# Predicting Global Internet Instability Caused by Worms using Neural Networks

**Elbert Marais**

A thesis submitted to the Faculty of Engineering and the Build Environment, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science.

Johannesburg, April 2005

## Declaration

I declare that this thesis is my own, unaided work, except where otherwise acknowledged. It is being submitted for the degree of Master of Science at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination at any other university.

Signed this ___ day of _____ 20___

_____

Elbert Marais

## Abstract

Internet worms are capable of quickly propagating by exploiting vulnerabilities of hosts that have access to the Internet. Once a computer has been infected, the worms have access to sensitive information on the computer, and are able to corrupt or retransmit this information. This dissertation describes a method of predicting Internet instability due to the presence of a worm on the Internet, using data currently available from global Internet routers. The work is based on previous research which has indicated a link between the increase in the number of Border Gateway Protocol (BGP) routing messages and global Internet instability.

The type of system used to provide the prediction is known as an autoencoder. This is a specialised type of neural network, which is able to provide a degree of novelty for inputs. The autoencoder is trained to recognise "normal" data, and therefore provides a high novelty output for inputs dissimilar to the normal data. The BGP Update routing messages sent between routers were used as the only inputs to the autoencoder. These intra-router messages provide route availability information, and inform neighbouring routers of any route changes. The outputs from the network were shown to help provide an early warning mechanism for the presence of a worm.

An alternative method for detecting instability is a rule-based system, which generates alarms if the number of certain BGP routing messages exceeds a pre-specified threshold. This project compared the autoencoder to a simple rule-based system. The results showed that the autoencoder provided a better prediction and was less complex for a network administrator to configure.

Although the correlation between the number of BGP Updates and global Internet instability has been shown previously, this work presents the first known application of a neural network to predict the instability using this correlation. A system based on this strategy has the potential to reduce the damage done by a worm's propagation and payload, by providing an automated means of detection that is faster than that of a human.

## Acknowledgements

I would like to thank Professor T. Marwala for the ongoing support and encouragement throughout this project. He created an environment that is conducive to work, and ensured that students kept focussed during their studies.

I would also like to thank ARMSCOR for the financial support, as well as the CSIR for technical discussions throughout the project.

*To Lisa and my family for the support; especially to my mom, who put everything in perspective.*

# Contents

# Table of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| BGP | Border Gateway Protocol |
| SCG | Scaled Conjugate Gradient |
| TCP | Transmission Control Protocol |
| UDP | User Datagram Protocol |

# Glossary

| | |
|---|---|
| *Autonomous System* | A collection of Internet gateways or routers that have the same administrative policies and use the same internal routing protocol. |
| *Border Gateway Protocol* | The Internet protocol specifying the manner in which Autonomous Systems exchange routing information. |
| *Core Internet Router* | These are physical routers situated on the backbone of the Internet. Able to transmit large amounts of data. |
| *Internet Weather* | The measure of health for the Internet, determined by factors such as latency and percentage of packets lost between points in the Internet. |
| *Intrusion Detection System* | A system that attempts to recognize attempts to break into a computers system by monitoring network packets, system files and log files. These systems try to identify attacks based on intruder's attack patterns, rather than known virus or worm signatures which are used by anti-virus software. |
| *Network Prefix* | The network prefix was introduced with *Classless Inter-Domain Routing* (CIDR), which allows for specifying the number of bits that identify the gateway or group of gateways in an IP address. The prefix is the contiguous set of bits at the most significant end of the IP address that defines the collection of systems. For example, in the IP address 192.20.100.00/18, the number 18 refers to the first 18 bits being used for the network part of the address, and the remainder for the specific host address. Networking prefixes are supported by BGP, and replace the older class A, B and C networks. |
| *Peer* | A networking unit that works at the same networking protocol layer as another device. |
| *Routing Path* | The complete collection of hops taken by data between two networking hosts. |
| *Routing Table* | A table stored by a routing device listing all known network |

| | |
|---|---|
| | addresses, as well as information allowing it to transmit data to other network devices. |
| *Scaled Conjugate Gradient Optimisation* | A supervised learning optimisation technique which uses gradient information to find the optimum. Requires significantly less iterations than techniques such as back propagation but is computationally more complex. |

# 1 Introduction

The Internet has grown into a vast structure combining thousands of smaller networks [1]. It is estimated that traffic across the Internet doubled approximately every year for most of the Internet's existence, and will continue to do so for the foreseeable future [2]. According to [3, 4], there were more than 285 million Internet hosts in July 2004 and an estimated 605 million users in September 2002. Applications such as the web and email are widely used for both business and personal use. Companies such as *Amazon.com* rely on the Internet for business, with any network downtime causing a direct economic impact to the company.

The current control of the Internet is decentralised, since it is made up of many smaller independent networks that link up to exchange information using various protocols and standards. Various routing policies have been used to transmit information about routes and data between the various networks in the Internet. The original routing policies were designed to centrally manage and distribute traffic between networks. As the Internet grew, a need arose for companies and institutions to manage their own routing policies and procedures and to provide a more scalable solution that didn't rely on a single backbone to transmit data. For these reasons the *Border Gateway Protocol* (BGP) was introduced, which provides a scalable and robust means of transmitting routing data in a standard way across the Internet. BGP has the ability to inherently find new routes for traffic when other routes become unavailable due to events such as router outages. This, although providing a high level of failover, performs non-optimal path selection for routes [5].

The ease of access and widespread use of the Internet has made it open to attacks by worms which can be propagated through the Internet [6, 7]. An Internet worm is an automated program that uses a network to propagate copies of itself on other host computers, by exploiting vulnerabilities of the software running on those hosts. Worms are capable of spreading rapidly, such as in the case of the SQL Slammer worm which infected approximately 75,000 hosts in under ten minutes [8]. They are able to cause damage with their payload, since once a host is infected the program has control of the host as well as its information. Their rapid spread is also often uncontrolled, which places a high load on the Internet. These factors contribute towards the economic impact caused by worms, which in the case of the Code-Red

worms of 2001 caused approximately $2.6 billion of damage [9]. This damage was not caused by a malevolent payload of the worm; instead it was largely from the downtime of the Internet as a whole for the period that the worm was most active.

The high load caused by certain worms has a documented effect in increasing the number of BGP Update messages sent between routers [10, 11]. The increase in traffic causes BGP sessions between routers to expire since messages are unable to be received in time. Routers then select alternative network paths and inform adjacent routers about the new routes, as well as about the loss of the previous route.

Various measures have been put in place to deal with worms such as easily updatable anti-virus software, automated patching of operating systems and intrusion detection software. These preventative countermeasures significantly decrease the chance of a computer becoming infected, but it has been seen that worms continue to propagate regardless. For this reason, this project has approached the problem of worms from a reactive standpoint, with the goal of automatically detecting the presence of a worm faster than humans are currently able to. The method used in this project is an autoencoder [34] with the inputs being the number of BGP Update messages received on a router. This method is preferred to that of monitoring stability on hosts at the Internet edge, which provide a localised view on instability.

The work performed in this study is the first known application of a neural network used to predict Internet instability by monitoring BGP routing.

## 1.1 Aim

The overall aim of this project is to predict the presence of worms on the Internet, specifically by monitoring data available from Internet routers. It is hoped that the higher load caused by worms propagating will be apparent sooner at the routers rather than at the Internet edge. As an extension of this, the project will test whether a single router's data is sufficient to detect Internet-wide instability. The study will also test the effectiveness of neural networks for providing the prediction.

## 1.2 Objectives

The objectives of the project are:

- To use the novelty detection of a trained autoencoder to provide an indication of the possibility of Internet instability, specifically that of worms.

- To determine if novelty detection using an autoencoder is able to predict Internet-wide instability sooner than a rule-based system. In using either an autoencoder or a rule-based system it is necessary to set a threshold on the level above which it is assumed that there is Internet instability.
- To determine if using the BGP Update messages alone as input to the autoencoder is sufficient to predict instability.
- To determine if a single router can effectively represent Internet-wide instability.

The practical application for this research is to install an automated advanced warning system to allow for earlier detection and reaction to Internet instability and worms. Companies and individuals that are capable of analysing worms and viruses could be aware of the presence of a worm earlier, and can create a countermeasure sooner. A forewarning in the case of a worm can save a substantial amount of time, money and data. A network administrator that is aware of the presence of a worm is able to take preventative action such as blocking ports or emails that are used by the worm, also reducing the damage caused by the worm.

# 2 Literature Survey

## *2.1 Internet and Routers*

Traffic sent on the Internet backbone during 1994 was approximately 15 terabytes per month, and in 2000 reached an estimated 20,000 to 30,000 terabytes per month [2]. In order to transmit this information across the Internet it is necessary to send data over several hops between routers. These routers are physical devices which rely on routing policies and procedures to know where to forward traffic to, and vary considerably in their physical location and the rules imposed by the network administrators. Various routing policies are used within local networks, but the current standard for intra-network routing is the Border Gateway Protocol (BGP), which utilises the TCP protocol for transmitting information between routers.

This section forms the basis for subsequent explanations of the effect that worms have on the Internet, by providing an overview of the Internet and routing principles. This if followed by a more detailed description of the BGP protocol, detailing the specific BGP messages and characteristics that are relevant to this project.

Please refer to the glossary for a description of the networking terms used.

### 2.1.1 Internet

The Internet has grown from a small academic network into a critical extension of the public telecommunications infrastructure [12]. In the short period of time since its commercial inception in 1995, the Internet has undergone massive expansion, increasing its internal complexity and causing a large demand on the underlying equipment used to supply connectivity.

The Internet originated from the funding of the ARPANET in 1969, which was designed to test a packet-switching network by the Advanced Research Projects Agency (ARPA) [1]. The goal was to find ways of robustly and reliably transmitting data, and led to many of the data communications techniques still in use today. Although this was an experimental project, it was already being used for day-to-day communications, and so was changed to an operational network in 1975. In 1983, the ARPANET was divided into two separate networks – the MILNET and a new smaller ARPANET, which then collectively became known as the Internet.

