Detecting Changes in Network Performance from Low Level Measurements

By

Mark Sandford

A doctoral thesis submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of Loughborough University

July 2001

Psalm 27 Of David.

The LORD is my light and my salvation-- whom shall I fear? The LORD is the stronghold of my life-- of whom shall I be afraid?

When evil men advance against me to devour my flesh, when my enemies and my foes attack me, they will stumble and fall.

Though an army besiege me, my heart will not fear; though war break out against me, even then will I be confident.

One thing I ask of the LORD, this is what I seek: that I may dwell in the house of the LORD all the days of my life, to gaze upon the beauty of the LORD and to seek him in his temple.

For in the day of trouble he will keep me safe in his dwelling; he will hide me in the shelter of his tabernacle and set me high upon a rock.

Then my head will be exalted above the enemies who surround me; at his tabernacle will I sacrifice with shouts of joy; I will sing and make music to the LORD.

Hear my voice when I call, O LORD; be merciful to me and answer me.

My heart says of you, "Seek his face!" Your face, LORD, I will seek.

Do not hide your face from me, do not turn your servant away in anger; you have been my helper. Do not reject me or forsake me, O God my Saviour.

Though my father and mother forsake me, the LORD will receive me.

Teach me your way, O LORD; lead me in a straight path because of my oppressors.

Do not turn me over to the desire of my foes, for false witnesses rise up against me, breathing out violence.

I am still confident of this: I will see the goodness of the LORD in the land of the living.

Wait for the LORD; be strong and take heart and wait for the LORD.

Abstract

The Internet and associated network technologies are an increasingly integral part of modern day working practices. With this increase in use comes an increase in dependence. For some time commentators have noted that given the level of reliance on data networks, there is a paucity of monitoring tools and techniques to support them. As this area is addressed, more data regarding network performance becomes available. However, a need to automatically analyse and interpret this performance data now becomes imperative. This thesis takes one-way latency as an example performance metric. The term 'Data Exception' is then employed to describe delay data that is unusual or unexpected due to some fundamental change in the underlying network performance. Data Exceptions can be used to assess the effect of network modifications and failures and can also help in the diagnosis of network faults and performance trends. The thesis outlines how Data Exceptions can be identified by the use of a two-stage approach. The Kolmogorov-Smirnov test can initially be applied to detect general changes in the delay distribution, and where such a change has taken place, a neural network can then be used to categorise the change. This approach is evaluated using both a network simulation and a test network to generate a range of delay Data Exceptions.

Acknowledgements

Many thanks to Dr. David Parish, my supervisor, for his help and encouragement. His continual support and advice have been a great source of strength. David, I hope you know how much you are appreciated by everyone in the group.

Also I'm very grateful to Dr. Iain Phillips whose technical expertise has alleviated many a problem. How many beers is it now Iain?

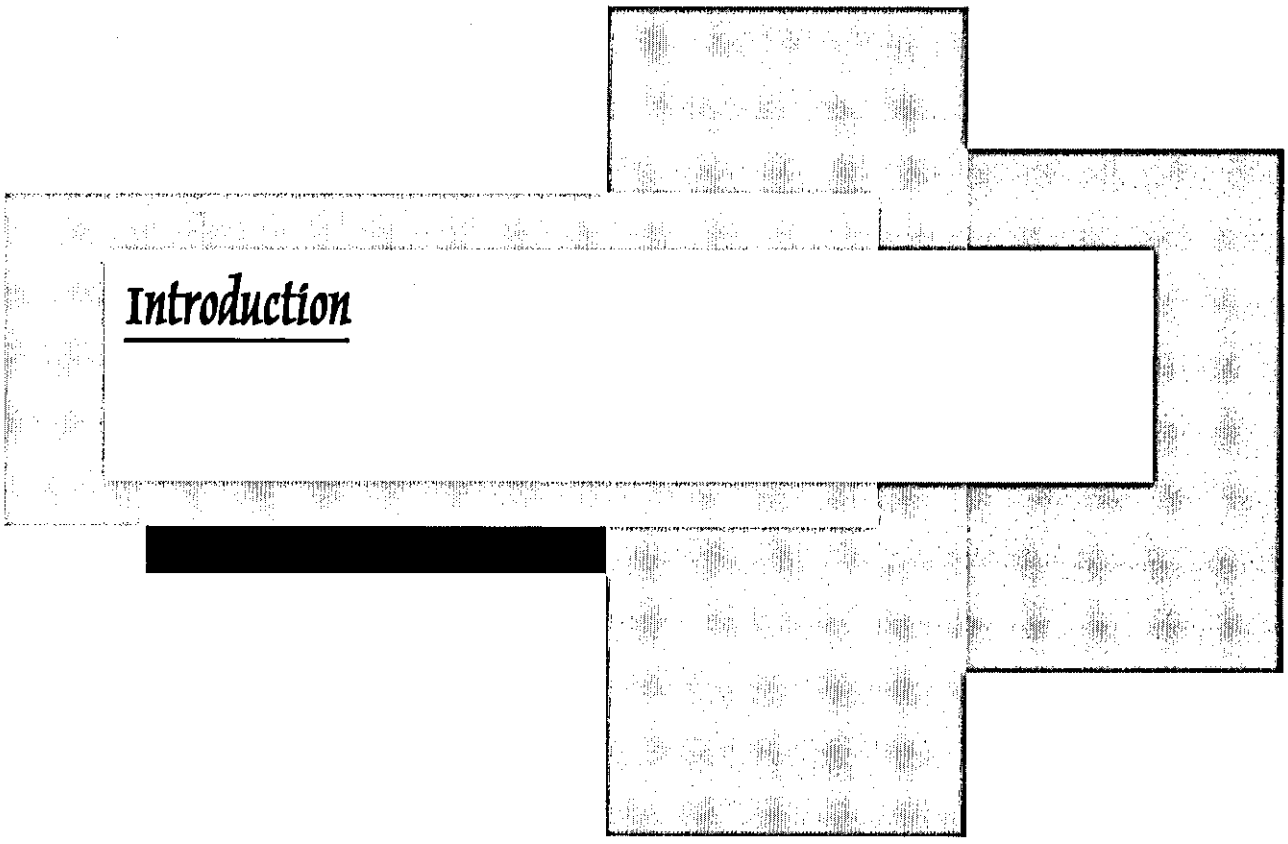
I should also mention British Telecom plc., who got me going in this whole area to start with and also Cisco Systems for their kind donation of network equipment.

Finally, Praise be to our Lord and Saviour Jesus Christ who has blessed us with every spiritual blessing. Glory and honour to his name because he is God and there is no other.

Abbreviations and Acronyms

AIR	Automatic Incident Reporting
AMP	Active Measurement Program
ASN.1	Abstract Syntax Notation One
ATM	Asynchronous Transfer Mode
BT	British Telecom.
CDF	Cumulative Distribution Function
FTP	File Transfer Protocol
GPS	Global Positioning System
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IGRP	Interior Gateway Routing Protocol
IP	Internet Protocol
IPPM	IP Performance Metrics
ITR	Internet Traffic Report
KS	Kolmogorov-Smirnov
MIB	Management Information Base
NAI	National Analysis Infrastructure
NAM	Network Animator
NIMI	National Internet Measurement Infrastructure
NLANR	National Laboratory for Applied Network Research
OFTEL	Office of Telecommunications
PC	Personal Computer

RIP	Routing Information Protocol
RTT	Round Trip Time
SLA	Service Level Agreement
SMDS	Switched Multimegabit Data Service
SNMP	Simple Network Management Protocol
TCL	Tool Command Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	University Research Initiative
WIC	Wide area Interface Card



Introduction

Introduction

This thesis presents a novel means of detecting changes in low level delay measurements taken from communication networks. These changes are termed Data Exceptions. The work originates from a measurement project undertaken by Loughborough University, funded by BT, where monitor stations are deployed to actively monitor unidirectional latencies across BT's SMDS network. A requirement of that project is that Loughborough University provide reports showing any significant changes to the measured delay values. These Data Exception reports are provided on a weekly basis, not in real time as the purpose of these reports is to provide information regarding the impact of network events rather than the detection of faults which would require real time analysis. The goal of this thesis then, is to present a novel means of automating the Data Exception Detection process and the approach presented is the central contribution of the work. The approach need not operate in real time but had to reliably detect Data Exceptions without raising large numbers of false alarms which undermine operator's confidence in such a system. Another desirable characteristic of the approach was that it should require a minimum amount of training or parameterisation.

The main approach presented employs the K-S test statistic to detect for the presence of a significant change in the data. The K-S test is particularly appropriate as it is distribution free, making no assumptions about the distribution of the delay. Another strength of the K-S test is that it tests for general changes, whether they are changes in location or spread. The test is also computationally light and simple to perform. It will be shown that this approach is a very proficient means of detecting changes in the data and requires no training or parameterisation.

The second phase of the Data Exception detection process uses a neural network to classify the change into one of several known categories. These are

- step changes
- changes in Time of Day Delay Variation

- spikes
- weekends

Once trained the neural network classifies the changes with a high degree of accuracy. Data Exception classification may prove to be useful as it allows related Data Exceptions to be grouped together. These groupings are termed Data Collections and may eventually allow for a system to output probable causes for the change in performance that has been observed. It is hoped that further research may show that a neural network may be trained to recognise generic Data Exception types independent of the network which they are from. This would enable the entire two-stage approach to be fully utilised without any training or parameterisation required. This would be a major benefit over rule-based approaches that need parameterisation.

The thesis is structured in the following way:

Chapter 1

In chapter one an overview of network management is given with specific reference to network performance monitoring. Consideration is given to different performance measures, the terminology and what the measures represent. Measurement tools are discussed and an overview of known tools that measure unidirectional latencies is provided.

Chapter 2

Chapter two details the background to the thesis in the form of an overview of the measurement work undertaken by Loughborough University and the measurement architectures currently deployed. Also attention is given to the AIR (Automatic Incident Reporting) system which incorporated a form of Data Exception detection. Work from this thesis contributed to publications on the AIR system. [**Phi99**][**PhiSan00**]

Chapter 3

The concept of Data Exceptions is discussed more fully in chapter three and examples are given both from monitoring systems at Loughborough and those of other research projects.

Chapter 4

Chapter four outlines the various data sources available to this project. Aside from the SMDS network, a simulation and a test network were used to generate delay data that could be used to develop and assess Data Exception detection methods.

Chapter 5

Two approaches to detecting Data Exceptions are discussed in chapter five. An early approach used a rule-based method that was implemented as part of the AIR system. A more sophisticated two-stage approach was subsequently developed which utilised the K-S test statistic to detect, and a neural network to classify Data Exceptions and it is this approach that forms the core of the work.

Chapter 6

In chapter six this two-stage approach is tested using data from both the simulation and the test network and the results are presented.

Chapter 7

The conclusions are given in chapter seven, along with suggestions for further work.

Chapter 1 - Network Management and Performance Monitoring

1. Network Management and Performance Monitoring

In this chapter a general background to the thesis is given. We take a broad look at the area of network management as a whole, the role of network performance monitoring within that and some of the research projects that conduct performance monitoring, specifically considering delay.

1.1 Network Management

With the rapid development of computer networks in recent times; network management has become an increasingly important area. Many businesses and organisations now have a high degree of dependence on network technology. Access to shared resources and information rely on computer networks and network failures can seriously impede working practices and even bring things to a standstill. The effect of network failures can be very serious, costing companies like banks large sums of money. As networks have grown in both size and complexity so the task of managing them has become more involved. Understanding the sophisticated devices and systems that make up a computer network and being able to take advantage of their features is now an integral part of network management [**Hel92**].

Network management incorporates many areas. Networks must be designed, planned out, installed, tested and then maintained and upgraded once the initial installation phases are complete. Network management is necessary at all levels of networking, from the physical layer, managing and maintaining the physical media that form the network, right up to the application layer and the software programs and processes that operate on the network.

1.1.1. SNMP

Network managers manage their network using software that allows them to monitor and control the network. Such software will allow a manager to garner statistics from routers and switches as well as hosts in the network. Software also allows control of these devices enabling a manager to change device configurations.

SNMP (Simple Network Management Protocol) is a commonly used protocol for network management. SNMP uses the client-server model incorporating agents and managers. An agent is a server that resides on any network node and contains performance information about that node. A manager is the client that retrieves information from the agents or changes some agent attribute. SNMP uses the fetch-store paradigm. This allows for just two commands; fetch, which retrieves information from the agent, and store which assigns some new value in the agent.

The agents store information in objects. Each object to which SNMP has access must have a unique name and be clearly defined. Collectively, the set of all objects that can be stored at an agent are described as a MIB (Management Information Base). The objects are accessed using ASN.1 (Abstract Syntax Notation One). SNMP does not define the objects stored in a MIB, but rather establishes a message format for communicating with a MIB.

1.2 Network Performance Monitoring

Network performance monitoring plays an important role in network management. Performance data can be used to spot potential or actual problems on the network. Trends may appear in performance data that could indicate long term problems. In the short term these may not give cause for fault alarms and consequently might remain undetected by network managers. Network performance monitoring allows such trends to be seen and dealt with before any serious problem occurs. Network performance data can be useful when analysing current problems as well, both in terms of characterising their nature and their impact.

Network performance data is also key in the planning and development of networks. Performance data allows network managers to assess the strengths and weaknesses of various network components and strategies and gives an up to date picture of how the network is utilised **[Cla96][Che00]**.

1.2.1. Hidden Failures

Interestingly, severe failures are often the easiest to diagnose and correct. Where connectivity is lost altogether, establishing the location of the problem, its cause and the

remedy may be relatively straightforward. In contrast, intermittent failures may be much harder to detect. Given that network protocols are designed to work with some measure of loss, hardware failures that occur only infrequently can remain hidden. Although the network hardware and protocol software contain procedures for dealing with errors (normally retransmission), network managers still work to detect underlying faults as they will inevitably have some impact on the overall network performance [**Com99**].

1.2.2. Improving Network Performance

When a network is performing badly the following loop (figure 1.2-1) is employed to attempt to improve performance. The relevant network parameters are measured, analysis is undertaken to try and understand what is taking place and then changes are made. This loop is repeated until either the performance is deemed good enough or alternatively there are no further improvements to be gained.

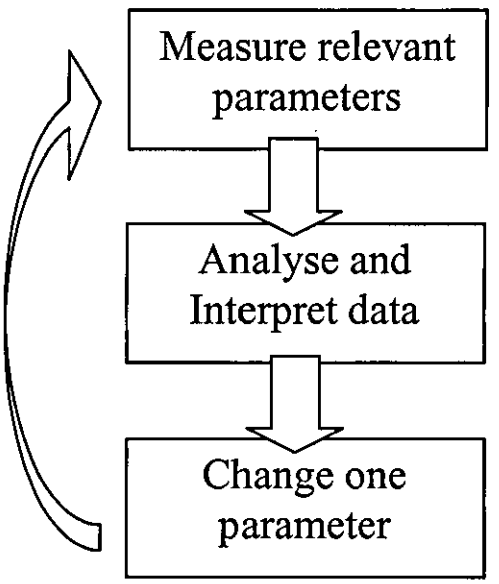


Figure 1.2-1 The Measurement Process

1.2.3. Performance Monitoring Techniques

There are three main techniques for evaluating network performance. These are analytical analysis, simulation and practical observation. Analytical models are mathematical representations, which relate system outputs to inputs by defining functional relationships between the two. The working of a network can be modelled to any desired level of detail if the necessary functional relationships are known.

Simulation is used when the model becomes so detailed that the analytical solution is too complicated. In these cases an experimental implementation of the network could be used but this would be a very expensive solution.

Analytical analysis and simulation both fall under the remit of predictive modelling. Predictive modelling can be very powerful and is necessary in designing and building a network. For the first implementation of a network no observed performance measures are available and so designers are completely dependent on predictive modelling techniques to evaluate likely performance. The limitations of predictive modelling stem from the simplifying assumptions that often need to be made, without which the problem can become intractable.

Once a network is built measurements can be made. Performance measurements are useful in that they monitor the actual, as opposed to the theoretical performance, thereby exposing design flaws and inefficiencies in the network in addition to effects due to user traffic.

1.2.4. Active and Passive Measurements

Performance measurements can be made in two ways, either actively (intrusively) or passively (non-intrusively). Active measurements involve injecting test traffic into a network for the specific purpose of monitoring some element (or elements) of that networks performance. Passive measurements make use of traffic already in the network and glean performance information from it. The advantage of active measurements over passive measurements is that test traffic is under the monitors control and therefore can be specified to any given parameters to provide the information required. The disadvantage is that by injecting test traffic into a network the traffic load is being increased and may potentially effect performance.

1.2.5. Performance Measures

1.2.5.1 Delay

The first performance measure we shall discuss is delay. Delay is one of the more fundamentally important measures of network performance and is also a central topic for

much of this thesis. The total amount of delay associated with information traversing a network can be thought of as compromising of four parts.

1.2.5.1.1 Propagation delay

This refers to that time which it takes for the signal to be sent along some physical medium, whether that be wire, fibre or even through the air. It is a fundamental delay that is essentially constant and characteristic of the medium.

1.2.5.1.2 Switching delay

This is introduced by electronic devices in the network such as routers, switches, bridges or hubs. This delay consists of a minimum element incurred due to the time it takes for all the bits of a packet to be received and then for the decision to be taken as to which destination the packet should be forwarded to.

1.2.5.1.3 Access delay

This arises when hosts have shared access to a network medium. An example would be a host on an Ethernet network that may have to back off and wait for the channel to become free before it is able to transmit. Similarly on a token ring network a host has to wait for the token before it can start to send packets.

1.2.5.1.4 Queuing Delay

This occurs when packets are waiting to be serviced at various points in the network. For instance, at a router where a significant amount of processing may be required to encapsulate and route packets, packets will be queued until they can be processed. Various queuing strategies are available to try and ensure a fair queuing system.

Often when delay measurements are required the Round Trip Time (RTT) is used as it can be easily obtained using tools such as ping. Round Trip Times refer to the total amount of time required for a test packet (normally an ICMP packet) to traverse across a network to some destination and to return again. The packet finally terminates at the

same host from which it was transmitted. This is advantageous as it eliminates any need for timing synchronisation. The same clock records the transmit time and the receive time.

Measuring one-way delay is difficult, as timing synchronisation is required. **[Sid89]** Some performance monitoring protagonists have argued that RTT is more than adequate **[Han00]** and that there is little to be gained by measuring end to end delay, given the complexities involved in achieving those measurements. However others have pointed out that network paths can be asynchronous, meaning that the respective delays encountered by outward bound packets and returned packets can be very different **[Cla93]**. In these cases, RTT could be misleading. Later discussion will look at performance monitoring projects that do measure end to end delay using GPS to give timing synchronisation.

1.2.5.2 Throughput

The second fundamental property of the network is throughput. This refers to the amount of data that can be sent through the network, or a section of the network. Throughput is generally measured in bits per second (or Megabits/sec or even Gigabits/sec). Another term that is used to describe throughput is bandwidth. As bandwidth or throughput are sometimes referred to as the speed of a network it is important to be clear that throughput and delay are distinct measures. The throughput describes the amount of data that can be put onto a link, the delay describes the amount of time the data will take to reach the destination. Comer **[Com99]** gives the example of a road that can accept one car every five seconds (throughput 0.2 cars/sec). The road has a delay of 30 seconds. Now if an extra lane was added to the road then two cars could join the road every five seconds (throughput has doubled to 0.4 cars/sec) but the delay on the road would still be 30 seconds.

Although throughput and delay are different measures and describe different aspects of network performance the two are associated. Congestion occurs on a network where the rate of traffic being sent over the network exceeds the throughput rate. A congested network results in packets being queued before they can be sent. It should be clear that this queuing time will affect the overall delay of the packet.

1.2.5.3 Jitter

Jitter (or delay variation) is the term used to describe the variance in delay. Jitter is of particular relevance to real time applications such as video or audio streaming software. In real time applications it's not enough for the mean delay to be low, the standard deviation also needs to be small so that there is no noticeable break in communication.

1.2.5.4 Loss / Errors

Although loss and packets containing errors are on the whole invisible to a user as they are generally dealt with by the hardware and software protocols that facilitate the network, they are still a very important network measure. Where packets contain errors or are lost they will often require retransmission. The more packets that are lost or corrupt the greater the inefficiency of the network.

1.3 Current Research / Existing Tools

There is a feeling among the academic community that Network Service Providers do not collect or make available sufficient performance measures concerning their networks **[Cla96][Pax98c][Che00]**. There seems to be two main reasons for this reluctance to monitor networks more comprehensively. The first difficulty is that no one seems too sure what to monitor and what statistics to collect. The technology involved is still in a fast evolving state and there is a suggestion that network service providers do not know with any great confidence what information would be of use to them and what would not. The second issue is simply that of priority. Enough effort is required in getting and keeping network infrastructure operational that performance statistics have been kept as a secondary concern. **[Cla96]** The issue is not so much whether generating performance data would be useful, rather that there is no spare time or clear direction to bring it about.

Despite the seeming lack of impetus, there is a growing belief that network performance information is important. The motivating factors come from both within and outside the telecommunication companies. Performance data is important for identifying the causes to network faults and for solving them. Detailed information allows faults to be analysed and understood. Performance data also allows trend analysis,

potentially allowing managers to identify hot spots before they occur and instigating preventative measures. These are good arguments for telecommunications companies to take the initiative. Pressure is also growing from external bodies. As networks are increasingly employed as a means of communication, users will want performance guarantees. There is also the introduction of government legislation by bodies such as OFTEL (UK) [**OFTEL**] ensuring that certain standards are met. Without performance measurements there is no means of satisfying these various parties that standards are being maintained.

1.3.1. Metrics and Methodologies

One of the areas being discussed presently is which performance measures to collect. There are a number of statistics that can be generated, some probably of greater use than others. Even where we know what statistics we want they still need to be carefully defined; so for example, what one person/organisation refers to as one-way delay is not misunderstood. For instance, when measuring delay, at what point does the 'clock' start counting? Ideally, these definitions would be standardised in such a way that a common set of measurements (or metrics) could be used universally [**Pax96**].

Methodologies are the means of obtaining metrics. Metrics can be defined where no methodology currently exists for obtaining them. Conversely there may be many methodologies for obtaining the same metric. It may be that there are known flaws in methodologies where the desired metric cannot be measured accurately but a good indication still obtained. This may still be useful but needs to be documented. IPPM (IP Performance Metrics) which is part of the IETF (Internet Engineering Task Force) have undertaken to draw up standards and ratify new metrics and methodologies as appropriate.

1.3.2. Performance Monitoring - Measurement Tools Overview

There are many measurement tools available, in addition to a quick look at ping and traceroute, we consider five here. These have been chosen as they are similar to that used in this project in that they measure similar metrics, namely one-way-delay (all except ping) and loss.

1.3.2.1 ICMP, Ping and Traceroute

The Internet Control Message Protocol (ICMP) is used to report network events. There are about a dozen types of ICMP messages and they are transported in IP packets. Some of the more important messages are listed in the table below (figure 1.3-1).

Message Type	Description
Destination Unreachable	Packet could not be delivered
Time Exceeded	Time to live field hit 0
Parameter problem	Invalid header field
Source quench	Choke packet
Redirect	Teach a router about geography
Echo request	Ask a machine if it is alive
Echo reply	Yes I am alive
Timestamp request	Same as Echo request, but with timestamp
Timestamp reply	Same as Echo reply, but with timestamp

Figure 1.3-1ICMP Messages

The ping program makes use of ICMP Echo request and Echo reply messages (or sometimes Timestamp request and Timestamp reply messages) to either test connectivity or round trip time. Ping is widely available and comes as standard under operating systems such as Microsoft Windows (95, 98, NT, 2000) and Linux. Ping was originally written by Mike Muuss.

Traceroute, developed by Van Jacobson, also uses ICMP. Unlike ping, traceroute outputs all the intermediate router hops it takes to reach the specified destination. Below is output taken from both ping (figure 1.3-2) and traceroute (figure 1.3-3).

```
PING www.bbc.net.uk (212.58.224.31) from 158.125.51.167 : 56(84) bytes of data.
64 bytes from www1.thdo.bbc.co.uk (212.58.224.31): icmp_seq=0 ttl=247 time=28.120 msec
64 bytes from www1.thdo.bbc.co.uk (212.58.224.31): icmp_seq=1 ttl=247 time=32.308 msec
64 bytes from www1.thdo.bbc.co.uk (212.58.224.31): icmp_seq=2 ttl=247 time=20.507 msec
64 bytes from www1.thdo.bbc.co.uk (212.58.224.31): icmp_seq=3 ttl=247 time=41.589 msec
64 bytes from www1.thdo.bbc.co.uk (212.58.224.31): icmp_seq=4 ttl=247 time=36.604 msec
64 bytes from www1.thdo.bbc.co.uk (212.58.224.31): icmp_seq=5 ttl=247 time=32.061 msec
64 bytes from www1.thdo.bbc.co.uk (212.58.224.31): icmp_seq=6 ttl=247 time=30.666 msec
64 bytes from www1.thdo.bbc.co.uk (212.58.224.31): icmp_seq=7 ttl=247 time=28.275 msec

--- www.bbc.net.uk ping statistics ---
8 packets transmitted, 8 packets received, 0% packet loss
round-trip min/avg/max/mdev = 20.507/31.266/41.589/5.829 ms
```

Figure 1.3-2 Ping Output (**ping www.bbc.co.uk**)

```
1 el-gateway-50.lut.ac.uk (158.125.50.1) 0.435 ms 0.318 ms 0.294 ms
2 emman-gw.lut.ac.uk (158.125.8.1) 0.987 ms 0.965 ms 1.024 ms
3 uon2-gw-v1.emman.net (194.82.121.130) 4.411 ms 6.112 ms 7.201 ms
4 uon1-gw-6.emman.net (194.82.121.42) 12.348 ms 10.344 ms 12.524 ms
5 ce-gw.ja.net (146.97.255.21) 20.145 ms 20.373 ms 23.826 ms
6 ext-gw4.ja.net (193.62.157.113) 21.743 ms 22.255 ms 23.573 ms
7 linx-gw.ja.net (193.63.94.249) 15.645 ms 22.558 ms 18.422 ms
8 rt-linx-b.thdo.bbc.co.uk (195.66.225.103) 20.297 ms 25.060 ms 26.070 ms
9  www1.thdo.bbc.co.uk (212.58.224.31) 34.144 ms * *
```

Figure 1.3-3 Traceroute Output (**traceroute www.bbc.co.uk**)

1.3.3. Ping based tools

Ping has been adapted and used by other Network Measurement systems. Although ping itself is a well-established tool it's still being used for research as part of larger schemes. Internet weather reports are provided by sites that poll other sites using ping on a regular basis and then assess the networks performance using the delay data gathered by these measurements. In the figure below (figure1.3-4), taken from Andover News Network ITR site (Internet Traffic Report) [**IntTR**], response time is plotted against time. The response time reflects ping results from several servers that the site uses to reflect the performance of the Internet.

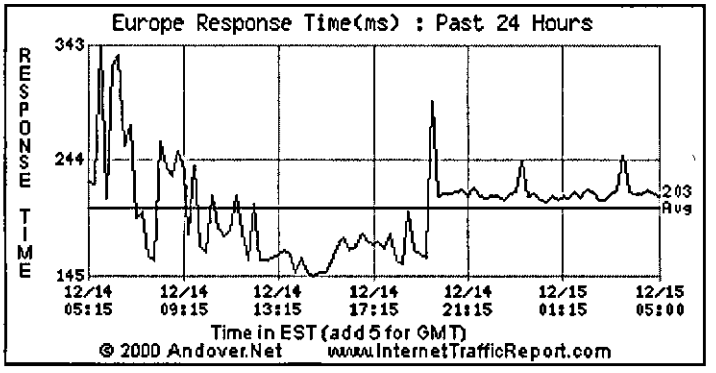


Figure 1.3-4 Response time graph from an Internet Weather site [IntTR]

1.3.4. NIMI

The NIMI (National Internet Measurement Infrastructure) project is a network measurement architecture based in the US [**Pax98b**][**Pax98c**]. The key focus of this project is to create a measurement infrastructure capable of monitoring very large networks. NIMI makes use of stations set as end-points for a set of measurements therefore allowing for end-to-end metrics to be measured. NIMI uses Traceroute

[**Jac89**], Treno [**Mat96**] and Poip (Poisson Ping) but is designed to allow for any tool to be used as part of its measurement suite. The NIMI project hopes to move towards diagnosis of problems by deploying probes at various points along the monitored network paths. This will allow problem areas to be identified more specifically.

1.3.5. Surveyor

Surveyor [**Kal98**] measures one-way delay and loss by sending UDP test packets (40 bytes including IP and UDP headers) between measurement test stations distributed around the Internet. Timing accuracy is achieved using GPS antenna. Test packets are sent, on average, every two seconds and then the results are summarised over a one-minute period. The summary statistics used include centile values for average delay and percentage loss. Surveyor was making use of some 38 machines in November 1998 having started with just 3. These are mainly at sites in the US but Europe is now included.

1.3.6. RIPE

The RIPE project [**Uij97**][**Uij98**] is similar to Surveyor in that it also uses active testing with GPS clocks for time synchronisation. The work is centred in Amsterdam and the testing is conducted mainly across Europe. Again like Surveyor, RIPE transmits packets using a Poisson sampling rate to avoid synchronisation between the test traffic and other events on the network. [**Flo94**]

1.3.7. NLANR – AMP

The National Laboratory for Applied Network Research (NLANR) has been working on creating a Network Analysis Infrastructure (NAI) [**McG00**]. This encompasses a variety of performance measures including a passive monitoring project and an active measurement project. NLANR has also been working on the collection of network management and control data. The Active Measurement Program (AMP) measures round trip time, topology and loss. They have deployed over 100 AMP monitors around the high-performance research networks in the United States (figure 1.3-5).

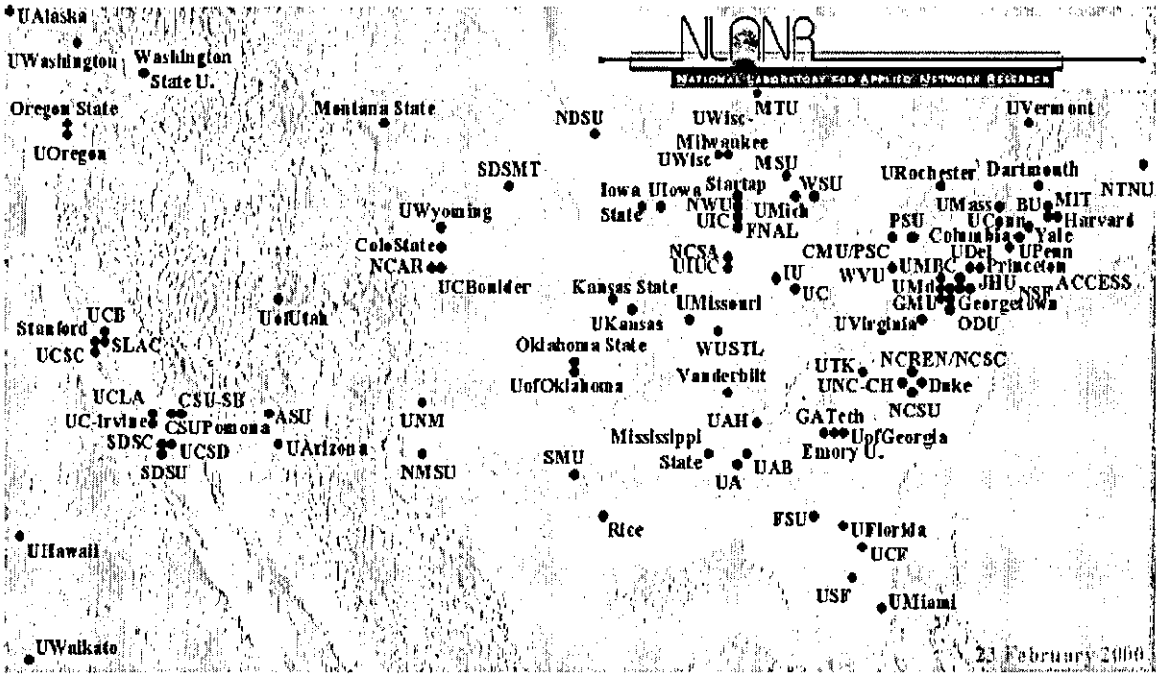


Figure 1.3-5 NLANR AMP Monitors [McG00]

1.4 Summary

In this chapter we have looked at the broad area of network management and given particular consideration to the role of network performance monitoring with that. Network performance monitoring forms an important part of network management, its characteristics include delay, throughput, jitter and loss. We've seen that One-Way Delay can be measured using intrusive measurements although some form of timing synchronisation is necessary. Examples of one-way delay measurement schemes are the RIPE project and the Surveyor project. Round Trip Time can also be measured using tools like ping. A monitoring scheme that measures RTT is the NLANR AMP project.

In chapter two we go on to look more specifically at performance monitoring work undertaken at Loughborough which provides the setting and background of the work discussed in this thesis.

Chapter 2 - Performance Monitoring at Loughborough

2. Performance Monitoring at Loughborough

2.1 Introduction

In chapter one a broad introduction was given to network management, network performance monitoring and specifically monitoring tools that measure delay. In this chapter we look more specifically at work conducted in this area at Loughborough University as this gives important background information and puts into context the purpose and aims of this thesis.

2.1.1. Background to Data Exception Detection

Researchers at Loughborough University have been engaged in the area of network performance monitoring for a number of years. Much of the work provides useful background to this thesis, as it should give a clear understanding of the motivation behind and the need for a means to detect Data Exceptions.

2.1.2. Chapter Overview

As has previously been mentioned, measuring one-way delay is a non-trivial task. Work has been carried out at Loughborough to develop a performance monitoring tool capable of measuring one-way delay and loss as part of the BT funded URI project. Two architectures are discussed here: The Walsall Test Architecture and the Portable test system. The Walsall test architecture is used to continuously monitor the SMDS network, providing an 'ever present' source of information. The portable test system is used for more specific, short-term tests (hence the portability for ease of installation). In both cases the aim is to indicate the type of performance experienced by the end user (i.e. one-way delay). This is particularly relevant when considering service level agreements that may exist between a network provider and their clients.

In addition to these test architectures a monitoring system, AIR (Automatic Incident Reporting) is also described. AIR incorporates several aspects of the monitoring process of which Data Exception Detection is one, hence the pertinence of the discussion to this thesis.

2.2 *Walsall Test Architecture*

2.2.1. Purpose and History

The Walsall Test Architecture was assembled and installed following the design and implementation of a unidirectional latency and loss tester, developed at Loughborough. The system was originally set in use, monitoring BT's SMDS network, in 1995. The purpose was to provide BT with performance measurements previously unavailable to them, namely one-way delay. This enabled BT to demonstrate their compliance with Service Level Agreements negotiated with their customers [**Phi95**][**Phi96**].

2.2.2. Physical Components and Layout

The architecture (figure 2.2-1) is made up of several components. Chiefly, there are three monitor stations based at a BT Network Operation Centre (NOC) in Walsall, an alarm station (also based at Walsall) and a control station based at Loughborough. Of the three monitor stations one is used as the primary test station, one is used to run focus tests and the third is an emergency backup. The test equipment is connected to the SMDS network at five separate locations via Megastream links. These locations are Birmingham, Bristol, Edinburgh, London and Manchester. An SMDS connection links the control station to the test stations.

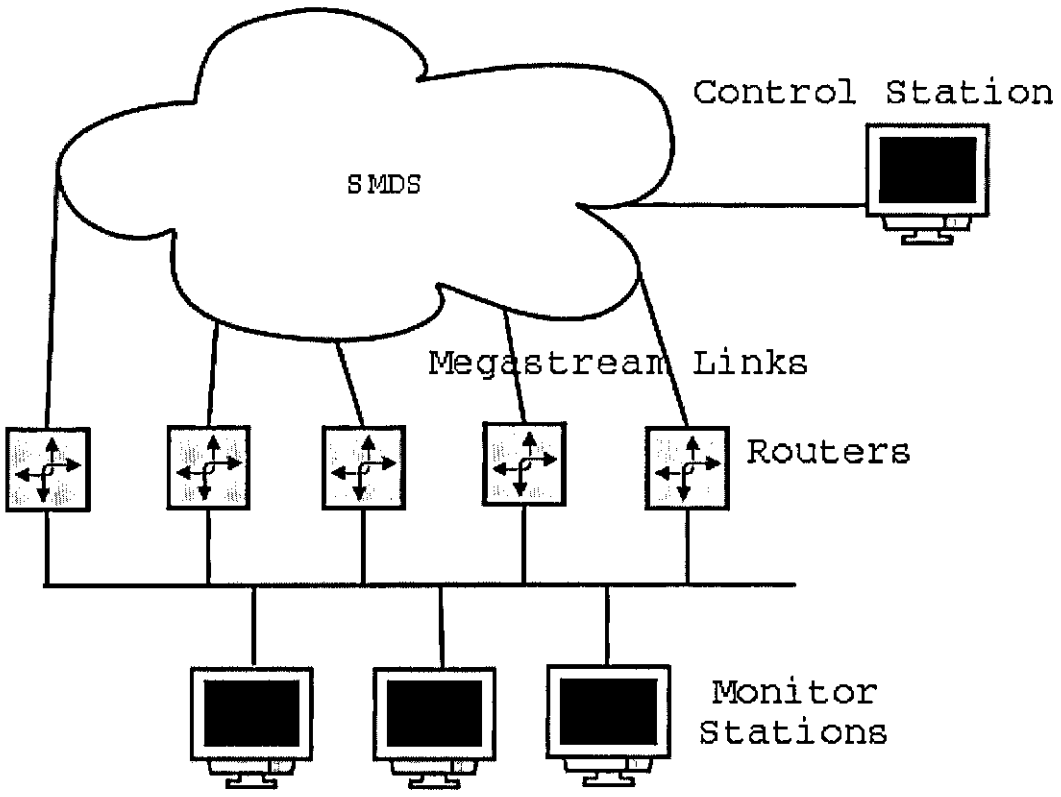


Figure 2.2-1 The Walsall Test Architecture

2.2.3. Testing Strategy

The test points provide twenty one-way paths across SMDS. Test packets are sent across the SMDS network, testing each route in turn with both small (64 Byte) and large (1500 Byte) test packets. These represent the smallest and the largest packet sizes available to us. Test packets are sent at a rate of one per second meaning that each test (individual packet size and route) is tested every forty seconds.

No clock synchronisation is required as the same machine logs the transmit time and the receive time. The only variable delay here is in the SMDS network itself; other delays involved are either negligible (e.g. propagation along the Ethernet at Walsall) or fixed (e.g. the megastream links). The access network is reserved solely for monitoring purposes. If this were not the case then the tests would be affected by traffic on the access network.

Loopback tests were conducted in conjunction with BT to measure the delay attributable to the non-SMDS element of the test circuit. By subtracting this figure from

the total delay we obtain a figure that solely represents the delay incurred traversing SMDS.

2.2.4. Data Storage and Processing

The test stations log packet transmit and receive messages in log files stored on their local hard drives. These files are retrieved to Loughborough typically twice a week. The data is then processed to calculate delay and loss information and stored in a database at Loughborough. The database uses a concept called intermediate information (figure 2.2-2). This essentially gives a speed up in response time as a trade off against storage space [**Bas98**].

