This can be clearly seen in Figure 6.1 where the DS readings for one lane over an entire weekday are displayed in graph format. The only major trends that are apparent from this data is the night-time lull between 02:00 and 05:00. During the day it is difficult to pick out features such as the morning and evening peaks. There is also no indication given of the times when SCATS can cope with the extra demand by giving extra time to a congested approach and the times when it can no longer cope. This means that the control room operators cannot tell whether a value peak is indicating a transient blip or something more serious that may require human attention.

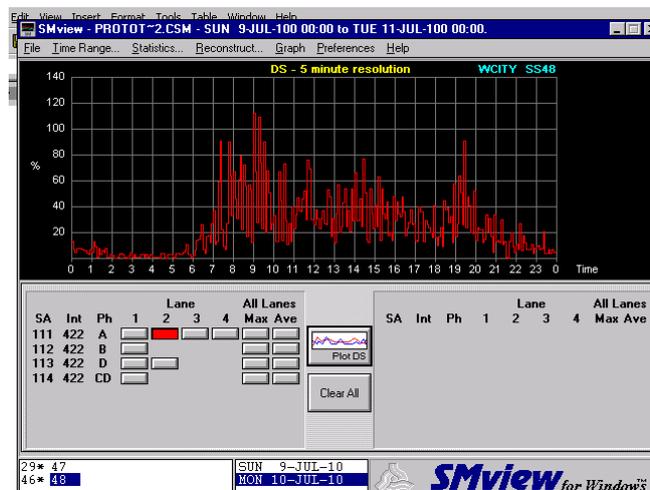


Figure 6.1: Graph of degree of saturation over a period of one day

As the DS values in Figure 6.1 were being collected the results of the congestion algorithm that was run on the same lane, at a frequency of once per minute, was also gathered. The results of the congestion measure are graphed in Figure 6.2 and as can be clearly seen the difference between the two graphs is quite obvious. The morning and evening peaks along with the night-time lull (which shows a brief increase around pub closing hours) are all clearly visible. During the day the figures oscillate between congestion measures of one and three except during midday when it remains settled at a measure of one.

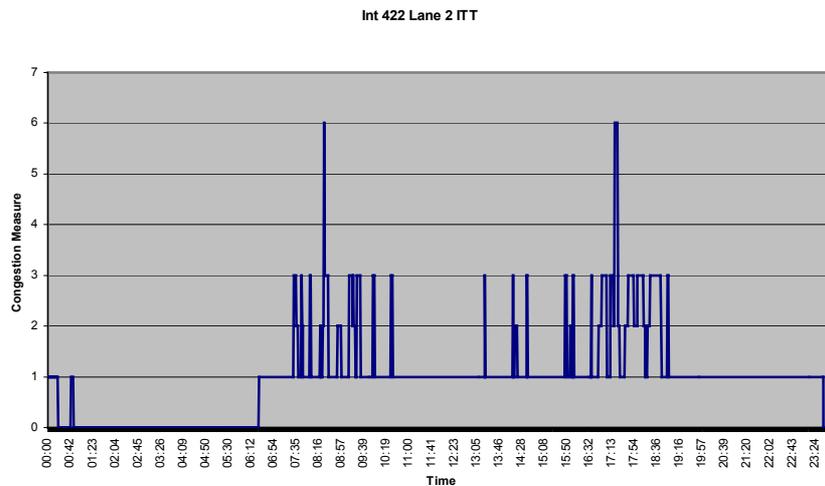


Figure 6.2: Graph of congestion measure over a period of one day

In conclusion, there can be little doubt that the data shown in Figure 6.2 is considerably more meaningful than the data of Figure 6.1. As well as graphing the data it is possible to visually compare the results of the congestion measure against the images from Dublin Corporation’s private network of traffic cameras. It would be difficult, if not impossible, to visually discern the difference between various median congestion levels, however it was possible to verify whether the maximum or minimum congestion measure being produced by the algorithm were warranted when compared to the situations seen on the cameras. In the vast majority of cases that were seen, if the congestion value was at six then traffic was invariably blocked up and moving extremely slowly, if at all.