In 1985 the National Science Foundation (NSF) created a new network called the NSFNet, which consisted of five smaller networks and connected these to the existing Internet. The NSF had a vision to allow all scientists and engineers across the world access to the Internet, and so created a new backbone in 1987. The Internet then evolved into a three tier-architecture, consisting of the backbone, regional and local networks. In 1995 the NSFNet stopped providing the central backbone, and the ARPANET officially ceased to exist. The original ARPANET evolved from a simple backbone, to a three-tier structure, and then into the huge interconnection of distributed hubs that currently form the Internet. It has doubled in size since 1983, and has continuously grown beyond its originally intended role and size.

### 2.1.2 Routing

The ARPANET used a simple routing algorithm to determine the path of data sent, which was based on determining the queuing delay at every link in the network, and then propagating these measurements to all routers. These routers then forwarded data packets along the routes that had the least delay [5]. This worked effectively under normal circumstances, but led to poor performance under heavy load. For this reason, a new method was introduced under high load, using the path-distance based on the number of hops for a packet to reach its destination [13]. This meant that the route which had the least number of routing devices on the path was used to send the traffic. This had poorer performance when measured explicitly, but was found to be more stable under high load.

One of the first routing architectures used by the Internet is shown in Figure 1. *Autonomous Systems* (AS) are connected to external routers, which then forwarded all traffic between the AS's using the Internets core routers [1]. *Autonomous Systems* are made up of a network or group of networks that have a common administration and routing policies [14], of which there are currently more than 10,000 [15]. By combining many hosts within a network to represent a single AS, routers are able to only store the addresses of the AS's, rather than storing the entire collection of IP addresses for the individual hosts that they contain.

It was necessary to send all traffic leaving the AS's via the core routers. The core routers shown in the figure were responsible for determining the best routes between AS's, which relied on the *reachability information* that is communicated between routers. These core routers exchanged this routing information using what was known

as the *Gateway to Gateway Protocol* (GGP). The *Exterior Gateway Protocol* (EGP) was used separately to pass information between AS's and the core. This implementation performed effectively for a small network, but was not able to scale as the Internet grew in size. As the Internet grew and multiple networks were added, it became important for these agencies to control their routers interaction with other routers [5], without the centralised control.

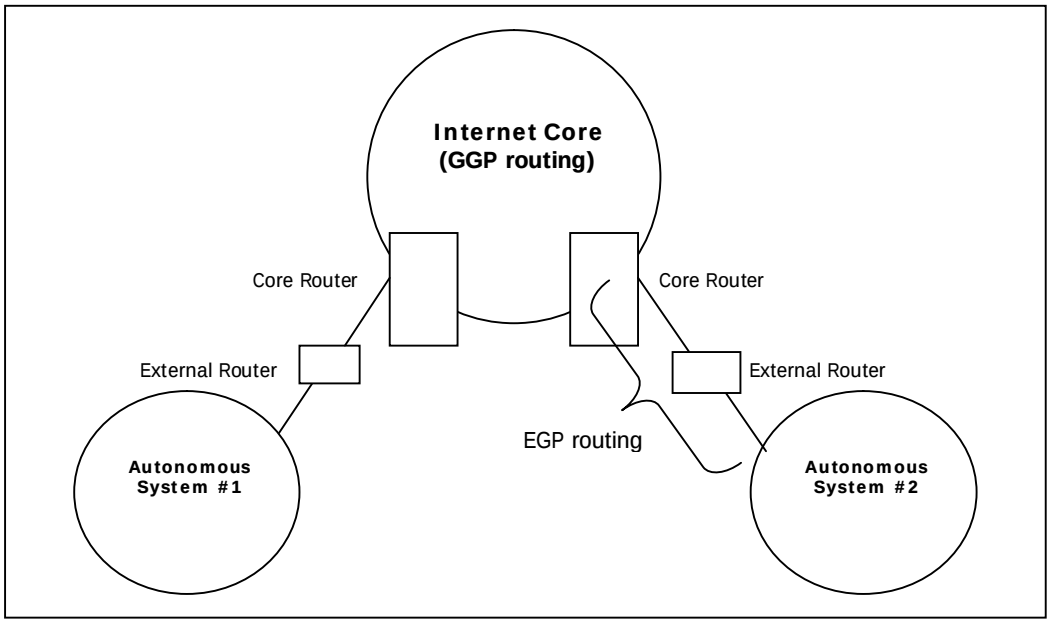


**Figure 1. Basic routing architecture of the original Internet. This structure required that all traffic leaving the AS was sent via the Internet Core's routers. An Exterior Gateway Protocol (EGP) was used to transmit information to the core, while core routers exchanged routing information using a Gateway to Gateway Protocol (GGP).**

The current state of the Internet as shown in Figure 2 contains a series of routing domains which consist of several AS's. These domains are responsible for determining the best routes between the various AS's, and use the Border Gateway Protocol (BGP) to share the routing information [5]. This system is decentralized, not requiring traffic to be sent through the core, and so scales better than the previous structure. Each AS is able to manage its own internal routing policies, with the smaller AS's opting for simple routing policies (such as raw hop counting), but larger AS's use more advanced techniques to avoid using routes with high delays [5].

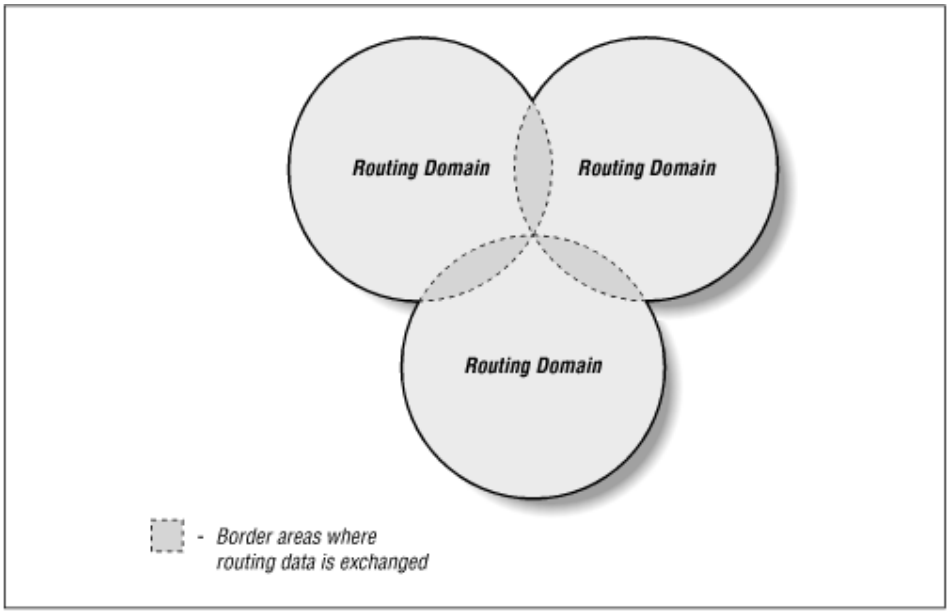


**Figure 2. Current state of Internet routing with a decentralised architecture. Each domain uses its own protocol to exchange routing information internally, but inter-domain routing uses the Border Gateway Protocol (from [1]).**

### 2.1.3 Border Gateway Protocol

Although AS's are capable of using their own internal routing policies, once information leaves the AS it is necessary for routers to communicate using the same protocol. The Border Gateway Protocol, now in version 4, is the current standard used by Internet routers to exchange routing information [16]. At a broad level, it is able to transmit routing information between many AS's, and is able to provide a high level of control and flexibility for inter-domain routing while still successfully managing policy and performance criteria. The protocol supports the *Classless Interdomain Routing* (CIDR) protocol, which provides for using *IP prefixes* and removes the older network classes that were previously used to distinguish networks (refer to the Glossary for *IP prefixes*).

The configuration of BGP allows for autonomous routing policies to be applied by a network administrator, based on requirements such as ensuring good end-to-end performance, catering for contractual obligations with clients, load balancing and cost reduction strategies [5, 15]

For these reasons, BGP allows the network administrator to indicate preferences that only partially include performance. From [15], it is noted that the BGP policies of the Internet represent the commercial relationships between AS's. AS pairs generally either form a customer-provider relationship, or a peer-to-peer relationship.

In the customer-provider relationship, the provider sells the connectivity to the customer (such as an ISP that charges home users for their connection), whereas a peer-to-peer relationship is between peers who provide connectivity to their customers (such as the connectivity provided between ISP's).

Use of the BGP policies allows a BGP device to select the list of AS's to use for sending information. The configurations allow for explicitly refusing to transmit data for certain AS's, but still allowing internal advertisement of these routes. Some performance parameters that can be controlled by the BGP configuration are:

- Shorter AS paths (i.e. number of AS hops) can be preferred to longer ones.
- Stable paths should be preferred to unstable paths.
- Preference of internal routes along an AS path to external routes.
- Quality of AS paths used to reach a destination. If multiple AS paths can be used to reach a destination, BGP will select a path based on parameters such as the link speed and tendency for congestion along the path. If a BGP router is aware that certain AS's within an AS path perform poorly, it can rather use other available AS paths.

AS's use BGP to advertise the router's current set of routes to neighbouring AS's – they never advertise the full set of paths available, but instead only inform other AS's of the subset that they are currently using to transmit data. BGP includes full AS path information in every route advertisement, thereby eliminating the possibility of infinite loops from host to host, and BGP does therefore not impose any topological restrictions on the ways that AS's are connected. For this reason the number and configuration of AS's is arbitrary to the BGP protocol, allowing a large network such as the Internet to scale well.

BGP devices exchange routing information (such as AS paths used) when the connection is established initially, and afterwards only using incremental updates. Information is only refreshed when updates are received, and so do not rely on any scheduled updates from other routers. The AS paths for the best routes are chosen based on a single metric for each available AS route, and this metric is calculated using the policies and properties discussed earlier. The path chosen is determined by the path with the lowest metric.

The BGP protocol runs on top of the TCP protocol [17], which already provides fragmentation, retransmission, acknowledgement and sequencing. TCP is already

supported by BGP routers since this is the most used transmission protocol for delivering data on the Internet. The default BGP port is port 179 [10].