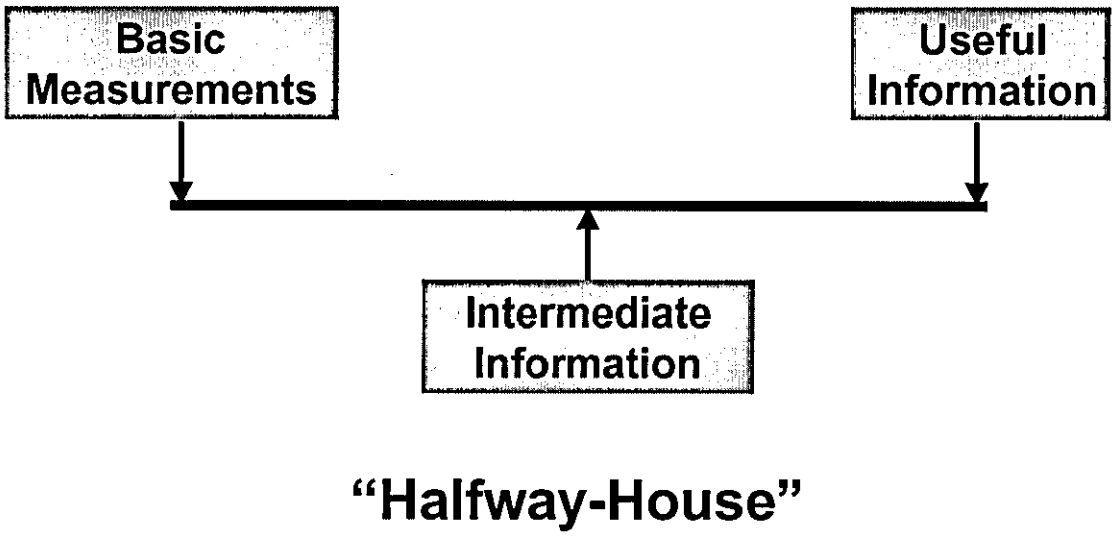


Figure 2.2-2 Intermediate Information

2.2.5. Data Reporting

From the monitoring information, delay graphs are produced that show 50th, 95th and 99th centile values, typically over a three hour window. These graphs are examined for anomalies (Data Exceptions) and where such anomalies are found a report is produced describing the change. These reports are generated historically and are produced on a weekly basis. In addition to the weekly reports the alarm station, based at Walsall, displays up to date performance information and also has some simplistic thresholds designed to give warning of any serious performance impairment [**Pag99**].

2.3 Portable Test Architecture

2.3.1. Purpose and History

As has been mentioned earlier end to end delay consists of two parts, access network delay and core network delay. Where end to end performance has degraded it might be due to either the access network or the core network or both. The portable test system was designed to measure end to end delay between two arbitrary points in the network and also to measure the delay on the access networks involved.

In practice the portable test system has been used to measure links between points within the core network to give benchmark figures for delay.

2.3.2. Physical Components and layout

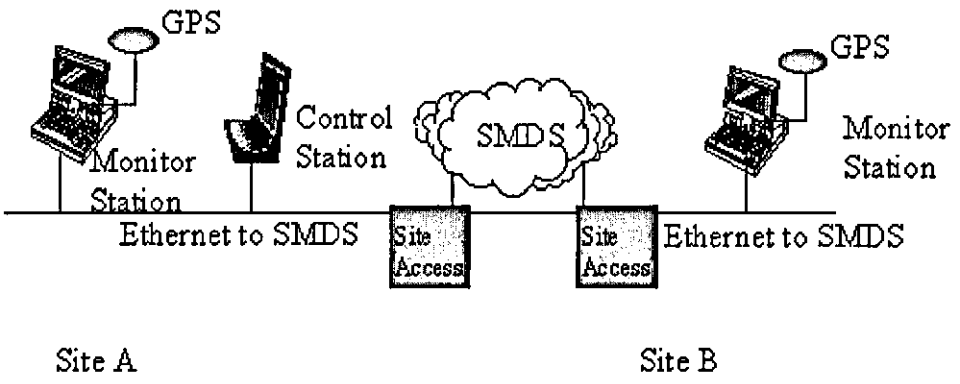


Figure 2.3-1 The Portable Test Architecture

The portable test system (figure 2.3-1) makes use of two test stations each receiving a timing pulse via GPS antenna. A laptop computer running Solaris is then used as the control station. The log files from the portable test stations can be retrieved to the control station where they can be stored and processed.

2.3.3. Testing Strategy

As was mentioned above, in addition to end to end delay measurements, the local site access is monitored. This is done by sending a test packet to the local gateway and back again. Hence there are four tests, one for each site access and two tests traversing the core network, one in either direction. The size of the test packets and the rate at which they are sent can be tailored to suit the specific requirements of the test.

2.3.4. Data Storage, Processing and Reporting

Data storage, processing and reporting is done in a similar manner to that described under the Walsall Test System. The control station for the portable system is nominally a laptop that can be taken to a remote site, however any Solaris based machine that can access the test stations remotely could be used as a control station.

2.4 The Automatic Incident Reporting system (AIR)

2.4.1. Purpose and history

Given that some monitoring information exists, further work can be done to collect, analyse and present this information so that it can be interpreted and if necessary, action can be taken. The AIR system addressed these issues with the specific intention of automating the process as far as possible **[Phi99][PhiSan00]**.

2.4.2. Layered Approach

Jain describes the layered approach referred to here (figure 2.4-1) **[Jai91]**. In his model there are seven layers, of which the top three generally involve some human element.

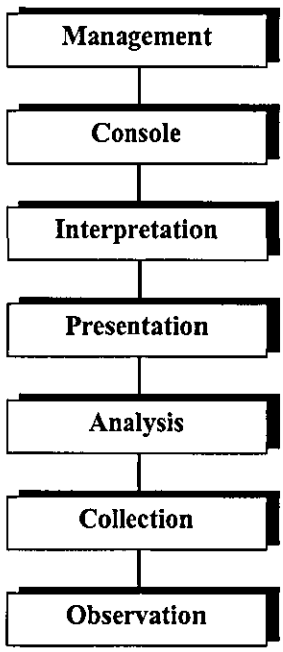


Figure 2.4-1 Jain Layered Approach [Jai91]

The observation layer gathers raw data on individual components of the system, in this case a network. These then need to be collected and analysed. Analysis at this stage may simply involve calculating statistical summaries; thresholds might also be used to trigger alarms. It is in this layer that Data Exception techniques might be implemented. The information summarised by the analysis layer then needs to be presented; this can be done in the form of reports, displays and alarms. The information must then be interpreted; this is generally done by human beings or perhaps an expert system. The console layer is the means by which managers can interface with and control the network, allowing the management layer, at the top, to make decisions regarding any action that may be required.

2.4.3. Observation

The AIR system was designed to incorporate any monitoring information source that happened to be available. The observation layer could therefore include many types of monitors. In practice two monitors have been used, the delay and loss monitors described above in association with the Walsall Test Architecture and also an ATM tester, that also monitors delay and loss, developed at BT's research laboratories at Adastrel Park, Martlesham.

2.4.4. Collection

The data is stored in log files on the monitor stations and then retrieved on a regular basis. It is stored in a database on a control station in a similar manner to that discussed in the previous two sections.

2.4.5. Analysis

Analysis of the data included the calculation of statistical summaries and then the application of a rule base to establish whether any Data Exceptions, that is anomalies in the data, had occurred. Data Exceptions will be discussed more fully in the next chapter and the rule-based approach used here is described in detail in chapter 5.

2.4.6. Presentation

Data Exceptions are gathered together in data collections that are then stored in a database which can be accessed using a viewer (figure 2.4-2).

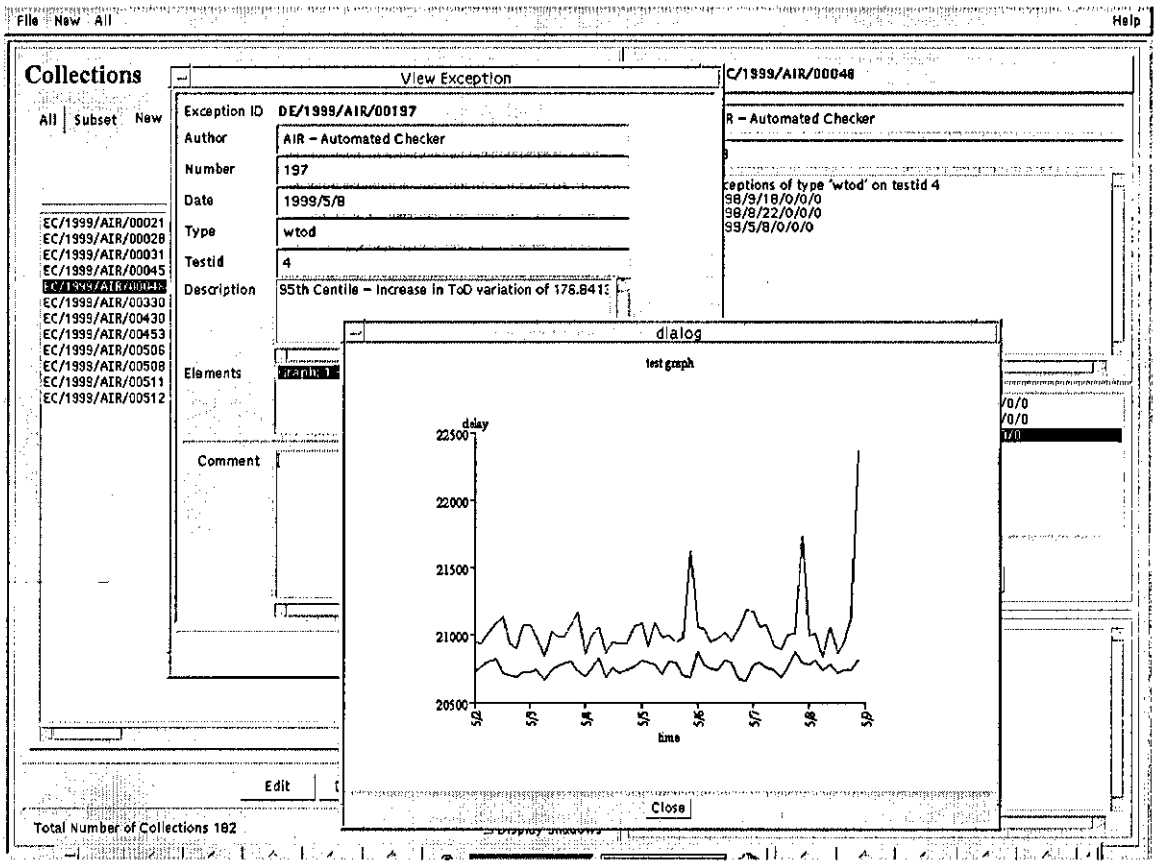


Figure 2.4-2 The AIR database viewer (screen shot)

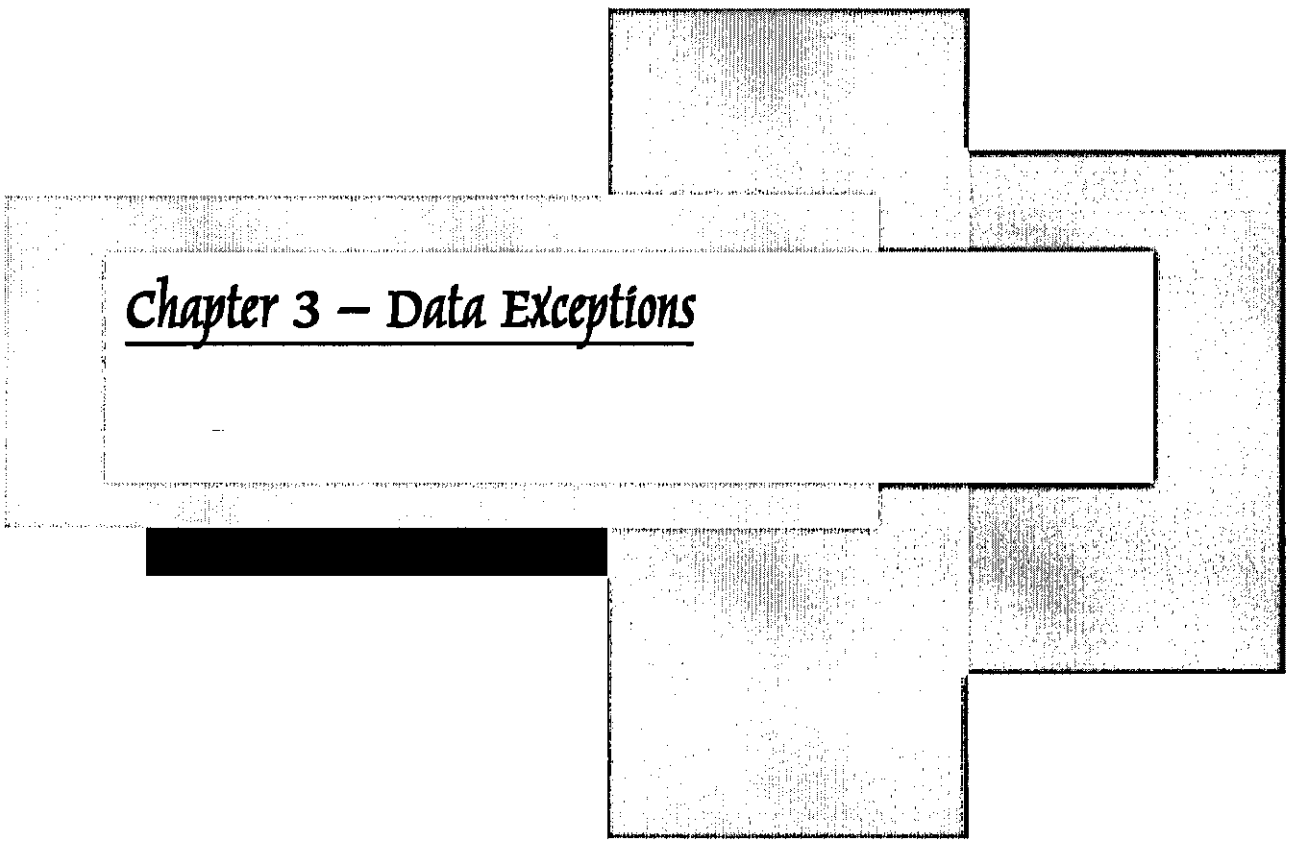
AIR executes these processes automatically. By collecting together all relevant Data Exceptions the aim is to make the interpretation of the data as simple a task as possible. This might even lead to the automation of this layer although no work has been attempted in this area. The top three layers are not addressed by AIR.

2.5 Summary

In this chapter we have looked specifically at research undertaken at Loughborough. Relevant work has led to two test architectures, (The Walsall Test Architecture and The Portable Test Architecture) and a monitoring system (The AIR system). This thesis aims to detect Data Exceptions in the delay data observed by monitor stations such as those discussed in the two test architectures. These Data

Exception detection techniques may then be used by a system such as AIR as part of a complete network management system.

In chapter three we discuss Data Exceptions in more detail. Examples of Data Exceptions are presented and their causes considered.



Chapter 3 – Data Exceptions

3. Data Exceptions

In this chapter the concept of Data Exceptions is discussed in more detail. Although the concept is not novel or exclusive to this piece of work, very little has been written describing the nature of Data Exceptions. In this chapter examples are given and the causes of the different types of Data Exception are discussed.

3.1 What are Data Exceptions?

A given network has an associated set of network performance characteristics **[Pax98]**. These may include any metric, which in some way characterises the performance of that particular network. These characteristics may be expressed as some kind of fixed value or alternatively as a distribution of values. Therefore the set of network performance characteristics for a network could include the minimum possible delay between two points on the network which would be expressed as a fixed value. Alternatively, a performance characteristic could be the observed throughput along a certain link, which would be expressed as a distribution of values.

An important assumption here is that these performance characteristics don't change except where the state of the network that they characterise changes. For this assumption to hold we must take in to account the usage characteristics of a network, as the way a network is used affects many network performance characteristics **[Hoo97]**. The use of a network may change without its state (in a physical sense) changing in any way. However for the purposes of considering and detecting Data Exceptions, the general network use will be included as part of the network state. If a network user were to introduce an excessively large volume of traffic so that it significantly altered the network performance this would be thought of as a change in network state.

Given that the underlying performance characteristics remain constant so long as the network is in an unchanged state, we now define Data Exceptions to be data that reflect a change in these underlying performance characteristics and therefore a change in the network. Data Exceptions are exceptions from the expected performance characteristics that define a given networks performance at a given time.

As was alluded to above there are changes in performance that are brought about by the changes in network use. Certain usage patterns are included in our definition of network state as they are consistent and predictable. Network usage on many networks increases during working hours of the day leading to Time of Day Delay Variation (which is discussed further on in this chapter). Further to the variation in delay observed during the day a similar principle is applicable to weekends where less variation in delay is observed than during the week. These changes are not Data Exceptions as they are expected and are included as part of the network state.

Data Exceptions reflect some change in network performance due to some change in network state. This may be due to any kind of network alteration, not just network faults. Data Exceptions could include planned works, upgrades, reconfigurations as well as unforeseen errors.

Data Exceptions could be found in any data source. This thesis deals specifically with one-way delay but there are many other metrics that are being actively monitored **[Alm99a][Alm99b][Pax98a]**. Data Exceptions, that is data that differs in some way from that which was expected, could be detected in any of these sources.

As there are often many different monitoring tools monitoring one network, a change in the network could trigger many Data Exceptions. Likewise, one monitoring tool measures many different paths or nodes on a network. Again, one network event might cause Data Exceptions in more than one of these measurements.

Where there are multiple Data Exceptions all referring to the same network event it would be useful to have some automated means of collecting these together. Although this is outside the remit of this thesis a basic means of collecting Data Exceptions together is discussed at the end of this chapter.

3.2 Examples of Data Exceptions

3.2.1. Step Change

A commonly observed Data Exception, known as a 'Step Change', occurs when the average delay either increases or decreases in such a way that all test packets are

affected (see figure 3.2-1 for an example). That is to say that delay is altered by a constant amount for each packet. Step changes can be any size but, as they become very small it becomes hard to be certain whether or not the change is a genuine step change or not. Step changes can occur when the network is reconfigured. Packets may travel over a different route with different associated latency or the same route might be upgraded (or even downgraded) in some way. **[Mat00][PhiSan00]**

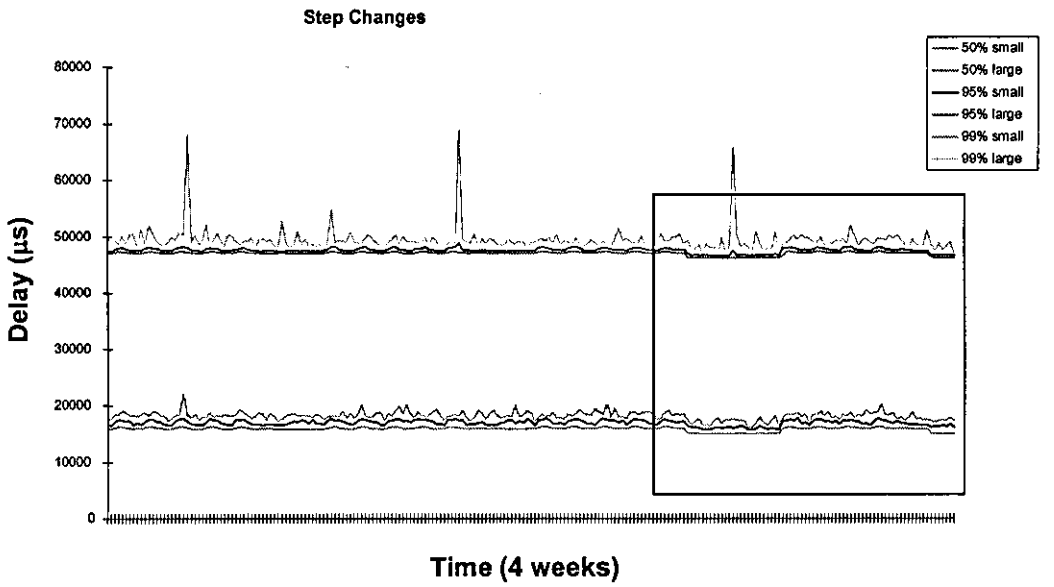


Figure 3.2-1 A Series of Step Changes

3.2.2. Time of Day Delay Variation Changes

As can be seen from the graphs, delay varies according to the time of day or to be more precise delay fluctuates according to the amount of loading on the network, which follows daily patterns. Therefore delay tends to be higher during the working hours of the day than at night or at the weekends. Other work has shown that network traffic has self-similar properties **[Lel94]**. This seemingly contradicts the notion of load varying according to some pattern, however the time scales in which the fractal-like nature of network traffic has been observed are considerably shorter than those being dealt with here. Leland himself notes the possible presence of a time of day cycle **[Lel94]** when the time scale is lengthened. This trend is extended further to bank holidays where again a difference in delay is noticeable. This effect has been termed 'Time of Day Delay

Variation' **[Phi96]**. Changes in Time of Day Delay Variation can occur when a link becomes more reactive to high load. That is to say that during the working period of the day, the effect of the high load on the network is more severe than previously. Another potential cause for a Time of Day Delay Variation change could be a significant increase in load due to client activity. Changes in Time of Day Delay Variation can also result from reconfiguring the network and it is not uncommon for step changes and changes in time of day variation to occur in tandem. Figure 3.2-2 shows four weeks of delay data. During the week delay increases during the working hours of the day. On the fourth week this additional delay increases.

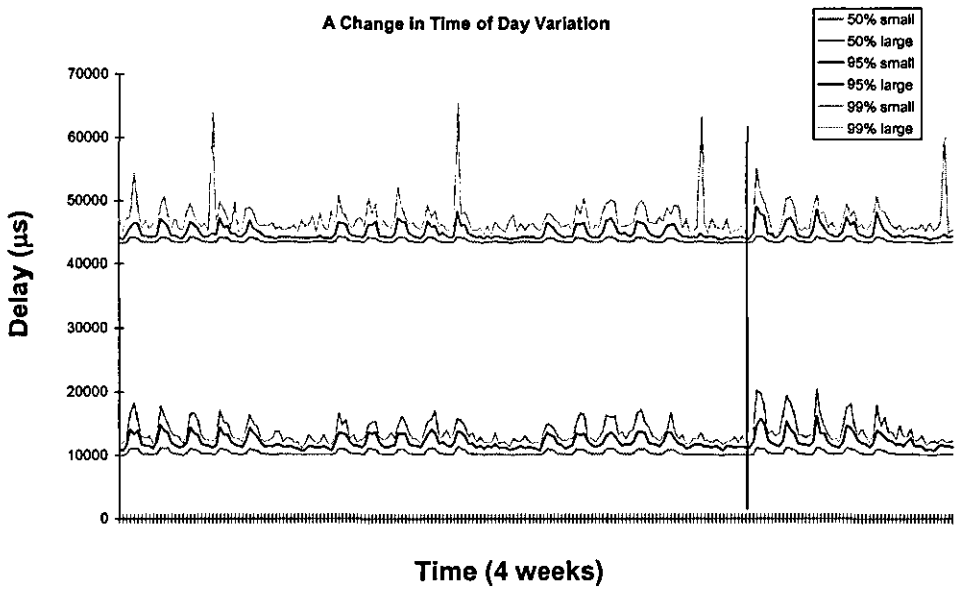


Figure 3.2-2 A Change in Time of Day Variation

3.2.3. Loss

Loss exceptions can take two forms, a continuous break or a period of high loss. Although this work is primarily concerned with detecting changes in delay, the following could be defined as loss Data Exceptions. If several packets are lost consecutively then we infer that a *short break* has occurred. Without testing continuously it is impossible to be certain that this is true but statistically speaking, if every packet sent over, say, a twenty minute period (30 packets) is lost then there is a high probability that a short break

has occurred. Alternatively if a high proportion of loss is observed over a specific period then this could be reported as a *high proportion* of loss. By high proportion this might be anything from 5% of packets upwards.

3.2.4. Delay Spikes

Spikes are sharp increases in delay that last for a relatively short period of time (one to two hours). The increase need not affect all the test packets although the more test packets it affects the more significant it is likely to be. Spikes can be of any size although the spikes need to be clearly observable above fluctuation in delay caused by load. Figure 3.2-3 shows an example of a delay spike.

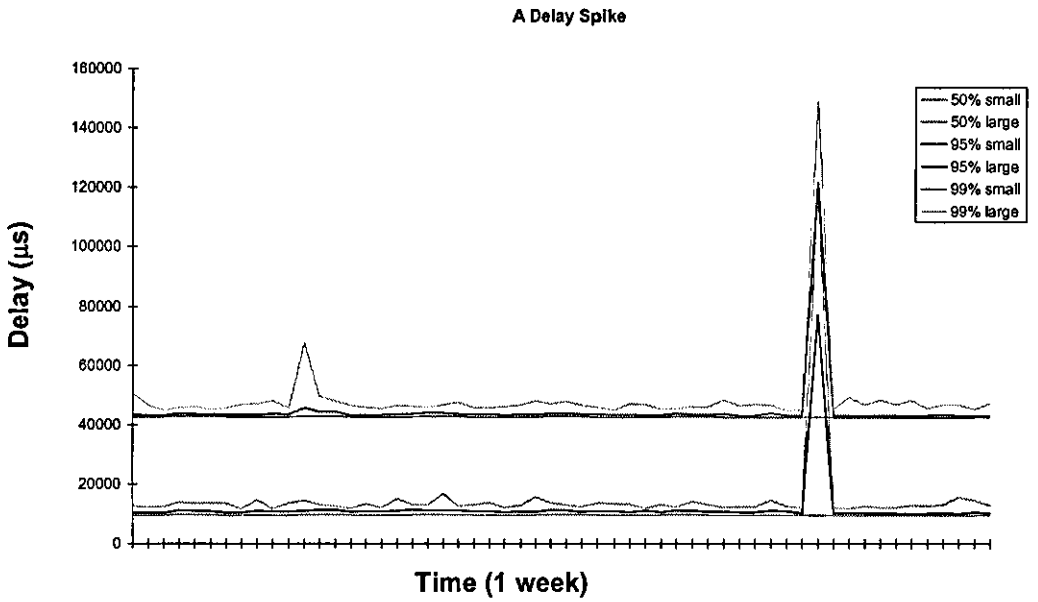


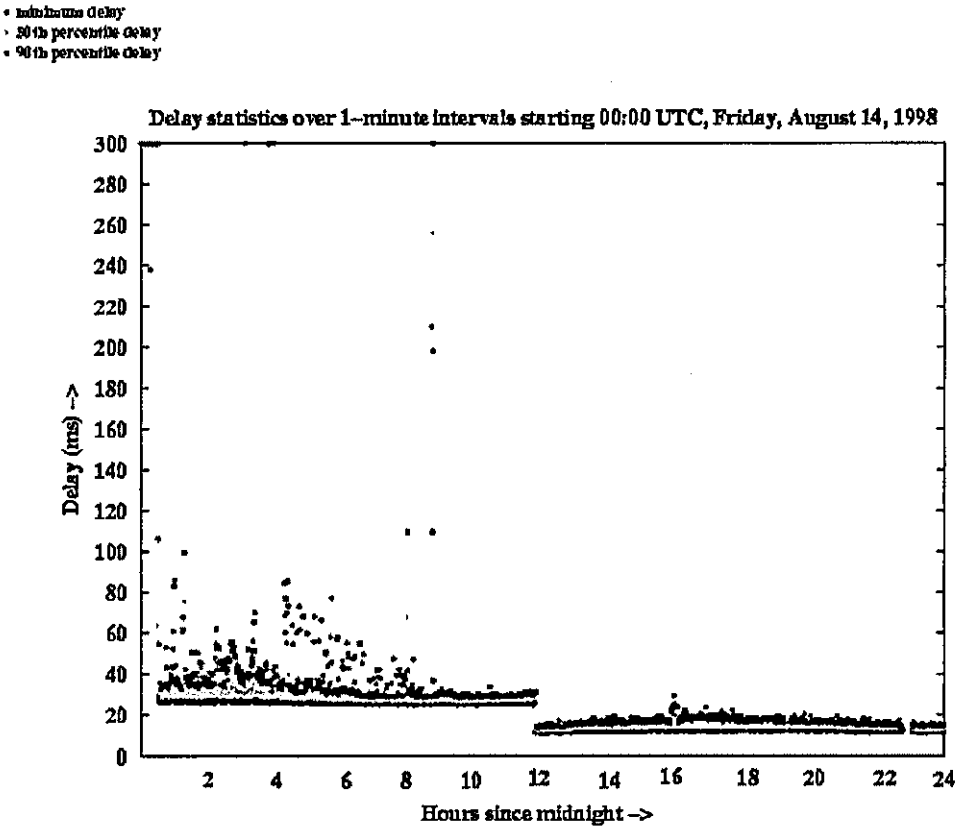
Figure 3.2-3 A delay spike

Other researchers have observed examples of Data Exceptions. Although the terminology is not standardised, phenomena such as step changes, 'time of day' variation changes and spikes have been noted by the researchers working on the AMP project as well as the Surveyor project. **[McG00][Kal98]**

This example of a step change (figure 3.2-3) comes from the Surveyor project.

"[This Graph] shows a routing change between two sites. The two sites are in the eastern US, so the change took place in the early morning, and is represented by a discontinuity in the delay. The minimum delay

along this path dropped by about ten milliseconds. The receiving site is multi-homed, and changed the way its network was advertised, resulting in the routing change."



Another example of a step change can be seen in the following graph (figure 3.2-5) taken from data gathered by the Ripe project. **[Uij98]**

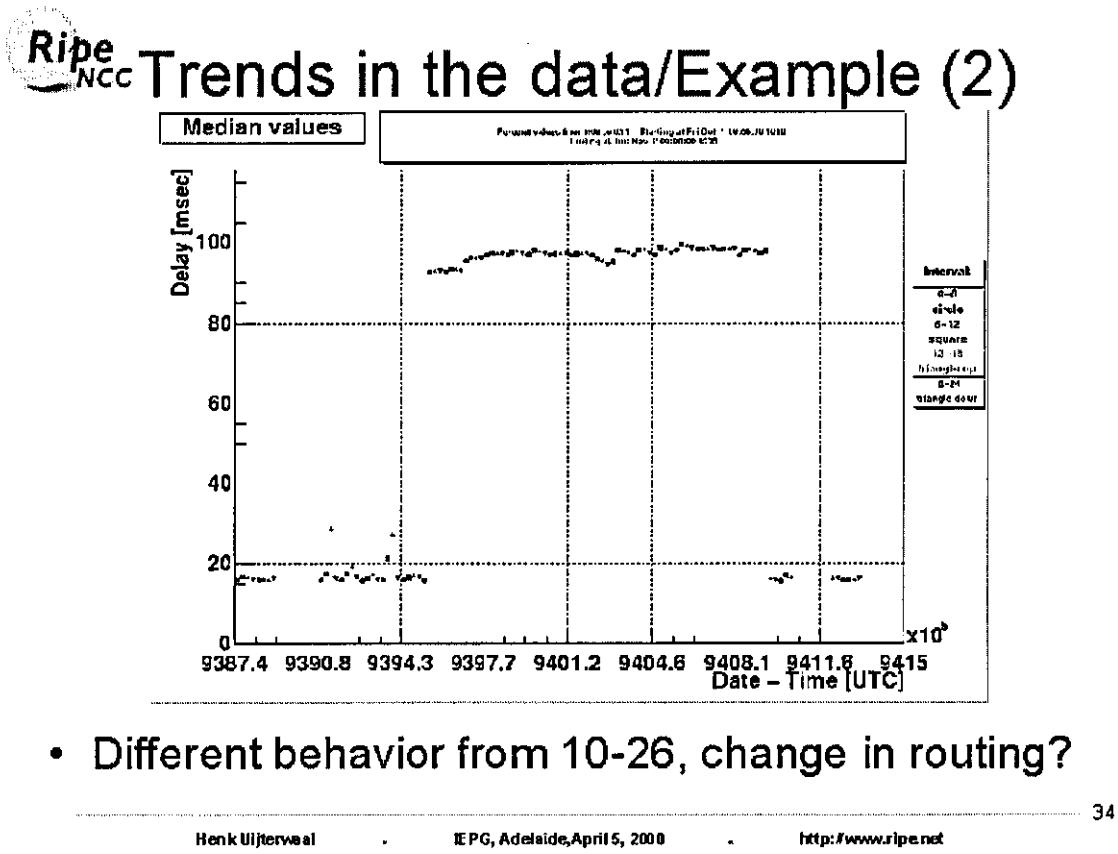


Figure 3.2-5 Step change example from Ripe [Uij98]

Not all network delay measurement schemes will show time of day variation in delay. A key difference between the research work described in this thesis and the other measurement projects discussed in chapter 1 is that the Loughborough work focuses on delay across a single network, owned by a single network provider. This allows the performance data to be linked to specific network events and gives greater ability to identify problem areas. Other network monitoring projects have been investigating delay across the internet thereby potentially spanning several network providers, usage patterns and even time zones which would make the time of day effect very much more complex. Another related key difference is that many of the other measurement projects being undertaken access the internet via a local network. Often additional delay incurred as a direct result of the access network is far higher than the delay incurred from network providers core network. The problem here is that the results from a measurement scheme accessing the internet via a local network (a campus network for instance as is often the

case for Surveyor) may be so dominated by that access network to make it difficult to infer much about the performance of anything other than the access network itself.

This following example (figure 3.2-6) from the AMP project shows two graphs, one of loss measurements, one of delay measurements. In the second graph time of day variation in delay is visible.

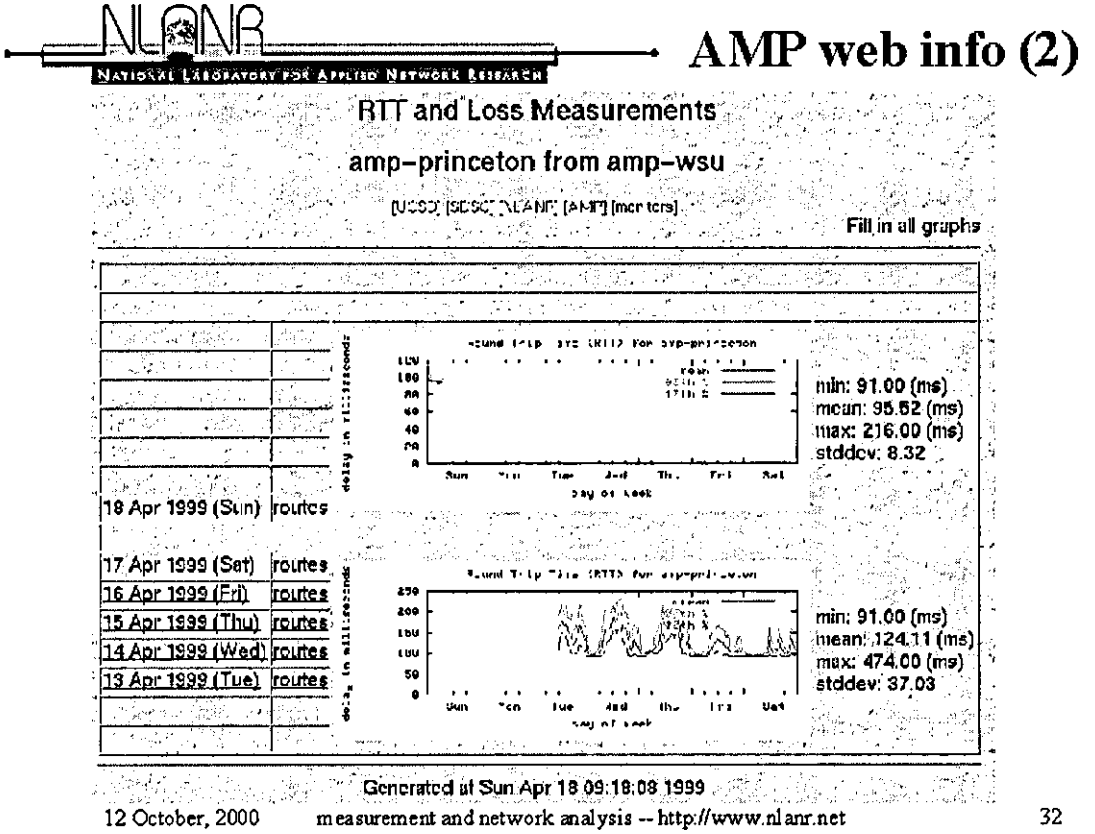


Figure 3.2-6 TodVar example from AMP [McG00]

3.3 How Can Data Exceptions be used?

3.3.1. Data Abstraction

Data Exceptions are an abstraction of the data. In this sense they are inherently useful as they present performance information in a condensed and accessible form. Information is only presented when something has changed from that expected and then only the change itself is presented. This is a key benefit as the volume of measurement

information increases. Sifting through the multitude of information for the relevant and informative is a formidable task. Skilled analysts are expensive to train and retain so any means of reducing or semi-automating the task is a significant step forward.

3.3.2. Gauging effects of network events

Data Exceptions are especially useful when gauging the impact of network events such as faults and planned alterations or re-routes. Models or simulations can be used to predict the effect that certain changes might have. Observing the direct effect on user perceived performance, in terms of latency, brings a greater degree of certainty that any alteration to the network has had the expected effects. It is surprising the degree to which network operators are in the dark regarding the actual performance that their networks are providing **[Che00][Cla97]**. The effects of alterations are often different from that which was anticipated. Data Exceptions show exactly what the effects of a change were.

3.3.3. Network event detection and diagnosis

Data Exceptions can also be used as part of the network event detection and identification process. The non-real-time aspect of the Data Exception detection methods discussed in chapter 5 make it unsuitable as a front line fault detection method. Fault detection at a fundamental level, detecting that a line has gone down for instance, is better done using network management tools built around SNMP. However faults can occur that are significantly harder to detect and that can remain unnoticed for some time **[Jai91]**. Faults of this nature might be detected using Data Exceptions. One network event will generally give rise to multiple Data Exceptions, as it is likely to affect multiple tests. To identify and interpret network events some means of correlating and collating the relevant Data Exceptions must exist.

3.4 Collections

Data Exception Collections contain all the Data Exceptions pertaining to one network event, or a series of connected network events **[PhiSan00]**. The aim here is to give as complete a picture as possible of a network event. Multiple views of the event may be necessary to determine information about the event. For instance if a test monitors the delay between two points on a network and a Data Exception occurs on that

test there is no immediate way of knowing whereabouts along the link an event has occurred. If tests are being conducted on several paths, some of which show the Data Exception and some do not, then it may well be possible, given that some topology information is known, to deduce whereabouts the network event that gave rise to the Data Exceptions occurred.