6.2 Performance

What is meant by performance in this section is the speed at which the implemented system can carry out its various tasks. Any potential performance bottlenecks that have been discovered during the implementation phase will also be examined.

The first and most obvious potential problem lies in the fact that the automated telnet session can only extract data from one intersection every second (see Chapter 5). Ideally the rate of data extraction for every intersection should be the same as its cycle length due to the fact that the SCATS data gets refreshed once per cycle. In practise most intersections around Dublin city operate at a cycle length of between 100 and 120 seconds but as the target cycle length for the city centre is 70 seconds this should also be the target data extraction rate that Tier 1 is capable of. A very simple solution to this problem would be to start a simultaneous telnet session on SCATS for every 70 intersections that have to be examined. At the time of writing there were 72 intersections spread over 10 routes in routesData.xml so there was no need to implement any extra telnet sessions. From a technical standpoint there are no reasons as to why such a feature would not be very easy to implement.

The second issue affecting performance stems from the system architecture itself. Each tier is a separate application and so they loop (completely refresh their data) at different rates. In order to calculate the maximum possible delay in getting data from SCATS to the screen of the end-user, all the loop times from each tier must be added together. At the time of writing the maximum delay was as follows:

$\text{Tier 1 (72s)} + \text{Tier 2 (60s)} + \text{Tier 3 (60s)} = \text{Max. Possible Delay (192 seconds)}$
--

If necessary this figure could be reduced by approximately 50 seconds or so if the second tier was placed on an infinite loop with no inbuilt delay. If it were deemed necessary the length of time between map refreshes in Tier 3 could also be significantly reduced, provided that the web-server or computer where the XML files were stored would be capable of handling the increased files request rate.

The third performance issue affects the dynamic map application in the third tier. This application is memory and processor intensive for three reasons:

1. The background map is nearly 400KB in size when compressed using JPEG, however when it is being used by the dynamic map application the image has to be uncompressed and takes up 9.1MB of memory.
2. As the number of intersections and routes to be displayed increases the number of calculations required to refresh the graphics and check whether the mouse is situated over an intersection increases significantly.
3. The application is written in Java which has significant inbuilt overheads as a result of being an interpreted language.

These issues do not become a problem where the machine being used has sufficient resources. As an example a computer with 128MB of RAM and an Intel Pentium III processor has proven to have more than adequate resources to run the application with no visible reduction in performance.

6.3 Scalability

There were three main criteria for which the proposed architecture had to be scalable. These criteria stated that the system could not place any limits on the number of intersections or routes, end-users, or end-user applications that could be added to the system.

By adding simultaneous telnet sessions (see previous section) and ensuring that the hardware at all stages has sufficient resources to handle the calculations then there are no practical limits on the numbers of intersections or routes that could be added to the system.

The number of end-users that the system can cope with depends solely on the ability of the web server to cope with the number of page requests it receives for the XML files and JPEG images. There are countless examples of websites that handle many multiples of the number of users that are expected to access this system. It should be noted however the number of page requests would more than likely peak during the morning and evening rush-hours when most journeys are taking place. Therefore the web server should be capable of handling these peaks rather than just an average number of “hits” per day. If necessary, the bandwidth of the Internet link could be increased along with the resources of the web-server to cope with greater numbers of end users.

Scaling the system to cope with added end-user applications is precisely the same problem as coping with additional end-users. This is because any extra applications will just increase the number of requests made to the web-server for the XML files.

6.4 Concurrency

The issue of concurrency is one that arises as a result of the three tiers operating simultaneously while still requiring good data from each other in order to operate correctly. A feature of the architecture that was used is that it does not deal with the possibility of “dirty data”. Dirty data is what results when corrupted data is read in from a file by one process as a result of a second independent process performing a write operation on the same file, at the same time, as the first read operation is taking place. While occurrences of dirty data have only been encountered on very rare occasions during testing, it is likely that the problem will increase as the number of users on the system increases.