In [5], the effectiveness of routing paths over the Internet is examined, which is primarily due to BGP routing, stating that the overall effect of routing protocols and policies on end-to-end performance are poorly understood. Five different datasets were to compare the path of data sent across the Internet. Metrics such as round-trip times and loss rates were used to determine that in 30 to 80 percent of the time a better path that could have been selected. This is partially due to factors such as "early-exit" routing, which sends traffic which is bound for an external network to the closest exchange point, thereby lowering the local network traffic but ignoring any shorter routes. Even with these concerns, BGP still provides a high level of flexibility and customisation, together with scalability and robustness.

#### 2.1.3.1 Types of BGP Messages

There are four types of messages supported by the BGP-4 protocol [17, 11]:

- The *Open* message is the first command sent between BGP devices and is used to establish a BGP session. Once this has been established, the devices are able to send the other types of messages.
- The *Update* message provides the latest information for the advertised routes used by the BGP device. It is used to simultaneously advertise new routes (known as *announcements*), and to inform of routes that have become unavailable (known as *withdrawals*). Advertised routes are those routes that the device is actively using for sending data, and does not include paths that it is aware of but not using.
- *Notification* messages are sent when an error forces the closure of the BGP session.
- The *Keep-Alive* message is transmitted between BGP devices to provide notification that the devices are still active. This is sent at timed intervals to ensure that the BGP session doesn't expire.

An additional *Route Refresh* message was added subsequent to the BGP-4 RFC 1711 [17] specification. This has been widely implemented and allows for requesting a full retransmission of a peer's routing information, and was added as a later BGP capability in RFC 2918 [18].

A BGP session is established after a TCP session has been initiated. This involves one peer sending the *Open* message, followed by a *Keep-Alive* message returned by the peer. Once the session has been established, each peer sends their entire set of advertised routing information. After this initial update, only incremental updates are exchanged.

These messages are relevant only for routing status and update information, and are independent of the data being transmitted between the routers. The message that is most relevant for this research is the *Update* message. BGP routers send Update messages to exchange network accessibility information, such as when a new router is present that causes the router to change its routing paths. These changes are what are propagated to the surrounding networks.

Two specific features of BGP are of specific interest to this project – the *Minimum Route Advertisement Interval* (MRAI) property and *BGP route flap damping*.

#### 2.1.3.2 Minimum Route Advertisement Interval

The original BGP protocol includes a property called the *Minimum Route Advertisement Interval* (MRAI). This is implemented by a BGP peer on the sender side, to limit the time between two successive BGP Update messages that provide information about the same AS. This reduces the amount of route exploration that is possible [12]. The timer is set to 30 seconds normally, limiting the interval between updates to at least 30 seconds. This is primarily to deal with instability in a time scale of tens of seconds [19].

#### 2.1.3.3 BGP Route Flap Damping

*BGP route flap damping* [19] is another property of interest that is implemented on the receiving side of router messages. This is designed to reduce the effect of edge instability propagating further through the Internet, as well as handling instability for longer periods of time than MRAI. It was introduced in the mid 1990's to prevent route flapping, which is a type of oscillation that occurs on BGP networks when a route's availability continuously changes state. This change of state may then cause a router to send frequent updates to surrounding routers about the change in best-routes, potentially initiating localised routing instability. The protocol handles this by only allowing a certain number of the updates to be received, after which they are suppressed for a period of time. This attempts to distinguish between routes that occasionally fail and those that are persistently unstable [20].

Each receiver keeps a penalty value for every peer-prefix combination that it has received. Whenever it receives an update, this penalty associated with the relevant peer-prefix is increased. When the penalty value reaches a preset *suppression threshold*, the router temporarily stops using this route. The penalty value undergoes continuous decay, so that when the penalty decreases to below *reuse threshold* the route will be available once again. Alternatively, when the *maximum suppression time* is reached, the route also becomes available. This ensures that no further updates will be sent when there are continuous changes on a route.

The default BGP route flap damping values for *Cisco* and *Juniper* routers are shown in Table 1. Both Cisco and Juniper are large network-equipment suppliers, with routers widely deployed across the Internet. The Cisco routers don't penalise routers re-announcing the same peer-prefix routes, but both default implementations will quickly suppress routes if a series of withdrawals for the same peer-prefix combination is repeated.

**Table 1. Default BGP route flap damping for Cisco and Juniper routers.**

| Damping Parameter | Cisco Value | Juniper Value |
|---|---|---|
| Re-announcement penalty | 0 | 1000 |
| Withdrawal penalty | 1000 | 1000 |
| Attributes change penalty | 500 | 500 |
| Suppression threshold | 2000 | 3000 |
| Half-life time | 15 | 15 |
| Reuse threshold | 750 | 750 |
| Max. suppression time (min) | 60 | 60 |

The research in [21] shows that the BGP route flap damping has not been fully deployed across Internet routers, but even if fully deployed the time taken to recover from instability would be extended by route flap damping. Also, the work suggests that this partial deployment may worsen the Internet's response under certain conditions. In [20], the authors also note that the presence of route flap damping can cause a substantial delay in convergence times for relatively stable routes.

## 2.2 Worms

### 2.2.1 Overview

The first worms were written as experimental programs starting in the late 1970's. The term 'worm' was first used in the programming context in 1979, and was inspired by a monster tapeworm from a John Brunner novel. A few newer definitions of worms are listed below:

- An autonomous intrusion agent, more specifically a software module which is able to automatically infect a computer, with the purpose of using it to infect another system, causing an exponential expansion [22].
- "A worm is a program that can run independently and can propagate a fully working version of itself to other machines. It is derived from the word tapeworm, a parasitic organism that lives inside a host and uses its resources to maintain itself." [23, 24]
- A worm is a stand-alone automated program designed to exploit the network to find susceptible computers so that they can infect them with copies of themselves. [7]

Worms are not parasitically linked to host programs, but capable of propagating by themselves, unlike Trojans and viruses which require human intervention to spread [25]. The term *worm* is often used interchangeably with *virus*, but the latter is defined as:

- A *virus* is a piece of code that adds itself to other programs, including operating systems. It cannot run independently – it requires that its "host" program be run to activate it. As such, it has an analogy to biological viruses – those viruses are not considered alive in the usual sense; instead, they invade host cells and corrupt them, causing them to produce new viruses [23].
- A *virus* replicates by attaching itself to another program so that the virus' instructions are called during execution of the host program [7].

Initial work was done to use worms to perform harmless tasks such as posting announcements [25]. The Morris worm in 1988 was the first malicious Internet worm, after which it became evident that malevolent or malfunctioning worms could be a serious threat. The increase in worms recently has coincided with the

improvement in computer networking, for which the authors in [7] suggest an urgent need for automated and coordinated protection measures.

This section provides an overview of general worm propagation, modelling and impact. This is followed by case studies of the Code Red worm, which is applicable to the dataset used for the results in the scope of this work.

#### 2.2.1.1 The Morris Worm

In [23], the author analyzed the first attack on the Internet by an Internet worm, which was initially thought to be a virus [25]. This worm was called the Morris worm, named after the suspected author. The worm first appeared on 2 November 1988, and only targeted machines with very specific hardware and operating systems.

The program consisted of 99 lines of source code, which were compiled on the host [22], and gained access to computers by exploiting flaws in the operating system. It then collected information about the host and network to allow it to infect other computers. The computer was able to continuously be re-infected, with each infection starting a new process that would attempt to spread. The effect of this was that the computer became increasingly overloaded by these processes, which administrators recognised by a gradual degradation in performance. If left unattended, these machines eventually failed due to their swap space or process tables becoming exhausted.

Although the worm did cause several machines to fail as well as a disruption to the Internet, there was no code within the worm that caused any explicit damage to the system that it ran on. Even though this was the case, it managed to cripple a large portion of the Internet [22].

### 2.2.2 Modelling

Several studies have attempted to model the propagation of an Internet worm [26, 27], based on data retrieved from previous worms such as Code Red and Nimda. Initial studies based the models on epidemiology research, such as [27]. Figure 3 shows the classic epidemic spread of a biological worm, which can be simply related to that for the spread of a computer worm. These models are *homogenous* [6], meaning that an infected host has an equal likelihood of infecting the susceptible hosts. Several models have expanded on this, such as [6], which expands on the epidemic model but includes the dynamic factors of Internet worms such as patching infected computers and the limitation of spread by network traffic.

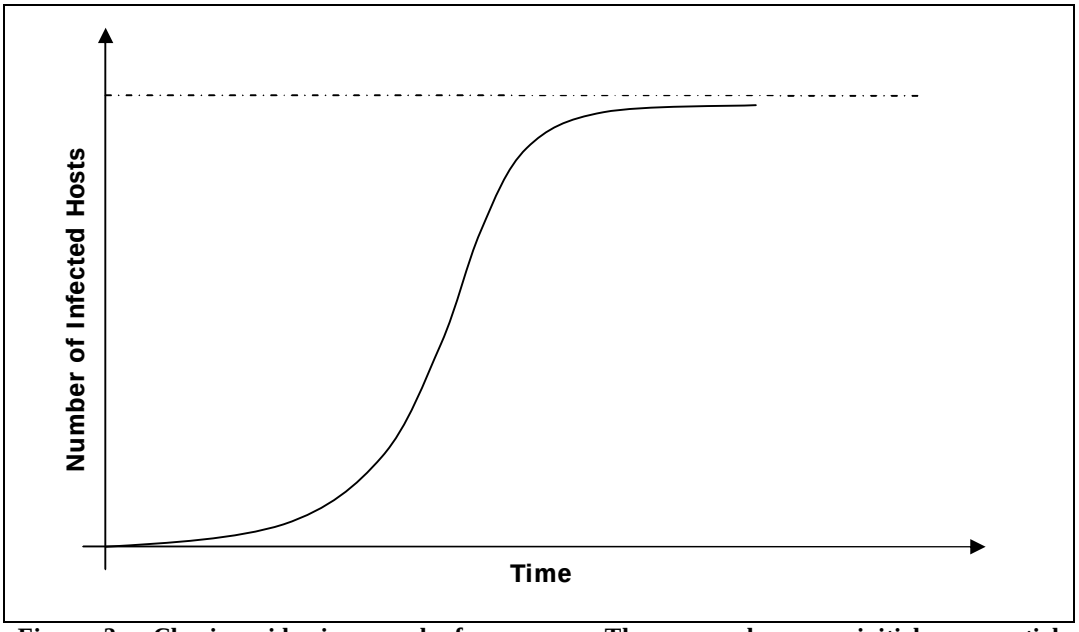


**Figure 3. Classic epidemic spread of a worm. The curve shows an initial exponential propagation, followed by an inverse exponential expansion as the number of susceptible hosts rapidly decreases.**

The dynamic factors that make a worm difficult to model include human countermeasures such as anti-virus programs which remove worms, the patching or upgrading susceptible computers making them immune to the worm and the configuration of firewalls and routers the block the worm traffic. An additional factor is the disconnection of computers from a network until a more effective method to remove the worm is made available [6].