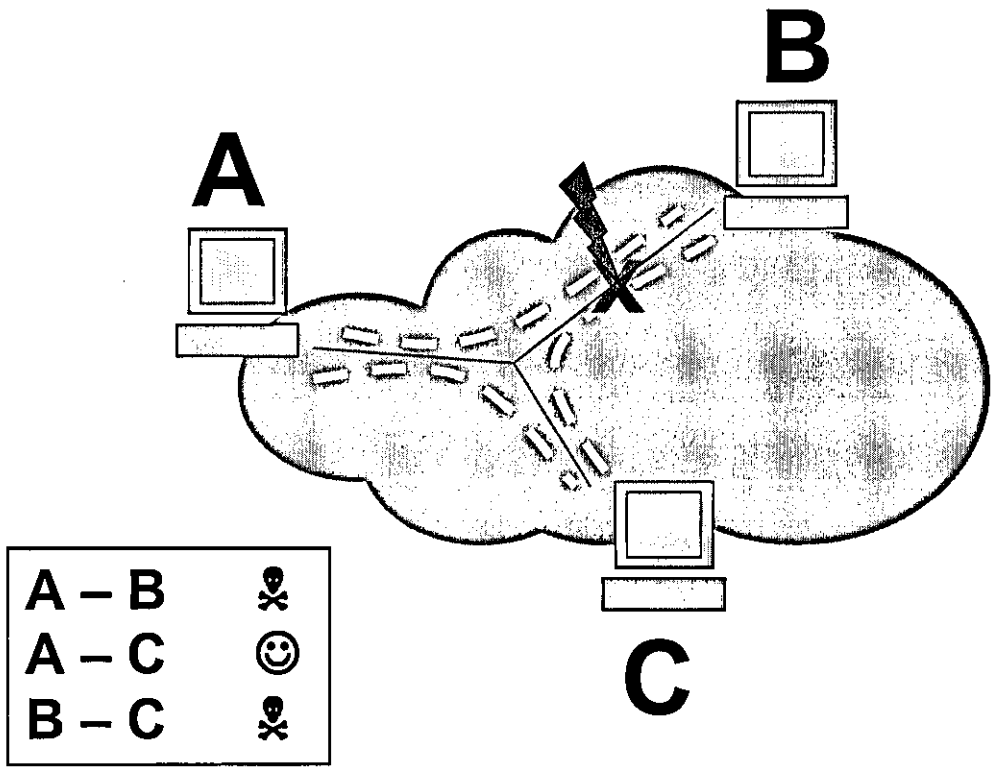


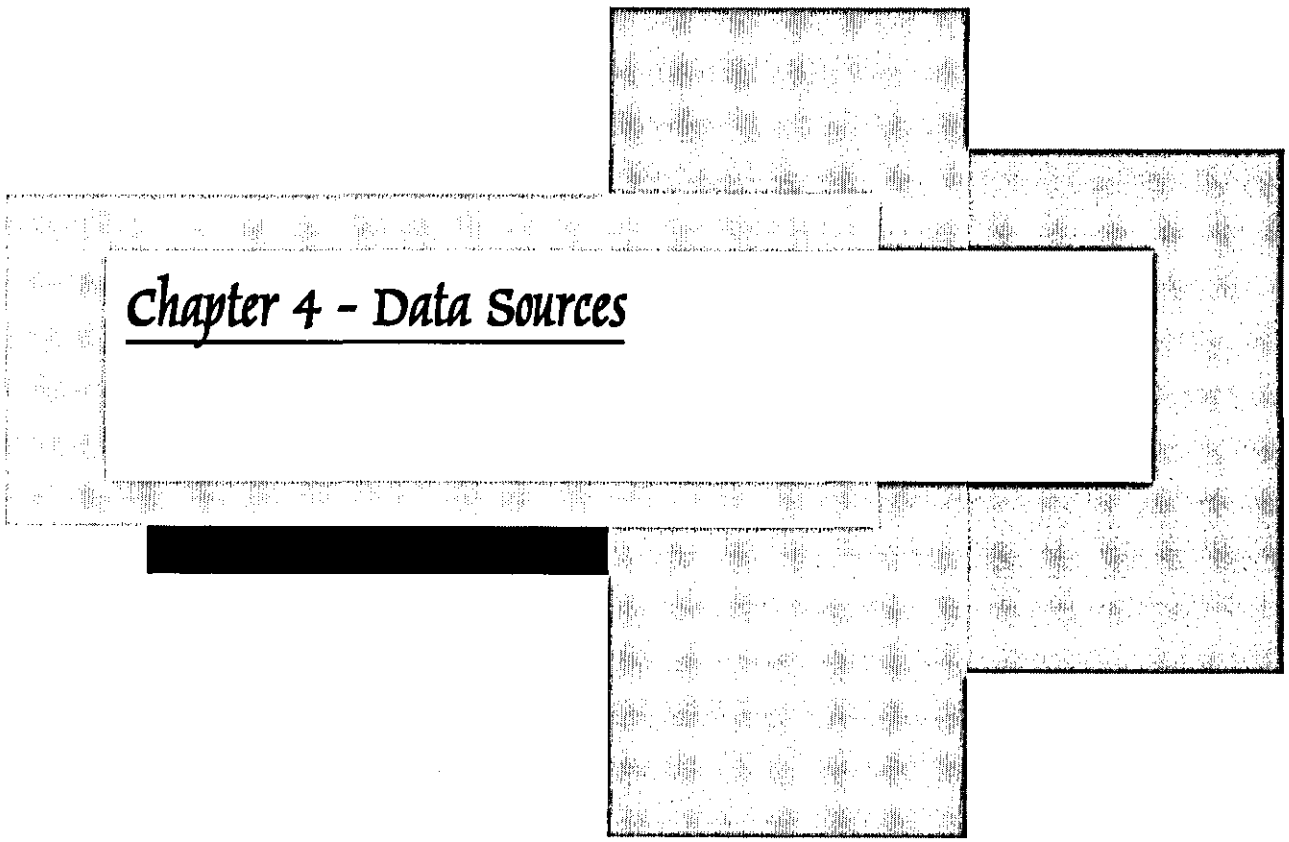
Figure 3.4-1 A simple example of the need for collections

In the example above (figure 3.4-1), if we were monitoring each of the paths A to B, A to C and B to C then both the Data Exceptions that should be reported on A to B and B to C would be necessary to deduce whereabouts the event had occurred.

The concept of collections was implemented in the AIR system mentioned in chapter 2 **[Phi99]**. Collections were formed by correlating Data Exceptions using three fields: testid (route information), time and type. If Data Exceptions matched on two of these three fields they were stored together in a collection. Although this is fairly rudimentary approach it served its purpose adequately.

3.5 Summary

In this chapter the concept of Data Exceptions has been examined in more detail, examples have been given and causes discussed. The idea of Collections has also been introduced as a possible further step. In the next chapter we consider the data sources available to the project. These data sources were used to develop and assess Data Exception detection methods.


Chapter 4 - Data Sources

4. Data Sources

In this chapter the data sources available to this project are discussed. These data sources were necessary in order for Data Exception Detection methods to be developed and tested.

4.1 Essential Criteria

4.1.1. Data Completeness

In order to develop and assess techniques for detecting Data Exceptions, performance data is required. The data is needed so that correlation between network events and network performance data can be observed, learnt and finally validated and tested. An essential criterion for the data is that it should be complete. It is of limited use to know what network events have occurred without having access to the corresponding performance data. Likewise the performance data is only partially useful without knowledge of the network it describes. Where the data is incomplete it may be possible, by estimation and inference, to make good the shortfall but this path is strewn with difficulty and danger and is undesirable. The problem here is that estimation and inference can lead to incorrect assumptions. Bogus data may have an adverse effect on the derivation of a detection method leading to faulty results.

4.1.2. Data Range

Another useful quality that the data set may have concerns the range of events it represents. It would certainly be of benefit if as many different types of network event as possible are recorded and represented. If during the monitoring process only one kind of event is observed out of a large set of possible events, then any detection technique based on this data will be tailored to deal with only that event. In a sense this is another kind of 'incompleteness'. Ideally all network events should be included but this is not possible as the range of possible things that may happen to a network is very large. However, a broad range of network events, in terms of their impact on the network, can and should be considered.

4.1.3. Controllable data

To make experimentation easier it is desirable that the data be in some way controllable. The ideal scenario is one in which data can be produced on demand allowing for any gaps or thinly covered areas to be supplemented. This necessitates the ability to generate or cause network events. One might say that this is a 'desirable' rather than 'essential' criterion. The point here is that it is impractical to simply wait for the right set of data to be gathered; it might be any length of time before certain network events occur. The preferable option is to be able to affect the network in such a way as to be able to cause network events and observe the resultant data.

4.2 Potential Data Sources – an overview

4.2.1. Commercial network

These criteria, that the data should be complete and that they should cover the full range of network events poses a problem - how can we obtain such a data set? Data from a large commercial data network is excellent in that it represents the actual problem in hand. However it may not meet the criteria set before us. Firstly, obtaining accurate and detailed information regarding network events from commercial data networks is difficult as operators are understandably reluctant to make such information public knowledge. Even where access is granted to collect performance data, full explanations of the network events may not be forthcoming, partly as the company may be reluctant to release such information, partly because they may not even know.

Even if such information were available our second criteria also presents a stumbling block. How can we guarantee covering a full range of network events during our monitoring period? This data source is definitely not controllable and considerable difficulty may be encountered in trying to persuade operations managers to allow live network equipment to be tampered with, simply so that performance data can be gathered!

Data from a commercial network (BT's SMDS network) has been available to this project. While the data on it's own is not sufficient to enable the design of a reliable exception detection method, it has been useful. It has been helpful to be able to verify that data generated by other methods (see below) are realistic. If a simulation tool, for

instance, were all that was available it would be hard to ascertain whether or not the output that was being produced behaved in a manner that was realistic and consistent with that of a commercial network.

The SMDS data was also useful in characterising the nature of delay data. Early work on this project investigated the distribution of delay data, of particular interest was whether or not the data were normally distributed (see Appendix E). Also Data Exceptions that could be grouped together and categorised were identified from the SMDS data.

4.2.2. Network Simulation

In addition to data from SMDS, two other solutions have been investigated. Modelling a network scenario has not been seriously considered due to the immense complexity involved in trying to model a network of the size we are considering.

The first solution is to simulate a network. Various packages exist, both freeware (often developed by academics) and commercial software, that allow network scenarios to be built, observed and recorded. Simulations have the advantage of allowing considerable complexity (in terms of network topology) to be generated with the minimum degree of difficulty. This makes them excellent in terms of experimentation as large networks can be generated and torn down again with ease, facilitating rapid adaptation and easy development to suit the nature of the problem.

4.2.3. Test Network

The second solution is to use a test network. This is a relatively expensive solution and is not as flexible as a simulation. A test network is advantageous though in that simulations, however good, are always going to be simplifications to some degree or another. A test network will present 'real' data, inclusive of any quirks that might exist.

Both solutions allow full control over network events and full access to the complete data set.

All three of the methods of obtaining data mentioned above were used. The data from BT's SMDS network was used primarily for the author's own benefit in understanding the nature of Data Exceptions, how they can occur and what they might look like and also in verifying work done using the simulation. The data from SMDS was not used further due to the limitations described earlier.

4.3 SMDS Data

SMDS data has been collected and stored by the High Speed Networks group at Loughborough over the past five years. The data is taken from twenty one-way paths across SMDS encompassing five ingress/egress points. For a more complete description of how this data is obtained please see chapter 2, 'Performance Monitoring at Loughborough'.

4.4 Simulation - NS

NS [**NS**], a free network simulator developed in the US at Berkeley University of California, was used to simulate a network. NS was chosen due to the readiness of its availability (it can be freely downloaded from the internet) and the level of support in the form of a comprehensive web site including tutorials, help pages, documentation and a mailing list. Of the other network simulation packages considered BONES [**BONES**] was in the throes of becoming obsolete, OPNET [**OPNET**] was an expensive alternative (although probably easier to use) and CNET [**CNET**] didn't appear to have the required level of development and support.

The network consisted of 8 peripheral nodes, a core of 4 nodes and 3 nodes attached at each point on the periphery (3 to each of the peripheral nodes) from which the various agents sent and received data. Each peripheral node had attached one node that was used to send telnet style traffic, one node to send ftp style traffic and one node to send monitor traffic. A peripheral node is shown in figure 4.4-1 and the overall topology in figure 4.4-2.

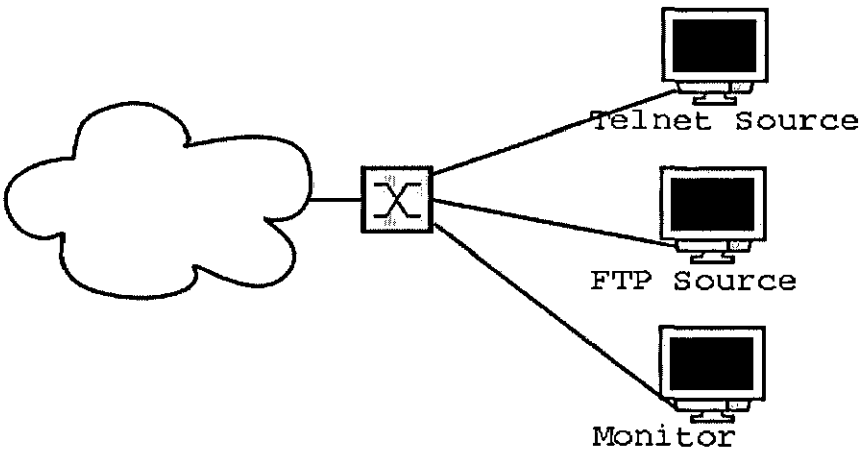


Figure 4.4-1 A peripheral node

The topology needed to contain sufficient inherent complexity to give rise to interesting Data Exceptions. This desire for complexity had to be tempered however with the practical considerations of implementing the simulation (running time, storage space). This structure seemed to allow reasonable flexibility for simulating network events whilst still being manageable.

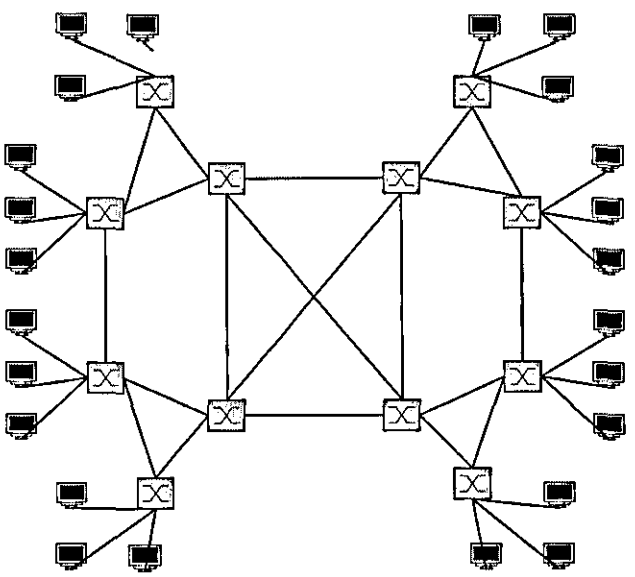


Figure 4.4-2 The NS Topology

Telnet sources were linked with a traffic sink at each of the different peripheral nodes, meaning that at any one peripheral node there were seven telnet sources (one for each of the remaining nodes)[1]. These were then set off at random. Similarly, ftp sources

[1] The seven telnet sources and the seven ftp sources are represented by just one node each on the diagram.

were established at each peripheral node to send to all the other peripheral nodes. Again, the traffic from the ftp sources was sent at random. Although both the ftp and the telnet sources generated traffic at a random rate, the probability of a source sending traffic at any one time was altered so that the it was higher during the working hours of the day. This gives the 'Time of Day Variation' effect discussed in earlier chapters. The ftp and telnet sources were chosen to load up the network as they offered contrasting traffic profiles and were relatively simple to implement.

The monitor agents transmitted a fixed size packet to each of the other monitor nodes at constant intervals. The latencies experienced by these packets were then recorded and later used for analysis.

The simulation was run over a 336 second period where each second represented an hour in real time (24 second days). The simulations are therefore representative of a two week period. On each simulation run (at least) one network event was introduced. The network events that were simulated were: links going down, nodes going down, links simulating faulty behaviour, traffic re-routes, links being introduced and the introduction of other erroneous traffic sources. An example of the TCL scripts that were used to specify these network scenarios can be found in Appendix A.

4.4.1. NAM (Network Animator)

The output from NS is flexible and can be specified. Standard functions exist for the generation of text files that are in a format that can be passed on to NAM (Network AniMator) (this file is called out.nam). This allows a complete viewing of the entire simulation. The NAM files were used mostly in the design phase as they provided a good overview of what was going on and could be accessed and understood instantly.

```
r -t 321.878 -s 6 -d 20 -p ack -e 40 -c 5 -i 1263144 -a 5 -x {35.2 20.8 4962 ------- null}
+ -t 321.878 -s 20 -d 6 -p tcp -e 1000 -c 5 -i 1263624 -a 5 -x {20.8 35.2 4973 ------- null}
- -t 321.878 -s 20 -d 6 -p tcp -e 1000 -c 5 -i 1263624 -a 5 -x {20.8 35.2 4973 ------- null}
h -t 321.878 -s 20 -d 6 -p tcp -e 1000 -c 5 -i 1263624 -a 5 -x {20.8 35.2 -1 ------- null}
- -t 321.878 -s 10 -d 32 -p tcp -e 1000 -c 5 -i 1263301 -a 5 -x {20.7 32.2 6281 ------- null}
h -t 321.878 -s 10 -d 32 -p tcp -e 1000 -c 5 -i 1263301 -a 5 -x {20.7 32.2 -1 ------- null}
r -t 321.878 -s 11 -d 10 -p cbr -e 333 -c 3 -i 1263489 -a 3 -x {33.13 30.13 6416 ------- null}
+ -t 321.878 -s 10 -d 30 -p cbr -e 333 -c 3 -i 1263489 -a 3 -x {33.13 30.13 6416 ------- null}
- -t 321.878 -s 10 -d 30 -p cbr -e 333 -c 3 -i 1263489 -a 3 -x {33.13 30.13 6416 ------- null}
```

Figure 4.4-3 'out.nam', text output from the NS simulation

The output shown in Figure 4.4-3 is formatted especially to be read by NAM so that it can be animated. Below, in Figure 4.4-4, is a screen shot of just such an animation.

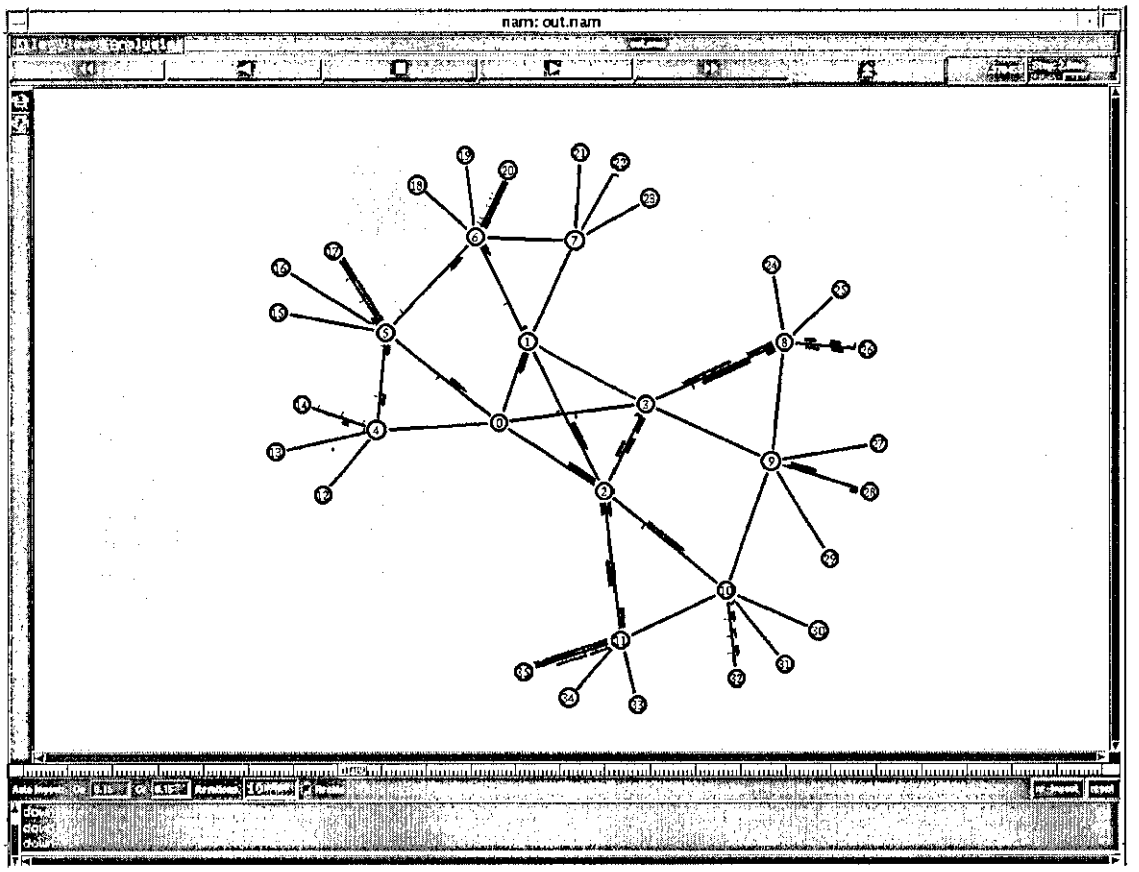


Figure 4.4-4 Screen shot from NAM

4.4.2. Test.out

A file called 'Test.out' was produced that simply represented the output of every action that occurred and the time at which it occurred (see figure 4.4-5). These files contained all the information allowing specific analysis but required further processing before that analysis could be easily achieved (The test.out files were around 1GB in size).

```
+ 335.999555 1 2 ack 40 ------- 5 23.13 35.10 5397 1387120
- 335.999555 1 2 ack 40 ------- 5 23.13 35.10 5397 1387120
- 335.999759 35 11 tcp 1000 ------- 5 35.13 32.13 4840 1387092
r 335.999776 5 0 cbr 333 ------- 3 15.7 33.1 6691 1387102
+ 335.999776 0 1 cbr 333 ------- 3 15.7 33.1 6691 1387102
- 335.999776 0 1 cbr 333 ------- 3 15.7 33.1 6691 1387102
r 335.999885 8 24 cbr 333 ------- 3 12.3 24.0 6690 1387039
r 335.999943 2 10 tcp 1000 ------- 0 25.9 31.4 5420 1387071
+ 335.999943 10 31 tcp 1000 ------- 0 25.9 31.4 5420 1387071
- 335.999943 10 31 tcp 1000 ------- 0 25.9 31.4 5420 1387071
```

Figure 4.4-5 'test.out', text output from the simulation

The data that were used to monitor the network were the latencies of the monitor packets. These had to be filtered out from the 'test.out' file, this was done in the first instance by simply using the 'grep' program in UNIX and the results were put into a file called 'monitor.out'. Delay files were generated, one for each of the paths monitored, by further processing the delay data. Initially this was done using an AWK script posted on the NS users mailing list. Rewriting the script in 'C' gave the process a considerable speed up. The overall process is shown below.

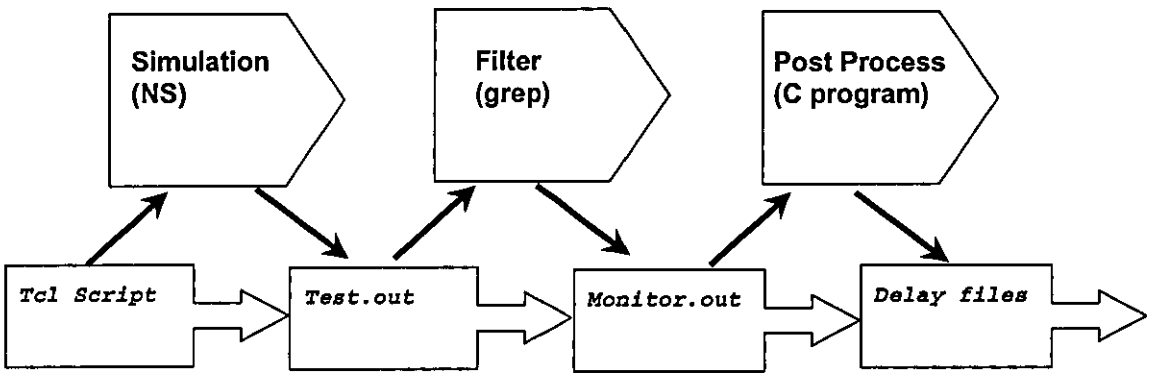


Figure 4.4-6 Getting Delay Data from an NS simulation

So for each simulation 56 delay files were produced, showing the one-way delay on different routes across the simulated network (The entire process is shown in figure 4.4-6 above). Each delay file has just two fields, the transmit time (hours since the beginning of the simulation) and the delay. Below (figure 4.4-7) is a sample delay plot.

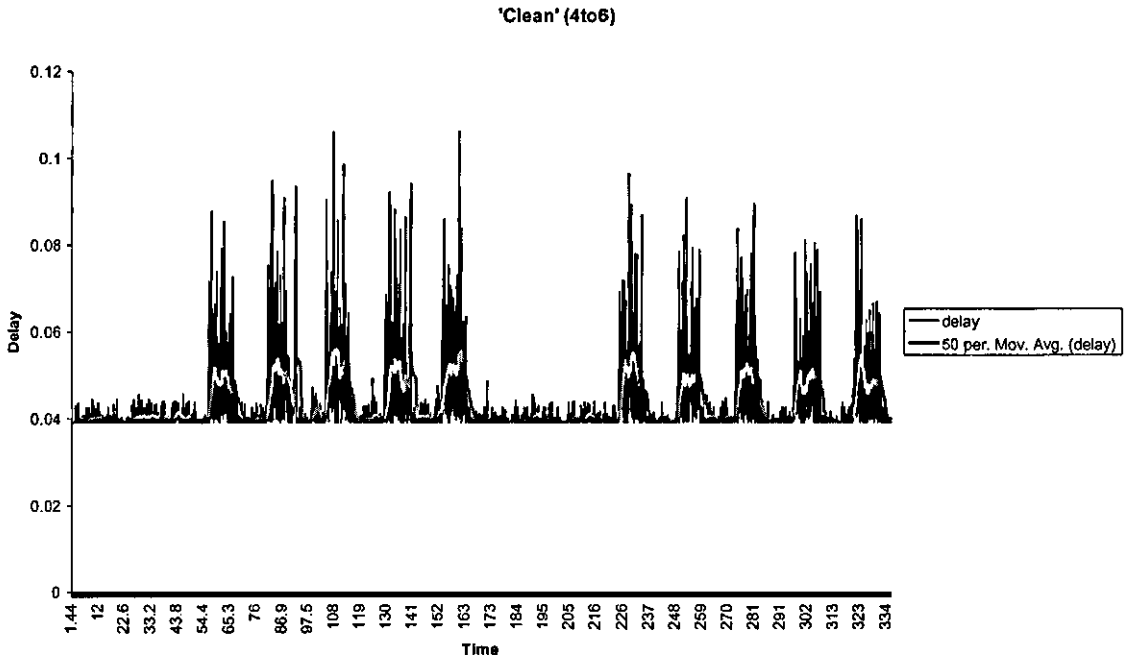


Figure 4.4-7 A sample delay plot from NS

4.5 Test Network

4.5.1. Network Design

A test network was designed and built using equipment donated by Cisco Systems. The network consists of eight 2600 series routers and four Catalyst 1900 series switches (figure 4.5-2). In designing the network several key aspects were taken into consideration. As with the simulation there needed to be sufficient complexity inherent in the design to allow for a variety of network events. Whereas with the simulation the limiting factor was processing power and storage space, the test network was restricted only by the cost of the equipment. This included network components such as the routers and the switches and end stations used for monitoring the network and generating traffic to load the network.

The Cisco 2600 routers have a network module slot and two WIC (Wide Area Interface Card) slots (see figure 4.5-1). This allows a number of different types of interfaces to be fitted to the router depending upon the requirements of the user.

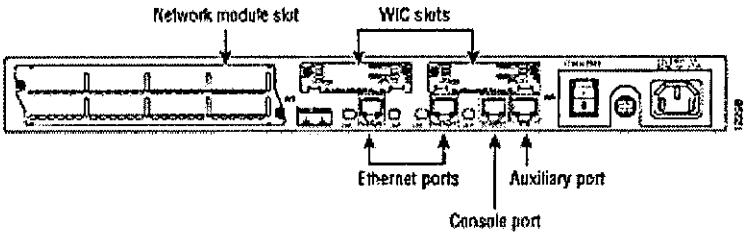


Figure 4.5-1 The rear of a Cisco 2611 router

Both the Network module slot and one WIC slot were fitted with serial interfaces giving a total of six serial interfaces, four on the Network module slot and two on the WIC slot. These were in addition to the two Ethernet ports that came as standard.

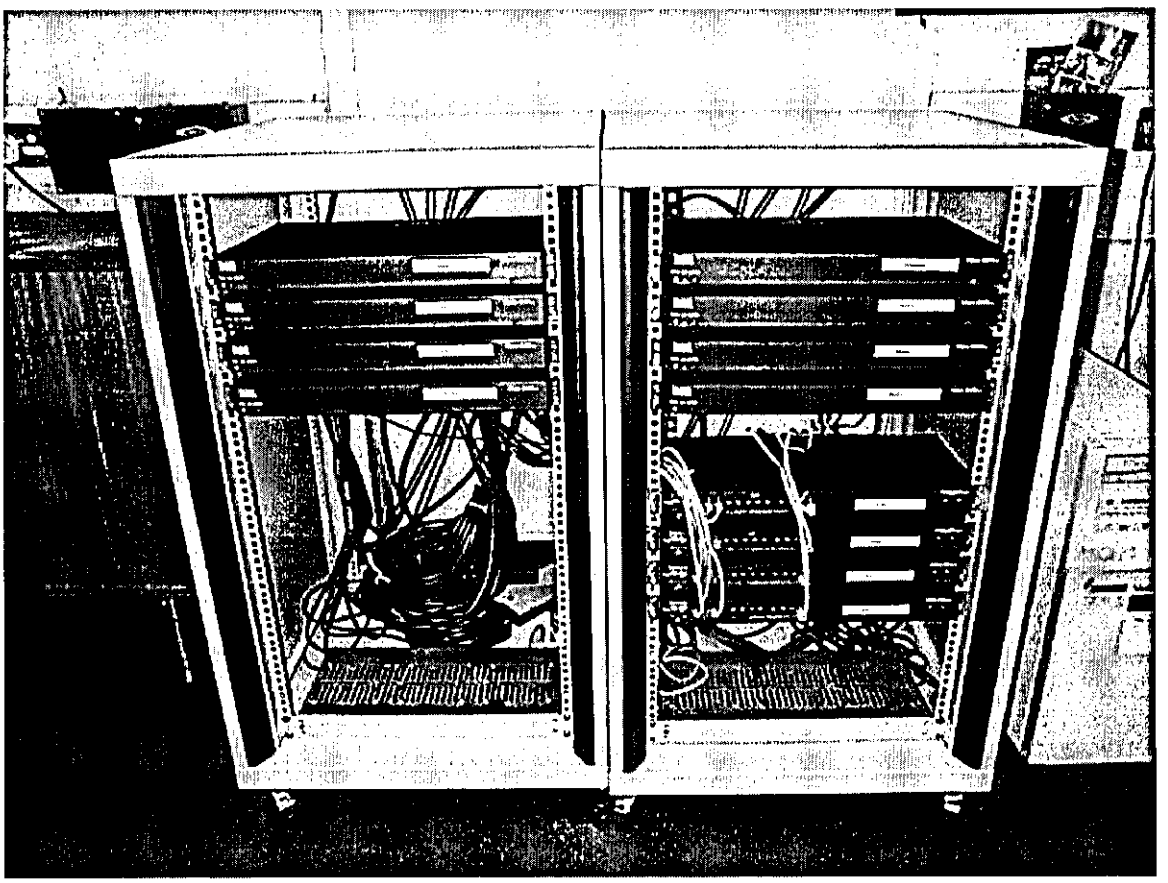


Figure 4.5-2 Routers and Switches

The design chosen has two meshes, each with four routers (figure 4.5-3). The aim here was to include as much redundancy as possible giving more scope for Data Exceptions. Mesh A interconnects the routers using point to point links. The protocol used here is LAPB running over serial cables. These links have a capacity of just 0.12Mb. This is ideal in that it makes the task of loading the network up to capacity far

easier. It should be remembered that the purpose of this section of the work is to generate Data Exceptions. These do not have to be generated at high data rates. If the test network had high performance difficulties would have been encountered in trying to generate sufficient traffic to load up the network in a realistic manner. Mesh B, in addition to the four routers, makes use of four Ethernet switches. The switches are used to connect the routers together. The links on this network are Ethernet and have a capacity of 10Mb, which is still relatively low performance (in comparison to the SMDS core for instance) but is significantly better than Mesh A, providing a good contrast. The two meshes, A and B are inter-linked again using LAPB although this time with a capacity of 4Mb.

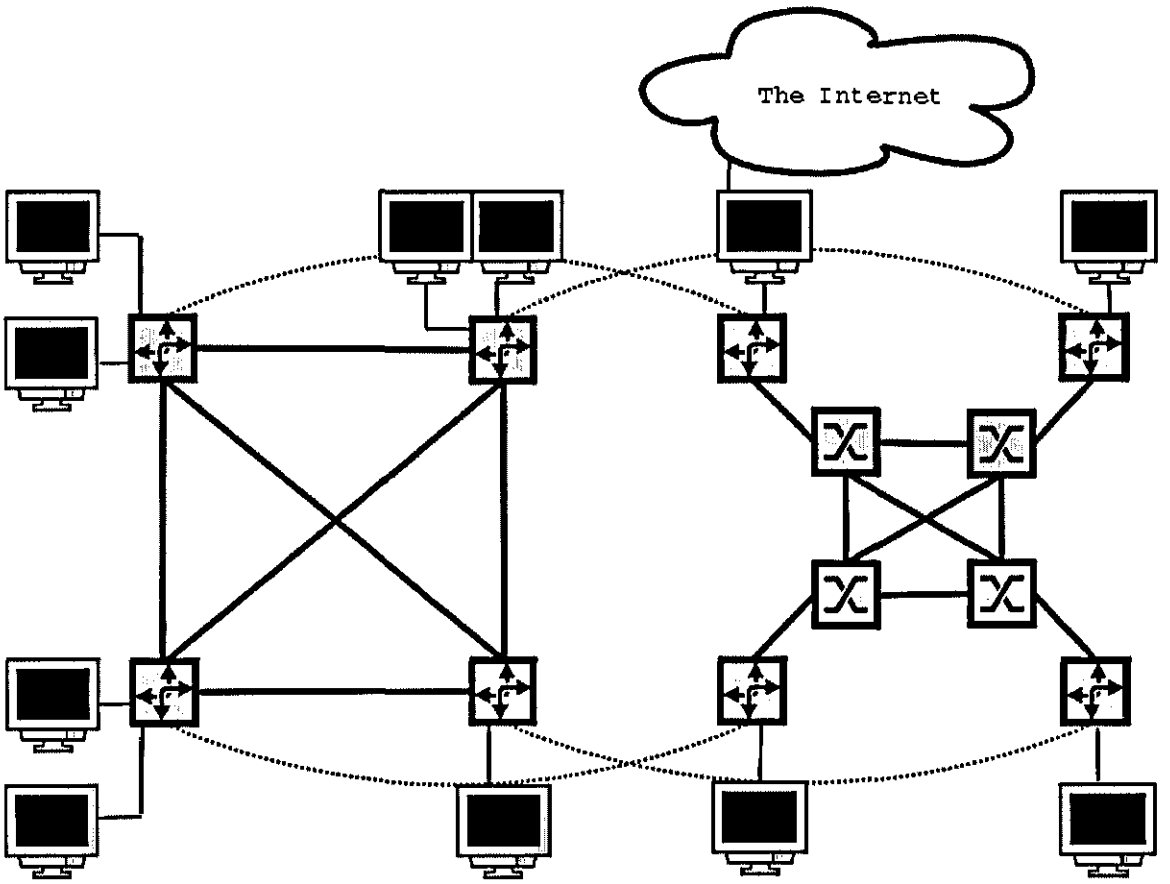


Figure 4.5-3 The test network design

4.5.2. Traffic Generation

Traffic was generated using six Linux based PCs. Although there is software available to perform traffic generation nothing could be found that would easily emulate

the varying load seen on wide area networks such as SMDS. Consequently a traffic generator was written that would fulfil the desired criteria. The important factors for the generator were that it should be able to vary its output in a random manner according to the time of day. The traffic generated did not need to emulate profiles observed over customer networks in terms of the micro detail such as traffic profile, 'burstiness', distribution or type; rather it needed to replicate the general volume of traffic, proportionally, as seen on commercial networks.

Each traffic generator ran a number of sessions. A session described the traffic generated between itself and one other generator. A session contained the following attributes (figure 4.5-4).

Name	Schedule
Traffic Level	Start Time
Day Length	Server Port Number
Quantity	Server IP Address
Day	Number of days to run
Transport Type	Transmission Rate

Figure 4.5-4 Table of session attributes

4.5.2.1 Schedule

A day consists of 1440 minutes. The schedule is an array containing 1440 elements and it determines whether or not the traffic generator will be active during each of these minutes.

4.5.2.2 Traffic Level

A day also consists of 24 hours. The Traffic Level is also an array and contains 24 elements, it is used when calculating the schedule. Where the Traffic Level is high, the probability that activity will be scheduled during any one of the minutes in that hour will also be high.

4.5.2.3 Start Time

The session can be configured to start at any time. This is governed by the system clock.

4.5.2.4 Day Length

A day need not be 24 hours! Virtual days can last from anything from one minute up to 24 hours. This allows results to be generated at a faster rate than real-time.

4.5.2.5 Port Number & IP Address

A server Port Number and IP Address have to be specified for the traffic to be sent to. If the transport type is set to UDP a server may not be running on this port number but the field must still be set. If the transport type is set to TCP then the port number must be set to the number that the server is listening on.

4.5.2.6 Quantity

This attribute sets the number of packets that will be sent each time the schedule activates the traffic generator. The packets will be of a random size, uniformly distributed between 100 and 1500 bytes. The Quantity should be changed depending on the day length. If the day length is set to 24 hours then the quantity of packets sent each time the traffic generator is called should be significantly larger than if the day length was set to 1 hour. Linear scaling is appropriate here.

4.5.2.7 Day

Each session starts at day 0. This is then incremented at the end of each virtual day until it equals the 'Number of Days to Run' attribute.

4.5.2.8 Transport Type

This can be set to either UDP or TCP. UDP is generally preferable as there is no need to configure a server since a connection does not need to be established. Also the TCP protocol prevents any overloading of the network.

4.5.2.9 Transmission Rate

The transmission rate is controlled using an active wait. The traffic generators are connected to the network via 10Mb/s links. This means that the total amount of traffic

produced by one traffic generator cannot exceed 10Mb/s. In practice 8Mb/s is a more realistic figure for the link speed. The traffic generators have not been tested on faster links to see what transmission rates they are capable of but there has been no issue utilising the whole of the bandwidth available where required.