Dirty data reads can potentially take place at either of the two interfaces between the three tiers. If Tier 1 is writing an XML file while Tier 2 is attempting to read from the same file then it is possible that Tier 2 will receive corrupted data. Tier 3 can potentially receive corrupt data in exactly the same manner if Tier 2 is writing to a file while Tier 3 is reading from it. At the time of writing there is an exception mechanism in place whereby if any of the SAX parsers encounters data that appears corrupted (e.g. an invalid intersection number) the parser restarts itself and rereads the file after an interval of approximately ten seconds by which stage any write operation will have been completed. Ideally however, the instances of dirty data should be prevented from happening in the first place.

Preventing occurrences of dirty data between Tiers 1 and 2 is easier than doing so between Tiers 2 and 3. This is because Tier 1 uses Perl which has a file locking mechanism as part of the core language. This means that before Tier 1 begins writing to an XML file it can first place a lock on it (providing nothing is already accessing the file) thereby preventing Tier 2 from reading the file. Then once Tier 1 is finished writing to the file it can release the lock and let Tier 2 access the file once more. Unfortunately, Java does not provide any such locking mechanism so an alternative solution has to be found for the interface between Tiers 2 and 3. Two alternative solutions involve the use of either “shadow paging” or a transactional database. When

using the technique of shadow paging, write operations are not performed on the same file that read operations take place on. The file that the writing takes place on is known as the shadow copy. Once all the write operations have been completed on the shadow copy it is renamed as the file that the read operations are to take place on. As the process of renaming a file is far quicker than actually opening, writing and closing a file there is less chance of the occurrence of dirty data. Unfortunately, even though the chances of a problem are greatly reduced there still is a chance that a problem could occur if a second process tried to access the file during the renaming process. The second alternative is where a transactional database is used as a buffer in the second tier. This means that instead of transferring information between the second and third tiers using files all the information is stored on a database. Any user or third tier application that wishes to access the data would then have to make a request to the web server which would in turn query the database and return the result. The advantage of using this approach is that the transactional database ensures that a file or unit of data that is being written to, cannot be read at the same time. The obvious disadvantage to such a system is the additional overhead and extra level of indirection that is added to the data path. There would, at the very least, be some reduction in the overall performance of the system.

A suitable compromise might be to use a combination of all the techniques mentioned above, i.e. file locking, shadow paging and a transactional database. File locking could be used between Tiers 1 and 2, and for immediate or current data shadow paging could be used between Tiers 2 and 3, where a small transient error that lasts for a few seconds is not of great significance. Having correct data is of far greater importance when it is being used for the purposes of analysis (i.e. historical data). As it is probable that a transactional database will be added at some future stage to the second tier for the purpose of storing historical data (see Chapter 7), the problem of dirty data would no longer be an issue.

Chapter 7 Conclusions

This chapter summarises the work that was carried out and the objectives that were achieved during the course of the project. It also mentions some of the insights that were gained through the various design decisions that had to be taken along the way and the various types of users who could benefit from this project. Finally, some of the many possibilities for future work, that could build upon the knowledge obtained during the course of the project, are discussed.

7.1 Achievements

In order to satisfy the main objectives and constraints specified at the start of this project a three-tier architecture was designed and implemented. The resulting work was then evaluated and compared against the initial requirements and any outstanding problems or features of the implementation were discussed. This evaluation led to the conclusion that all three of the major objectives laid out at the start of the project had indeed been achieved. These three major objectives were as follows:

1. The extraction and conversion of SCATS data into a universally usable and extendable format.
2. The creation of an algorithm that would be capable of using the extracted data to generate a meaningful measure of road traffic congestion.
3. The presentation of all this data to a wide variety of users in a clear and meaningful format.

Arguably, the most important of these objectives was the SCATS data extraction application (Tier 1). This was important as it enabled third party access to large quantities of data that had been previously “locked up” in a SCATS proprietary format. Once the data had been extracted the key to making it accessible to as many users as possible was through the use of XML. Once the SCATS data has been translated into XML it becomes possible for any application to take the XML file and make use of the data.