These models do help to explain how worms spread, but do not include worm-specific traits (such as the Code Red worm which was programmed to stop propagation at a specific time). It is also difficult to model the effects of a worm, since the data from the effect of the multiple worms cannot be separated. This was the case with the Code-Red I and Code-Red II worms, where the two worms were actively disrupting the Internet at the same time [27]

### 2.2.3 Propagation

Although it is probably not the intention of the writers of worms to slow down the Internet, since this slows the rate at which worms spread, worms do have a significant effect on the Internet as a whole. Ports that are being used to transmit the worm can be temporarily blocked at firewalls and routers to limit the worm spread, but this also disrupts the flow of any valid traffic using these ports.

If abnormally high load is caused, the infrastructure that is used to send information can become overloaded (such as Internet routers), which in turn degrades all transfer of data on the Internet. This includes traffic that is not related to propagation of the worm, i.e. traffic that uses different ports to propagate.

Some of the factors that contribute to the propagation of worms are [6]:

- The human countermeasures discussed above. Examples include patching computers, blocking worm traffic and fire-walling worm traffic.
- As the Internet becomes overloaded, the worm takes longer and longer to transmit itself to new hosts, due to the high traffic on the Internet.
- As the number of infected hosts increase, the number of susceptible hosts correspondingly decreases.

An interesting side-effect of either disconnecting the computers from the Internet, or the difficulty imposed in communicating with other computers due to the slowness of the Internet, is that a computer may be unable to retrieve the information required to immunise the computer. An example of this was with the Morris worm [23], when a temporary way of immunising the computer was posted on a news group. The spread of this solution was hampered because many computers were disconnected from the Internet to ensure that they wouldn't become infected, and as a result were unable to remove the worm.

### 2.2.4 Impact

Recent worms, such as the Blaster and SQL Slammer worms in 2003, have shown that worms continue to spread even with widely used anti-virus software, firewalls and the newer Intrusion Detection Software (IDS) [7]. Worms have also evolved to the stage where they are now able to attack computers using multiple types of infections, and are able to update themselves dynamically from the Internet and actively attack anti-virus software.

Once a worm has spread to a host, the attacker has effective control over all the infected computers. With that control, the attacker is able to launch a distributed denial-of-service attack against one or many hosts, crippling the host's ability to serve content. This has a severe impact on companies such as *Amazon.com* which rely on the Internet for their business. The worms are also able to access any sensitive information on the computer, possibly retransmitting this information so that it is available to the attacker. This sensitive information could include credit card

numbers, email address books, archived email correspondence or any other information on the computer [27]. Additionally, information can be corrupted by the worm, creating confusion and a lack of trust for all the information available on the computer.

The *Lirva* worm, released in January 2003, had the ability to email Windows dial-up networking passwords to the worm author. It also connected to websites to download and install the Back Orifice software, which provides full control of the computer to the hacker over the Internet [7].

The cost effect of worms and viruses is difficult to estimate, since it is a combination of the time spent analysing and patching the problem, as well as the loss of productivity by end-users who are unable to use their computers or the Internet. The effect of worms in the public and private sector has been estimated to cost millions of dollars. The *Code-Red* worm which spread on 19 July 2001, together with subsequent strains, caused approximately $2.6 billion [9]. The estimated cost of SQL Slammer was estimated at $1 billion [28]. From [7], the cost of the *LoveLetter* worm caused $7 billion in damages, and the *Melissa* email worm from 1999 created at least $80 million in damages.

The following is an indication of the speed at which worms are able to spread:

- The *Code-Red II* worm required less than 14 hours to reach its saturation of more than 359,000 infections, with the worms spread peaking at just over 2,000 infections per minute [25].
- The *Nimda* worm, which had five different methods of attacking hosts, managed to spread to 450,000 hosts in its first 12 hours of propagation [7].
- The *SQL Slammer* worm targeted a buffer overflow vulnerability in computers running Microsoft's SQL Server. The worm infected at least 75,000 hosts over a period of approximately 10 minutes [29].

From the examples above, worms are capable of causing a large amount of damage simply through their rapid propagation. Anti-virus software and intrusion detection will continue to improve, but are limited in their ability to detect novel worms. A significant number of computers connected to the Internet don't have anti-virus software running or the latest operating system patches, and are therefore susceptible to worms. For these reasons, worms will continue to remain a threat.

### 2.2.5 Prevention

The study in [9] concludes by listing the methods required in containing a worm. The first, *Reaction Time,* is the time taken to detect the worm which they suggest should be automated to be effective. The second method is the *Containment Strategy*, which is the means to isolate a computer from susceptible hosts. The last method is the *Blocking Location*, which is how the countermeasure system is deployed. The study concludes that this should be implemented across almost all Internet paths for the system to be effective, requiring a large amount of cooperation between companies and organisations.

Some of the methods used to prevent worms from [30] are:

- *Compile-time software protection* is the ideal approach whereby programs are released without bugs that can be exploited by worms.
- *Run-time software protection* which is the ability to detect misuse by analysing signatures of worms, which relies on the tool used to have the patterns already installed. Anomaly detection is also included in this category, and relies on detecting of some of the patterns of a worm's behaviour.
- *Network filtering* is the means of blocking worm traffic on a network. Address blacklisting is used to block traffic from suspected worm sources using firewalls. Advance worm filtering is a preventative method of blocking vulnerabilities before a worm has been detected or written. *Content filtering* is used to detect worms by scanning traffic sent through a network, but this method is difficult to deploy due to the high load of traffic that should be scanned and the problems with dealing with false positives. *Honeypots* are computers that are deployed on the Internet with no purpose except being made available to attacks. It is assumed that any traffic sent to them might be malicious (such as a worm), and this traffic can be analysed to detect worms.

## 2.3 Code Red Case Study

### 2.3.1 Overview

A serious vulnerability was announced by Microsoft on June 18th 2001 [6], which affected Microsoft Windows systems running Internet Information Service (IIS), Microsoft's web-server. Since the purpose of a web-server is to provide content to client browsers such as Internet Explorer and *Mozilla Firefox*, many of the computers running the web-server were accessible to the Internet, although some may have been used solely for internal company Intranets.

In less than a month after the vulnerability was announced on July 12th 2001, the Code Red I worm emerged which exploited this IIS vulnerability [21]. A newer strain of the worm called Code Red I v2 began to propagate on July 19th 2001. This version of the worm was designed to stop propagating at 0:00 on the 20th of each month. On the 4th of August 2001, a new worm emerged, called the Code-Red II worm.

This section provides a more detailed description of the Code-Red I and Code-Red II worms which infected hosts during 2001. This specific worm has been selected for analysis, since it occurs within the time period for the data used in this project.

### 2.3.2 Chronology of Events

The following table summaries the incidents relating to the Code Red worm [6, 25]:

**Table 2. Significant dates for the spread of the Code-Red worms.**

| Date | Event |
|---|---|
| June 18, 2001 | eEye provided information about a buffer-overflow vulnerability in Microsoft's IIS web server. |
| June 26, 2001 | Microsoft released a patch for the vulnerability. |
| July 12, 2001 | Code Red I v1 started to exploit the vulnerability. |
| July 19, 2001<br>Approx 10:00 UTC | The Code Red I v2 variant emerged, which was observed to probe new hosts, sharply increasing the number of infected hosts. |
| August 4, 2001 | A new worm emerges, named Code Red II. |

The worm was only capable of spreading on Windows 2000 with IIS installed, part of the code is invalid on the Windows NT operating system [9] and causes these and other systems to crash [31].

#### 2.3.2.1 Code Red I

After infecting a computer, the first version of Code-Red initially determines if the date on the computer is between the 1st and the 19th of the month. In this case, a random list of IP addresses is generated, and each is attacked in order to allow the worm to propagate.

A serious limitation of this version was that the seed used to generate the list of IP addresses was static, and so each computer continuously generated identical lists. This permitted the worm to propagate to the computers in the list, but once these hosts were infected, the computer was unable to spread to additional computers. This caused the worm to spread slowly.

Between the 20th and 28th of the month, the worm was designed to stop spreading, and instead launch a denial of service attack against the website http://www1.whitehouse.gov. The worm is not active after the 28th of the month, but begins spreading again on the 1st day of the following month.

The Code Red I v2 variant appeared on July 19th, 2001. This variant was similar to the previous version, except that it used a random seed to generate the list of hosts to infect. The worm polled computers in the list of random IP addresses at approximately 11 hosts every minute. Another difference between this version and the previous version is the text "Hacked by Chinese" inserted at the top of some web pages generated by the IIS web server.

Both the initial and variant versions of the Code Red I worm are not stored on the computer, so a simple reboot completely removes the worm from the computer. But, since the vulnerability is still present on restart, the computer is open to being re-infected. The worm omits the loopback and multicast IP addresses.

#### 2.3.2.2 Code Red II

Although similarly named to the Code Red I worm, this worm is not a variant or new version of the Code Red I worm. It was named due to some of the source code containing the phrase "CodeRedII". This worm exploits the same Microsoft IIS vulnerability as the Code Red I worm [25].