4.5.3. Monitoring the Test Network

Monitor stations had previously been developed at Loughborough for work on the SMDS network (see Chapter 2) and these were used to monitor the Test Network. Four monitoring stations were employed, this was the maximum number that could be supported without producing more timing cards. Timing synchronisation was achieved by configuring three of the monitor stations to draw their timing pulse from the fourth. As the network is entirely situated in one place the monitor stations can be placed adjacent to one another allowing the timing synchronisation issue to be localised. Where the network is more widely distributed timing synchronisation can be obtained using GPS (see chapters 1 & 2). The monitor stations, two on each mesh, monitor paths to each other giving twelve one-way test routes, each being tested with both 64byte and 1500byte packets. These results are stored in log files on the monitor stations. A control station is used to retrieve the log files, process them into a database and draw summaries from them as required. A picture of the monitor stations, together with the traffic generators is shown in figure 4.5-5.

Network events were introduced to produce Data Exceptions. These included changing router configurations, changing routing priorities, unplugging cables, introducing extreme traffic levels.

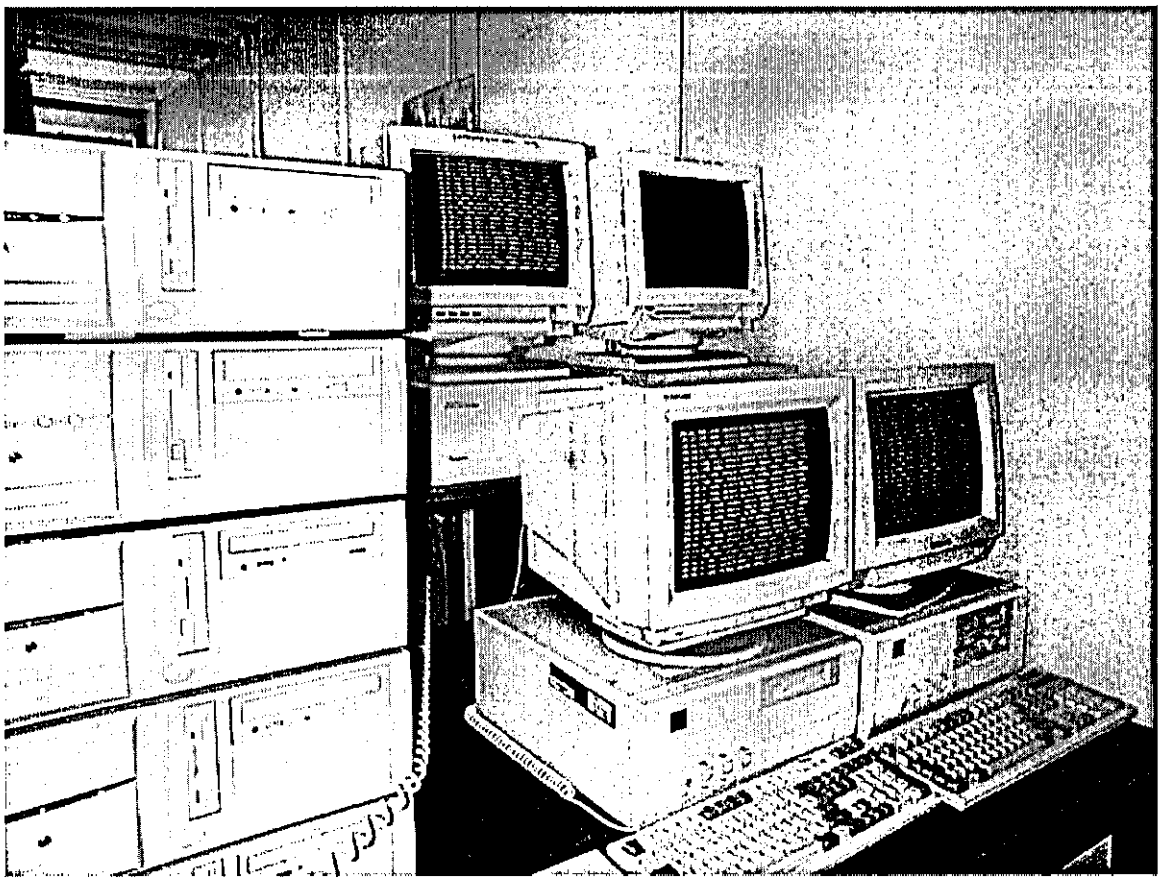


Figure 4.5-5 The Traffic Generators (left) and the Monitor Stations (right)

The log files containing the raw information are stored on the monitor stations. A command language was previously developed to control the monitor stations and this allows the log files to be retrieved to a control station where they are processed into a database. Querying this database then gives the end to end delay. An example plot is given in figure 4.5-6.

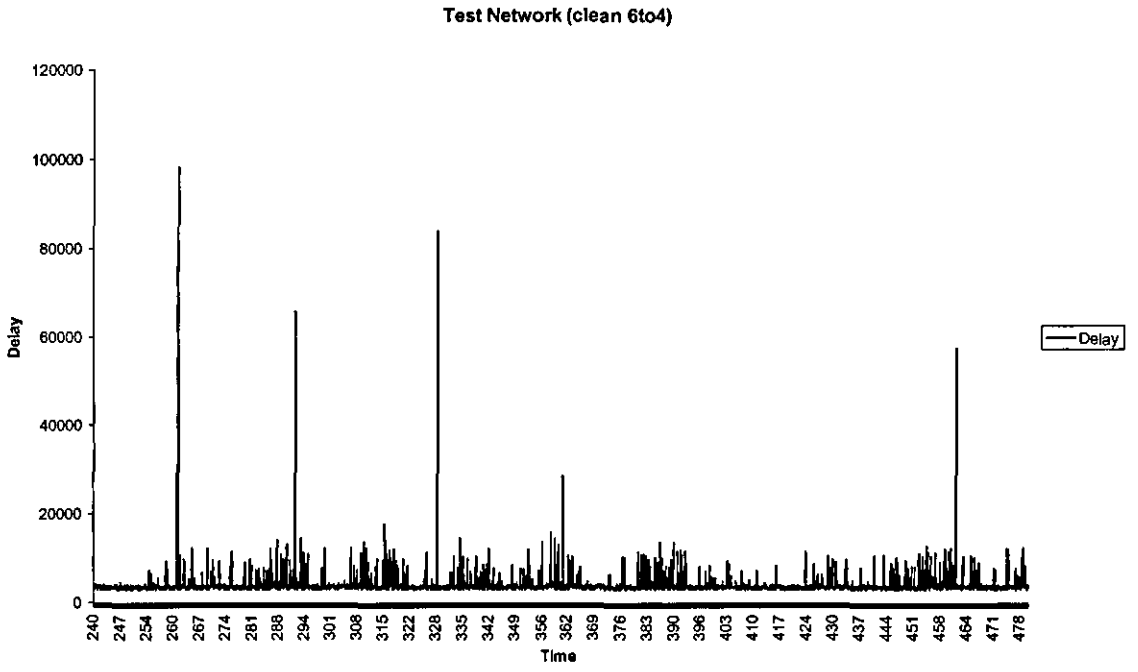
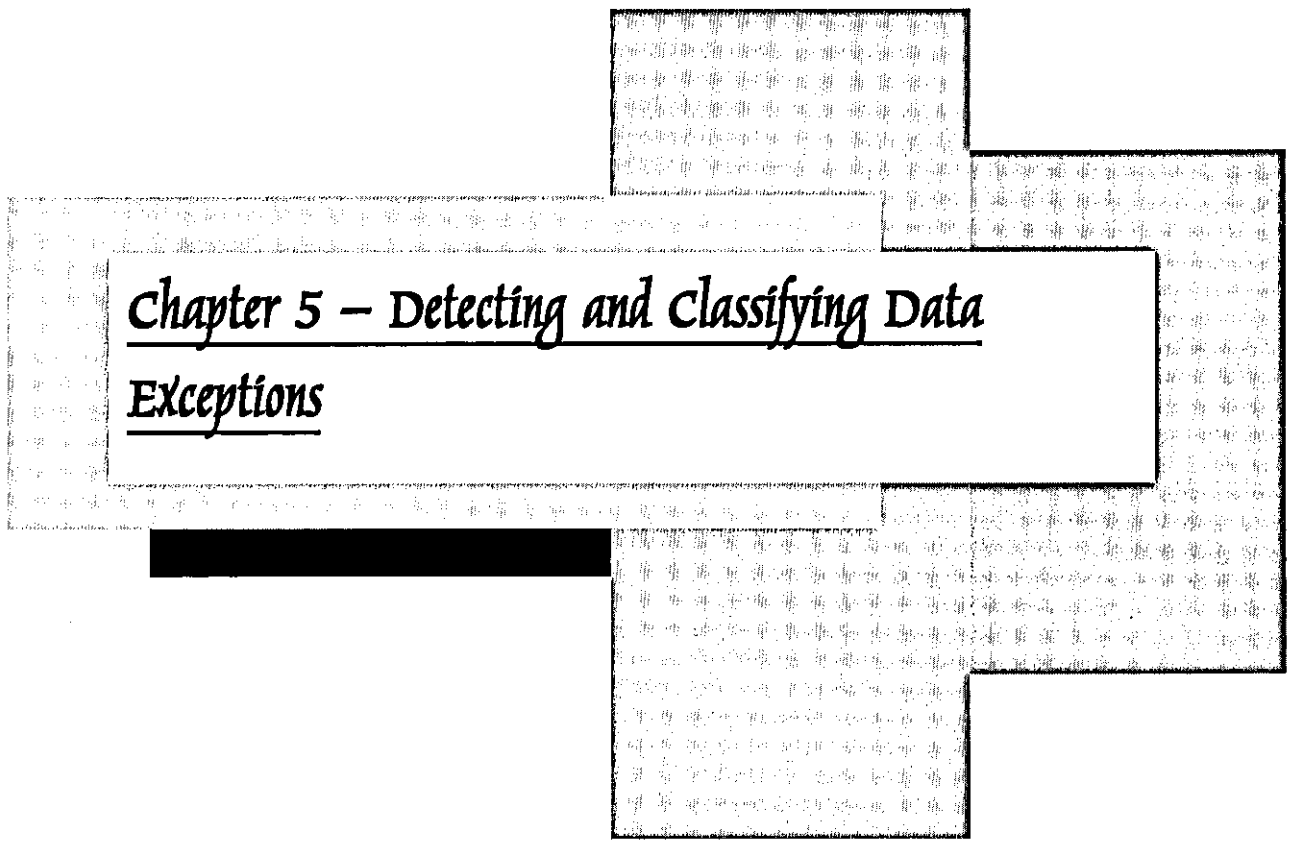


Figure 4.5-6 Delay plot from the test network

4.6 Summary

In this chapter we have looked at the data sources available to this project. Delay data is necessary for training and testing exception detection methods. Three sources of data have been available to this project. Data from BT's SMDS network has been used to learn about the nature of Data Exceptions and validate data produced by a network simulation and a test network. NS was used to simulate a network and network events were incorporated into these simulations to give Data Exceptions. A test network was also constructed and then monitored using monitoring tools discussed in chapter 2. Network events were then introduced again to give Data Exceptions.

In the next chapter we go on to look at two methods of detecting Data Exceptions.



Chapter 5 – Detecting and Classifying Data Exceptions

5. Detecting and Classifying Data Exceptions

5.1 Introduction

Two methods for detecting and classifying Data Exceptions are discussed in this chapter. The first, a rule-based approach, was a first attempt at solving the problem of Data Exception detection. It was implemented as part of the AIR system **[Phi99]**. The second approach is more refined and makes use of a two-phase process. The K-S statistic is used to initially detect the presence of a Data Exception in the data. A neural network is then used to classify the type of Data Exception that has occurred.

5.2 Rule-based Approach

A rule-based solution has two significant advantages. Firstly, it is the simplest solution and secondly it is a predictable solution and easy to trace. If a rule set can be found that accurately defines Data Exceptions then it would be the obvious solution and presumably the fastest one. There is no sense in devising a complicated process such as a neural approach or a statistical approach when it will not better a simpler, readily available process such as a rule set **[Tar98]**. A rule-based system also scores highly in that it is predictable. Given a set of inputs the output can be calculated and understood. This can be a requirement of 'mission critical' software solutions and is advantageous in tracking any errors that may occur.

A rule-based solution was devised that attempts to describe the different Data Exceptions accurately, in such a way that rules can be put in place to test for the different exception types. The rules make use of a 'feature set'. A feature set contains various features, or statistical properties that describe the data (such as the mean or the standard deviation). Features are calculated for new data and a feature set is held over from previous data to provide a benchmark for comparisons. These features are combined to give 'indicators'. Indicators are, in essence, higher level features that can be tested directly against thresholds to check for exceptions. Indicators can be features themselves, a combination of current features or a combination of current and historical features. The rules return an exception type or no exception if none are found.

As Hood notes [**Hoo97**], the use of thresholds to test for faults is quite common both in practice [**Mad94**][**Wal91**] and in research [**Den93**][**Gol95**] for detecting unusual network behaviour. These are generally used to test whether some variable (often stored in the MIB) has drifted significantly out of bounds.

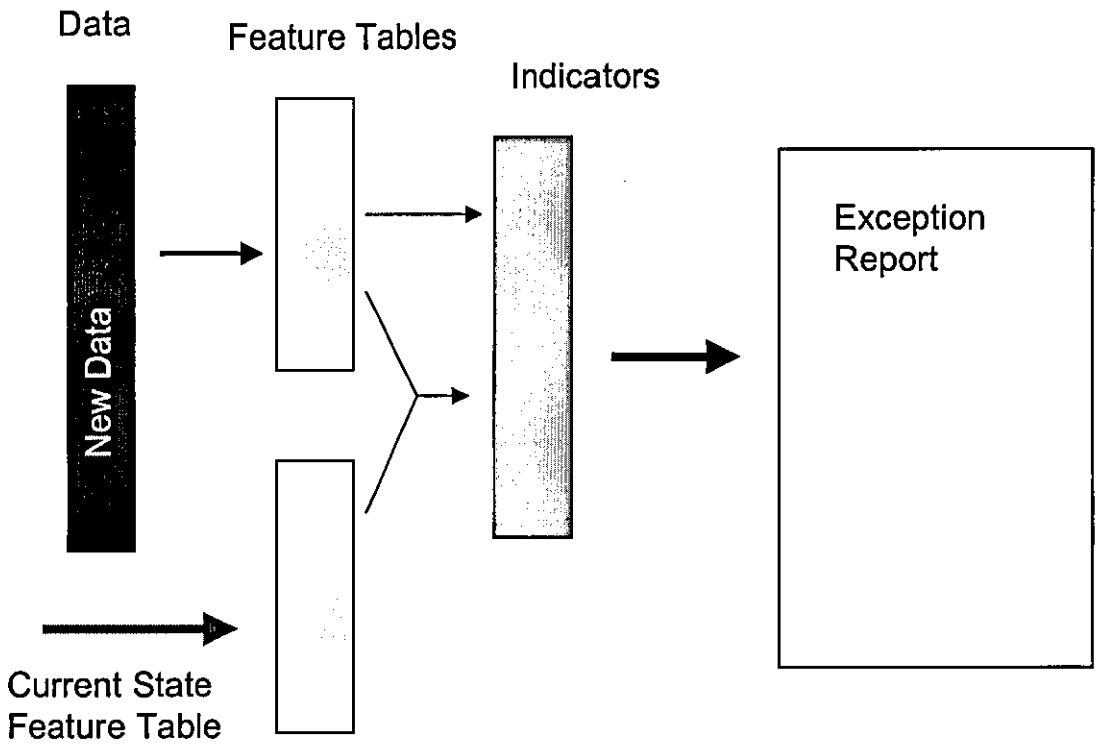


Figure 5.2-1 Exception Report Generation

A complete list of the features and their derivations is given below in figure 5.2-2. Each of these features is calculated over a day.

Feature	Derivation	Description
Mean (μ)	$\mu = \frac{\sum d_i}{n}$	The mean will be the sum of all the delay measurements, divided by the total number of delay measurements.

Standard Deviation (from the mean) (θ)	$\theta = \frac{\sum(d_i - \mu)^2}{n}$	The standard deviation is derived by summing the squares of the differences between each value and the mean value.
Median (M)	$M = d_{\frac{n+1}{2}}$ where $d_i \leq d_{i+1} \quad \forall i$	The median is the (n+1/2)th value if the values are put in rank order.
Maximum (max)	$\max = d_c$ where $d_c \geq d_i \quad \forall i$	The maximum from the set of delay data.
Second (sec)	$\text{sec} = d_c$ where $\max > d_c \geq d_i \quad \forall i$	The second highest value from the set of delay data.
Minimum (min)	$\min = d_c$ where $d_c \leq d_i \quad \forall i$	The minimum from the set of delay data
Change in delay (Cd)	$Cd_i = \lvert d_i - d_{i+1} \rvert$	A second set of values can be derived, that being the change in delay (over time) at each point
Mean Change in delay ($C\mu$)	$C\mu = \frac{\sum Cd_i}{n}$	As for mean delay
Standard Deviation of Change in delay (from the mean) ($C\theta$)	$C\theta = \frac{\sum(Cd_i - C\mu)^2}{n}$	As for standard deviation of delay

Median Change in delay (*MC*)	$CM = Cd_{\frac{n+1}{2}}$ where $Cd_i \leq Cd_{i+1} \quad \forall i$	As for median delay
Maximum Change in delay (*C* max)	$C\max = Cd_c$ where $Cd_c \geq Cd_i \quad \forall i$	The maximum from the Change in delay set (Cd_i)
Minimum Change in delay (*C* min)	$C\min = Cd_c$ where $Cd_c \leq Cd_i \quad \forall i$	The minimum from the Change in delay set (Cd_i)
Total Change in delay (*C* tot)	$C\,\text{tot} = d_1 - d_n$	The Total Change in delay is the difference between the average delay at the start of the day and the average delay at the end of the day

Figure 5.2-2 Table of Features

These indicators are then calculated:

Indicator	Derivation	Description
Peak (P)	P = max – sec	The difference between the maximum and the second highest delay value
Spike Ratio (SR)	SR = P / (sec – min)	The ratio of the difference between the maximum and the second highest delay value against the difference between the minimum and

		the second highest delay value
SD Change (SDC)	$SDC = SD_c - SD_o$	The difference between the current standard deviation and the observed standard deviation
SD Ratio (SDR)	$SDR = SD_c / SD_o$	The ratio of the current standard deviation against the observed standard deviation
Maximum Change (Max2Max)	$Max2Max = Max_c - Max_o$	The difference between the current maximum and the observed maximum
Minimum Change (Min2Min)	$Min2Min = Min_c - Min_o$	The difference between the current minimum and the observed minimum

Figure 5.2-3 Table of Indicators

These indicators are then compared against thresholds which, if exceeded, flag up exceptions. These thresholds can be set differently depending on the source of the data being examined. The values to date have been based on experience and 'rule of thumb' rather than using any optimisation technique although this would be a useful area to explore further.

The changes in maximum and minimum and the total change in delay are all used to detect step changes. Total change in delay is measured over the last day and is the simplest of the three measures to relate to a step change. The total change in delay is not sufficient on its own to detect step changes. One potential scenario is where there are two step changes within the day period. The total change from the beginning of the day to the end of the day may be negligible but two step changes would have in fact taken place. The changes in maximum and minimum are used to pick this up. In the case of a decrease in delay followed by and increase the minimum delay for the day decreases.

This is a good measure in that it doesn't suffer much variation, the minimum delay is a fairly consistent value relative to step changes. The maximum however is a lot less stable and therefore less useful in detecting step changes.

The spike ratio and the peak value are used for detecting spikes. On some routes, high delay variation can make it difficult to detect spikes. The idea here is that the increased delay from the spike should be, relatively and numerically significantly higher than other observed delay values.

The changes in time of day variation are detected using the change in standard deviation value and the ratio between the current standard deviation and the observed standard deviation. This is not a wholly satisfactory measure, as it takes no account of when the periods of high delay take place during the day, a key feature of time of day variation. However as a temporary feature it does indicate when the time of day variation shifts significantly.

The entire rule set that was implemented in Java as part of the AIR system and is given below in extracts from the function. The key objects are the two feature sets, one that is the currently stored information (**curr**) and the other being the information that has just been taken from the test (**test**). These contain all the features mentioned in the above table. The thresholds are stored in a **TestParameters** object called **limit**. The six indicators are declared at the start as **float** variables. Other variables of note are **type** which refers to the type of exception that has been detected, **weekend** which is a Boolean value indicating whether the test feature set is taken from a weekend and **holiday** which carries over the value of the standard deviation from the last weekend date.

```
if ( curr.maximumDelay == 0 ) {
    if ( test.maximumDelay != 0 ) {
            exception += "Testing Started\n";
            test.standardDeviation = 0;
            holiday = test.standardDeviation;
            type = "start";
    }
  }
  else {
    if ( lastDelay == 0 && test.maximumDelay == 0 ) {
            exception += "No Data";
            test.standardDeviation = curr.standardDeviation; // StandardDeviation won't be updated
            type = "loss";
    }
  }

  if ( !type.equals("start") ) {
    if ( test.minimumDelay == 0 && test.maximumDelay != 0) {
```

```
            exception += "No Delay Data available for three hour period\n";
            test.standardDeviation = curr.standardDeviation; // StandardDeviation won't be updated
            type = "loss";
        }
    }
```

In this first code excerpt (above) the function looks for missing data (loss) and for the possibility that the monitoring has just started. This is done primarily using the maximum and minimum values. Where the maximum value for a feature set is zero, no delay data can have been received for that day. Where the minimum value for a feature set is zero there has been at least a three hour period where no delay data has been received. These scenarios are unusual, certainly for a commercial network, but must still be catered for.

```
if ( !type.equals("loss") && !type.equals("start") ) {
    if ( min2min > limit.stepMax ) {
            exception += "Decrease in average delay of " + min2min + " micsecs\n";
            type = "step";
            magnitude = (int)min2min;
            test.standardDeviation = curr.standardDeviation; // StandardDeviation won't be updated
    }
    else {
            if ( test.totalChangeInDelay <-(limit.stepMax) ) {
                exception += "Decrease in average delay of " + (-1 * test.totalChangeInDelay) + " micsecs\n";
                test.standardDeviation = curr.standardDeviation; // StandardDeviation won't be updated
                magnitude = (int)(-1 * test.totalChangeInDelay);
                type = "step";
            }
    }
    if ( min2min <-limit.stepMax ) {
            exception += "Increase in average delay of " + (-1 * min2min) + " micsecs\n";
            type = "step";
            magnitude = (int)(-1 * min2min);
            test.standardDeviation = curr.standardDeviation; // StandardDeviation won't be updated
    }
    else {
            if ( test.totalChangeInDelay > (limit.stepMax) ) {
                if ( (lastDelay - test.maximumChangeInDelay) == delaySet[6] && min2min >-limit.stepMax) {
                        if (test.totalChangeInDelay > limit.peakMax) {
                            exception += "Increase in average delay likely to be a spike \n";
                            type = "spike";
                            magnitude = (int)peak;
                        }
                        else {
                            exception += "Increase in average delay likely to be a step change \n";
                            type = "step";
                            magnitude = (int)test.totalChangeInDelay;
                            test.standardDeviation = curr.standardDeviation; // StandardDeviation won't be updated
                            test.minimumDelay = 0; // minimumDelay will be set to the next day's minimum (see above)
                        }
                }
                else {
                        exception += "Increase in average delay of " + test.totalChangeInDelay + " micsecs\n";
                        type = "step";
                        magnitude = (int)test.totalChangeInDelay;
                        test.standardDeviation = curr.standardDeviation; // StandardDeviation won't be updated
                        test.minimumDelay = 0;          // minimumDelay will be set to the next day's minimum (see
above)
                }
            }
    }
```

```
}
```

Given that there is no exception of type 'loss' or type 'start' this next section (above) looks for the possibility of a step change. As was mentioned earlier, the primary features and indicators used in this process are the change in minimum, maximum and total change. One difficult scenario is where the last delay point of the day increases abnormally. It is difficult to judge whether or not the increase will be sustained (i.e. a step change) or whether the delay will decrease quickly again (i.e. a spike). The judgement is made here on how large an increase has occurred as spikes, on the whole, tend to be larger increases than step changes. This is by no means a certainty and is used to give a 'best guess' given the available data.

This next section (below) then tests for the presence of a spike, given that exceptions of type 'step', 'loss' and 'start' have not already been detected. The spike ratio describes the scale of the spike in proportion to the standard deviation. This is then used along with the peak value (effectively the size of the spike) to determine whether an exception of type 'spike' has occurred.

```
if ( !type.equals("start") && !type.equals("loss") && (!type.equals("step") && !(test.totalChangeInDelay > limit.peakMax)) ){
  if ( (spikeRatio > limit.spikeRatioMax) && (peak > limit.peakMax)) {
        exception += "Spike of " + peak + " microseconds\n";
              type = "spike";
        }
}
```

The final section taken from the **detectException** function contains the rules for detecting changes in Time of Day Delay Variation. There are two categories of Time of Day Delay Variation. One deals with a change during the working days of the week, the second category flags up a change in the Time of Day Delay Variation on subsequent weekend days. This would compare the first day of the weekend with the last day of the previous weekend and then the next day of the current weekend.

```
if (!weekend) {
    if ( !type.equals("step") &&
          !type.equals("loss") &&
          !type.equals("spike") &&
          !type.equals("start") ) {
        if ( SDRatio > limit.todVarDecrease && SDChange > limit.todVarLimit) {
          if (!wTodVarEx) {
                exception += "Decrease in ToD variation of " + SDChange + " micsecs\n";
                curr.standardDeviation = test.standardDeviation;
                todVarEx = true;
          }
```

```
                else {
                        check = true;
                }
                type = "tod";
            }
            if ( SDRatio <limit.todVarIncrease && SDChange >limit.todVarLimit) {
                exception += "Increase in ToD variation of " + SDChange + " micsecs\n";
                curr.standardDeviation = test.standardDeviation;
                todVarEx = true;
                type = "tod";
            }
        }
    }
    else {
        if ( !type.equals("step") &&
                !type.equals("loss") &&
                !type.equals("spike") &&
                !type.equals("start") ) {
            if ( SDRatio >limit.todVarDecrease && SDChange >limit.todVarLimit ) {
                if (!todVarEx) {
                        exception += "Decrease in ToD variation of " + SDChange + " micsecs\n";
                }
                else {
                        check = true;
                }
                type = "wtod";
            }
            if ( SDRatio <limit.todVarIncrease && SDChange >limit.todVarLimit) {
                if (!todVarEx) {
                        exception += "Increase in ToD variation of " + SDChange + " micsecs\n";
                }
                else {
                        check = true;
                }
                type = "wtod";
            }
            if (check) {
                todVarEx = false;
                check = false;
            }
        }
    }
```

Figure 5.2-5 shows the 'Exception Database Viewer' developed at Loughborough University displaying exceptions detected using rules based on the above. The AIR system was a significantly wider project (see Chapter 2 and also [**Phi99**]) which incorporated some of the early research work conducted for this thesis.

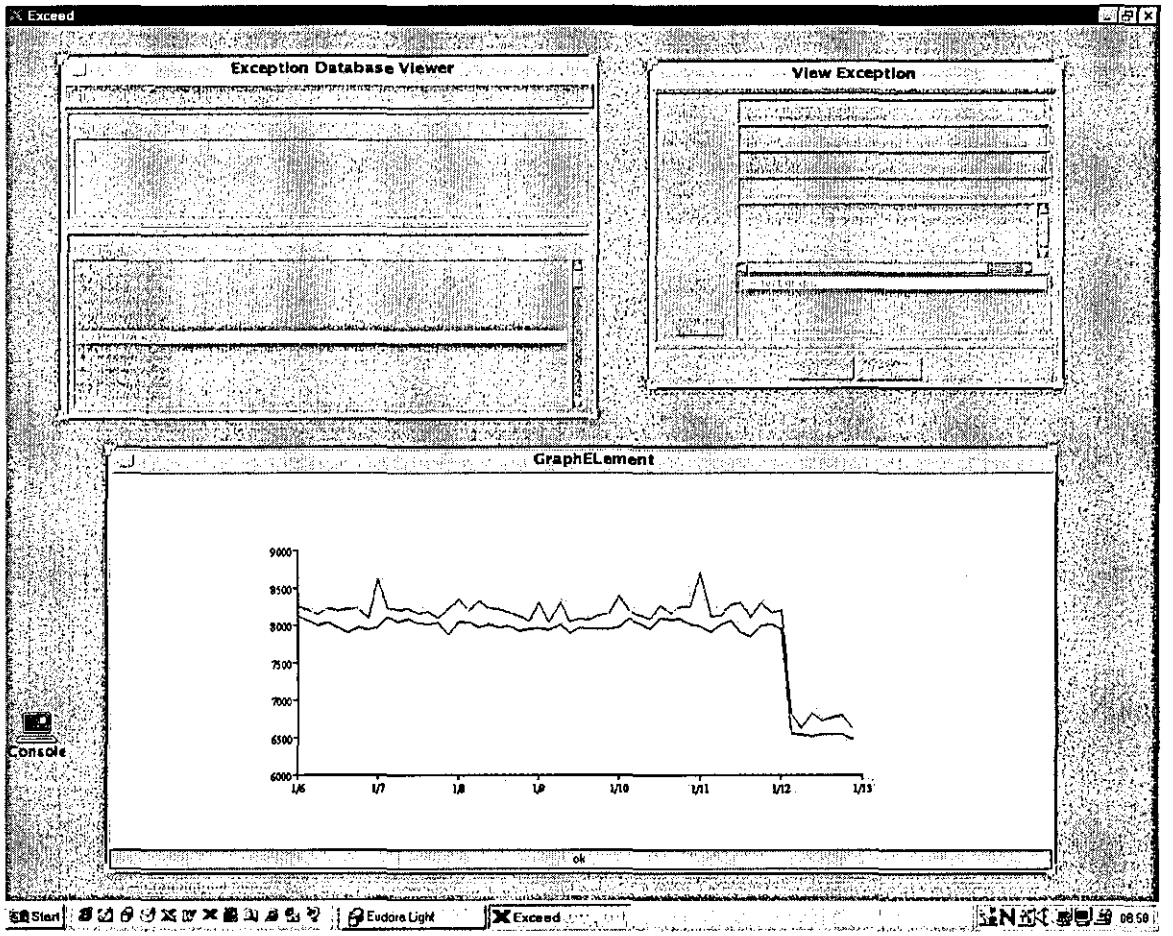


Figure 5.2-5 The Exception Database Viewer

The rule-based approach, as has been mentioned, was implemented as part of the AIR system and installed at BT labs Martlesham. Although the rule-based solution performed adequately as a rudimentary exception detection method it was limited as to what it could detect and also unadaptable. The rule-base catered for a specific set of tightly defined Data Exceptions, should new Data Exceptions be encountered or network characteristics change a new rule-base would have to be constructed. An alternative approach was deemed necessary that would be more adaptable and more accurate.

5.3 KS Test/Neural Approach

The second approach uses two methods, the K-S (Kolmogorov-Smirnov) Statistic and a neural network. The K-S statistic is used to identify changes in the network delay data. This stage however, does not in any way characterise what change has occurred.

These changes are presented to the neural network which is used to classify the changes into one (or more or none) of seven Data Exception types.

The K-S statistic has been chosen as it is the best known of several distribution-free procedures which compare two sample CDFs (Cumulative Distribution Functions) in order to test for general differences between two distributions [**Nea88**][**Ste70**]. The K-S statistic is more generally used to assess the probability that a sample comes from a normally distributed data set. Although more powerful than other goodness-of-fit tests such as the chi-squared [**Mas51**], the Shapiro-Wilk's test has been shown to be more powerful still [**Sha68**]. With regard to testing for general differences between any two distributions however, the K-S statistic is both easy to calculate and powerful. As there is no assurance that the delay distribution will be in any way standard, our choice of test is restricted to distribution-free techniques. More powerful distribution-free tests exist if only one aspect of the distribution is of interest (for instance the mean), but the K-S statistic is particularly appropriate when testing for general differences [**Nea88**]. The K-S test compares two samples CDFs using the maximum vertical distance between them as a test statistic. A normalised example of this is shown in Figure 5.3-1. Any kind of substantial difference between the two distributions should show up as a significantly large difference between the sample CDFs. Such differences may be in location, spread or may be more general differences in the shape of the distributions.

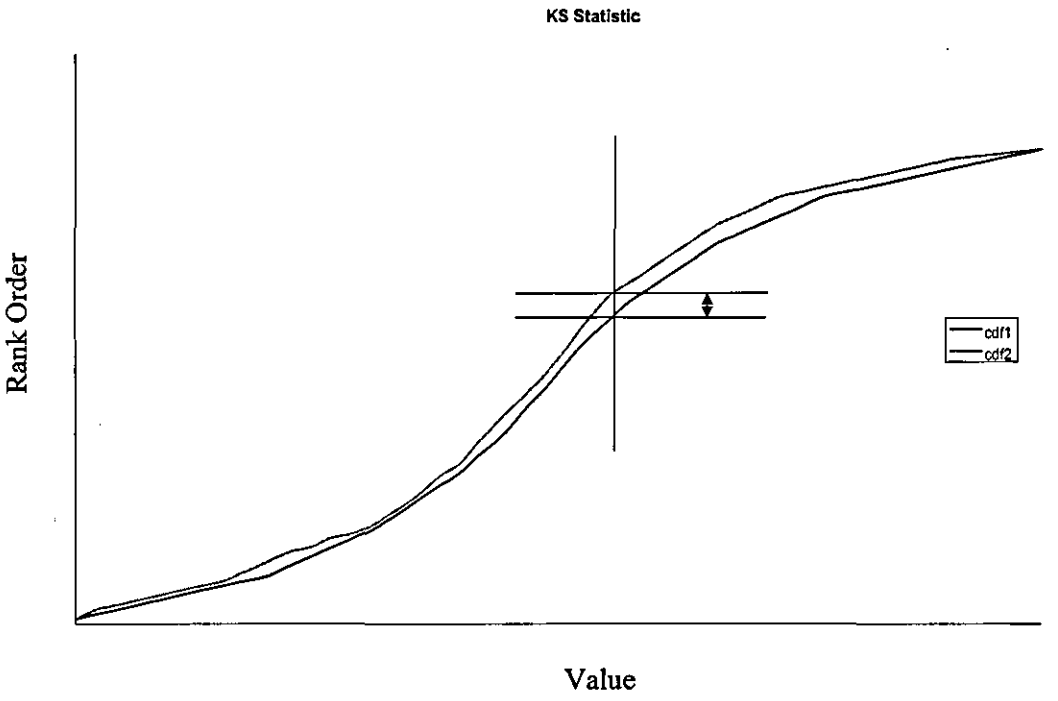


Figure 5.3-1 The K-S Statistic

The K-S statistic is calculated as follows:

Delay distributions are taken from 24 hours worth of data before and after some point in time and the K-S test is then applied to determine any differences. Delay increases during the working hours of the day, when load is high, but this increase is not exceptional. If distributions were chosen using a time period other than complete days this change during the working part of the day would be picked up on by the K-S test. By comparing data taken from two entire days this issue is circumvented. This issue is also relevant when considering the impact of weekends. Delays on a Monday are higher than of those on a Sunday but this is not considered to be exceptional. Using the current approach the K-S test does flag the beginning and end of weekends as significant. The second phase of the system, the trained neural network, will then categorise this change. This means that the neural process could filter out exceptions flagged due to the decrease in delay at weekends.

The implementation computes the K-S statistic every hour, using the previous 24 hours of delay data, and the following 24 hours of data. When a Data Exception occurs the K-S statistic may remain significant for several hourly points. In such cases the maximum point is taken to be the time that the exception occurred. Where two exceptions occur at similar or even identical times these would be passed to the neural classifier as one detected change, but there is scope within the neural process to categorise the change as being the product of two or more exception types. In Figures 5.3-2, 5.3-3 & 5.3-4, the K-S test is plotted on the same graph as delay (these are raw delay values). The scale on the primary y-axis refers to delay while the scale on the secondary y-axis refers to the K-S test (which will always return a value between 0 and 1). The x-axis is the number of hours from the beginning of the simulation.

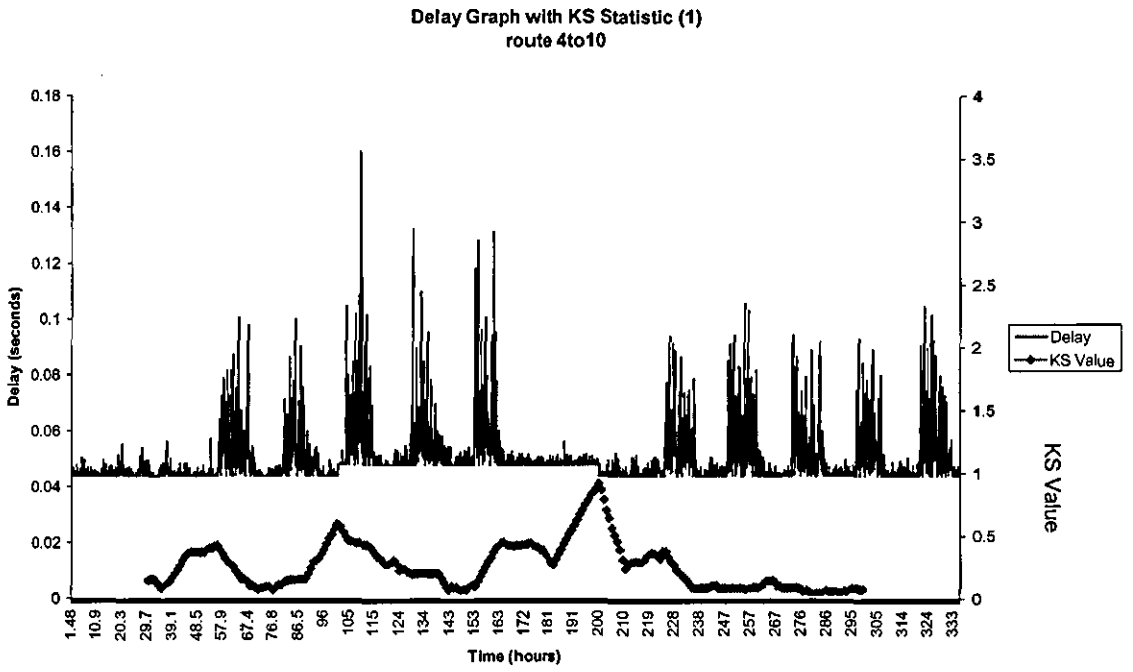


Figure 5.3-2 The K-S Test applied to delay data (1).

In Figure 5.3-2 the highest two peaks have been caused by the step changes present but the K-S values are also high at the beginning and end of each weekend. In Figure 5.3-3 the K-S values are again high at the beginning and end of each weekend, there is also a peak marking the point where the time of day variation increases.