As an adaptive UTC, SCATS can adjust the timing of the lights to allow greater or fewer numbers of vehicles through an approach, depending upon the prevailing traffic conditions. Therefore, to create a meaningful measure of congestion it was important

to know how well SCATS could cope with the current situation or whether it was reaching the limits of what it could do. This knowledge resulted in a congestion measure that made use of both the current state of SCATS as well as measurements of the traffic flow itself. During the evaluation phase (Chapter 6) it became apparent that this congestion measure was, at the very least, far more meaningful than any single measurement currently available from the SCATS.

The third primary objective was achieved through the use of three different user interfaces that were designed to serve the needs of both internal Corporation staff and public Internet users. A dynamic congestion map that gives immediate access to both the overall congestion map as well as highly detailed information on each intersection was implemented to serve the needs of the traffic control room in Dublin Corporation as well as any external users that may require constant access to traffic information. Two types of static image producers were also implemented for users that may require a quick view of the current state of congestion. The first static image producer creates JPEG images of various sections of the overall congestion map. The second static image producer creates graphics showing the state of congestion on individual routes. Both of the static producers refresh their images once a minute to ensure that the user receives an up-to-date image.

As well as the main objectives of the project, several constraints were also present. These constraints included the need for any implementation to be modular, scalable and easily configurable. The need for the system to be modular ensured that the project would be usable with any future versions of SCATS, that any changes to the congestion algorithm could be easily implemented and that future end-user applications could be incorporated with relative ease. The project had to be scalable to ensure that the implementation could cope with any extra intersections, routes and end-users that were required. Finally the design had to allow the set-up of the routes and intersections to be easily configured by a system administrator. All of these constraints were met by using a combination of a three-tier architecture and XML configuration files.

7.2 Potential Users

There are at least four categories of users that will benefit from the work carried out in this dissertation. These include:

1. Traffic control room operatives in Dublin Corporation
2. Traffic news reporters
3. General road users
4. Road traffic researchers

At present the control room operatives in Dublin Corporation have no overall picture of the current traffic conditions around the city. Their primary sources of information include the network of traffic cameras and the SCATS user interface that displays information on one intersection at a time. A SCATS congestion map is available but is rarely, if ever, used due to its lack of directional information. Therefore the dynamic congestion map ought to be extremely helpful in indicating the areas to which the controllers should direct their attention. Traffic news reporters would undoubtedly find the dynamic congestion map very useful in helping to compile more accurate and up-to-date traffic reports. The static congestion maps should be helpful to the general public as they will be able to check for themselves whether there are any major congestion problems on the routes they are about to travel on. Potential future extensions to this project such as a WAP application (see next section) would be of considerable help to people wishing to check on potential congestion problems while on the move.

7.3 Future Work

Due to the combined use of XML and the three-tier architecture this project leaves open countless possible avenues of future work and research. The most immediately beneficial of these possibilities would be the addition of a database to the second tier. This database would store all the historical data from SCATS which could then be used for the purpose of analysis. At the time of writing the only method of getting historical data from SCATS is to run a “collection” routine. A collection is where SCATS is told in advance to save all the data from a certain intersection between certain times and tends to be used very infrequently. There is no facility to save system-wide traffic data on an on-going basis. The existence of a permanent store of past data would enable traffic engineers to analyse historical data and spot trends over significant periods of time. An interesting possibility that could be used in conjunction with such a database might be the implementation of a trained neural network that could find heretofore unnoticed patterns in the historical data. An

example of this would be the fact that sometimes there can be “good” traffic days while other days can be classified as “bad” traffic days even though there may be no discernable cause that differentiates between the two. Finding the significant differences between two such days is a task that would be ideally suited to the pattern-matching skills of a trained neural network. Another interesting use for a historical database might be to link it to a visualisation tool such as a virtual-reality system that would allow traffic engineers to “fly” over a three dimensional congestion map. The benefits of this would in allowing the user to picture large amounts of complicated data in an intuitive manner.