When infecting a new machine, the worm checks to see whether the worm is already present on the host. If not, it creates a backdoor and then waits for a day, creating a new vulnerability. After this wait is over, the machine reboots itself and the worm starts to spread.

The worm has various rules for determining its behaviour when propagating itself:

- If the system language is 'Chinese (Taiwanese)' or 'Chinese (PRC)' it uses 600 threads to propagate, otherwise 300 threads are used.
- When selecting which host to probe, a random IP address is created and modified according to various rules to predominately infect computers within the same area. One in eight probes will use the randomly generated IP address (same as Code Red I), three-eights of attacks will modify the random IP to fall within the same /8 IP prefix of the network and the remaining half of the attacks will be within the same /16 IP prefix of the network. An example of the /8 IP selection is that a host with address 66.102.11.99 will then poll addresses starting with 66.xxx.xxx.xxx.
- The loopback and multicast IP addresses are ignored.

This version doesn't modify any web pages or launch Denial-of-Service attacks, but allows for any code to be executed using the backdoor that the worm creates. This provides the means for the worm writer to exploit the system at any point later until the worm is removed. The Nimda worm used this backdoor as one of its methods of propagation.

### 2.3.3 Impact and Patching

From the detailed analysis performed in [25], the Code Red I v2 worm infected more than 359,000 computers in less than 14 hours. The overall cost of Code Red, including all strains cost an estimated $2.6 billion. The infection rate reached more than 2,000 infected hosts per minute. The primary defence against the Code-Red worm was ad-hoc containment by individual networks by system administrators – such as by blocking inbound traffic on TCP port 80, or filtering content to find the worm signature [9]. From the experimental evidence in [25], it is evident that human administrators are not able to slow the spread of a worm in under 24 hours.

A web server survey performed by Netcraft indicated that there were approximately 6 million Windows IIS instances at the end of June 2001 [32]. Conflicting estimates of the percentage of hosts infected by Code-Red I v2 to those that were susceptible differ, with [27] concluding that almost all online susceptible hosts were infected, while later research in [6] estimates approximately 60% were infected.

From the work done in [25], network monitors were installed to gather data about the spread of the Code Red I and II worms. By mapping the IP addresses of infected

host machines to countries, they were able to conclude that the worm was an international event – physical and geographical boundaries did not affect the spread of the worm. Home and small business web-servers were as severely impacted as corporations, demonstrating the impartiality of the worm.

In the same study, the rate at which users patched infected computers showed that the number of patched computers increased slowly towards the end of July 2001, while the worm was not propagating. On the 1st of August 2001, only about 32% of infected hosts had been patched, even though there was widespread media coverage of the worm and its effects. Only once the worm had restarted its propagation cycle at the beginning of August 2001 did the rate at which infected hosts were patched increase, reaching approximately 64% of hosts patched by 28 August 2004.

# 3 Novelty Detection using Neural Networks

This chapter provides a brief introduction to general neural networks and novelty detection. This if followed by a more detailed description of the autoencoder, which is a specialised type of neural network used for novelty detection.

There are several methods available to solve a problem for approximating a system's behaviour, with the choice of solution dependant on the type of system and the data available. A problem-solving system is used to map data from the input space to the domain space, and then into the solution space. The ideal mapping from the input space to the domain space is known as the objective function of the system. The following list shows some of the methods available [33]:

- *Statistical methods* are useful when the data can be represented statistically, and the underlying objective function is already defined.
- *Symbolic AI rule-based systems* for well defined problems with fixed rules, and when the variation of a more advanced system is not required or too difficult to construct.
- *Fuzzy systems* for when the problem knowledge is heuristic. These systems are vague and ill-defined but are useful when little data is available.
- *Neural networks* are suited for when the problem knowledge is not well understood and the objective function is unknown. These require data for training and validation, from which the network extracts the problem knowledge.

## *3.1 Neural Networks*

A neural network is a computational model based on the biology of the human brain [33]. The processing elements within the model are neurons, which are interconnected to perform the calculations. The connections between the various neurons are weighted, to provide the long-term memory for the system. The neurons combine the values received from each of the connections, and perform calculations before sending values to other neurons via more weighted connections.

The structure of a basic neural network called a Multilayer Perceptron (MLP) is shown in Figure 4, with $m$ input neurons, two hidden neurons and $n$ output neurons. The input layer's neurons are connected to each of the hidden layer's neurons, which

are in turn connected to the output neurons. Each connection carries a certain weight which is determined during training. Neural networks can be connected in several ways, the figure shows the most commonly used generic example.

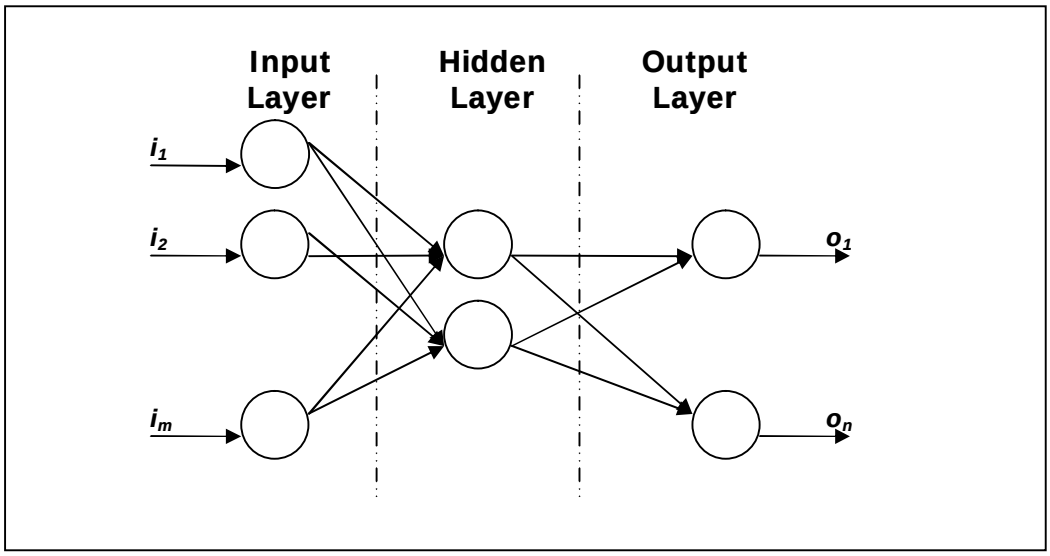


**Figure 4. Basic structure of a neural network. The various layers are connected using several weighted connections which are assigned values during training.**

MLP's were used only after methods of training them had been developed. A simple training method is called the *backpropagation* algorithm, which is used to continuously adjust the network connection weights based on the error that is fed back when learning from training examples. A more advanced training technique is the *Scaled Conjugate Gradient* (SCG) method, which is able to reach a local minimum in orders of magnitude faster than the backpropagation technique, but is far more complex.

Some of the general features of a neural network are [33]:

- *Learning* - the network is capable of learning without any previous knowledge. This is performed by training the system with a set of training input-output pairs (supervised learning), or with only input data (unsupervised learning).
- *Generalisation* – the network is able to generate a best output when presented with input data that the network hasn't seen before.
- *Massive parallelism* – when computing the output values, the network is able to process many neuron calculations simultaneously.

- *Robust* – if some of the neurons within the network have been poorly trained, the network as a whole may still be able to generate strong results.

A properly configured and trained neural network is able to implicitly represent the underlying hidden knowledge of a system, which may not be obvious to a human observer. It is also able to use noisy, missing and corrupted data and still provide good output [33].

## *3.2 Novelty Detection*

Novelty detection is the ability to distinguish a level of novel behaviour from what is considered normal behaviour [34]. This approach only requires normal behaviour to be defined, from which abnormalities can then be identified against this description [35]. Systems trained to detect novelty are unable to detect faults, but can robustly identify anomalies even when the system has never seen the fault before.

As noted for the tests performed in [35], novelty detection can be effective in fault detection and prediction for a single type of fault, but attempting to predict additional faults may not be as effective. The tests performed in their work used a neural network to detect real faults, and were found to retrospectively provide an effective measure of a single fault. It was essential that the inputs to the neural network were not corrupted by any faults, otherwise the results became unreliable.

### 3.2.1 Autoencoder

The auto-associative neural network (or *autoencoder*) is a specialised type of neural network that has the ability to detect novel behaviour. The network is trained using only normal behaviour, so that when data is presented that falls outside of the acceptable trained behaviour, the network is able to show this novelty. This behaviour is not necessarily "good" or "bad", it simply falls outside of the data that the network has been trained with [34].

The basic structure of an autoencoder consists of the same number of inputs as outputs. The training requires teaching the network to attempt to match each output to the corresponding input, so that the error between inputs and outputs is low for the normal or training data, but varies when the input dataset is unrecognised.

The autoencoder has the following features [34]:

- Principal Component Analysis (a widely-recognised method for dimensionality reduction, applications include pattern recognition and image processing [36])

- Information compression
- Recovery of missing sensory data
- Implicitly learn characteristics of the underlying data

The structure of an autoencoder is shown in Figure 5. The auto-associative nature of the network is provided in the training phase, with the same values used for inputs and outputs. The hidden layer provides a "bottleneck", mapping the input space onto a reduced dimensional space. The output of a trained network will be similar to the inputs if the input is similar to the training dataset. If the data provided to the trained network is not similar to the training dataset, the outputs will differ from the inputs.