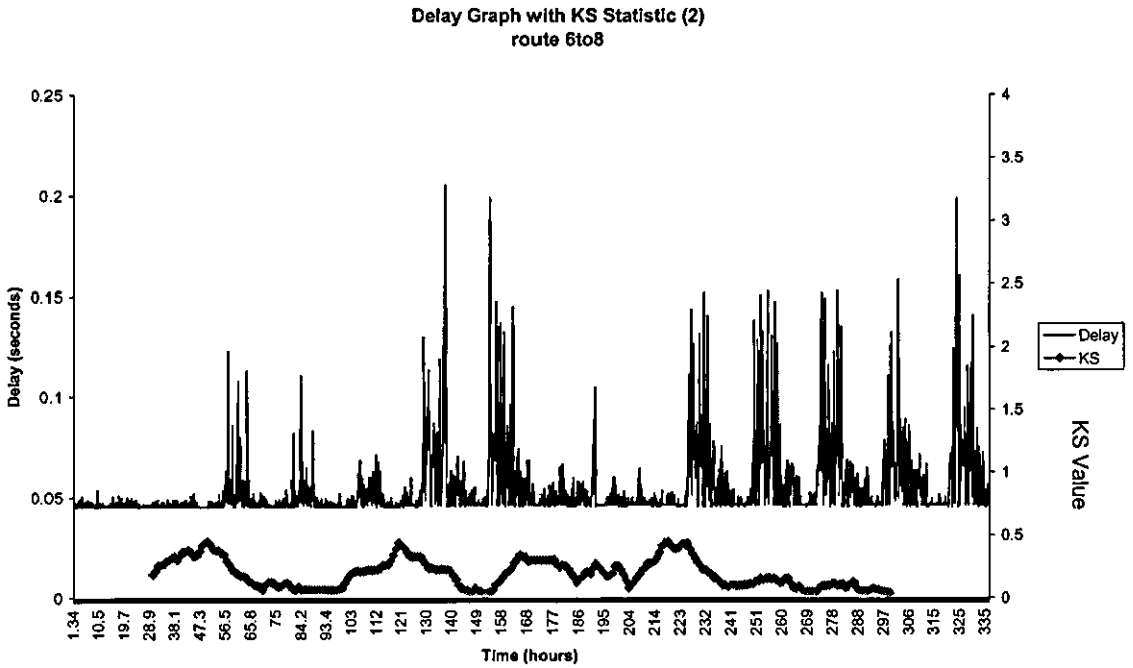


Figure 5.3-3 The K-S Test applied to delay data (2).

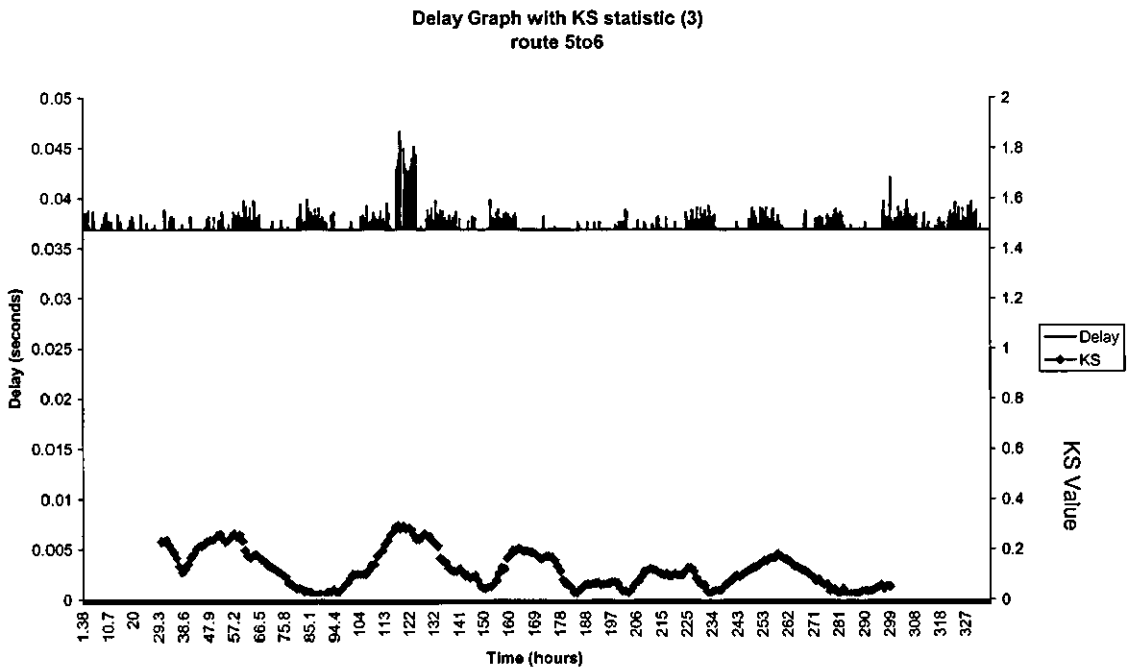


Figure 5.3-4 The K-S Test applied to delay data (3).

In Figure 5.3-4, the route shown is less heavily loaded. There are five relatively minor peaks in the K-S values. Three of these peaks relate to the beginning and end of

weekends, one refers to the spike and one is erroneous. Erroneous events flagged up by the K-S statistic are filtered out by the neural process.

Note that the nature of the test means that this approach cannot be adapted to work in real time, although with modification, the approach could be made to work on shorter time windows. The motivation behind the work stems from the desire to automate the exception detection process, which is currently done by a human operator offline in order to investigate the impact of recent changes to the network. Real time event detection is geared towards alerting operators to immediate faults. While delay information could be used in this way there are other more readably accessible performance metrics, often held by network nodes in MIBs, that can be used to identify faults. For real-time detection methods some preliminary approaches are being examined that borrow techniques from industrial statistics such as process charts [**McG00**]. These however cannot identify the range of Data Exceptions detected by the approach proposed in this Thesis as the time window is shorter. For example, a meaningful Time of Day Delay Variation exception is only identifiable after a 24 hour period.

5.4 Neural Network

A neural network was selected as a means of classifying the changes in performance detected by the K-S test. A means of classifying the change in performance is desirable as it provides a means for changes, that is Data Exceptions, to be grouped together (see section 3.4). A neural network was selected as a method of classifying the Data Exceptions. Neural networks are particularly appropriate where some relationship exists between the input and output (in this case between the data representing the Data Exception and the classification of the Data Exception) that cannot be expressed as function or as a set of rules [**Tar98**].

The neural network is trained using a standard back propagation algorithm (for a description of this see [**Fau94**]). The network is fed by an input vector that includes representations of the time-of-day and day-of-week that the exception occurred; the route on which the exception occurred; and the delays either side of the time that the exception occurred. The inputs are scaled so that, on the whole, they are in the range –1 to 1. In every case some kind of transformation takes place. Input features whose values have no

relative meaning compared to one another are represented as binary vectors in order to ensure that no one input is given significance over another by virtue of an arbitrarily assigned value. For instance, one egress point is of no greater or lesser significance than another egress point. If the egress points had been represented using one input, relative value would have been attributed which would be misleading.

The time-of-day needs to be represented in such a way that 23:59 is next to and not the furthest possible point away from 00:00. To achieve this a Sine function is used. Considering time of day in hours the transformation is:

$$time = \sin(\pi \frac{time - 12}{12})$$

The day of the week, the ingress point and the egress point are all represented as vector inputs. Therefore the day of the week is represented by seven inputs where one input will be 1 (the day the exception actually occurred) and the others –1.

The delays are represented by 96 inputs, allowing 48 for each day. The 50^{th} and 95^{th} percentile values are taken from each hour of the day and these are then scaled so that generally they fall within the range 1 and –1. The scaling has been set so that it is possible for large delay values to transform to values greater than 1. This is so that the delays on the whole do not all scale to similar values but that reasonable spread is attained. If the maximum observed delay (on any test, on any route) were to be represented as 1, the vast majority of delays would fall in a very narrow band.

From the above we have 8 inputs to represent the ingress point, 8 for the egress point, 7 for the day of the week, 1 for the time of day, 96 for the delay values and 1 input for the K-S value itself. The total length of the input vector is therefore 121. These inputs are fed into a hidden layer containing 150 nodes and then into an output vector of length 7, representing the seven classes of exception (Figure 5.4-1). The choices made here are somewhat arbitrary and could be the subject of further work to determine which representation of the data, and number of hidden units will give the greatest accuracy in classification.

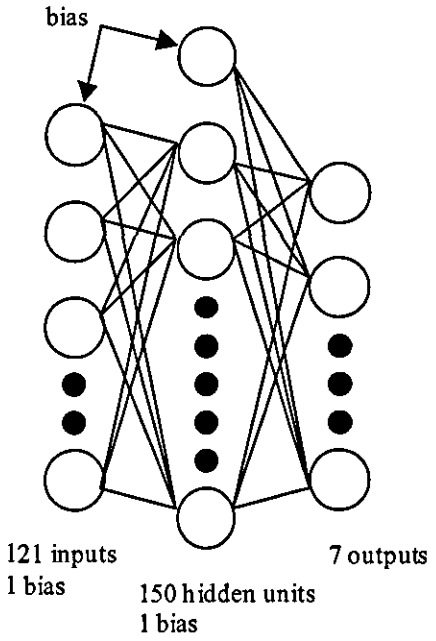


Figure 5.4-1 The neural network

The activation function used at each node is given below.

$$f(x) = \frac{2}{1+\exp(-x)} - 1$$

which has the derivative (necessary for backpropagation of the error)

$$f'(x) = \tfrac{1}{2}[1+f(x)][1-f(x)]$$

Weights are initialised at random.

The output vector represents the class of Data Exception. It might be identified as none, one or more of the following classes:

- The beginning of a weekend
- the end of a weekend
- a step change up
- a step change down
- an increase in time-of-day delay variation
- a decrease in time-of-day delay variation
- a spike

Each component of the output vector is rounded to either 1 or –1 to indicate whether or not the Data Exception falls under that classification.

5.5 Summary

In this chapter two approaches to detecting Data Exceptions have been discussed and described. The first, a rule-based solution, was implemented as part of the AIR system. While a rule-based solution is advantageous in that it is simple, executes quickly and is predictable it was thought to be inflexible when adapting to new circumstances and was sufficiently inaccurate to necessitate an alternative approach.

The second approach made use of the K-S statistic to determine when a change in network performance, that is a Data Exception, had occurred. These changes were then presented to a neural network, which was used to classify them into Exception Types.

In the following chapter the results are presented from testing the K-S/Neural approach with the available data sources. The test schedule is discussed, delay graphs are given showing the corresponding K-S values and the classification accuracy is given.

Chapter 6 – The K-S/Neural Approach - Results

6. The K-S/Neural Approach Results

6.1 Introduction

The K-S/Neural approach was applied to data taken from both the simulation and the test network. Network Events were introduced in the manner described in Chapter 4. The K-S Test was applied to the resultant delay data and these Data Exceptions were categorised manually using **label**, a purpose-built program. The list of categorised Data Exceptions was split into two files, one for training the neural network and one for testing the neural network. The training and testing phases were carried out and the results evaluated. This chapter describes this process in more detail and presents the results.

6.2 Simulated Data

Delay data was generated using the NS simulation package as described in Chapter 4. The simulation was run 24 times, each run lasting fourteen virtual days. At least one network event was introduced into each run of the simulation. The figure below (Figure 6.2-1) is given for ease of reference.

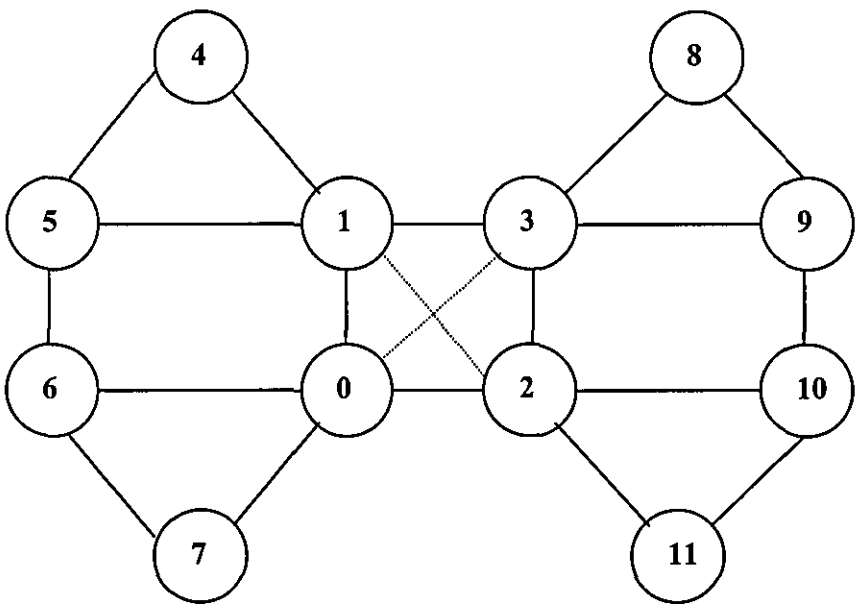


Figure 6.2-1 The NS simulation topology

The simulation runs were chosen so that all the different types of Data Exception would occur, at different types and on different routes. On the simulation only a limited

number of events could be introduced. Links could be taken down or introduced, traffic could be re-routed and links could be made to drop packets intermittently. At least one of these events was introduced, sometimes in quick succession to cause Data Exceptions of type Spike. The simulation runs are summarised below.

1) Core Change 1	2) Core Change 2	3) Routing Change
4) Link 1 to 2 Down	5) Node 0 Down	6) Link 2 to 3 Down
7) Link 0 to 4 Down	8) Link 6 to 7 Down	9) Link 9 to 10 Down
10) Link 10 to 11 Down	11) Link 2 to 10 Down	12) Core fails intermittently
13) Traffic increase	14) New link 4 to 1	15) Core Re-route 1
16) Core Re-route 2	17) Spike 1	18) Spike 2
19) Spike 3	20) Spike 4	21) Spike 5
22) New link 0 to 2	23) New link 3 to 1	24) Link 2 to 11 down

These represent the entire set of changes that could be made. The network events have not been applied exhaustively, more links could have been taken down for instance, but further changes would not introduce any additional types of Data Exceptions. The following sections detail the 24 simulation runs, describing the network events that were introduced.

6.2.1. Core Change 1

At time 120 a link was introduced between nodes 1 and 3. At time 122 the links between nodes 1 and 2 and between 0 and 3 were brought down. This caused Data Exceptions of several types.

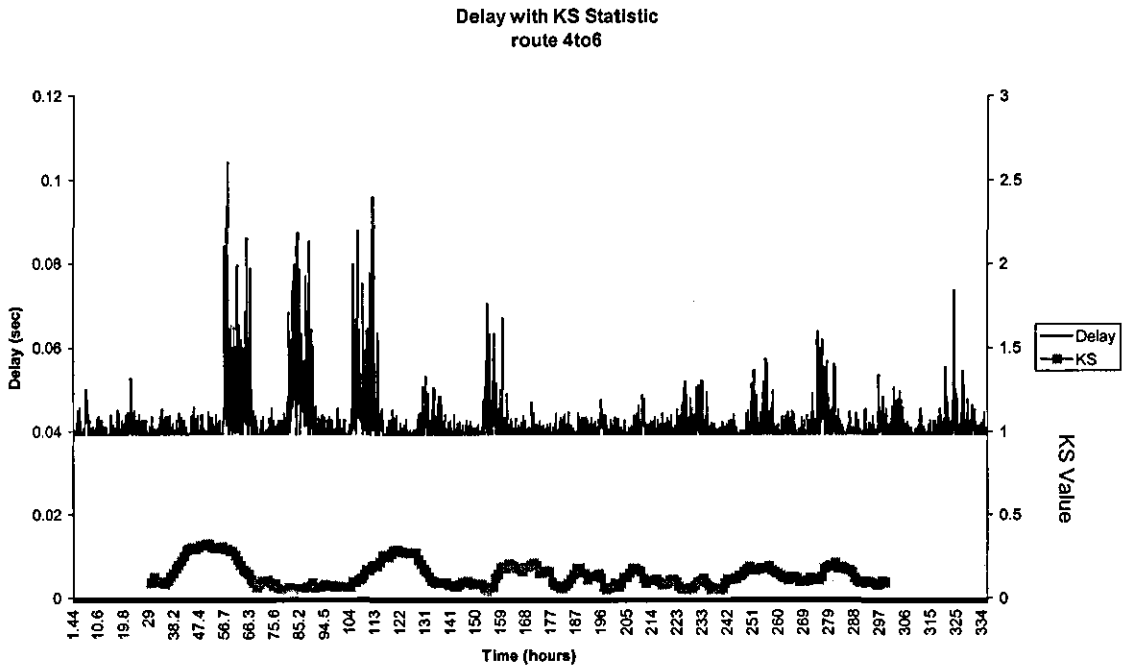


Figure 6.2-2 Delay/KS graph route 4to6

This first graph (figure 6.2-2) shows a change in Time of Day Delay Variation. This is indicated by a peak in the K-S test (at time 122). The K-S test also has a peak representing the end of the first weekend. The K-S test is also significant at the beginning of the second weekend (time 168) and once during the second week (time 281). The last of these values does not represent any network event. This would be labelled as 'not an exception' so that the neural network can be trained to filter out such anomalies.

The route shown in figure 6.2-2, between node 4 and 6, is one of the least affected as the monitoring traffic between the two nodes (4 and 6) traverses the same links before and after the changes introduced at times 120 and 122. However the traffic load on this route is lightened by the re-routing that the changes cause.

The changes in performance on other directly effected routes were more dramatic. Figure 6.2-3 shows the impact the alterations had on the route between nodes 6 and 11. As the traffic now has to take a longer route to reach its destination there is a step change in delay. Also as there is now only one link (1 to 3) connecting the two halves of the

network and this link has a capacity of 2Mb/s as opposed to 5Mb/s of the other core links, it is consequently very heavily loaded.

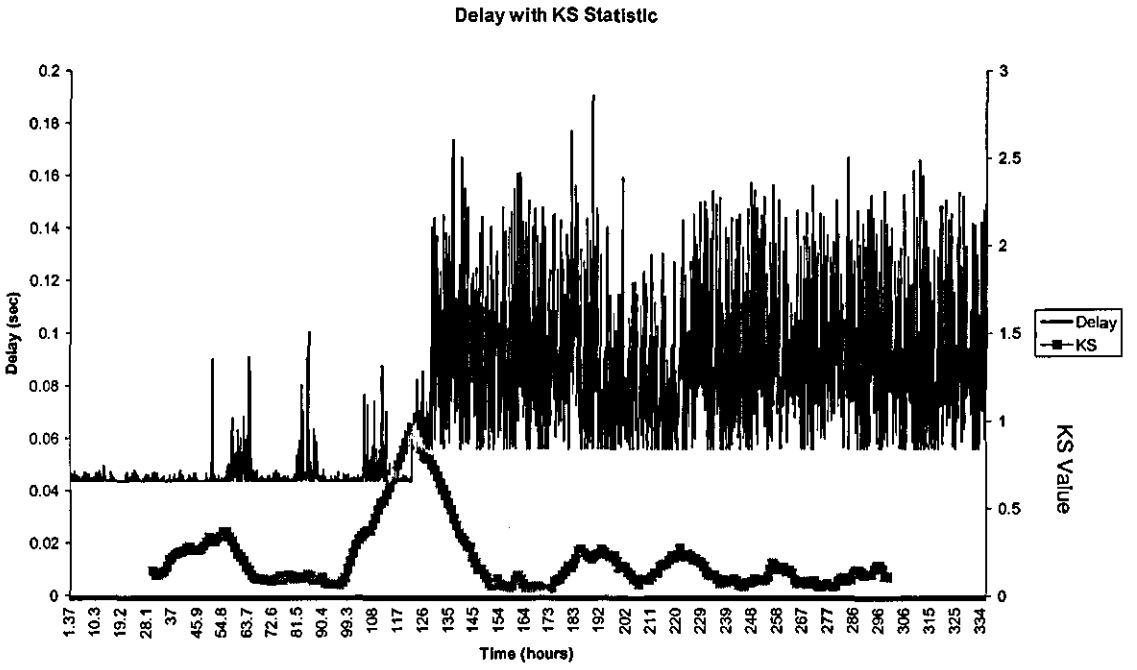


Figure 6.2-3 Delay/KS Graph route 6to11

6.2.2. Core Change 2

At time 120 the routes from 1 to 3 and from 0 to 2 were activated. At time 122 the links between 1 and 2 and between 0 and 3 were taken down. This event was similar to the one above except that the introduction of two new links avoided the extreme congestion seen in the previous section. Figure 6.2-4 below gives a comparison on the route from node 6 to node 11.

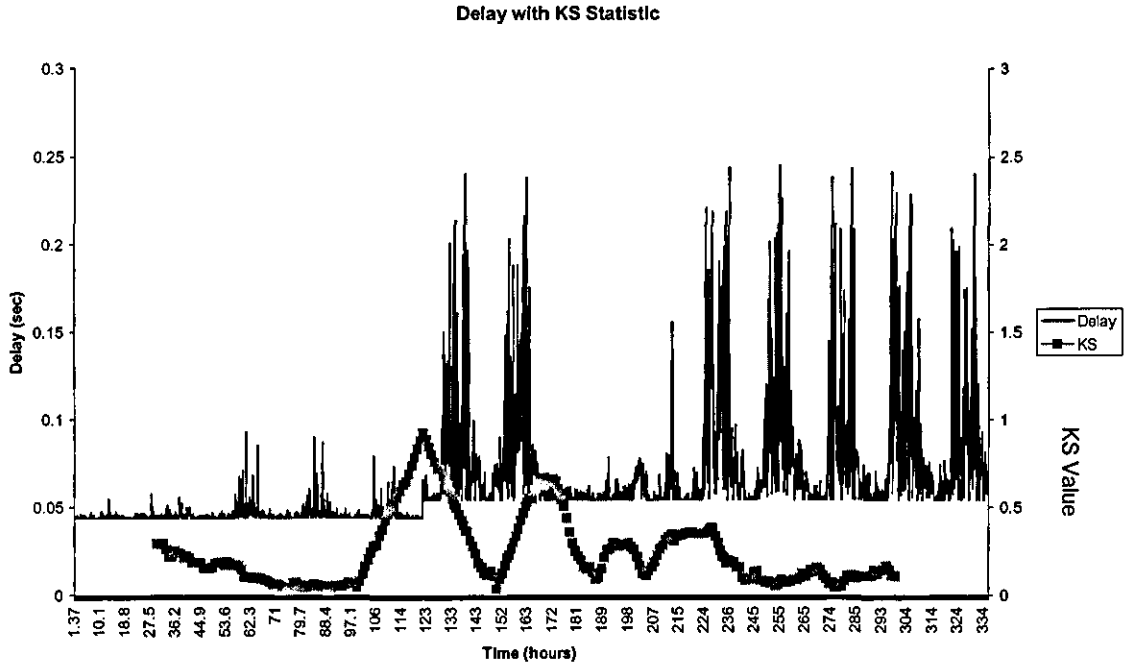


Figure 6.2-4 Delay/KS Graph route 6to11

6.2.3. Routing change

At time 138 the routing is changed by assigning a 'cost' value to the links from 4 to 5, from 5 and 6 and from 6 to 7 of 1 – reduced from a previous value of 5. The 'cost' value is returned to 5 on these links at time 200. The effects of this change can be seen in the graph below which plots the monitored delay data from node 5 to node 7. The costing of the routes was implemented asymmetrically. That is to say that the route from 7 to 5 does not exhibit the same characteristics. The step changes in figure 6.2-5 reflect the increased latency attached to the different route.

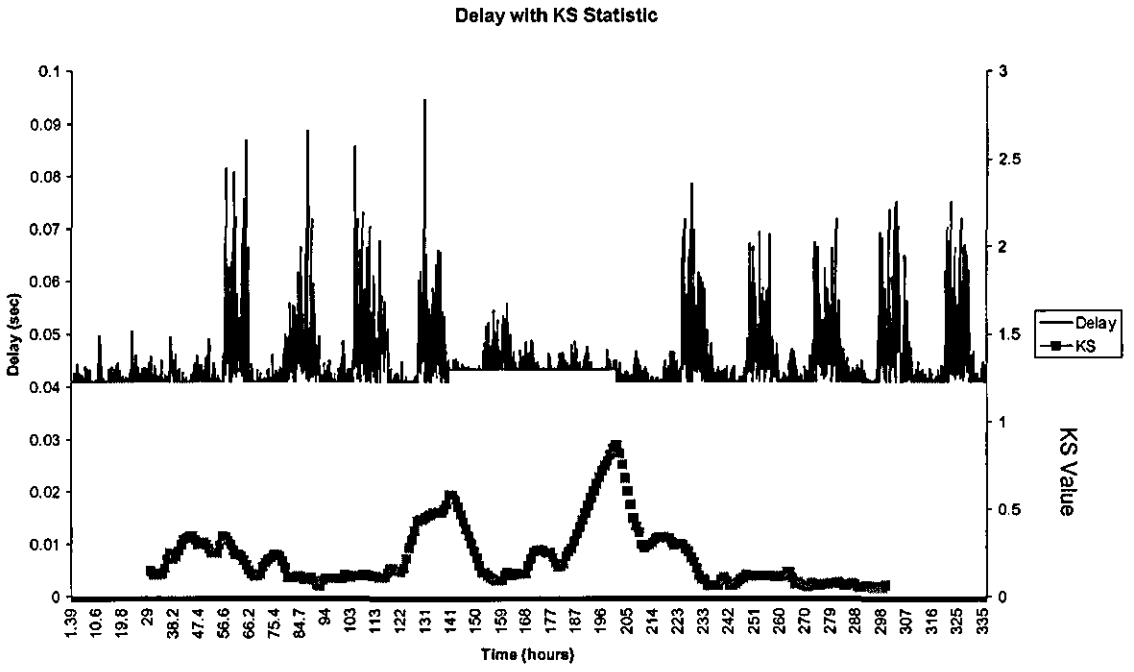


Figure 6.2-5 Delay/KS Graph route 5to7

6.2.4. Link Down 1 to 2

The link between nodes 1 and 2 was taken down at time 170. Although this reduced the number of links supporting traffic between the two halves of the network down to 1 (the link between nodes 0 and 3), the capacity of that remaining link meant that the congestion was not as high as that experienced in the 'Change Core 1'. In figure 6.2-6 we see the effects on the route from node 11 to node 7. There is a large step change at the time of the event and also a noticeable difference in the Time of Day Delay Variation for the subsequent week.

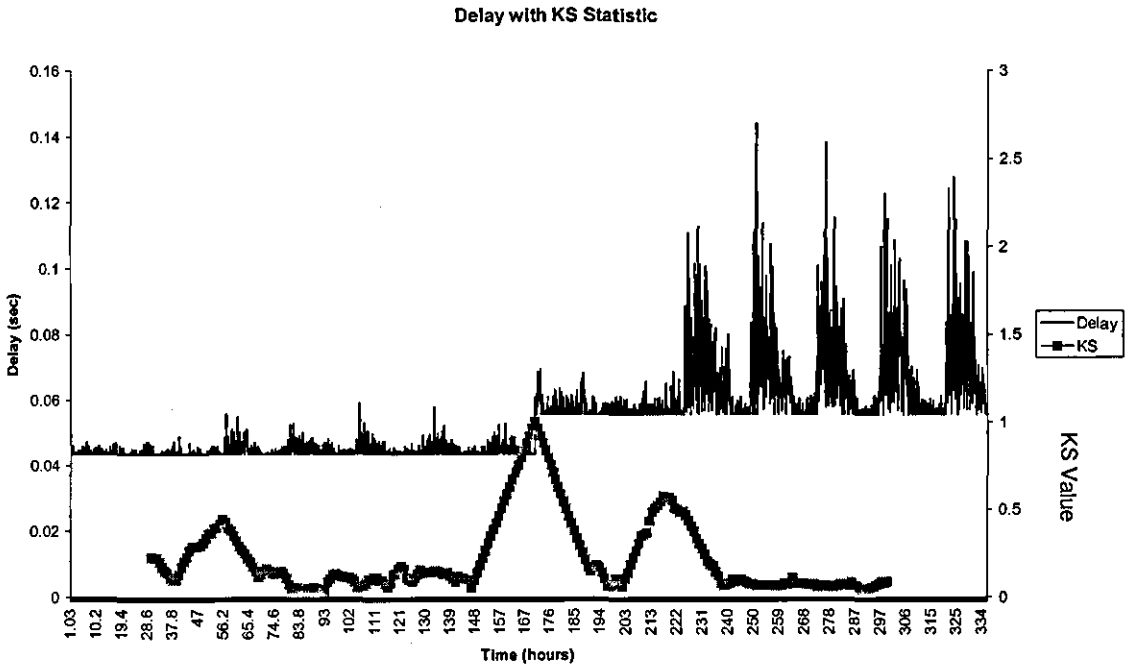


Figure 6.2-6 Delay/KS Graph route 11to7

6.2.5. Links down from node 0

At time 300 all the links attached to node 0 were taken down. Those links are 0 to 1, 0 to 3, 0 to 4 and 0 to 5. This caused step changes and changes in Time of Day Delay Variation on various routes.

6.2.6. Link down 2 to 3

At time 133 the link from node 2 to node 3 was taken down. This resulted in step changes and Time of Day Delay Variation changes on certain routes.

6.2.7. Link down 0 to 4

At time 200 the link from node 0 to node 4 was taken down.

6.2.8. Link down 6 to 7

At time 260 the link from node 6 to node 7 was taken down.

6.2.9. Link down 9 to 10

At time 82 the link from node 9 to node 10 was taken down.

6.2.10. Down 10 to 11

At time 124 the link between node 10 and node 11 was taken down. This caused step changes on the routes between nodes 10 and 11.

6.2.11. Down 2 to 10

At time 10 the link between node 2 and node 10 was taken down. This link was reinstated at time 78 and then taken down again at time 254. This gave multiple step changes as can be seen in figure 6.2-7.

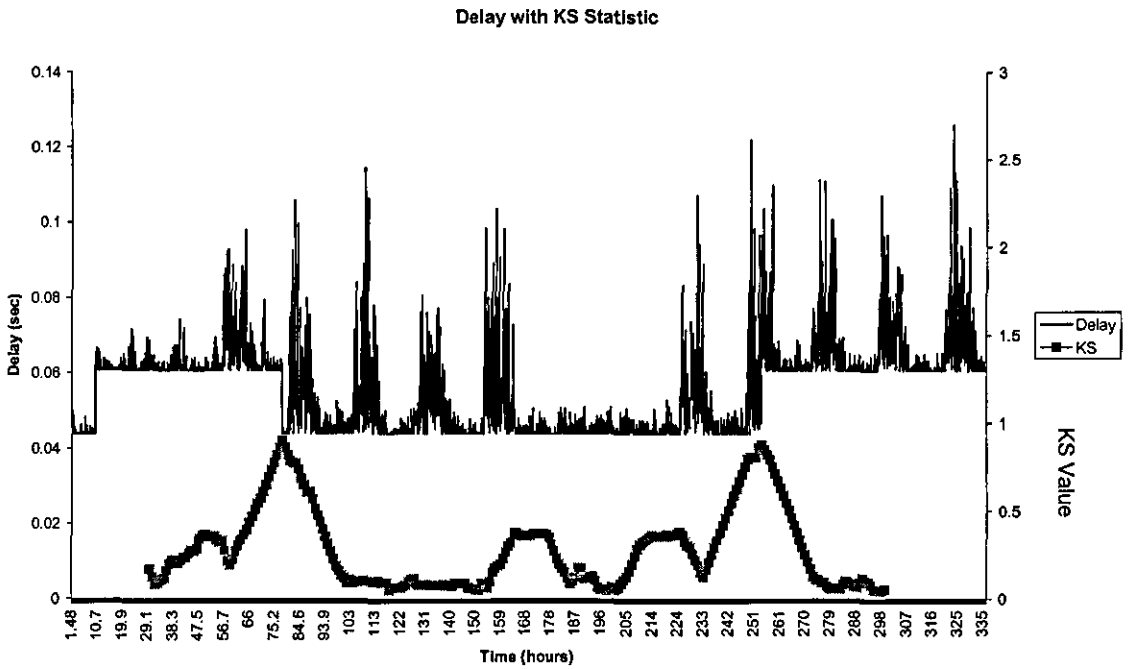


Figure 6.2-7 Delay/KS Graph route 4to10

6.2.12. Faulty Core

At time 55 the core links, between node 0 and node 1, node 0 and node 3, node 1 and node 2 and node 2 and node 3 were caused to fail intermittently. At time 95 the link between node 1 and node 3 was brought up. The failure rate was 33%. At time 100 the core links were returned to their normal state. Figure 6.2-8 shows the kind of effect the faulty links had.

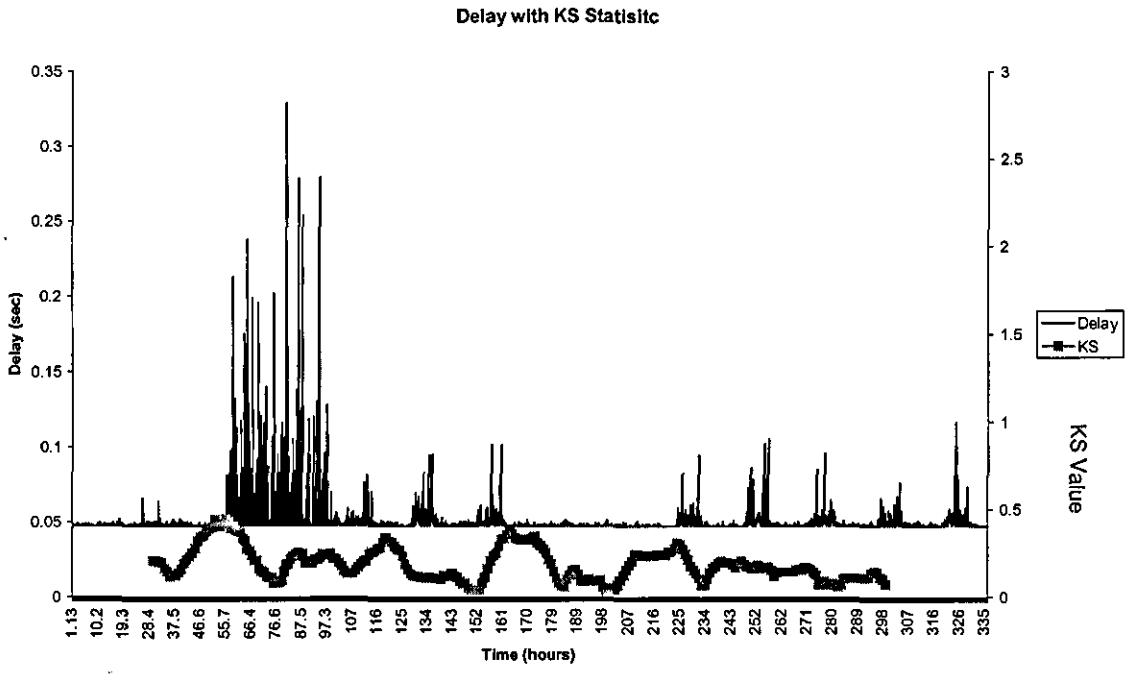


Figure 6.2-8 Delay/KS Graph route 9to4

6.2.13. Increase 5 to 6

At time 168 the traffic sources on the link between node 5 and node 6 start to transmit at a higher rate. This had no impact on the delay data. Presumably the link was under-utilised and could bear the extra traffic load.

6.2.14. New link 4 to 1

At time 75.9 a new link is introduced between node 4 and node 1. This causes a step change down on certain links to and from node 4. It also causes a reduction in Time of Day Delay Variation on some routes where the load has been decreased.

In figure 6.2-9 (below), the step change reflects the shorter route now available. The lightening of the load on other links is shown in figure 6.2-10. In figure 6.2-10, the K-S statistic does not show the seeming change in Time of Day Variation. The occasions where this is the case are very rare.

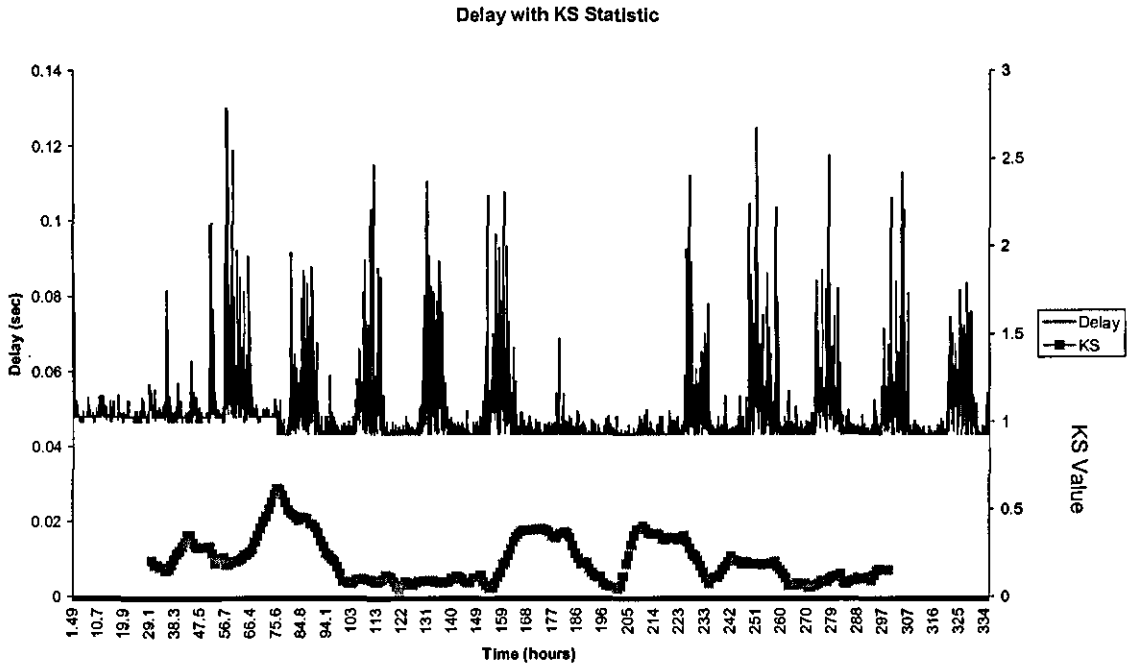


Figure 6.2-9 Delay/KS Graph route 4to11

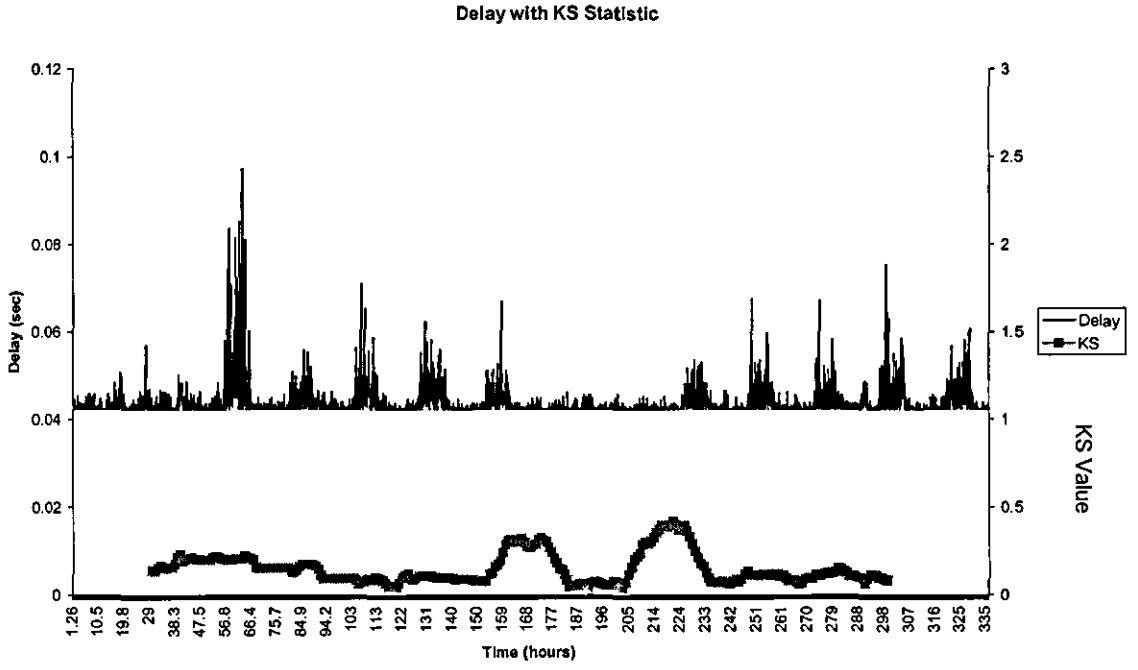


Figure 6.2-10 Delay/KS Graph route 7to5

6.2.15. Core Re-route 1

The links between node 0 and node 1 were given an increased cost at time 100. The cost on this link was returned to normal at time 200. Also at time 200 the cost on the links between node 2 and node 3 was increased. This causes step changes on certain links that traverse the core (nodes 0, 1, 2 & 3) and changes in time of day variation in others.