Another interesting possibility for future work would be the addition of a Wireless Application Protocol (WAP) interface. This would be situated on the third tier and would probably be limited to producing a simpler version of the static route images due to the limited resources current available on WAP enabled phones. As mobile phones with better resources and higher data transfer rates (e.g. UMTS) become available the amount of data available could be increased. An additional third tier application that could be added might be a personalised routes service. As more intersections are brought onto the system it would become possible for people to request an image showing congestion levels on their own personally selected routes. Alternatively, it would also be possible for an application to automatically create a map showing a user the least congested route between two user-defined points within the city. To have such a system linked into a mobile device such as a phone would then become very useful indeed.

Finally, as has been mentioned in Chapter 2, a measure that general road users are really interested in is travel time, i.e. how long will it take to get from A to B? At the time of writing, Dublin Corporation was about to install a journey time system on two routes within the city. This system utilises image processing to recognise the number plates of vehicles at two different points along a route which means that the journey time and average speed between the points can then be calculated. By knowing the congestion measures that correspond to different average vehicle speeds, journey times for other similar routes, that don't have the image processing system, could be extrapolated and presented to the road users.

References

Athens Real-Time Traffic Map. Retrieved May 5th, 2000 from the World Wide Web:
<http://www.transport.ntua.gr/map/>

Aubert D., Bouzar S., Lenoir F., Blosseville J.M., “*Automatic Vehicle Queue Measurement at Intersections using Image-Processing*”, Eighth International Conference on Road Traffic Monitoring and Control, (Conf. Publ. No.422), IEE, pp.100-104, London, UK, 1996.

Diaz M.E., Ferris R., Cavero V., Martinez J.J., Guillen S., Fuertes A., “*Evaluation of a computer vision based automatic incident and congestion detection system in an urban context*”, Transportation Systems: Theory and Applications of Advanced Technology. A Postprint Volume from the IFAC Symposium. Pergamon. Part vol.1, pp.459-64 vol.1. Oxford, UK, 1995

Bouzar S., Lenoir F., Blosseville J.M., Glachet R., “*Traffic Measurement: Image Processing Using Road Markings*”, Eighth International Conference on Road Traffic Monitoring and Control (Conf. Publ. No.422), IEE, pp.105-109. London, UK, 1996.

Brennan S., “*Visualisation of Dublin Traffic on the Web*”, Final Year Computer Science Project, Trinity College, Dublin, May 2000.

Cherrett T.J., Bell H.A., McDonald M., “*The use of SCOOT type single loop detectors to measure speed, journey time and queue status on non SCOOT controlled links*”, Eighth International Conference on Road Traffic Monitoring and Control (Conf.Publ. No.422), IEE, pp.95-99, London, UK, 1996.

The Definitive Table of Real-Time Traffic. Retrieved May 5th, 2000 from the World Wide Web: <http://www.nawgits.com/rtable.html>

Dion F., Yagar S., “*Real-Time Control of Signalised Networks – Different Approaches for Different Needs*”, Eighth International Conference on Road Traffic Monitoring and Control (Conf.Publ. No.422), IEE, pp.56-60. London, UK, 1996.

Evans R.G., Bell M.C., “*The identification of recurrent urban traffic congestion.*”, Eighth International Conference on Road Traffic Monitoring and Control (Conf. Publ. No.422), IEE, pp.183-7, London, UK, 1996.

Gartner N.H., “*OPAC: A Demand Responsive Strategy for Traffic Signal Control*”, Transportation Research Record 906, pp.75-81, 1983.

The Georgia NAVIGATOR. Retrieved May 5th, 2000 from the World Wide Web:
<http://georgia-navigator.com/>

Gross Neil.R., “*SCATS Adaptive Traffic System*”, Adaptive Traffic Control Workshop, TRB Committee A3A18, July 1998.