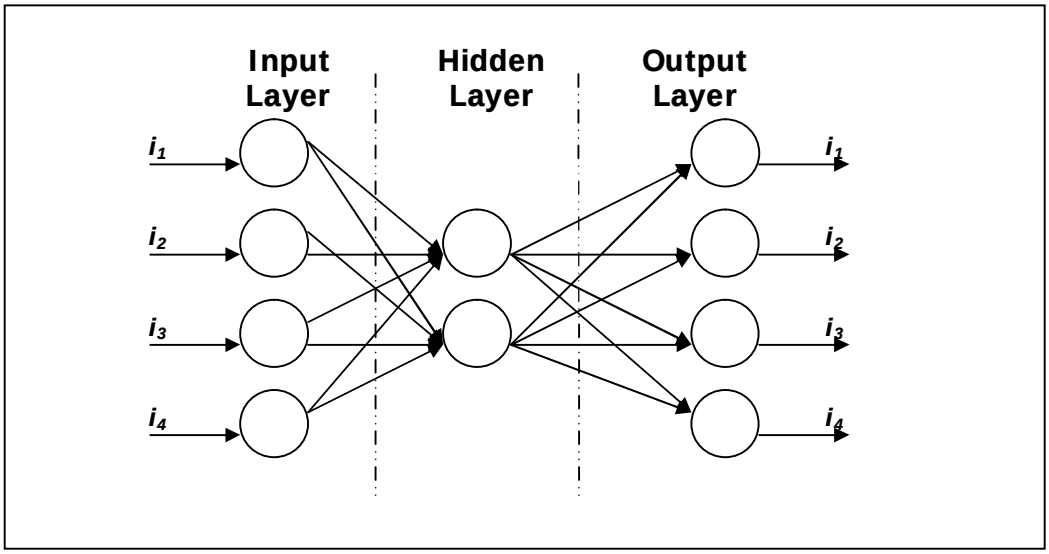


**Figure 5. Autoencoder structure with training inputs equal to outputs. The outputs are trained to match the inputs as closely as possible. The number of hidden neurons should be kept to less than the number of inputs.**

Since there is often more than one input-output pair for a neural network, a method is required to provide a single measure for the degree of novelty of the input data. The method used in [34] simply takes the sum of the differences between the inputs and the outputs. Another method is to use the mean-square value for the differences between the inputs and the outputs, as in (1).

$$e = \frac{1}{n}\sum_{1}^{n}\left((t_n - i_n)^2\right) \tag{1}$$

where $n$ is the number of inputs and outputs, $t_n$ is the $n^{th}$ output and $i_n$ is the $n^{th}$ input. The resultant error value $e$ is the level of novelty of the combined outputs.

The novelty output of the autoencoder is a relative value, since it is only comparable to other novelty values from the same trained autoencoder. Therefore, a relative threshold value should be determined, so that any values above this threshold are novel, and values below this can be considered normal. The threshold can then be set by looking at the outputs for real data, and deciding based on the range of the outputs which seem most likely to be novel. Otherwise, if a novel value is already known for the real dataset, the novelty output of the autoencoder for this data can be used as a guideline for the threshold.

### 3.2.2 Testing the Autoencoder

The autoencoder has been tested using both simulation and real-world data in [34], with promising results. The first test was performed by training a neural network using generated Gaussian white noise. The network consisted of 20 input and output neurons, and 18 neurons in the hidden layer. The input-output pairs represented a sliding window of the input data, with the first neuron representing the current data value, the second the value for the previous discrete time frame, and so on for all subsequent inputs. The network was trained using approximately 20,000 training datasets of generated noise. The following were performed to test the usefulness of this autoencoder:

- Another noise pattern to that used for training was applied to the input (generated in the same way as the training data); with the output showing very little change throughout, indicating that the input contained no significant novelty (see Figure 6).
- The second test used two Gaussian-shaped pulses in combination with the generated noise. It is evident from the novelty values shown in Figure 7 that the autoencoder is able to successfully distinguish these pulses from the noise.
- The next test used a simulated noisy sinusoidal wave as input (see Figure 8). The novelty result of the output from the network showed the sinusoidal pattern of the input far more clearly than the input noisy sinusoidal wave. The purpose for this test was to show that the system is capable of detecting more than just pulses.

In the same paper, the authors used real world data gathered from CPU load of a computer which was measured in intervals. The results were promising in that the

autoencoder was able to implicitly learn the parameters of this system, and correctly identify novel behaviour.

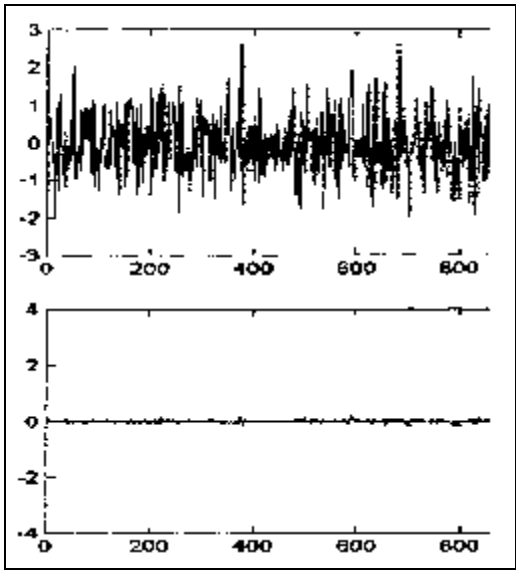


**Figure 6. Generated noise applied to the autoencoder. The top graph shows the generated noisy input, and the resulting novelty value is shown below (from [34]).**

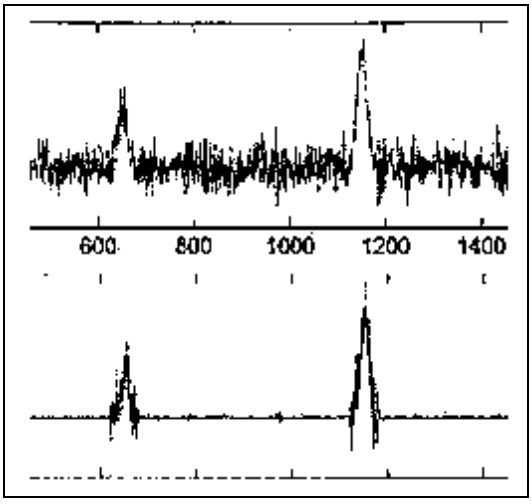


**Figure 7. Noisy generated Gaussian-shaped pulse applied to the autoencoder. The top graph shows this input, and the resulting novelty value is shown below (from [34]).**

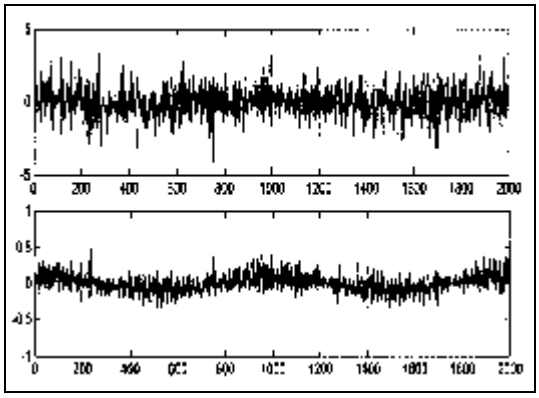

**Figure 8. Noisy generated sinusoidal wave applied to the autoencoder. The top graph shows this input, and the resulting novelty is shown below. This demonstrates the networks ability to filter out the noise and extract the sinusoid, which is not clearly visible from the input graph (from [34]).**

A disadvantage of the autoencoder is that the set of normal data-points must be selected. This technique is often reliant on human selection of the standard data-points.

### 3.2.3 Normalisation

A commonly used method of pre-processing the data for a neural network is to use normalisation. This involves adjusting the values supplied to the inputs of the neural networks to the same scope [33], usually to the range [0,1] or [-1,1]. Without normalisation, the variable with the largest scale will dominate the output, whereas with normalisation the inputs have equal importance. It is necessary to perform a reverse transformation on the output values to transfer the values back into the original data range, if required.

Two methods of normalisation are *linear* and *logarithmic* [33]. *Linear normalisation* uses the formula:

$$x_{ij,norm} = \left(x_{ij} - x_{i,\min}\right) / \left(x_{i,\max} - x_{i,\min}\right) \quad (2)$$

where $x_{ij}$ is the value being normalised, $x_{ij,norm}$ is the normalised value and $x_{i,min}$ and $x_{i,max}$ are the minimum and maximum values for the variable $x_i$.

*Logarithmic normalisation* is effective when the domain used for an input is very wide, so applying a logarithmic normalisation to the input reduces the dynamic range of the data. This technique has not been used in this project since linear normalisation is sufficient, so no further explanation is provided.

## 3.3 Effect of Worms on BGP

This section forms the basis of the research for this project, by providing a justification for the choice of input for the neural network, which is explained later.

Worms have the ability to not only affect the computers at the endpoints of the Internet, but are also able to impact the infrastructure connecting these computers together [10]. From the research performed in [10], it was shown that the *Code-Red* and *Nimda* worms were responsible for the global Internet outages during 2001. Of particular interest to this project, it was also evident that the number of BGP announcement messages increased significantly during these times. Edge network administrators reported "insane ARP storms", router failures, congestion slowdowns and connectivity problems during the *Nimda* attack.[11]

It is difficult to determine Internet weather from an endpoint, since this only gives an impression at that point [11]. Even monitoring load in the Internet's core (such as load through routers) doesn't provide a reliable impression for the end-users experience of the Internet speed. For this reason, research has shifted to examining the routing data itself, rather than the load though the routers. The most surprising effect of this research is that outages within the Internets core – such as router failures or fibre cuts – do not significantly affect the routing data sent by routers.

According to [11], there are two main goals for using routing information to monitor Internet instability – *reachability* and *rates of change*. *Reachability* is used to determine how many autonomous systems can be reached by a router. *Rates of change* are used to measure the number of BGP Update messages (announcements and withdrawals) that have been sent in a specified time. Since BGP has a limit on the number of messages that can be sent from a router about a route change within 30 seconds (which is the *BGP route flap damping* effect described previously), if the route continuously changes in each time period this signifies that the diversity of the routes is increasing.

### 3.3.1 BGP Impact from Code-Red and Nimda Worms

From the research performed in [11], which was analysed during the period June to September 2001, the authors analysed several occurrences that affected the Internet:

- The Baltimore train wrecks on 18 July 2001, which eliminated many fibre links.
- The Code-Red II worm which started affecting the Internet on 19 July 2001.

- The World Trade Centre attacks on 9 September 2001.
- The Nimda worm which occurred on 18 September 2001.