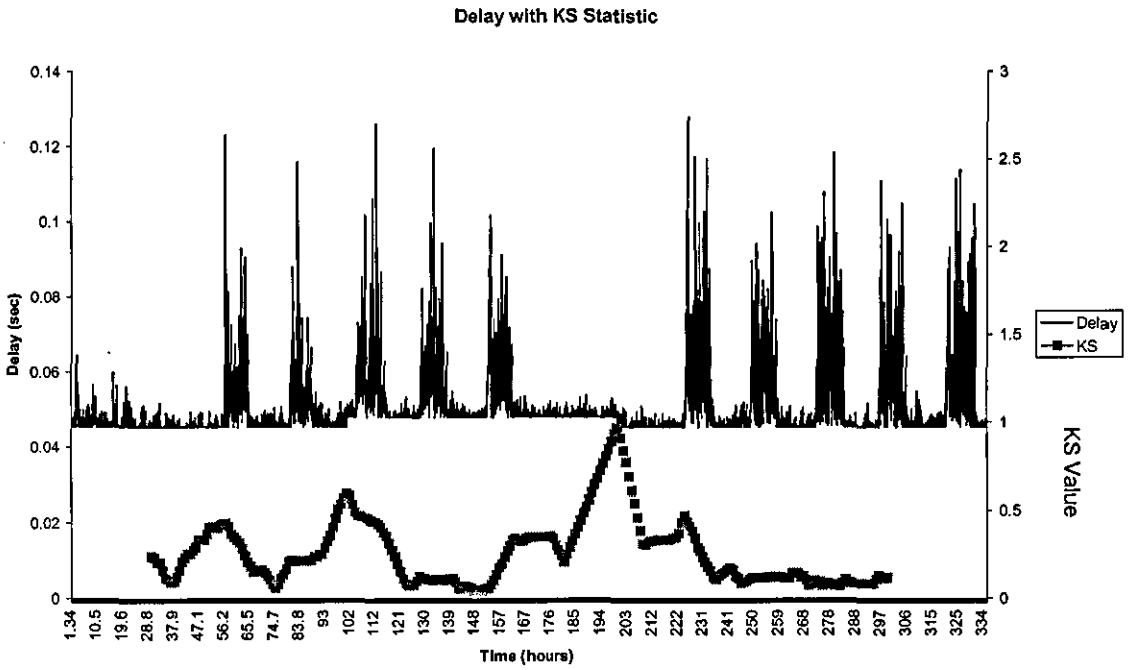


Figure 6.2-11 Delay/KS Graph route 6to8

In figure 6.2-11 (above) the two changes cause step changes as the test packets are sent over different links. In figure 6.2-12 (below) there are no step changes present but the time of day variation in delay changes as traffic is routed away from this link. This change is marked by a line corresponding to the peak in the K-S Statistic.

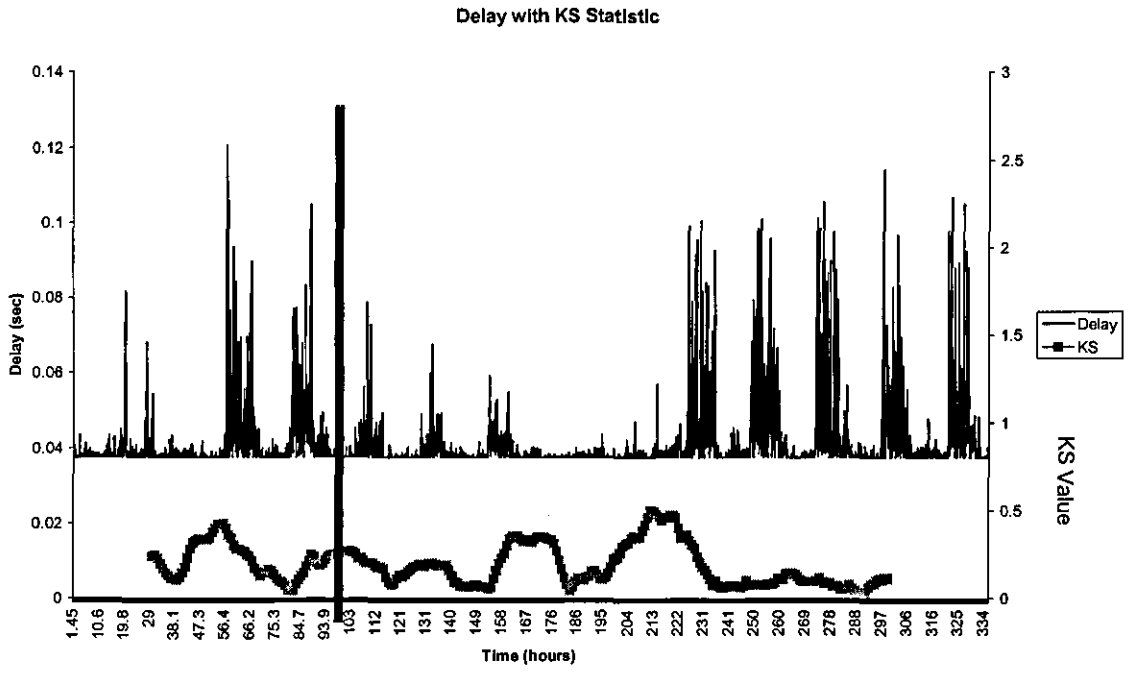


Figure 6.2-12 Delay/KS Graph route 7to5

6.2.16. Core Re-route 2

At time 100 the cost of the link between node 0 and node 3 was increased thereby routing traffic away from this link. At time 200 the cost of the link between node 0 and node 3 was restored to its normal value but the cost of the link between node 1 and node 2 was increased. Step changes and time of day variation in delay changes were seen similar to those above.

6.2.17. Spike 1

At time 115 the links between node 5 and node 6 were made faulty until time 125. This caused a spike in the delay on these links as shown in figure 6.2-13.

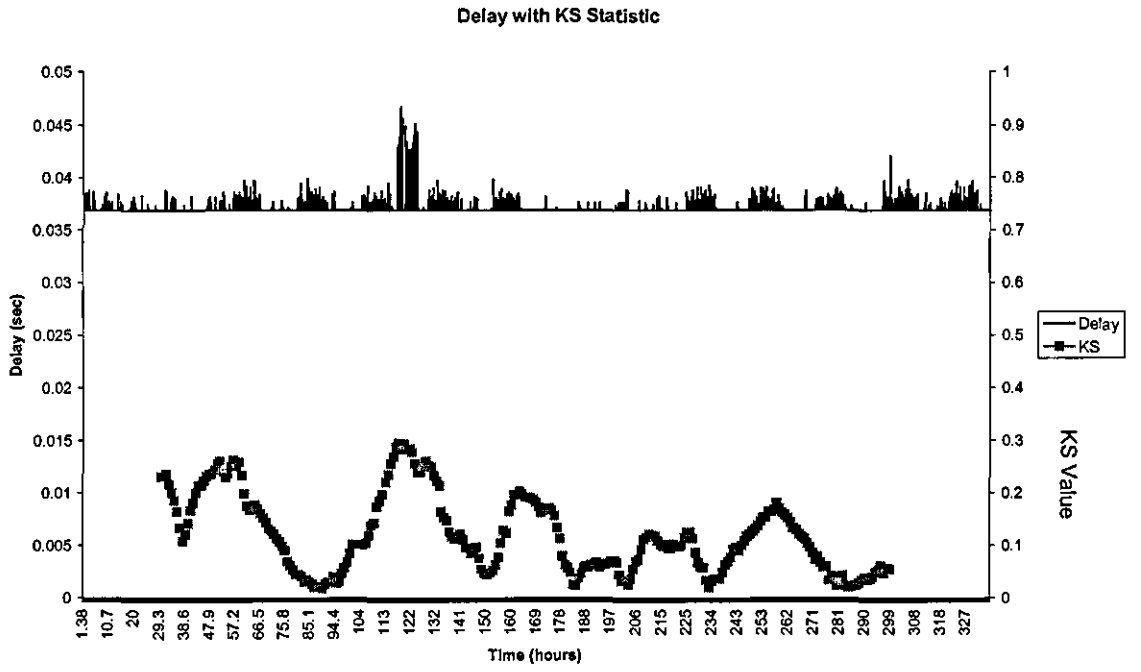


Figure 6.2-13 Delay/KS Graph route 5to6

6.2.18. Spike 2

At time 165 a fault was introduced on all links connected to node 0. This caused step changes and spikes on several routes passing through node 0.

6.2.19. Spike 3

At time 260 the links between node 0 and node 1 and between node 1 and node 2 are taken down. At time 264 these links are restored. This caused a large step change up followed quickly by a step change down. This is classified as a spike.

6.2.20. Spike 4

At time 205 the links between node 1 and node 2 and between node 2 and node 3 are taken down. At time 213 these links are restored. The effects are similar to those described for spike 3.

6.2.21. Spike 5

At time 245 a fault is introduce to the links between node 0 and node 3 and between node 2 and node 3. This resulted in some small spikes and step changes on various routes.

6.2.22. New link between 0 and 2

At time 95 a new links was introduced between node 0 and node 2. This caused step changes on several routes as shown in figure 6.2-14.

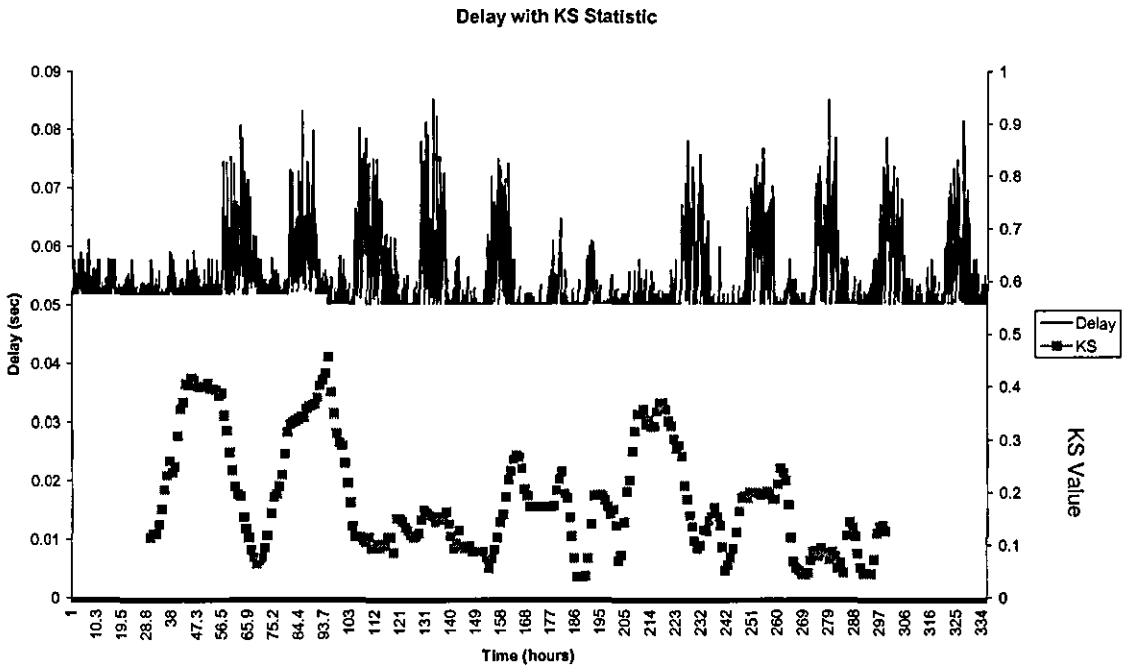


Figure 6.2-14 Delay/KS Graph route 11to4

6.2.23. New link between 3 and 1

A new link is introduced at time 5 between node 3 and node 1. This link is taken down again at time 87 and then re-established at time 260. The effects are minimal although there are changes in time of day variation on certain routes that would incorporate this link.

6.2.24. Link down then up 2 to 11

At time 82 the link from node 2 to node 11 is taken down. The link is restored at time 216. This causes step changes on all routes from or to node 11 (figure 6.2-15).

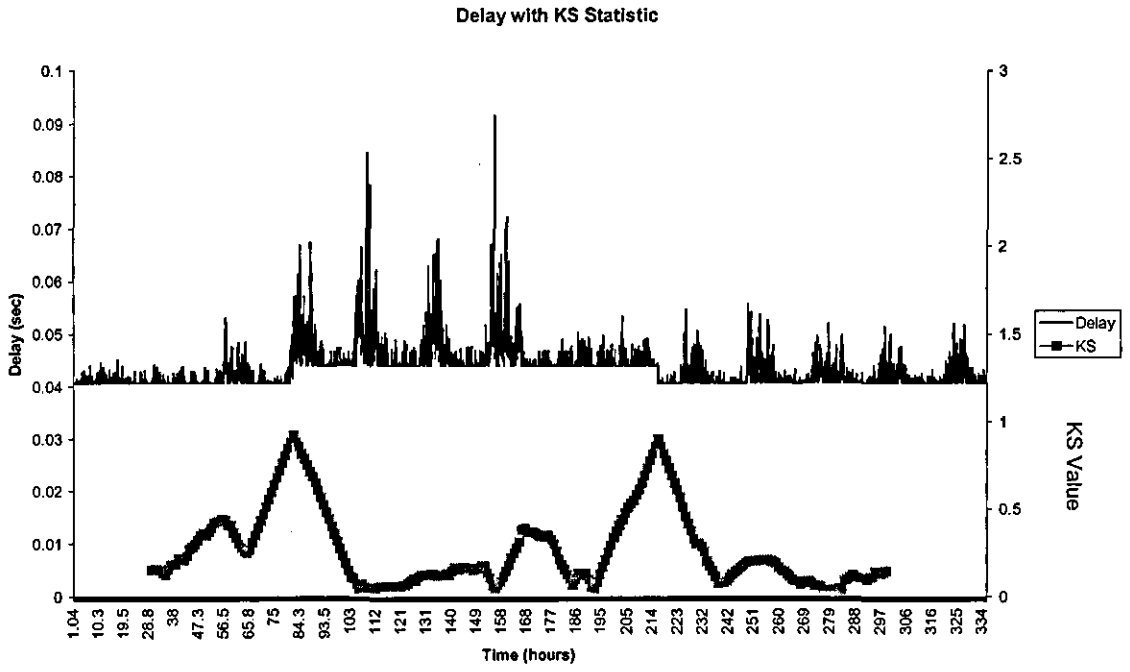


Figure 6.2-15 Delay/KS Graph route 11to8

6.3 Data from the Test Network

Delay data was generated using the test network as described in Chapter 4. The network was used to generate 21 sets of data, each lasting fourteen virtual days. Each data set contained at least one monitored event. Figure 6.3-1 is given for ease of reference.

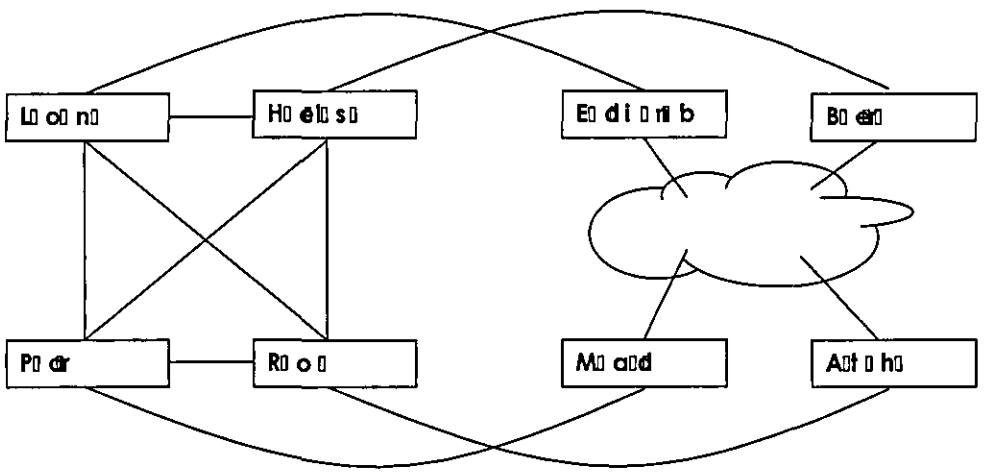


Figure 6.3-1 Test Network Layout

The test network could be altered in a number of different ways. The routers could be configured enabling the test network to have a very large number of possible configurations. Many of these had no perceptible impact on the network performance as monitored by delay. Events that gave clear Data Exceptions were disabling interfaces on the routers, restarting a router, changing the bandwidth on a link, reconfiguring the routing and changing the queue length on a certain link.

The table below summarises the events that were introduced.

1) Berlin interface disabled	2) Increase in Athens traffic	3) Berlin Ethernet down
4) Berlin Serial down/up	5) Clean (no events)	6) Edinburgh Bandwidth 1
7) Edinburgh Bandwidth 2	8) Edinburgh Queue change	9) Edinburgh Ethernet down/up
10) Edinburgh Serial down	11) Edinburgh Serial up	12) 3 serial links taken down
13) London reload	14) Madrid Ethernet down	15) Madrid Ethernet up
16) Network 11 down + routing	17) Network 11 down	18) Paris Serial up
19) Routing change	20) Rome serial up	21) Rome serial up 2

The following sections detail the 21 data sets, describing the network events that were introduced.

6.3.1. Berlin Serial interface disabled

At time, the Serial interface connecting Berlin to Helsinki was enabled (it had previously been disabled). This caused Step Changes and Time of Day Variation in Delay Changes. These can be seen in Figure 6.3-2 below.

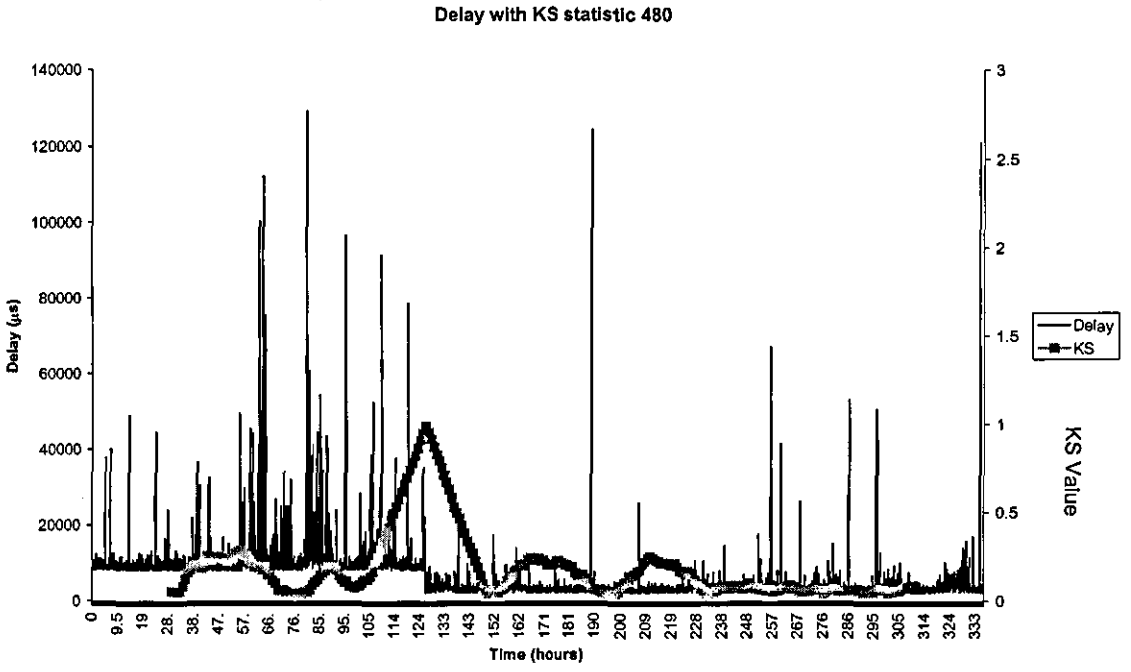


Figure 6.3-2 Delay/KS Graph route Helsinki to Berlin (small packets)

6.3.2. Increased Athens Traffic

Additional traffic sources were introduced from the Athens traffic generator for the last three virtual days. The traffic level was increased for the last of these three days. This caused an increase in Time of Day Variation in Delay. This can be seen in figure 6.3-3.

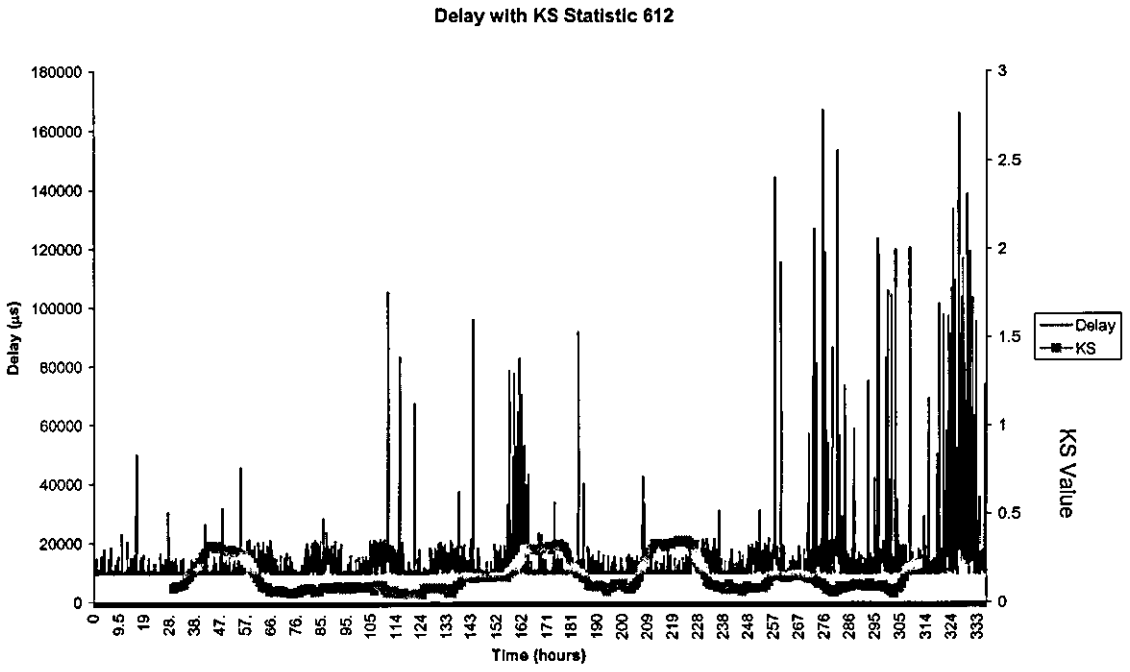


Figure 6.3-3 Delay/KS Graph route Madrid to London (large packets)

6.3.3. Berlin Ethernet Port down

The Ethernet port connecting the Berlin router to network 11 was taken down and then brought back up again. This caused Step Changes and Time of Day Variation in Delay changes on various routes as below (figure 6.3-4).

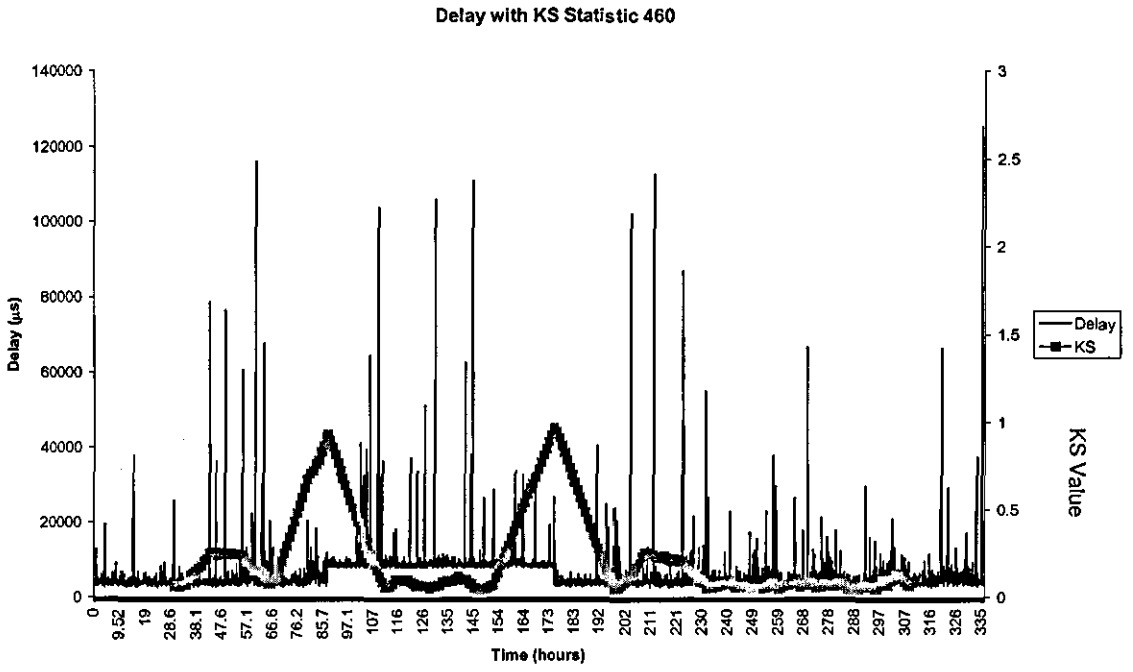


Figure 6.3-4 Delay/KS Graph route Helsinki to Madrid (small packets)

6.3.4. Berlin Serial Port down and up

The Serial Port on the Berlin router connecting to Helsinki was taken down and then restored again. This introduced Step Changes and changes in Time of Day Variation in Delay.

6.3.5. Clean (no events)

No events were introduced. Only weekend exceptions were evident.

6.3.6. Edinburgh bandwidth 2Mb to 4Mb

The bandwidth on the serial link connecting Edinburgh to London was increased from 2 Megabits per second to 4 Megabits per second. This gave rise to Step Changes and changes in Time of Day Variation in Delay.

6.3.7. Edinburgh bandwidth 4Mb to 2Mb

The bandwidth on the serial link connecting Edinburgh to London was reduced from 4 Megabits per second to 2 Megabits per second. This gave rise to Step Changes and changes in Time of Day Variation in Delay.

6.3.8. Edinburgh Queue length reduced

The Queue length on the Serial interface connecting Edinburgh to London was reduced to six packets and then brought back to it's default length of seventy-five packets. This caused some minor Time of Day Delay Variation Changes.

6.3.9. Edinburgh Ethernet Port down and up

The Ethernet Port connecting the Edinburgh router to network 11 was disabled and then re-enabled hours later. This caused spike exceptions as shown in figure 6.3-5 below.

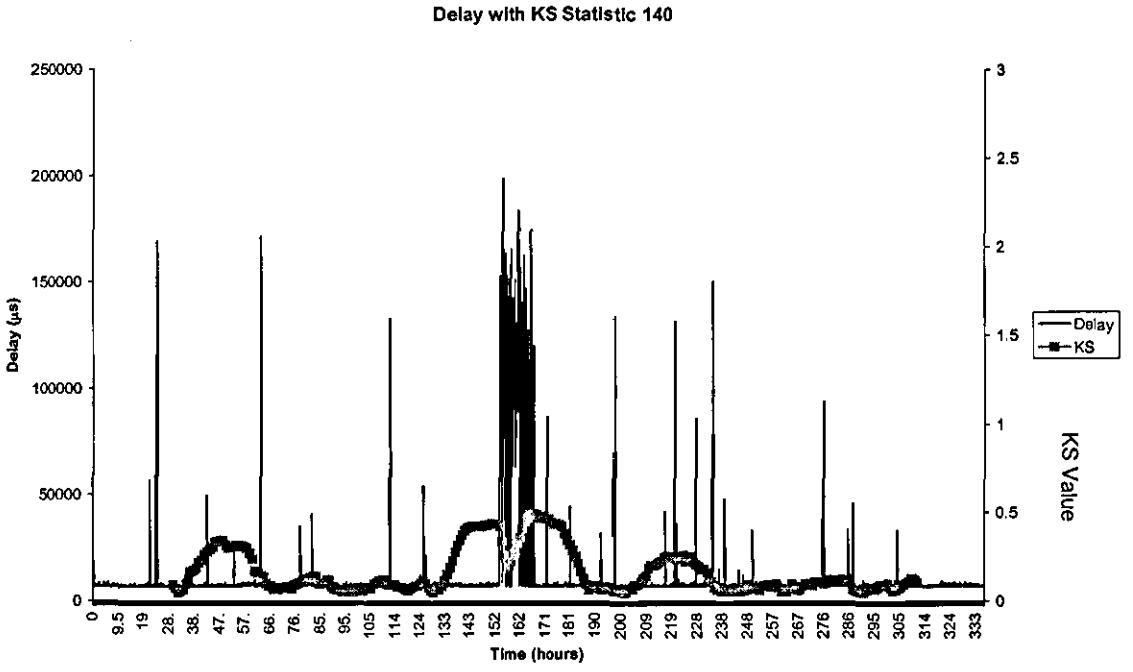


Figure 6.3-5 Delay/KS Graph route London to Helsinki (small packets)

6.3.10. Edinburgh Serial Port down

The Serial Port on the Edinburgh router connecting Edinburgh to London was taken down at time 61. This caused several Step Changes and changes in Time of Day Delay Variation.

6.3.11. Edinburgh Serial Port up

The Serial Port on the Edinburgh router connecting Edinburgh to London was brought up at time 191. This caused several Step Changes and changes in Time of Day Delay Variation.

6.3.12. Three Serial links taken down

The serial links from Edinburgh to London, from Berlin to Helsinki and from Madrid to Paris were all taken down for two short periods. This resulted in spikes on several routes. See Figure 6.3-6 below.

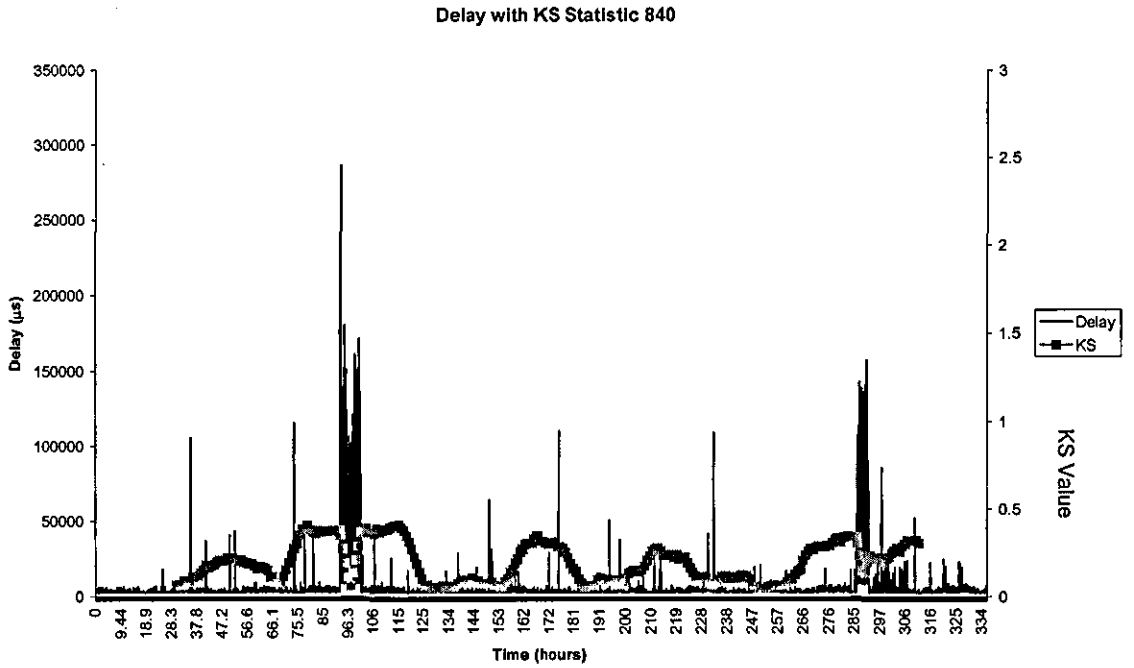


Figure 6.3-6 Delay/KS Graph route Berlin to Helsinki (small packets)

6.3.13. London Reload

The London router was restarted. This caused a small spike in delay.

6.3.14. Madrid Ethernet Port down

At time 126 the Ethernet port that connects Madrid to the 11 network is brought down. This causes step changes on links to and from the Madrid test station and significantly increased Time of Day Delay Variation. These are shown in Figure 6.3-7.

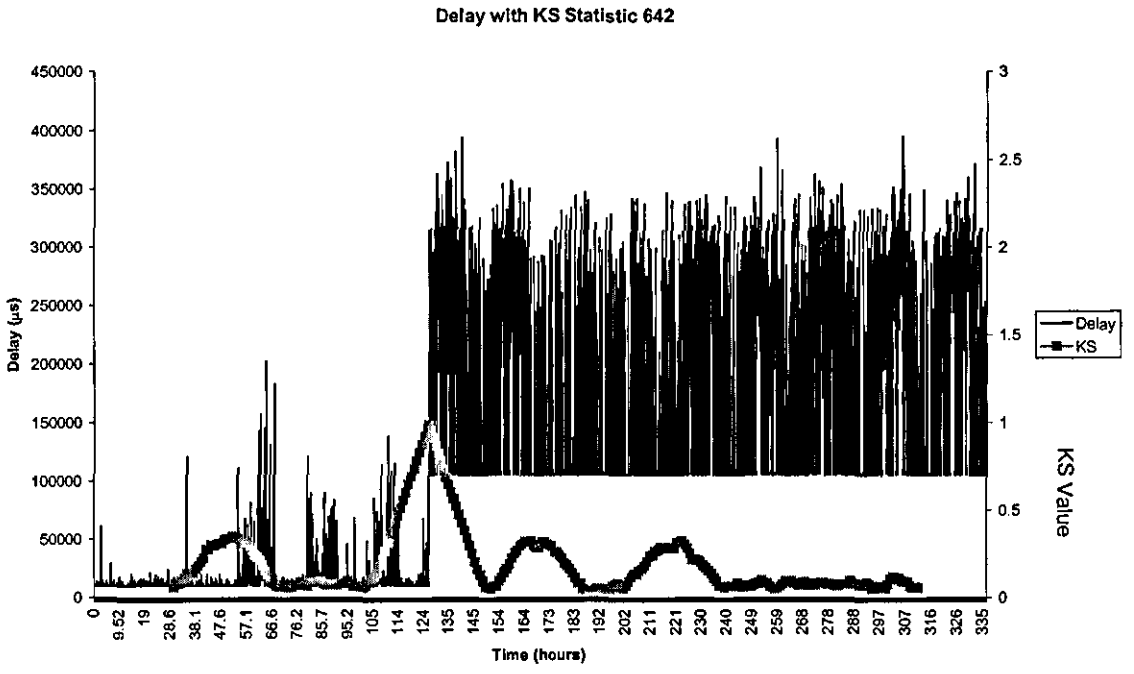


Figure 6.3-7 Delay/KS Graph route Madrid to Helsinki (large packets)

6.3.15. Madrid Ethernet Port up

The Ethernet port that connects Madrid to the 11 network is brought up. This causes step changes on links to and from the Madrid test station and significantly decreased Time of Day Delay Variation.

6.3.16. Network 11 down and routing changes

At time 93 the 11 Network was taken down for a short period. At time 252 the routing protocol was switched from IGRP to RIP. These events caused spikes and Step Changes respectively where Time of Day Delay Variation changes sometimes accompanied the step changes.

6.3.17. Network 11 down

The 11 Network was taken down for a short period. This caused some spike exceptions.

6.3.18. ParisS0up

The Serial Port connecting Paris to Madrid was brought up causing Time of Day Delay Variation Changes.

6.3.19. Routing Change

The routing protocol was changed from RIP to IGRP at time 307. This caused several Step Changes and Time of Day Delay Variation Changes.

6.3.20. Rome Serial Port up

The serial port connecting Rome to Athens was brought up causing a change in Time of Day Delay Variation.

6.3.21. Rome Serial Port up 2

The serial port connecting Rome to Athens was brought up causing a change in Time of Day Delay Variation.

6.4 Generating the training files

A c program used to implement the K-S test is first applied to the delay data and corresponding .ks files are generated for each delay file. These values are plotted on the graphs shown above. Another program, written in Java, is then used to aid the data labelling process. A neural network requires training data. The several thousand exceptions generated by the network events described above required labelling to provide a training set and also a test set of data for the neural network. The Java program, label, is shown below in figure 6.4-1. It searches for significant K-S values in the data, presents these to the operator who must then classify the Data Exception and move to the next one. The output files contain information about the route, the day, the type and the time for each exception.