- Henry J.J., Farges J.L., “*PRODYN*”, Procs. 6th IFAC/IFIP/IFORS Symp. on Control, Computers, Communication on Transportation, Paris, France, pp.253-255, 1989.
- Hunt P.B., Robertson D.I., Bretherton R.D., Royle M.C., “*The SCOOT On-Line Traffic Signal Optimisation Technique*”, International Conference on Road Traffic Signalling, IEE, pp.59-62. London, UK, 1982.
- Longley D.A., “*A control strategy for congested computer-controlled traffic networks*”, Transportation Research, 2, pp.391-408, 1968.
- Lowrie, P.R., “*The Sydney Co-ordinated Adaptive Traffic System – principles, methodology, algorithms*”, Proc. IEEE International Conference on Road Traffic Signalling, London, pp67-70, 1982.
- Luk J.Y.K., “*Two traffic-responsive Area Traffic Control methods: SCAT and SCOOT*”, Traffic Engineering and Control, Vol. 25, No. 1, Jan. 1984.
- Mauro V., “*Road Network Control*”, Concise Encyclopaedia of Traffic & Transportation Systems, Pergamon Press, pp.361-366, 1991.
- Nielsen J., “*Designing Web Usability: The Practice of Simplicity*”, New Riders Publishing: Indianapolis, 2000.
- Nielsen J., “*Usability Engineering*”, Academic Press: Boston, 1993.
- Online Simulation Downtown Area Duisburg. Retrieved May 5th, 2000 from the World Wide Web: <http://www.traffic.uni-duisburg.de/OLSIM/olsim.html>
- Papageorgiou M., “*Concise encyclopedia of traffic & transportation systems*”, Oxford: Pergamon, 1991.
- Petty K.F., Bickel P., Ostland M., Rice J., Schoenberg F., Jiang J., Ritov Y., “*Accurate estimation of travel times from single-loop detectors*”, Transportation Research Part A - Policy & Practice, vol.32A, no.1, pp.1-17, UK, Jan. 1998.
- Pignatoro L.J. *et al.*, “*Traffic control in oversaturated street networks*”, NCHRP Report 194, Transportation Research Board, Washington, DC, USA, 1975.
- Roads and Traffic Authority of New South Wales, Australia, “*SCATS Message Formats*”, (RTA-TC-226), June 1999.
- Robertson D.I., Grower P., “*Use of TRANSYT Version 6*”, Transport and Road Research Laboratory Supplemental Report 255, Crawthorne, 1977.
- Shibata J., Yamamoto T., “*Detection and Control of Congestion in Urban Road Networks*”, Traffic Engineering and Control, Vol. 25, No. 9, pp. 438-444, 1984.
- Sims A.G., Dobinson K.W., “*The Sydney Coordinated Adaptive Traffic (SCAT) System Philosophy and Benefits*”, IEEE Transactions on Vehicular Technology, Vol. VT-29, No.2 , pp130-137, 1980.
- Superoute 66. Retrieved May 5th, 2000 from the World Wide Web: <http://travel.labs.bt.com/route66/>

TRENDS – Real Time Road Traffic Information for Gothenburg. Retrieved May 5th, 2000 from the World Wide Web: <http://trends.ics.uwe.ac.uk:5678/>

Tzafestas SG, Laliotis LN, Protonotarios M. *Decision support and artificial intelligence in GIS: overview and applications*. [Conference Paper] Control Applications and Ergonomics in Agriculture (CAEA'98). Proceedings volume from the IFAC Workshop. Elsevier Sci., pp.53-60. Kidlington, UK, 1999.

Webster F.V., Cobbe B.M., *“Traffic Signals”*, Road Research Laboratory Technical Paper No. 56, London, 1966.

Wolshon B., Taylor W.C., *“Analysis of intersection delay under real-time adaptive signal control”*, Transportation Research Part C-Emerging Technologies, vol.7C, no.1, pp.53-72. Publisher: Elsevier, UK, Feb.1999.

Yagar S., Han B., *“A Procedure for Real-Time Signal Control that Considers Transit Interference and Priority”*, Transportation Research-B28, pp. 315-331, 1984.