The graph shown in Figure 9 shows a logarithmic plot of the total number of aggregated BGP prefixes announced (or BGP announcements) between 1 June 2001 and 24 September 2001. These were collected from a single router that has 13 RIPE NCC [37] collection points which are based in London, Paris, Amsterdam, Geneva and Vienna. The router provides an interconnection between many small European networks as well as many global 1-tier providers. Each point on the graph represents the number of BGP prefixes announced for a 30-second time slice. These collection points continuously log information about the traffic passing through, including the BGP messages such as the number of announcements.

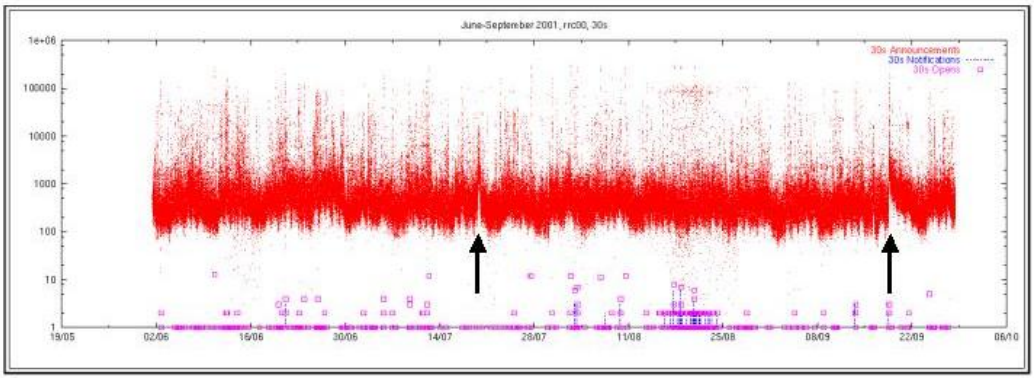


**Figure 9. Number of BGP announcement messages sent from June to September 2001 from 13 RIPE NCC collection points. The arrows indicate the increase in messages caused by the Code-Red and Nimda worms (from [11]).**

The authors in [11] observed significant daily and weekly trends in this data, possibly due to fluctuations in traffic load, or otherwise due to network administrators performing maintenance shutdowns of routers (which makes routes via the router unavailable). The non-periodic effects that are evident are the sharp increases on July 19th and September 18th, which correspond to the Code-Red II and Nimda worms respectively. It is interesting to note that the Baltimore train wreck and the World Trade Centre attacks do not feature noticeably on this graph.

The graph in Figure 10 shows an expanded view of the number of BGP announcements from the *RIPE NCC* routers on 19 July 2001. The y-axis is logarithmic, showing an eight-fold exponential increase in the number of updates. The increase was present for approximately eight hours, and was initially thought to

be a delayed reaction to the Baltimore train wreck which had occurred on the previous day.

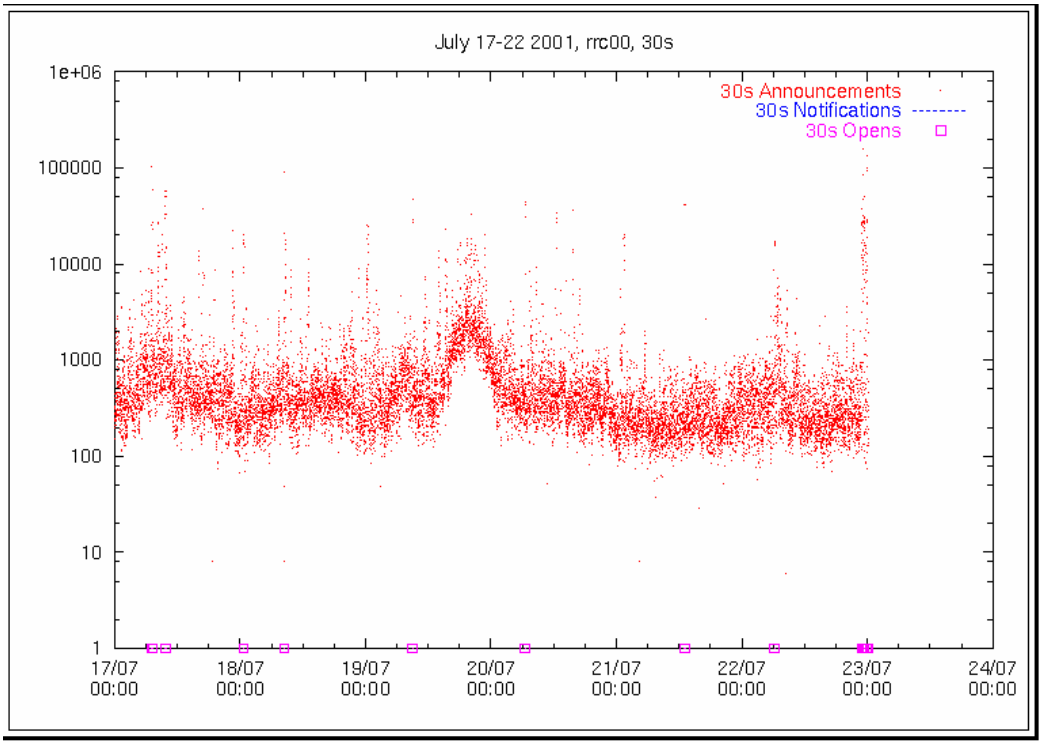


**Figure 10. Increase of BGP announcements from several RIPE NCC routers around the time of the Code-Red II worm (from [11]).**

The increase in BGP announcement messages caused by the Nimda worm is shown below in Figure 11. The y-axis on this plot is also logarithmic, and over a period of two hours increase exponentially by a factor of 25, after which it slowly decreased for the next few days and reached pre-worm levels only on 24 September 2001.

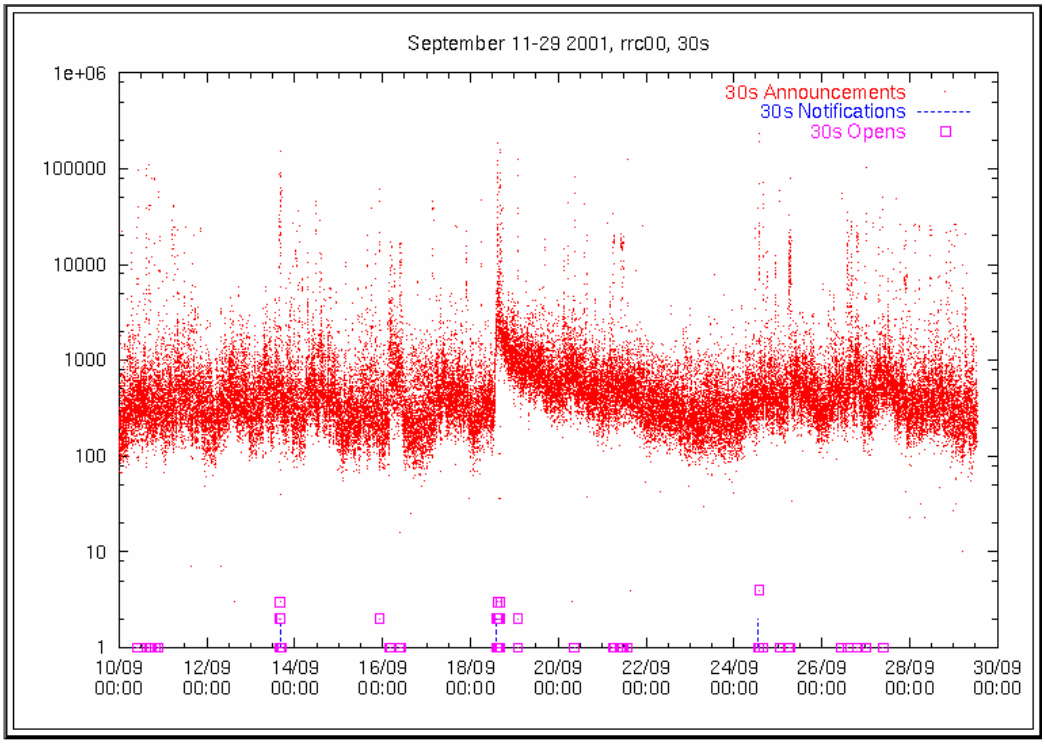


**Figure 11. Increase of BGP updates from the RIPE NCC routers caused by the Nimda worm on 18 September 2001 (from [11]).**

### 3.3.2 BGP Impact from SQL Slammer Worm

The SQL Slammer worm first appeared on 25 January 2003, and exploited hosts running the Microsoft SQL Resolution Service [10]. The worm was able to infect over 75,000 hosts in just over 30 minutes, becoming the fastest spreading worm in history [8]. The worm was capable of scanning at 55 million hosts per second at its peak, and doubled the number of scans every eight minutes, compared to Code-Red II's doubling rate of about 37 minutes [10]. As in the case with the Code-Red and Nimda worms, the SQL Slammer worm wasn't directly targeting the Internet routing infrastructure. It also caused extensive damage and downtime for many users, even those not directly infected by the worm.

The SQL Slammer worm used the UDP protocol to propagate, unlike the Code-Red and Nimda worms which relied on the TCP protocol. Even with this difference, the number of BGP messages sent by routers was similarly affected to that of the Code-Red and Nimda increases [8, 38], which is also shown in Figure 12. Using data collected from several AS's, the decrease in the number of routes due to SQL Slammer is plotted in Figure 13.