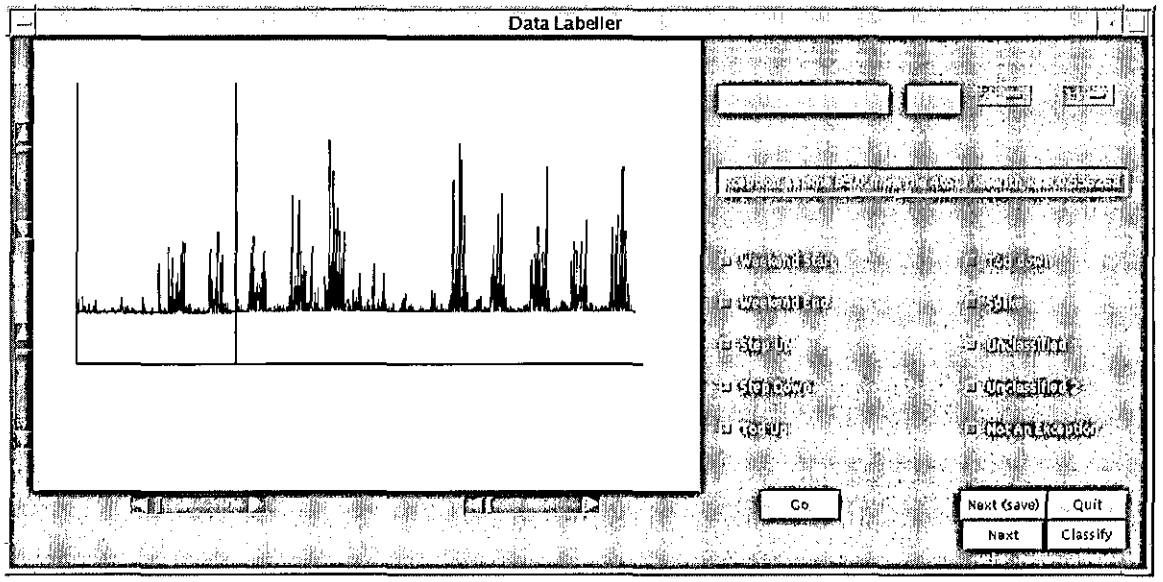


Figure 6.4-1 The label program

An example of the training files (.tr) which are the output of the label program is given below in figure 6.4-2. The first field is the time, the second field represents the date as a (1,7) binary matrix, the third and fourth fields represent the ingress and egress points respectively and the fifth field represents the type of exception as a (1,10) binary matrix. This final field includes more types of exceptions than we currently classify. This is to give scope for new types of exceptions.

```
48.0 (0 0 1 0 0 0 0) 5 8 (0 1 0 0 0 0 0 0 0 0)
171.0 (1 0 0 0 0 0 0) 5 8 (1 0 0 0 0 0 0 0 0 0)
217.0 (0 0 1 0 0 0 0) 5 8 (0 1 0 0 0 0 0 0 0 0)
299.0 (0 0 0 0 0 1 0) 5 8 (0 0 1 0 1 0 0 0 0 0)
```

Figure 6.4-2 Output from a training file

The training files are then collected together, mixed randomly and split into different sets; one set for training and one set for evaluation. The file that is actually presented to the neural network is of the format shown below in figure 6.4-3.

```
211.000000 (0 1 0 0 0 0 0) 5 4 (0 1 0 0 0 0 0) down4to0 0.489583
211.000000 (0 1 0 0 0 0 0) 10 7 (0 1 0 0 0 0 0) newlink4to1 0.376227
56.000000 (0 0 1 0 0 0 0) 8 9 (0 1 0 0 0 0 0) up2to0 0.266667
55.000000 (0 0 1 0 0 0 0) 9 7 (0 1 0 0 0 0 0) down4to0 0.536603
290.000000 (0 0 0 0 0 1 0) 7 8 (0 0 0 0 0 0 0) down2to10 0.258333
209.000000 (0 1 0 0 0 0 0) 8 4 (0 1 0 0 0 0 0) d_u_2to11 0.259195
100.000000 (0 0 0 0 1 0 0) 6 7 (0 0 0 0 0 1 1) faultycore 0.195833
75.000000 (0 0 0 1 0 0 0) 8 7 (0 0 0 0 0 0 0) up2to0 0.238227
100.000000 (0 0 0 0 1 0 0) 10 7 (0 0 0 0 0 1 1) faultycore 0.308029
128.000000 (0 0 0 0 0 1 0) 7 6 (0 0 0 0 0 0 0) up3to1 0.325000
```

Figure 6.4-3 Output from the exceptions file

The format here is as for the training files except that the exception type field has been reduced to a (1,7) binary matrix, removing unused classifications and also the directory name and the K-S value have been appended in fields 6 and 7. The directory name is necessary as the neural network uses the delay values as well as the information contained in the exceptions file. The program containing the neural algorithm retrieves the delay values based on the information from the exceptions file.

The above process is described in figure 6.4-4 below.

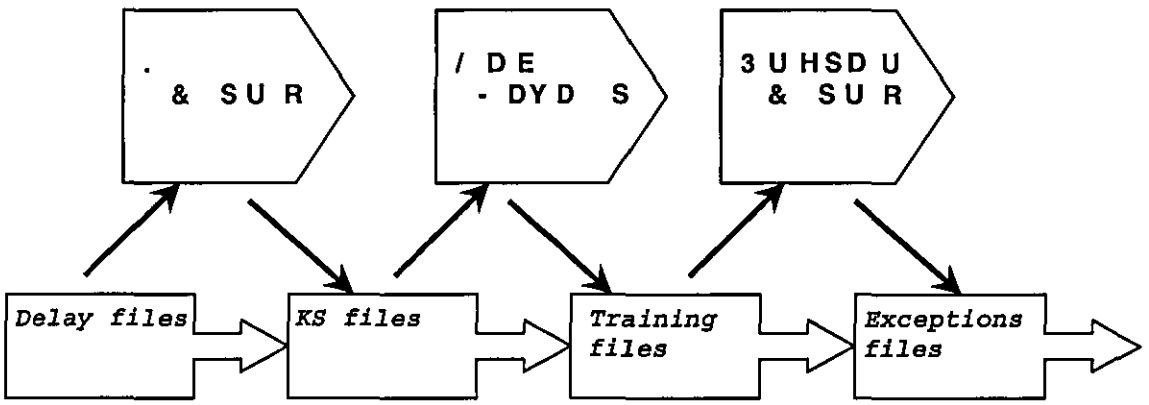


Figure 6.4-4 From Delay files to Exception files

6.5 Results – Simulation

The K-S Test has proved to be very effective in detecting the presence of a Data Exception. Using the simulation previously described, 210 days of delay measurements were generated per route, containing on average a Data Exception every 5 days per route. The distribution of labelled Data Exception types is given in figure 6.5-1.

Of the simulated events, the K-S Test correctly identified over 99.5% as Data Exceptions. Of the changes signalled by the K-S Test, around 70% were correctly identified as Data Exceptions with around 30% being false positive identifications. Although it may seem that the K-S Test is labelling a lot of changes incorrectly as Data Exceptions, these figures are entirely satisfactory. The K-S test is only the first phase of the detection process and the first priority is that Data Exceptions should not be missed at this point. As such, it is preferable at this first stage to over identify rather than to miss Data Exceptions. The second phase, the neural network, is then able to conduct further filtering to reduce the number of misclassified exceptions.

Exception Type	Training Set	Test Set
Weekend Begin	229	28
Weekend End	231	31
Step Change Up	40	23
Step Change Down	35	32
ToDVar Up	42	34
ToDVar Down	32	19
Spike	29	15

Figure 6.5-1 Exception Types

	Mean Square Error	Classification
Training Data Set	0.006	98.94%
Validation Data Set	0.108	80.09%

Figure 6.5-2 Classification Error

Using the previously described parameters for the neural network, the mean square error of the output vector after 2000 epochs was 0.006; this gives a classification accuracy of 98.94% (see figure 6.5-2). Using this trained network on the validation set of Data Exceptions, the mean squared error was 0.108 giving a classification rate of 80.09%. The chart below (Figure 6.5-3) breaks up the classification statistics. For each Data Exception type the percentage of correct identifications and the percentage of correct rejections are shown.

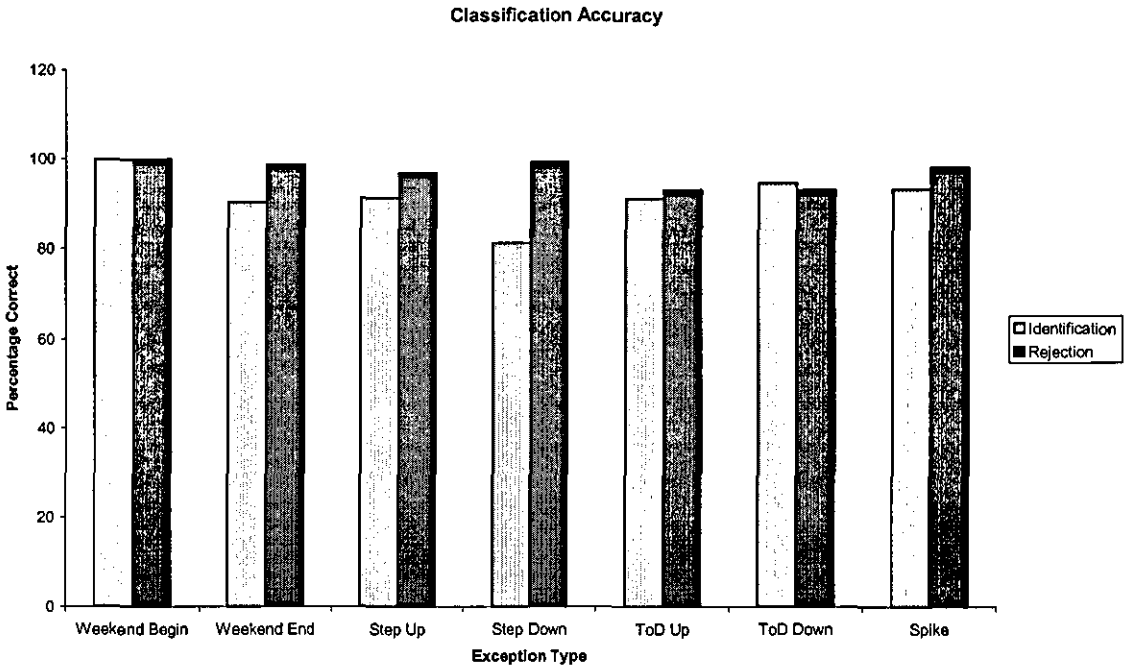


Figure 6.5-3 Data Exception Classification (Simulation)

As can be seen the results are encouraging. The neural network correctly classified each exception type over 90% of the time in every case with the exception of Step Down exceptions which were correctly classified with only a degree of accuracy of 81.25%. The neural network is not generating large numbers of false alarms with fewer than 7% of false positives for any given Data Exception type.

6.6 Results – Test Network

The K-S Test was again a very effective means of detecting that a change had occurred. An interesting difference to the results generated from the simulation is that the K-S Test was rarely significant (less than 1% of Data Exceptions) where no event had occurred. It's hard to know why the K-S Test should work better when applied to the test network in a real world scenario but it would appear that the delay distributions are more stable than those created by the simulation. It maybe that delay values generated by the simulation could have been subject to changes that were inherent characteristics of the simulation. An example of this is where the absolute time for the simulation crosses the 100 hour barrier. Previously this caused a loss of accuracy in the delay measurements as the simulation only worked to a certain number of significant figures and subsequent

values were rounded down. This problem was corrected by editing the simulation code and recompiling the simulation (an advantage of using an open source simulation). However, similar issues may exist that were not identified or corrected.

Exception Type	Training Set	Test Set
Weekend Begin	231	29
Weekend End	231	30
Step Change Up	33	12
Step Change Down	31	24
ToDVar Up	50	22
ToDVar Down	53	49
Spike	39	31

Figure 6.6-1 Exception Types

	Mean Square Error	Classification
Training Data Set	0.005	99.36%
Validation Data Set	0.184	72.58%

Figure 6.6-2 Classification Error

The parameters described previously were used to train the neural network with an additional input to allow for the two different packet sizes used for monitoring the test network. The number of each Data Exception types that were used is given in figure 6.6-1. The mean square error of the output vector after 2000 epochs was 0.005; this gives a classification accuracy of 99.36% (see figure 6.6-2). Using this trained network on the validation set of Data Exceptions, the mean squared error was 0.184 giving a classification rate of 72.58%. The chart below (Figure 6.6-3) breaks up the classification

statistics. For each Data Exception type the percentage of correct identifications and the percentage of correct rejections are shown.

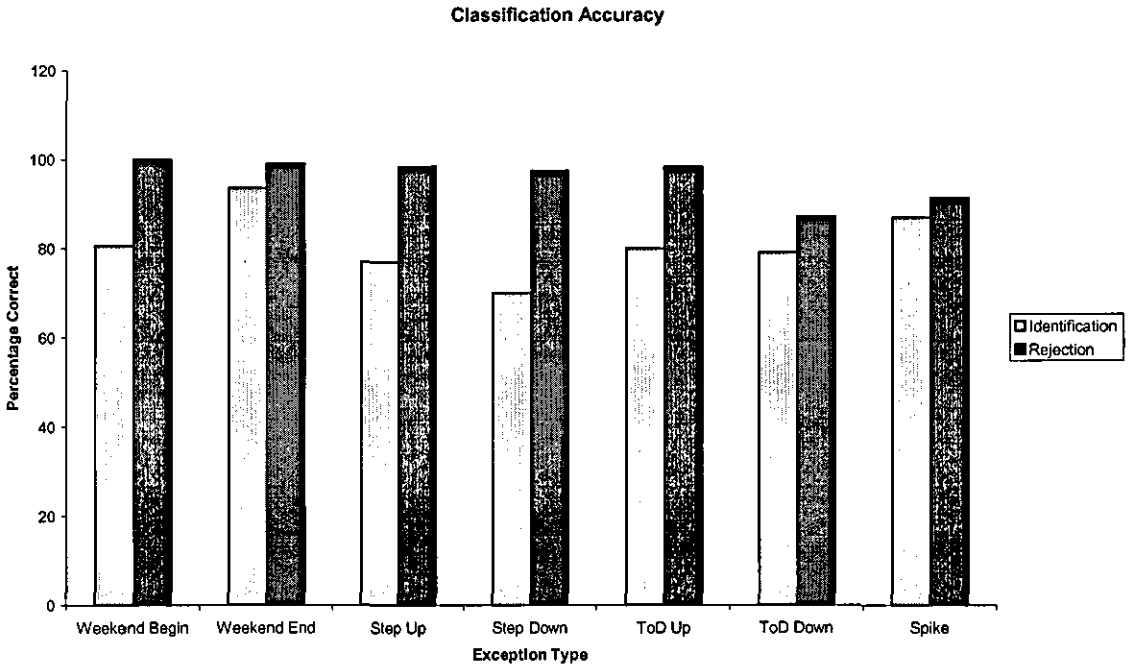


Figure 6.6-3 Data Exception Classification (Test Network)

As with the simulation results the neural network had most trouble identifying Data Exceptions of type Step Down but again the results are a positive indication of how a neural network could be used to classify Data Exception types.

6.7 Summary

In this chapter the K-S/neural approach has been tested and evaluated using two data sources, a simulation and a test network.

The underlying objective is to present a network operator with key information. The combination of the K-S test and a neural network is reliably identifying that an event of some kind has occurred and this is a significant step forward, potentially saving an analyst valuable time. The subsequent classification of the Data Exceptions into types will be useful in grouping related Data Exceptions together and perhaps even automatically diagnosing the type of event that has occurred. In the next chapter final conclusions are drawn and possible further work is discussed.

Chapter 7 – Conclusions and Discussion

7. Conclusions and Discussion

Given the speed in which communications networks, and particularly the Internet, have become an integral part of every day life, it is perhaps unsurprising that network management tools and techniques have been unable to keep pace with development. The Internet is being used in ways that far outstrip the perceived objectives at its inception. People use the Internet to communicate with one another, to find information, to buy and sell, to advertise, to share tools and ideas, with the number of services available increasing at a startling rate. Ten years ago public awareness of the Internet was limited to a small percentage of people with specialised interests, now it is a global phenomenon and ten years from now it may well be the leading means of communication, the most prominent provider of entertainment and the foremost facilitator of trade and commerce.

This explosion has had to be matched by technology that is able to support the many and varied services for which the Internet is now used. This has meant communications companies investing in infrastructure so that high bandwidth, high speed connections are available to businesses and home users alike. Much research has been geared towards providing protocols, coding algorithms and technologies that either increase the available bandwidth or decrease the need for it. The driving motivation has been to establish network technology that can meet the requirements, both present and predicted, of the Internet age.

While advances in network management have been made and research is being conducted into these areas, the relentless pursuit of high performance networks has so dominated that there are now significant gaps in management areas such as network security and network performance monitoring. While work is being done to redress this situation it will take a change in the priorities of communications companies before these gaps will be closed up. In the current climate such a change of priorities is unlikely. The emphasis will remain on expanding network services until such a time as most of the likely avenues for Internet use have been explored and exploited and this could be some time away.

While management issues are of secondary importance to network providers at present that is not to say that they are neglected altogether. As companies increasingly use the Internet for business the need for networks to be reliable and well managed grows. Network performance data is needed to better understand the behaviour of the monitored network as well as to detect faults and identify 'hot-spots' allowing network operators to manage networks in an informed manner.

This thesis has investigated means of detecting Data Exceptions in delay data. Data Exceptions are a useful concept for abstracting, summarising and presenting network performance information and for potentially identifying network events. Common Data Exceptions that relate to delay measurements are Step Changes, changes in the Time of Day Delay Variation and Spikes. They reflect some real change in the network. Several Data Exceptions may result from a single network event reflecting the several tests that may be conducted on that network.

This thesis has presented two methods of detecting Data Exceptions. The first approach utilised a rule base that compared summary statistics from the most recent measurements with those of previous measurements. Rules were then applied that tested for the presence of the various types of Data Exceptions. Although the rule base was integrated as part of the AIR system and had moderate success in detecting Data Exceptions there are weaknesses in this approach. Firstly, while the rules could cater for the most common Data Exception scenarios, unusual cases could case the rule base to fail to classify the Data Exceptions accurately. Secondly, the rule base required parameterisation for a specific network before being applied to data from that network.

A second approach made use of a combined method using the K-S Test and a trained neural network as a means of detecting and classifying delay Data Exceptions. The K-S test identifies that a change in network performance has taken place. The neural network is then used to classify the changes as specific types of Data Exception.

The K-S Test has proved to be a very effective means of detecting the presence of Data Exceptions in the data. Although the nature of the test necessitates a non real time approach in order to detect the presence of Time of Day Delay Variation changes, this is acceptable for the purpose. Where nearer to real time information is required and Time

of Day Delay Variation changes are less significant the approach can be modified to work in closer to real time. The K-S Test requires no parameterisation or training and can consequently be applied to arbitrary data sets and accurately detect changes in that data set. The use of the K-S Test for the purpose of detecting changes in network monitoring information is both novel and powerful. Data Exceptions were reliably detected with virtually no false alarms. This is an important criteria since network operators may lose confidence in a system that either misses events or repeatedly raises reports where no event has taken place.

The neural process has been shown to be an effective means of classifying Data Exception types. Although the neural network currently needs training it is hoped that in future a neural network could be trained to detect the generic types of Data Exceptions in any data source. This would then make the entire process completely generic, allowing for it to be applied to unfamiliar networks without any training.

For a neural solution to identify Data Exception types in arbitrary data sources the neural network may need training data from a variety of sources. At present only the data sources mentioned in this thesis, that is the simulation, the test network and the commercial network are available for use. Additional sources may come from further simulation or from other measurement projects. Artificial data could also be generated and this may prove to be particularly useful remembering that the aim here is to aid the neural network training process to learn generic Data Exception types.

Further possible development of the work includes extending the idea of Data Exception Collections. Data Exceptions can be collected together according to factors such as type, time and route to give a complete picture of a network event. It must be remembered that a Data Exception refers to data on a single path. A network event may impact the perceived performance as measured by several monitoring agents leading to several Data Exceptions. A means of reliably correlating Data Exceptions so that all the relevant information regarding a particular network event is reported together would be beneficial. Further, once such correlation is achieved, Data Collections may be used to give information regarding probable causes and locations of network events. For this to be attained, some additional research is necessary to link Data Exception Collections to

network events. Other information, such as the topology of the network, may also be necessary to establish details such as the location of the event.

Specific further plans include implementation of the method described in this thesis, or a modified version of it, at BT Network Operations Centre, Walsall. An alarm station that is currently deployed at Walsall could be updated to incorporate a rapid Data Exception feedback facility implemented using the K-S Test. In this scenario the K-S Test Statistic would be calculated more frequently and would compare data from a shorter time period such as an hour. In this deployment no provision would be made for detecting changes in Time of Day Delay Variation, instead the system would concentrate on giving information pertaining to step changes and spikes, reporting such events within an hour of the time the event occurred.

Another related area which may provide interesting research is methods of gaining meaningful one-way delay measurements. All the measurement schemes presented in this thesis achieve time synchronisation by using either the same clock to record the transmit and receive times or alternatively by using GPS synchronised clocks. At present these seem to be the only viable possibilities where a degree of accuracy is required within one hundred microseconds. However, as these delays are often averaged and summarised it may be that a lower degree of accuracy could be tolerated in order to achieve greater flexibility in terms of use and deployment. Relying on GPS can be cumbersome as the GPS antenna need to be positioned so that they are in view of GPS satellites (e.g. by a window). This is not always convenient. Alternative means of timing synchronisation include the use of NTP (Network Time Protocol). If timing synchronisation could be achieved to a sufficient degree using an NTP based monitoring station then these stations would be far easier to install and far more cost effective to deploy. If monitoring stations could be developed along these lines then an increase in the number of monitoring stations a network operator would be willing to utilise is likely as the cost to the network operator is significantly less. A means of analysing the data, such as the work presented in this thesis, would then become even more pertinent.

References

Alm99a Almes G et al, *"A One-way Packet Loss Metric for IPPM"*, RFC 2680, September 1999

Alm99b Almes G et al, *"A One-way Delay Metric for IPPM"*, RFC 2679, September 1999

Bas98 Bashir O, *"Management and processing of network performance information"*, Loughborough University PhD Thesis, 1998

Che00 Chen TM, *"Network Traffic Measurements and Experiments"*, Guest Editorial IEEE Communications Magazine, Vol. 38, No. 5, page 120, May 2000

Cla96 Claffy KC, *"'But some data is worse than others': Measurement of the Global Internet"*, National Laboratory for Applied Network Research (NLANR), August 1996 (available from http://www.caida.org/outreach/Papers/telegeog96.html)

Cla93 Claffy KC et al, *"Measurement Considerations for Assessing Unidirectional Latencies"*, in:Internetworking, Research and Experience, Vol. 4, No. 3, pp 121-132, January 1993

Cla97 Claffy KC, Monk T, *"What's Next for Internet Data Analysis? Status and Challenges Facing the Community"*, Proceedings of the IEEE, Vol. 85, No. 10, pp 1563-1571, October 1997

Cnet *"Cnet Simulator"*, http://www.cs.uwa.edu.au/pls/cnet/

Com99 Comer EC, *"Computer Networks and Internets"*, Prentice-Hall, 1999

Dij00 van Dijk P, *"0657420 Interim Report"*, The University of Waikato, 2000

Den93 Deng RH, Lazar AA, Wang W, *"A Probabilistic Approach to Fault Diagnosis in Linear Lightwave Networks"*, IEEE Journal on Selected Areas in Communications, Vol. 11, No. 9, pp 1438-1448, December 1993

Fau94 Fausett LV, *"Fundamentals of Neural Networks: architectures, algorithms and applications"*, Prentice-Hall, 1994

Flo94 Floyd S, *"The Synchronisation of Periodic Routing Messages"*, IEEE/ACM Transactions on Networking, Vol. 2, No.2, pp 122-136, April 1994

Gol95 Goldszmidt G, Yemini Y, *"Distributed Management by Delegation"*, Proceedings of the 15th International Conference on Distributed Computing Systems, pp 333-340, June 1995

Han00 Hansen T, *"Active Measurement Data Analysis Techniques"*, International Conference on Communications in Computing (CIC), page 105, June 2000

Hel92 Held G, *"Network Management"*, John Wiley & Sons Inc., 1992

Hoo97 Hood CS et al, *"Beyond Thresholds: An alternative Method for Extracting Information from Network Measurements"*, IEEE Global Telecommunications Conference, Vol. 1, pp 487-491, October 1997

IntTR *www.InternetTrafficReport.com*, Andover.net

Jac89 Jacobson V, *"traceroute"*, ftp://ftp.ee.lbl.gov/traceroute.tar.Z, 1989

Jai91 Jain R, *"The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modelling"*, John Wiley & Sons Inc., 1991

Job91 Jobson JD, *"Applied Multivariate Data Analysis"*, Springer-Verlag, 1991

Kal98 Kalidindi S, *"Participants guide to Surveyor daily summary reports"*, Advanced Network and Services, November 1998 (Available at http://telesto.advanced.org)

Lel94 Leland WE, *"On the Self-Similar Nature of Ethernet Traffic (Extended Version)"*, IEEE/ACM Transactions on networking, Vol. 2, No. 1, pp 1-14, February 1994

Mad94 Madruga EL, Tarouco LMR, *"Fault Management tools for a Cooperative and Decentralized Network Operations Environment"*, IEEE Journal on Selected Areas in Communications, Vol. 12, No. 6, pp 1121-1130, August 1994

Mas51 Massey FJ, *"The Kolmogorov-Smirnov test for goodness of fit"*, Journal of the American Statistical Association, Vol. 46, pp 68-78, 1951

Mat00 Matthews W Cottrell L, *"The PingER Project: Active Internet Performance Monitoring for the HENP community"*, IEEE Communications Magazine, Vol. 38, No. 5, pp 130-137, May 2000

Mat96 Mathis M et al, *"Diagnosing Internet Congestion with a Transport Layer Performance Tool"*, Proc. INET, June 1996

McG00 McGregor T et al, *"The NLANR Network Analysis Infrastructure"*, IEEE Communications Magazine, Vol. 38, No. 5, pp 122-128, May 2000

Nea88 Neave HR Worthington PL, *"Distribution-Free Tests"*, Unwin Hyman, 1988

NS *The NS Simulator*, http://www-mash.cs.berkeley.edu/ns/

Oftel *UK Government Office of Telecommunications web site*, http://www.oftel.gov.uk

Opnet *Opnet (Optimum Network Performance Modeller)*, http://www.opnet.com/

Pag99 Pagonis A, *"An efficient visualisation mechanism for communication network monitoring information"*, Loughborough University PhD Thesis, 1998

Pax98a Paxson V, *"Framework for IP Performance Metrics"*, RFC 2330, May 1998

Pax98b Paxson V, *"Creating a Scalable Architecture for Internet Measurement"*, Proc. INET, July 1998

Pax98c Paxson V, *"An Architecture for Large-Scale Internet Measurement"*, IEEE Communications Magazine, Vol. 36, No. 8, pp 48-54, August 1998

Pax96 Paxson V, *"Towards a Framework for Defining Internet Performance Metrics"*, Proc. INET, June 1996

Phi96 Phillips IW et al, *"On the Monitoring and Measurement of Quality of Service of SuperJanet"*, 13th UK Teletraffic Symposium IEE, pp 16/1-16/9, March 1996

Phi95 Phillips IW et al, *"SMDS Network Performance Measurement"*, IEE Colloquium on "Practical Experience with SMDS", PP7.1-7.4, October 1995

Phi99 Phillips IW et al, *"Generic Performance Management of Multiservice Networks"*, Integrated Network Management IEEE/IFIP, pp 943-944, May 1999

PhiSan00 Phillips IW, Sandford JM et al, *"Processing Network Delay Measurements into Network Events"*, NOMS 2000 IEEE/IFIP Networks Operation and Management Symposium, pp 955-956, April 2000

Sid89 Siddiqui MH, *"Performance Measurement Methodology for Integrated Services Networks"*, Loughborough University PhD Thesis, 1989

Sha68 Shapiro SS, Wilk MB, Chen HJ, *"A comparative study of various tests of normality"*, Journal of the American Statistical Association, Vol. 63, pp 1343-1372, 1968.

Ste70 Stephens MA, *"Use of Kolmogorov-Smirnov, Cramer-Von Mises and Related Statistics without Extensive Tables"*, Journal of the

Royal Statistical Society Series B, Vol. 32, No. 1, pp 115-122, 1970

Ste74 Stephens MA, *"EDF Statistics for Goodness of Fit and Some Comparisons"*, Journal of the American Statistical Association, Vol. 69, No. 347 Theory and Methods section, September 1974

Ste94 Stevens WR, *"TCP/IP Illustrated Volume1 - The Protocols"*, Addison-Wesley, 1994

Tan96 Tanenbaum AS, *"Computer Networks"*, Prentice-Hall, 1996

Tar98 Tarassenko L, *"A Guide to Neural Computing Applications"*, Arnold, 1998

Uij97 Uijterwaal H et al, *"Internet Delay Measurements using Test Traffic"*, RIPE-158.ps, June 1997 (available from http://www.ripe.net/ripe/docs/azdocument.html)

Uij98 Uijterwaal H et al, *"Internet Delay Measurements using Test Traffic: First results"*, 1998 (available from http://www.ripe.net/ripe/docs/azdocument.html)

Wal91 Waldbusseer S, *"Remote network monitoring management information base"*, RFC 1271, Nov 1991

Appendix A – tcl script for NS

```
set ns [new Simulator]
$ns use-scheduler Heap
$ns trace-all [open test.out w]
$ns rtproto DV

$ns color 1 Blue
$ns color 2 Red
$ns color 3 Yellow
$ns color 4 Green
$ns color 5 Brown

set rng [new RNG]

set numberOfDays 14
set SamplingRate 0.05

proc finish {} {
        global ns
        $ns flush-trace
        exec grep 333 test.out > monitor.out
        exit 0
}

proc remainder { num1 num2 } {
        while { $num1 >= $num2 } {
                set num1 [expr $num1 - $num2];
        }
        return $num1;
}

proc attach-expoo-traffic { node sink size burst idle rate } {
        set ns [Simulator instance]
        set source [new Agent/CBR/UDP]
        $ns attach-agent $node $source(
        set traffic [new Traffic/Expoo]
        $traffic set packet-size $size
        $traffic set burst-time $burst
        $traffic set idle-time $idle
        $traffic set rate $rate
        $source attach-traffic $traffic
        $ns connect $source $sink
        return $source
}

proc attach-telnet-traffic { node sink interval } {
        set ns [Simulator instance]
        set tcp [new Agent/TCP]
        $ns attach-agent $node $tcp
        set tcpsink [new Agent/TCPSink]
        $ns attach-agent $sink $tcpsink
        $ns connect $tcp $tcpsink
        set telnet [new Application/Telnet]
        $telnet set interval_ $interval
        $telnet attach-agent $tcp
        return $telnet
}

proc attach-ftp-traffic { node sink } {
        set ns [Simulator instance]
        set tcp [new Agent/TCP]
        $ns attach-agent $node $tcp
        set tcpsink [new Agent/TCPSink]
        $ns attach-agent $sink $tcpsink
        $ns connect $tcp $tcpsink
        set ftp [new Application/FTP]
        $ftp attach-agent $tcp
        $tcp set fid_ 5
```

The **ns** object outputs the network events as a trace file called **test.out.**

rng is a random number generator used later.

The variables **numberOfDays** and **SamplingRate** define the length of the simulation and the test packet transmission rate respectively.

finish is called when the simulation is completed. The file **test.out** is parsed for lines containing the numerical sequence 333. This is the size of the test packets and it significantly reduces the amount of processing required later on.

remainder returns the remainder from **num1** divided by **num2.**

attach-expoo-traffic, attach-telnet-traffic, attach-ftp-traffic and **attach-monitor-traffic** all create a traffic source and attach it to a node.

```
        return $ftp
}

proc attach-monitor { from to interval size } {
        set ns [Simulator instance]
        set monitor [new Agent/CBR]
        $ns attach-agent $from $monitor
        set sink [new Agent/Null]
        $ns attach-agent $to $sink
        $monitor set interval_ $interval
        $monitor set packetSize_ $size
        $monitor set fid_ 3
        $ns connect $monitor $sink
        return $monitor
}

for {set i 0} {$i < 36} {incr i} {
        set n($i) [$ns node]
}

# Set up the network core
# -----------------------

$ns duplex-link $n(0) $n(1) 5Mb 3.5ms SFQ
$ns duplex-link $n(2) $n(3) 5Mb 6.8ms SFQ
$ns duplex-link $n(0) $n(3) 5Mb 8.1ms SFQ
$ns duplex-link $n(1) $n(2) 5Mb 7.4ms SFQ

$ns duplex-link $n(1) $n(3) 2Mb 11.4ms SFQ
$ns duplex-link $n(0) $n(2) 2Mb 12.9ms SFQ

$ns rtmodel-at 0.1 down $n(0) $n(2)
$ns rtmodel-at 0.1 down $n(1) $n(3)

# Set up the perimeter
# --------------------

$ns duplex-link $n(0) $n(4) 6Mb 4.5ms SFQ
$ns duplex-link $n(0) $n(5) 6Mb 8ms SFQ
$ns duplex-link $n(4) $n(5) 4Mb 11.7ms SFQ
$ns duplex-link $n(5) $n(6) 4Mb 12.2ms SFQ
$ns duplex-link $n(1) $n(6) 6Mb 5.8ms SFQ
$ns duplex-link $n(1) $n(7) 6Mb 5.4ms SFQ
$ns duplex-link $n(6) $n(7) 4Mb 7.1ms SFQ
$ns duplex-link $n(3) $n(8) 6Mb 3.2ms SFQ
$ns duplex-link $n(3) $n(9) 6Mb 9.1ms SFQ
$ns duplex-link $n(8) $n(9) 4Mb 8.3ms SFQ
$ns duplex-link $n(9) $n(10) 4Mb 14.1ms SFQ
$ns duplex-link $n(2) $n(10) 6Mb 3.9ms SFQ
$ns duplex-link $n(2) $n(11) 6Mb 6.6ms SFQ
$ns duplex-link $n(10) $n(11) 4Mb 5.3ms SFQ

# Some routing priorities
# -----------------------

$ns cost $n(4) $n(5) 5
$ns cost $n(5) $n(4) 5
$ns cost $n(6) $n(5) 5
$ns cost $n(5) $n(6) 5
$ns cost $n(7) $n(6) 5
$ns cost $n(6) $n(7) 5
$ns cost $n(9) $n(8) 5
$ns cost $n(8) $n(9) 5
$ns cost $n(9) $n(10) 5
$ns cost $n(10) $n(9) 5
$ns cost $n(11) $n(10) 5
$ns cost $n(10) $n(11) 5

$ns cost $n(0) $n(4) 3
$ns cost $n(4) $n(0) 3
```

The nodes are created using a single **for** loop. Links are then set up between the nodes. The links include latency and throughput values. Links are created between node 0 and node 2 and between node 1 and node 3. These are then taken down immediately. This is so that they can be introduced at a later point.

Routing priorities are given here. The default value (cost) of a link is 1. Routes are calculated based on the total cost of the route.

```
$ns cost $n(0) $n(5) 3
$ns cost $n(5) $n(0) 3
$ns cost $n(1) $n(6) 3
$ns cost $n(6) $n(1) 3
$ns cost $n(1) $n(7) 3
$ns cost $n(7) $n(1) 3
$ns cost $n(3) $n(8) 3
$ns cost $n(8) $n(3) 3
$ns cost $n(3) $n(9) 3
$ns cost $n(9) $n(3) 3
$ns cost $n(2) $n(10) 3
$ns cost $n(10) $n(2) 3
$ns cost $n(2) $n(11) 3
$ns cost $n(11) $n(2) 3

# Set up periphery
# ----------------

# First some traffic related variables
set scale 10
set scale2 0.5
set scale3 0.5
set scale4 4

set tr(4) 0.05
set tr(5) 0.3
set tr(6) 0.2
set tr(7) 0.1
set tr(8) 0.1
set tr(9) 0.3
set tr(10) 0.08
set tr(11) 0.3

set fr(0) 1
set fr(1) 4
set fr(2) 3
set fr(3) 2
set fr(4) 2
set fr(5) 4
set fr(6) 1.5
set fr(7) 4

set rng [new RNG]
$rng seed 0

# What follows are a list of the nodes on the periphery
# traffic source(s attached to the nodes and the patterns
# of traffic output connected to them

# From node 4

$ns duplex-link $n(4) $n(12) 2Mb 10ms SFQ
$ns duplex-link $n(4) $n(13) 6Mb 10ms SFQ
$ns duplex-link $n(4) $n(14) 6Mb 10ms SFQ

        # Monitor traffic
        # sent every 0.2 seconds, probing different paths in the network

        set source(12,1) [attach-monitor $n(12) $n(15) $SamplingRate 333]
        set source(12,2) [attach-monitor $n(12) $n(18) $SamplingRate 333]
        set source(12,3) [attach-monitor $n(12) $n(21) $SamplingRate 333]
        set source(12,4) [attach-monitor $n(12) $n(24) $SamplingRate 333]
        set source(12,5) [attach-monitor $n(12) $n(27) $SamplingRate 333]
        set source(12,6) [attach-monitor $n(12) $n(30) $SamplingRate 333]
        set source(12,7) [attach-monitor $n(12) $n(33) $SamplingRate 333]

        $ns at 1.43 "$source(12,1) start"
        $ns at 1.44 "$source(12,2) start"
        $ns at 1.45 "$source(12,3) start"
        $ns at 1.46 "$source(12,4) start"
```

Traffic levels are scaled according to the day of the week (lower at weekends) and according to the route. **scale**, **tr**() and **fr**() are combined to calculate the traffic level for a particular link. **tr** defines the level of telnet traffic, **fr** defines the level of ftp traffic.

The traffic sources are then created and started. This is done at each node although the code for only one node is shown here (node 4).

```
        $ns at 1.47 "$source(12,5) start"
        $ns at 1.48 "$source(12,6) start"
        $ns at 1.49 "$source(12,7) start"

        # Telnet traffic, loading up the network trying to follow daily patterns

        set source(13,1) [attach-telnet-traffic $n(13) $n(16) 0.5]
        set source(13,2) [attach-telnet-traffic $n(13) $n(19) 0.5]
        set source(13,3) [attach-telnet-traffic $n(13) $n(22) 0.5]
        set source(13,4) [attach-telnet-traffic $n(13) $n(25) 0.5]
        set source(13,5) [attach-telnet-traffic $n(13) $n(28) 0.5]
        set source(13,6) [attach-telnet-traffic $n(13) $n(31) 0.5]
        set source(13,7) [attach-telnet-traffic $n(13) $n(34) 0.5]

        set source(14,1) [attach-ftp-traffic $n(14) $n(17)]
        set source(14,2) [attach-ftp-traffic $n(14) $n(20)]
        set source(14,3) [attach-ftp-traffic $n(14) $n(23)]
        set source(14,4) [attach-ftp-traffic $n(14) $n(26)]
        set source(14,5) [attach-ftp-traffic $n(14) $n(29)]
        set source(14,6) [attach-ftp-traffic $n(14) $n(32)]
        set source(14,7) [attach-ftp-traffic $n(14) $n(35)]

# set the telnet traffic sources off

for {set day 0} {$day < $numberOfDays} {incr day} {
        for {set i 0} {$i < 8} {incr i} {
                for {set j 1} {$j < 8} {incr j} {
                set i2 [expr $i * 3 + 13]
                        set i3 [expr $i + 4]
                        set j2 [expr $j + 3]
                        if {$i < $j} {
                                set j2 [expr $j + 4]
                        }
                        if {$day == 0} {
                                $ns at 0.0 "$source($i2,$j) start"
                        }
                        set thisDay [remainder $day 7];
                        if {$thisDay > 1} {
                                $ns at [expr $day*24 + 8.0] "$source($i2,$j) set interval_ [expr ($tr($i3) +
$tr($j2))/$scale]"

# These next lines are for changing the traffic rate
#                               if {$i == 1 && $j2 == 6 && $day > 6} {
#                                       $ns at [expr $day*24 + 8.1] "$source($i2,$j) set interval_ 0.001"
#                                       puts "telnet from $i to $j2 set to interval 0.001"
#                               }


                                $ns at [expr $day*24 + 17.0] "$source($i2,$j) set interval_ [expr ($tr($i3) +
$tr($j2))/$scale2]"
                                if {$i == 1 && $j2 == 6} {puts "telnet from $i to $j2 set to interval [expr ($tr($i3) +
$tr($j2))/$scale2]"}
                        }
                }
        }
}

# set the ftp traffic sources off randomly

for {set day 0} {$day < $numberOfDays} {incr day} {
        for {set k 0} {$k < 8} {incr k} {
                set k2 [expr 3*$k + 14]
                for {set i 1} {$i < 24} {incr i} {
                        for {set j 1} {$j < 8} {incr j} {
                                set rate2 [expr int(($fr($k) + $fr($j))/$scale3)]
                                set test [$rng integer $rate2]
                                set thisDay [remainder $day 7];
                                if { $i > 7 && $i < 18 && $thisDay > 1 } {
                                        set rate1 [expr int(($fr($k) + $fr($j))/$scale4)]
```

The telnet and ftp sources are set off at random times. These loops calculate the start and finish times for the telnet and ftp sessions. They make use of the random number generator (**rng**) and of the **scale**, **fr()** and **tr()** variables.

```
# This next line is for changing the traffic rate
#                                       if { $k == 1 && $j == 2 && $day > 6 } {set rate1 1}

                                        if {$rate1 == 0} {set rate1 1}
                                        set test [$rng integer $rate1]
                                }
                                if { $test == 0 } {
                                        set stop($k2,$j,$i) [$rng exponential]
                                        set start($k2,$j,$i) [expr $day*24 + [$rng uniform $i [expr $i + 1]]]
                                        set stop($k2,$j,$i) [expr $stop($k2,$j,$i) + $start($k2,$j,$i)]
                                        $ns at $start($k2,$j,$i) "$source($k2,$j) start"
                                        $ns at $stop($k2,$j,$i) "$source($k2,$j) stop"
                                }
                        }
                }
        }
}

#$ns rtmodel Exponential {55 0.2 0.1 100} $n(3) $n(2)
#$ns rtmodel-at 120 up $n(0) $n(2)
#$ns rtmodel-at 120 up $n(1) $n(3)
#$ns rtmodel-at 24 down $n(9) $n(0)
#$ns rtmodel-at 35 up $n(9) $n(0)
#$ns rtmodel-at 60 up $n(1) $n(6)
#$ns rtmodel-at 300 down $n(0) $n(5)
#$ns rtmodel-at 260 down $n(6) $n(7)
#$ns at 168 "$ns cost $n(4) $n(5) 1"
$ns at 100 "$ns cost $n(0) $n(1) 2"
$ns at 100 "$ns cost $n(1) $n(0) 2"
$ns at 200 "$ns cost $n(0) $n(1) 1"
$ns at 200 "$ns cost $n(1) $n(0) 1"
$ns at 200 "$ns cost $n(2) $n(3) 2"
$ns at 200 "$ns cost $n(3) $n(2) 2"
$ns at [expr $numberOfDays*24] "finish"
$ns run
```

Routing priorities are given here. The default value (cost) of a link is 1. Routes are calculated based on the total cost of the route.

Appendix B – PostProAll

PostProAll takes the file monitor.out generated by the simulation and creates delay files for the different routes that are monitored across the network.

The file **monitor.out** is interpreted a line at a time. The file is read using **fgets** and then parsed using **strtok**.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

void process_data(char fname[50]);

int main(void)
{
        char fname[50] = {"monitor.out"};

        process_data(fname);
        return(0);
}

void process_data(char fname[])
{
        FILE *infile;
        FILE *outfile;
        char outname[30];
        char temp[150];
        float starttime[30];
        int pid[30];

        char action[5];
        char time[20];
        char node_1[5];
        char node_2[5];
        char src[10];
        char size[7];
        char flow_id[10];
        char node_1_address[10];
        char node_2_address[10];
        char seq_no[10];
        int packet_id;

        int source;
        int dest;
        int sc,dt;

        int ref=0;

        float duration;

        for (source = 4; source <12; source++) {
          for (dest = 4; dest <12; dest++) {
                if (source != dest) {
                        infile = fopen(fname,"r");
                        sprintf(outname,"%dto%d.dly",source,dest);
                        outfile = fopen(outname,"w");
                  sc = source * 3;
                  dt = dest * 3;

                        printf("Doing file %s\n",outname);
                  while(feof(infile)==0)
                  {
                          fgets(temp,150,infile);
                          if(strlen(temp) >10) {
                                  strcpy(action,strtok(temp," \n"));
                                  if(strcmp(action,"r") == 0 || strcmp(action,"+") == 0 || strcmp(action,"-") == 0) {
                                          strcpy(time,strtok(NULL," \n"));
                                          strcpy(node_1,strtok(NULL," \n"));
                                          strcpy(node_2,strtok(NULL," \n"));
                                          strcpy(src,strtok(NULL," \n"));
                                          strcpy(size,strtok(NULL," \n"));
                                          strcpy(flow_id,strtok(NULL," \n"));
                                          strcpy(flow_id,strtok(NULL," \n"));
```

```
(int)atof(node_1_address) == sc) {

ref); */

                                        }
                                        }
                                }
                        }
                        fclose(infile);
                        fclose(outfile);
                }
        }
}

int getref(int packet_id, int pid[30])
{
        int x = 0;
        int finished = 1;
        int result = -1;

        while(x <30 && finished == 1)
        {
                if(packet_id == pid[x])
                {
                        result = x;
                        finished = 0;
                }
                x++;
        }
        return result;
}
```

```
strcpy(node_1_address,strtok(NULL," \n"));
strcpy(node_2_address,strtok(NULL," \n"));
strcpy(seq_no,strtok(NULL," \n"));
packet_id = atoi(strtok(NULL," \n"));

if(atoi(size) == 333 && (int)atof(node_2_address) == dt &&

        if(getref(packet_id,pid) == -1) {
                starttime[ref] = atof(time);
                pid[ref] = packet_id;
                if(ref == 29){ref = 0;} else {ref++;}
        }

if(strcmp(action,"r") == 0 && atoi(node_2) == dt) {
        duration = atof(time) - starttime[getref(packet_id,pid)];
        fprintf(outfile,"%f %f\n",starttime[getref(packet_id,pid)],duration);
        /*printf("%f %f %d\n",starttime[getref(packet_id,pid)],duration,

        pid[getref(packet_id,pid)] = 0;
}
```

Packets are logged in an array that contains start times. When they reach their final destination the delay is calculated and then written to a file. This file takes the format "<src>to<dest>.dly"

getref is a function that searches the array of packets that have been sent for a **packet_id**. The array is continually overwritten as packet information is not needed once the delay is calculated.

Appendix C – K-S implementation

The K-S implementation varied slightly for the two main data sources. The implementation given here was used to calculate the K-S statistic for the data generated by the Cisco Test Network.

```
# include <stdio.h>
# include <string.h>

typedef struct {
        float delay;
        int distribution;
        float cdf1;
        float cdf2;
} tableEntry;

typedef struct {
        float delay;
        float time;
} fileEntry;

int readData(char filename[40], fileEntry store[7000]);
int createTable(int time, fileEntry store[7000], tableEntry table[1000], int dist, int startingPoint);
void sortTable(tableEntry table[1000], int length);

int main(void) {
        fileEntry store[7000];
        tableEntry table[1000];
        int length1;
        int length2;
        int length;
        int storeLength;
        int time;
        char filename[40] = "4to10.dly";
        char outname[40] = "4to10.ks";
        int x;
        float diff;
        float K-S = 0;
        int cdf1 = 0;
        int cdf2 = 0;
        FILE *outfile;
        int egress;
        int ingress;
        int node[4];
        int size;

        node[0] = 1;
        node[1] = 4;
        node[2] = 6;
        node[3] = 8;

        for (ingress = 0; ingress <4; ingress++) {
                for (egress = 0; egress <4; egress++) {
                        if (ingress != egress) {
                                for (size = 0; size <3; size+=2) {
                                        sprintf(filename, "%d%d%dtest.dly", node[ingress], node[egress], size);
                                        sprintf(outname, "%d%d%dtest.ks", node[ingress], node[egress], size);
                                        printf("Doing %s .... ", outname);
                                        outfile = fopen(outname, "w");
                                        storeLength = readData(filename, store);
                                        for(time = 30; time <312; time += 1) {
                                                ks = 0;
                                                cdf1 = 0;
                                                cdf2 = 0;
```

```
                                                length1 = createTable(time, store, table, 1, 0);
                                                length2 = createTable(time+24, store, table, 2, length1);
                                                length = length1 + length2;
                                                sortTable(table, length);
                                                for(x = 0; x <length; x++) {
                                                        if(table[x].distribution == 1) {
                                                                cdf1++;
                                                        }
                                                        else {
                                                                cdf2++;
                                                        }
                                                        table[x].cdf1 = (float)cdf1/length1;
                                                        table[x].cdf2 = (float)cdf2/length2;
                                                        diff = table[x].cdf1 - table[x].cdf2;
                                                        if (diff <0) {
                                                                diff *= -1;
                                                        }
                                                        if (table[x].delay != table[x-1].delay && diff >ks) {
                                                                ks = diff;
                                                        }
                                                }
                                                fprintf(outfile, "%d %f\n", time, ks);
                                        }
                                        printf("done\n");
                                        fclose(outfile);
                                }
                        }
                }
        }

        return 0;
}

int readData(char filename[40], fileEntry store[7000]) {
        FILE *infile;
        int x = 0;

        infile = fopen(filename,"r");
        while(feof(infile)==0 && x <7000) {
                fscanf(infile, "%f%f", &store[x].time, &store[x].delay);
                        x++;
        }
        while(store[x-1].time <= 0) {
                x--;
        }
        fclose(infile);

        return x;
}

int createTable(int time, fileEntry store[7000], tableEntry table[1000], int dist, int startingPoint) {
        int x = 0;
        int y = startingPoint;

        while(store[x].time <time-24) {
                x++;
        }
        while(store[x].time <time) {
                table[y].delay = store[x].delay;
                table[y].distribution = dist;
                x++;
                y++;
        }

        return y- startingPoint;
}

void sortTable(tableEntry table[1000], int length) {
        tableEntry exchange;
        int x, y;

        for(x = 0; x <length; x++) {
```

```
                    for(y = x; y < length; y++) {
                        if(table[x].delay > table[y].delay) {
                            exchange = table[x];
                            table[x] = table[y];
                            table[y] = exchange;
                        }
                    }
                }
}
```

Appendix D – Neural Network Code Implementation

```
/* This is the implementation of neural net algorithm which takes
  .tr, .dly and .ks files from /ee/hsn4/eljms/tclScripts/testNetwork...
  /testRuns/14daytests and then classifies the exceptions highlighted by
  the K-S test. The .tr are training data (pre-classified data).

        This is version3 which changes the input vector so that the delays
        are put in the same place in the vector according to the hour
        of the day. This version also reduces the number of output
        classes to 7. We no longer consider either ramps or troughs
        in this version.
*/

# include <stdio.h>
# include <stdlib.h>
# include <math.h>
# include <string.h>

typedef struct {
        float x[122];
        int t[7];
        float rndNum;
} trainingPair;

float getDelay(float delay[100], int centile, int count);
float f(float x);
float f2(float x);
trainingPair copy(trainingPair tp);
void writeWeightVectorV(char filename[30], float v[122][150], int x, int y);
void writeWeightVectorW(char filename[30], float w[151][7], int x, int y);

int main(void) {

        trainingPair tp[2000];                  /* training Pairs*/
        int noc = 7;                                    /* number of classes */
        int noi = 121;                                  /* number of inputs */
        int nohu = 150;                                 /* number of hidden units */
        float y[noc];                                   /* Output vector */
        float y_in[noc];                        /* input to unit y */
        float z[nohu + 1];                      /* Hidden Layer */
        float z_in[nohu + 1];                   /* input to hidden layer unit */
        float v[noi+1][nohu];                   /* weights applied to x */
        float w[nohu+1][noc];                   /* weights applied to z */
        float yerror[noc];                      /* error in output y */
        float zerror[nohu+1];                   /* error in output from z */
        float zerror_in[nohu+1];
        float dv[noi+1][nohu];                  /* correction matrix for v */
        float dw[nohu+1][noc];                          /* correction matrix for w */

        float delay[100];

        trainingPair dummyTP;

        int ingress;
        int egress;

        FILE *tr;
        FILE *dly;
        FILE *outfile;

        char fname1[20];
        char fname2[100];
        char directory[30];

        float time, tm;
        int n, count, temp1, temp2, i, j, k;
```

```
        int timeThreshold;
        char ch;
        float r;
        float learningRate = 0.005;
        int epoch;
        float mserror;
        int exceptions;
        int index;

/* Initialise weight vectors */

        srand(2);

        /* weight vector v */
        for (j = 0; j <nohu; j++) {
                for (i = 0; i <= noi; i++) {
                        r = (float)rand()/RAND_MAX - 0.5;
                        v[i][j] = r;
                }
        }

        /* weight vector w */
        for (k = 0; k <noc; k++) {
                for (j = 0; j <= nohu; j++) {
                        r = (float)rand()/RAND_MAX - 0.5;
                        w[j][k] = r;
                }
        }

/* Read in training Pairs */

        exceptions = 0;
        strcpy(fname1, "Exceptions2.txt");
        if((tr = fopen(fname1, "r")) == 0) {
                printf("Problem opening file Exceptions2.txt\n");
                exit(1);
        }
        else {
                printf("Opened Exceptions2.txt\n");
        }

        while(!feof(tr)) {
                tp[exceptions].x[0] = 1;                /* set bias */
                /*** Express time as a number between -1 and 1 ***/
                fscanf(tr, "%f", &time);
                tp[exceptions].x[1] = (float)((div((int)time,24).rem - 12) / 12.0 + (time - (int)time));
                tp[exceptions].x[1] = (float)(sin(tp[exceptions].x[1]*M_PI));

                /*** Read the day of the week and convert to 1 or -1 ***/
                fscanf(tr, " (%f", &tp[exceptions].x[2]);
                tp[exceptions].x[2] = (tp[exceptions].x[2] == 0)? -1 : tp[exceptions].x[2];
                for(count = 3; count <8; count++) {
                        fscanf(tr, " %f", &tp[exceptions].x[count]);
                        tp[exceptions].x[count] = (tp[exceptions].x[count] == 0) ? -1 : tp[exceptions].x[count];
                }
                fscanf(tr, " %f)", &tp[exceptions].x[8]);
                tp[exceptions].x[8] = (tp[exceptions].x[8] == 0)? -1 : tp[exceptions].x[8];

                /*** Read the target values and convert to 1 or -1 ***/
                fscanf(tr, " %d %d (%d", &ingress, &egress, &tp[exceptions].t[0]);
                for(count = 0; count <(noc-2); count++) {
                        fscanf(tr, " %d", &tp[exceptions].t[count+1]);
                        tp[exceptions].t[count+1] = (tp[exceptions].t[count+1] == 0)? -1 : tp[exceptions].t[count+1];
                }
                fscanf(tr, " %d) ", &tp[exceptions].t[noc-1]);
                tp[exceptions].t[noc-1] = (tp[exceptions].t[noc-1] == 0)? -1 : tp[exceptions].t[6];
                tp[exceptions].t[0] = (tp[exceptions].t[0] == 0)? -1 : tp[exceptions].t[0];
                fscanf(tr, "%s ", &directory);
                fscanf(tr, "%f\n", &tp[exceptions].x[121]);     /* K-S statistic */

                /*** turn the ingress and egress points into inputs ***/
                for (count = 9; count <17; count++) {
```

Initialise weights (set to small random numbers).

Read in the training pairs (.tr files)

```
                    tp[exceptions].x[count] = (count - 4 == ingress) ? 1 : -1;
                }
                for (count = 17; count <25; count++) {
                    tp[exceptions].x[count] = (count - 4 == egress) ? 1 : -1;
                }

                /* Now fetch the delay values either side of the target time.
                  These need to be summarised into a limited number of
                  inputs. */

                sprintf(fname2, "/ee/hsn4/eljms/tclScripts/testNetwork/testRuns/14dayTests/%s/%dto%d.dly", directory,
ingress, egress);
                if ((dly = fopen(fname2, "r")) == 0) {
                    printf("Error opening file %s\n", fname2);
                    exit(1);
                }

                /* Centile values are calculated over a time period of one hour */
                timeThreshold = 1;
                count = 0;
                while(!feof(dly) && timeThreshold <49) {
                    fscanf(dly, "%f %f\n", &tm, &delay[count]);
                    if (tm >(time - 24) && tm <= (time +25)) {
                        count++;
                        if (tm >=timeThreshold + (time -24)) {
                            index = (int)div(tm, 24).rem;
                            index++;
                            index *= 2;
                            if (tm <time) {
                                index += 23;
                            }
                            else {
                                index += 71;
                            }
                            tp[exceptions].x[index] = getDelay(delay, 50, count - 1);
                            tp[exceptions].x[index + 1] = getDelay(delay, 95, count - 1);
                            timeThreshold++;
                            count = 0;
                        }
                    }
                }
                fclose(dly);
                exceptions++;
        }
        fclose(tr);

/* Having got the input vector we are now ready to proceed with the  training/use of the neural
  network */
        outfile = fopen("error.txt", "w");
        for (epoch = 1; epoch <2000; epoch++) {
            for (count = 0; count <exceptions; count++) {
                tp[count].rndNum = rand();
            }
            for (temp1 = 0; temp1 <exceptions; temp1++) {
                for (temp2 = temp1 + 1; temp2 <exceptions; temp2++) {
                    if (tp[temp1].rndNum >tp[temp2].rndNum) {
                        dummyTP = copy(tp[temp1]);
                        tp[temp1] = copy(tp[temp2]);
                        tp[temp2] = copy(dummyTP);
                    }
                }
            }
            printf("epoch = %d\n", epoch);
            mserror = 0;
            for (n = 0; n <exceptions; n++) {
                /* Compute hidden layer input */
                for (j = 1; j <= nohu; j++) {
                    z_in[j] = v[0][j-1];
                    for (i = 1; i <= noi; i++) {
                        z_in[j] += tp[n].x[i]*v[i][j-1];
                    }
                }
```

The exceptions (or training pairs) are sorted into a random order at the start of each epoch, which aids the learning process.

Feedforward: The values are passed forward to the hidden layer that then applies the activation function.

```
                        /* Compute hidden layer output */
                        z[0] = 1;
                        for (j = 1; j <= nohu; j++) {
                                z[j] = f(z_in[j]);
                        }

/* Having calculated the outputs of the hidden units we can now calculate the output vector */

                        for (k = 0; k <noc; k++) {
                                y_in[k] = w[0][k];
                                for (j = 1; j <= nohu; j++) {
                                        y_in[k] += z[j]*w[j][k];
                                }
                        }

                        /* compute the output */
                        for (k = 0; k <noc; k++) {
                                y[k] = f(y_in[k]);
                        }
```

Once the outputs have been calculated they can then be compared to the target results to give an error value.

```
/* The errors in the output are now calculated (training only) */

                        for (k = 0; k <noc; k++) {
                                yerror[k] = (tp[n].t[k] - y[k])*f2(y_in[k]);
                                mserror += (tp[n].t[k] - y[k]) * (tp[n].t[k] - y[k]);
                        }

                        for (k = 0; k <noc; k++) {
                                for (j = 1; j <= nohu; j++) {
                                        dw[j][k] = learningRate * yerror[k] * z[j];
                                }
                                dw[0][k] = learningRate * yerror[k];
                        }

                        for (j = 1; j <= nohu; j++) {
                                zerror_in[j] = 0;
                                for (k = 0; k <noc; k++) {
                                        zerror_in[j] += yerror[k] * w[j][k];
                                }
                                zerror[j] = zerror_in[j]*f2(z_in[j]);
                        }

                        for (j = 1; j <= nohu; j++) {
                                for (i = 1; i <= noi; i++) {
                                        dv[i][j-1] = learningRate * zerror[j] * tp[n].x[i];
                                }
                                dv[0][j-1] = learningRate * zerror[j];
                        }
```

Backpropagation: The weight corrections are calculated using the error information term. These are then applied below.

```
/* Now that the updates have been calculated they can be performed on the weight matrices */

                        for (j = 0; j <= nohu; j++) {
                                for (k = 0; k <noc; k++) {
                                        w[j][k] += dw[j][k];
                                }
                        }

                        for (i = 0; i <= noi; i++) {
                                for (j = 0; j <=nohu; j++) {
                                        v[i][j] += dv[i][j];
                                }
                        }
                }

                mserror = mserror / (exceptions * noc);
                fprintf(outfile,"%d %f\n", epoch, mserror);
        }
        writeWeightVectorW("W.txt", w, nohu+1, noc);
        writeWeightVectorV("V.txt", v, noi+1, nohu);
        fclose(outfile);
}
```

```
float getDelay(float delay[100], int centile, int count) {
        int ref, a, b;
        float temp;

        for (a = 0; a <count; a++) {
                for (b = a+1; b <count; b++) {
                        if (delay[a] >delay[b]) {
                                temp = delay[a];
                                delay[a] = delay[b];
                                delay[b] = temp;
                        }
                }
        }

        ref = (int)(count * centile / 100) - 1;

        if (count >0)
                return (delay[ref]/0.05) - 1;
        else
                return -1;
}

float f(float x) {

        return (float)2/(1 + exp(-x)) - 1;
}

float f2(float x) {

        return (float)((1 + f(x))*(1-f(x)))/2;
}

trainingPair copy(trainingPair tp) {

        trainingPair rtn;
        int count;

        for (count = 0; count <122; count++) {
                rtn.x[count] = tp.x[count];
        }
        for (count = 0; count <7; count++) {
                rtn.t[count] = tp.t[count];
        }
        rtn.rndNum = tp.rndNum;

        return rtn;
}

void writeWeightVectorV(char filename[30], float w[122][150], int x, int y) {
        int a, b;
        FILE *outfile;

        outfile = fopen(filename, "w");
        for (a = 0; a <x; a++) {
                for (b = 0; b <y; b++) {
                        fprintf(outfile, "%f ", w[a][b]);
                }
                fprintf(outfile, "\n");
        }

        fclose(outfile);
}

void writeWeightVectorW(char filename[30], float w[151][7], int x, int y) {
        int a, b;
        FILE *outfile;

        outfile = fopen(filename, "w");
        for (a = 0; a <x; a++) {
                for (b = 0; b <y; b++) {
                        fprintf(outfile, "%f ", w[a][b]);
                }
```

These functions, **f** and **f2** are the activation function and it's derivative. **getDelay** is used to get the delay information that will be used to train the neural network and **copy** is used to copy **trainingPairs** during the random sorting process.

The two functions **writeWeightVectorW** and **writeWeightVectorV** are used to write the two weight matrices to file. These are the trained matrices that can then be used for classification purposes.

```
                fprintf(outfile, "\n");
        }
        fclose(outfile);
}
```

Appendix E – Investigation into the Distribution of SMDS Delay Data

Work done at Loughborough prior to the installation of the Walsall test system topology involved using IP ICMP echo request and reply messages (known as 'Ping') to characterise the SMDS network delay profile. The results showed the effect of loading on delay. During the working week the delay distribution differed significantly from that observed at weekends, heavier tails signifying that more packets were experiencing longer delay. **[PHI95]**

An early implementation of the Walsall test architecture gave a more unexpected result. The observed delay distribution appeared to be dual peaked. In fact, these dual peaks corresponded to a planned change on the network and so the observed distribution contained data from the network in two different states. By plotting the delay distribution at different points in time, the shift from one network state to another was highly visible, which in itself, was a useful result. **[PHI96]**

With Data Exception detection in mind, we consider the delay distribution. To deploy certain statistical methods it is often necessary to assume that the data is normally distributed. In practice no real data set will be normal as the data will always be discrete and bounded but a good approximation to the normal distribution is sufficient to maintain a practical level of accuracy. If the sample size is relatively large a weaker normal goodness of fit can be tolerated although care should be taken if there is significant departure from the normal distribution. Consequently it was thought to be beneficial to investigate the delay data from SMDS with respect to normality and furthermore, to design and implement software for conducting such an investigation and any similar investigations that might be deemed beneficial, in the future.

Various procedures exist for examining the closeness of a sample to the normal distribution. The most commonly discussed of these (although not necessarily the most powerful) is the Kolmogorov-Smirnov (K-S) test. Essentially the K-S test compares the empirical distribution function (EDF) with a theoretical distribution function (in this case the normal).

In addition to analytical techniques, such as the K-S test, there also exist graphical techniques for assessing normality. The quantile-quantile plot (Q-Q plot) compares the observed quantiles (Q(p)) to the theoretical quantiles (Q*(p)). The Q-Q plot will be a straight line if the data are normally distributed.

The software developed by the author implements both the K-S test and the Q-Q plot. The procedures are described below but for a more comprehensive explanation, refer to Jobson [**JOB91**].

The K-S Test for Normality

To calculate the K-S statistic, first the standardized order statistics z_i are computed.

$$z_i = (x_i - \bar{x}) / s$$

Then the corresponding theoretical cumulative probabilities are determined (generally by tables) and denoted by F_i. The K-S test statistic, D, is given by

$$D = \max(|i/n - F_i|)$$

The Q-Q Plot

The Q-Q plot, plots the observed quantiles, Q(p), against the theoretical quantiles Q*(p) where

$$Q(p) = x_i$$

To calculate Q*(p), first calculate p where

$$p = (i - 0.5) / n$$

then Z(p) where Z(p) is the standardized order statistic from which Q*(p) can be derived using tables.

Normal Test Application

The software reads in a file of values and then assesses for normality using the described methods. There are no restrictions on the file although the software will take no account of numbers beyond the first 250 (The tests lose significance for large samples). The data is sorted and then read into fields in two records, one for the K-S test

and one for the Q-Q plot. The other fields in these records are calculated from this data (see above). Where the normal density function is required, tables are used (from file) with linear interpolation.

Fig E.1 The K-S statistic is outputted to the screen with a text message commenting on its significance.

For the SMDS delay data, a plug in procedure has been added to sub sample according to parameters such as the time of day and the centile fastest packets (such as the 95% fastest or 5% slowest). This can be removed for the general case or perhaps modified to cater for other cases that might require sub sampling in a similar way.

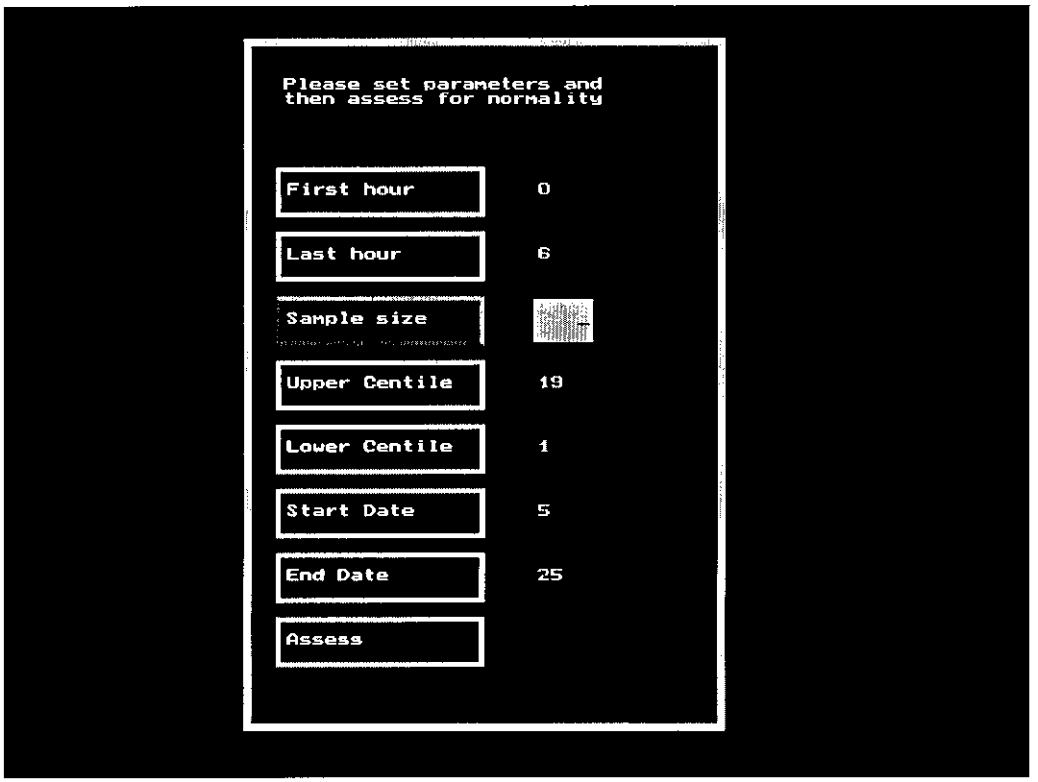


Fig E.2 These parameters can be modified to allow tests on sub samples from the delay data.

The software outputs the K-S (or D) statistic and gives a text interpretation of the significance of this statistic. The user has options to view the Q-Q plot or the delay distribution. The Q-Q plot is drawn as described above, with the least squares estimate of the linear relationship superimposed to give a better indication of how close the plot is to being linear and hence the data to being normal. The delay distribution, it is emphasized, is a rough guide to the shape of the density function and shouldn't be used independently to assess normality. It is intended to supplement the K-S test and the Q-Q plot by giving further clues as to where problems with the sample might lie.

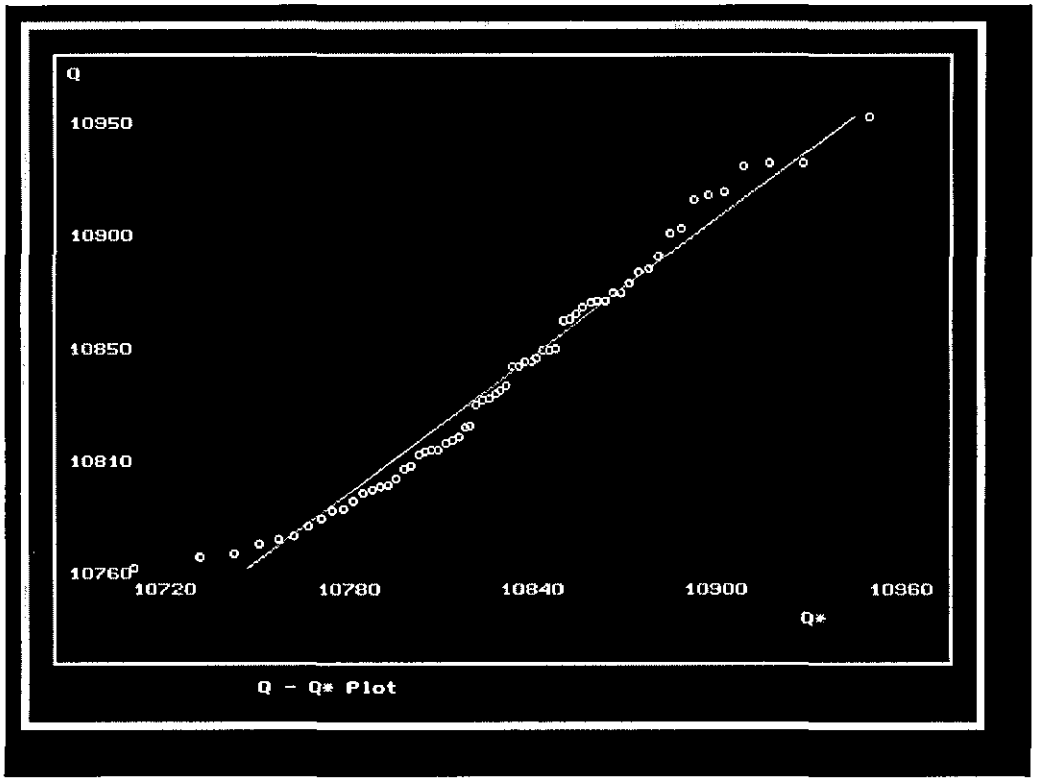


Fig E.3 The Q-Q plot. The data should lie in an approximately straight line for normal distribution

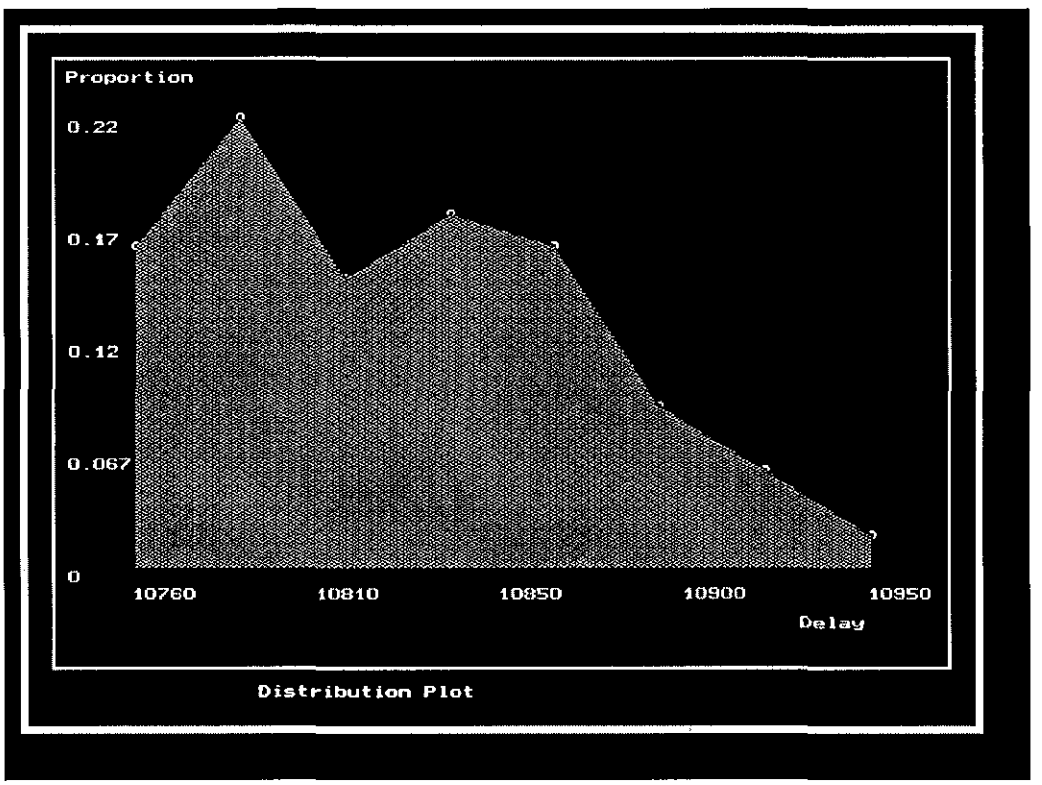


Fig E.4 The delay distribution. This is as a rough guide only, and shouldn't be used independently for analysis.

Results

The software was used to investigate the delay distribution of SMDS test packets. Data samples were taken from each test route, both for small (64 bytes) and large (1500 bytes) test packets transmitted over a three week period from 5 January 1998 to 25 January 1998. It should be noted that during this period there were instances of exceptional data that might effect the results. Results that were subject to exceptional data are shown in the tables in Italics.

As was expected, there is a poor correlation between the SMDS delay data and the normal distribution. Even allowing for the large amounts of data available for sampling, it would be unwise to compose tests requiring the normality assumption for the delay data. The distributions tended to be skewed to the right with heavy tails and occasional outliers. Improvements could be made by considering only the 95% fastest test packets and by restricting the sampling to times of the day with similar load (i.e. over night). Although this might be of some use it severely restricts the scope for tests requiring the normality assumption.

Test I

Small / ~~Large~~ Packets. Percentile 100% (all packets) .

Time (of day) 0 to 24 . (hours)

Dates from 5 to 25 . (of Jan 1998)

	Birmingham	Bristol	Edinburgh	London	Manchester
Birmingham		2.117	2.429	1.518	1.238
Bristol	1.702		*2.536*	1.905	1.555
Edinburgh	2.164	2.065		*2.61*	1.402
London	*2.971*	*2.582*	*3.604*		*2.791*
Manchester	1.63	1.886	1.5	0.6711	

Comments on Distribution.

There is highly significant evidence to suggest that the data are not normally distributed.

The distribution plot showed that the delay distribution is skewed to the right.

Test II

~~Small~~ / **Large** Packets. Percentile 100% .

Time (of day) 0 to 24 .

Dates from 5 to 25 .

	Birmingham	Bristol	Edinburgh	London	Manchester
Birmingham		1.242	1.654	1.527	1.058
Bristol	1.492		*2.464*	2.066	1.292
Edinburgh	2.042	1.774		*2.221*	0.919
London	*2.86*	*2.477*	*3.461*		*2.712*
Manchester	1.216	1.215	1.471	1.25	

Comments on Distribution.

There is highly significant evidence to suggest that the data are not normally distributed.

The distribution plot showed that the delay distribution is skewed to the right.

The results for large packets closely mirror those for small packets. This was found to be consistent throughout the testing.

Test III

~~Small~~ / **Large** Packets. Percentile 95% fastest .

Time (of day) 0 to 24 .

Dates from 5 to 25.

	Birmingham	Bristol	Edinburgh	London	Manchester
Birmingham		2.066	2.732	1.627	1.203
Bristol	2.795		*3.289*	1.971	1.679
Edinburgh	2.944	2.733		*2.701*	1.704
London	*3.617*	*3.216*	*3.713*		*2.686*
Manchester	2.532	2.175	3.112	2.523	

Comments on Distribution.

There is highly significant evidence to suggest that the data are not normally distributed.

The distributions were varied.

Test IV

Small / ~~Large~~ Packets. Percentile 5% slowest .

Time (of day) 0 to 6 .

Dates from 5 to 25.

	Birmingham	Bristol	Edinburgh	London	Manchester
Birmingham		1.649	2.195	2.333	1.298
Bristol	1.909	.	*2.264*	2.448	1.562
Edinburgh	2.576	2.456		*3.442*	1.75
London	*1.654*	*1.402*	*2.105*		*1.011*
Manchester	1.061	1.275	2.005	1.357	

Comments on Distribution.

There is highly significant evidence to suggest that the data are not normally distributed.

The distribution plot showed that the delay distribution has a very long and heavy tail.

7.1.1. Test V

Small / ~~Large~~ Packets. Percentile 95% fastest .

Time (of day) 0 to 6 .

Dates from 5 to 25 .

	Birmingham	Bristol	Edinburgh	London	Manchester
Birmingham		0.6516	0.5544	0.4424	1.289
Bristol	0.8289		*3.029*	0.9215	0.9657
Edinburgh	0.6375	1.211		*0.9131*	0.8099
London	*0.6238*	*1.421*	*1.467*		*1.05*
Manchester	0.9471	1.067	0.9993	1.596	

Comments on Distribution.

There is some evidence to suggest that the data are not normally distributed.

Distribution plots varied.