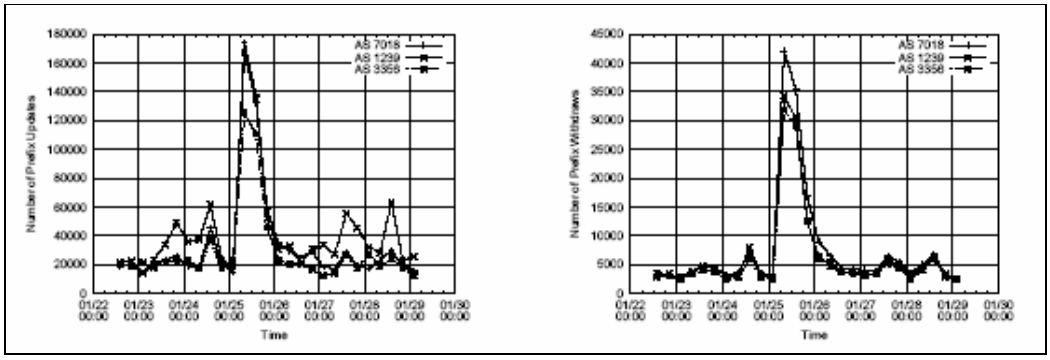


**Figure 12. Number of BGP announcements (left) and withdrawals (right) during the SQL Slammer attack in 2003. This information was extracted from routers in three different autonomous systems (from [21]).**

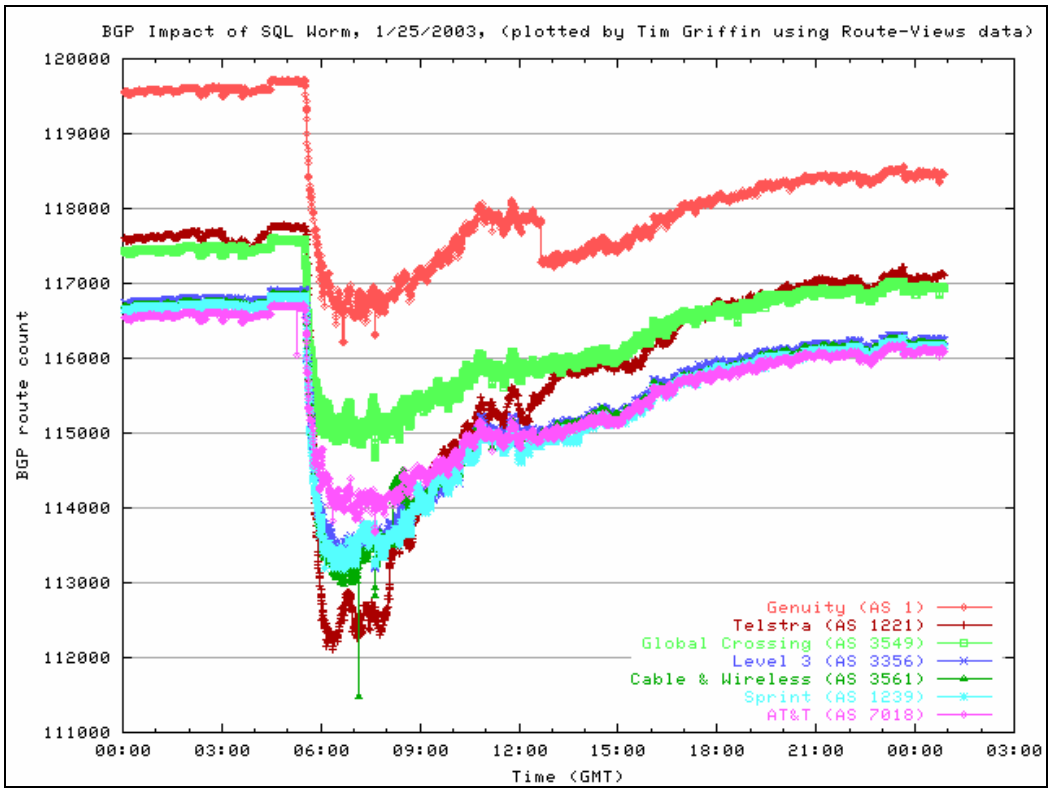


**Figure 13. Available best-routes for several Autonomous Systems at the time of the SQL Slammer worm (from [10]). The number of routes drops significantly for the various AS's at the time of the attack.**

The analysis performed in [21] shows how a local disturbance on the Internet can be propagated globally, with the edge networks affected by the SQL Slammer worm causing global routing problems. It was also noted that the presence of BGP route flap damping assisted in reducing the size of the SQL Slammer BGP storm, but in

itself was not enough to stop the storm. From the analysis, it was also clear that many of the routers did not have damping installed or enabled during the worm outbreak. This was evident by the penalty values for the damping far exceeding the default values when routes should have been suppressed. The research also points out that implementing damping at all core Internet routers (the RFC in [19] recommends deploying damping only at core routers) will make a significant improvement. Further, this limits the number of BGP Updates that can be sent between routers closer to the Internet edge. The study concludes by saying that the partial implementation of route damping is only part of the problem because for certain network topologies the partial implementation of damping could actually worsen routing performance.

Another study in [20] showed that BGP damping was also responsible for slow convergence times for recovering from edge instability. It was seen that instability from one end of a network is able to trigger BGP damping in another end of the network, which is undesirable since this increases the convergence time for the network.

### 3.3.3 BGP Behaviour for Worms and Outages

The constant stream of BGP announcement and withdrawal messages is called a "BGP message storm" [11], and additionally causes increased Internet load. The effects of these storms can be summarised by the following list of suspected causes from [11]:

- BGP session timeouts caused by increased load
- BGP session failures due to high CPU loads on the routers
- Disconnection of networks by administrators protecting against the worm
- Failure of equipment at the Internet edge (such as DSL routers)

As a further justification of the fourth suspected cause above, it was noted in [10] that Cisco Systems Inc released an advisory notifying customers that hardware not using Microsoft IIS server had been affected during the time of the Code-Red II outbreak. The hardware had been impacted by the side affects of the worm.

In [39], the storms are referred to as "route flap storms", which are caused when routers fail during instability. This is described by router's long periods of instability causing them to frequently update their routing tables. Many Internet routers are based on older processors, which are sufficient under normal load but for the longer

periods with higher load the routers can either fail or simply take too long to send BGP Keep-Alive messages. In either event, this causes the router's peers to drop it from any AS paths that it is used in. This in turn forces these peer routers to select alternate routes that don't use the unavailable router, and in doing so several BGP announcement messages are sent from these routers. When the router either reboots or recovers from the instability, it attempts to re-establish connections with the peer routers which try to revert to using it again in their AS paths. This causes more BGP withdrawal messages to be sent. This is the scenario that the *BGP route flap damping* has been designed to assist with.

# 4 Methodology Implemented

This section describes the methods used to collect the data from the router, to convert the data into normalised data for the autoencoder, as well as the way in which the neural network was trained.

The autoencoder has been selected to provide the prediction of the presence of a worm. This was used rather than standard neural-network classification, since much of the data appeared to be normal. There were also concerns that the remaining data caused by a worm would be treated as noise by a standard neural network classifier. As explained earlier, the autoencoder's inherent ability to implicitly determine the system parameters without prior knowledge is useful for this application, since little is understood about the underlying behaviour of the data.

There has been no attempt to model or characterise the behaviour of worms or their effect on the Internet, due to the complexities of the various types of worms and their means of propagation. These complexities include the difficulties of isolating the effects of individual worms at a global routing level, and the different types of worm's pre-programmed idiosyncratic behaviours (such as the Code-Red I worms preset ability to stop propagating on the 20$^{th}$ of the month). By using neural networks, it is hoped that the network is able to learn the system characteristics of the Internet's routing behaviour without requiring a model. This should make the system more robust in the manner in which it would be able to detect new worms, since no assumptions have been made about previous worms. Also, the training of the autoencoder doesn't require any data or information about worms, so the results shown are independent of any training for a specific worm. The system should perform in a similar manner when presented with a worm it hasn't seen before.

## *4.1 Data Collection*

The single router that was chosen is the *rrc00.ripe.net* router [40], which collects data from 15 peer routers, and is based at *RIPE NCC* [41] in Amsterdam. The RIPE NCC project collects data from more than 300 IPv4 and IPv6 peers at 12 data collection points throughout the world. The routers run a program to continuously collect routing data, which is stored in the MRT binary format [42]. This is compressed information containing all the BGP routing information for the router. The information is stored in two ways – either with all BGP packets, or alternatively

with the entire BGP routing tables which are dumped every eight hours. The data for this router is available from October 1999, and is available without restrictions when used for research purposes.

Since this project is interested with the BGP Update messages for this router, the entire BGP data available for this router was downloaded for the period June 2001 to September 2001, which correlates to the same time period as the effect of worms on BGP which was discussed in the literature. Software was written to extract the BGP messages from the binary data, using available MRT libraries. The number of BGP Update messages – both announcement and withdrawal messages – was extracted from the raw data, together with a timestamp for each message and was extracted into a MySQL database. Information such as the AS from which the data was received has been removed. The structure of the raw data stored in the database is shown in Table 3. This data has been logged by the data collection program as it is received. There was no grouping based on time, which is evident from the values shown in the table since the timestamp for all entries is the same.

**Table 3. Sample BGP Update data extracted from the rrc00.ripe.net router's raw data. These values were received during the same second.**

| Timestamp | BGP Announcements | BGP Withdrawals |
|---|---|---|
| 2001-06-10 00:12:44 | 2 | 0 |
| 2001-06-10 00:12:44 | 0 | 5 |
| 2001-06-10 00:12:44 | 1 | 0 |
| 2001-06-10 00:12:44 | 3 | 0 |

The data is not useful as input in this format, so the data was then grouped into various time buckets. This involved splitting the time period of interest into fixed time intervals, and then counting the total number of announcement and withdrawal messages that fall within this time period. Three new datasets were created using this method, with one-, five- and fifteen-minute time intervals for each bucket. An extract from the fifteen-minute dataset is shown in Table 4.

**Table 4. Sample BGP Update data when grouped into intervals of fifteen minutes.**

| Start of Time Interval | Total BGP Announcements | Total BGP Withdrawals |
|---|---|---|
| 2001-06-10 00:00:00 | 9805 | 1997 |
| 2001-06-10 00:15:00 | 10772 | 1582 |
| 2001-06-10 00:30:00 | 16576 | 1406 |
| 2001-06-10 00:45:00 | 7424 | 1385 |
| 2001-06-10 01:00:00 | 10757 | 1667 |

## 4.2 Autoencoder Structure

The basic structure for the autoencoder used is shown in Figure 14. The number of input-output pairs is chosen initially, together with the number of neurons in the hidden layer. The way that the data collected is represented in the neural network is by taking half of the input neurons for the BGP announcements, and the other half for the BGP withdrawals. The data from the previous section is normalised and reorganised into the structure shown below:

$$i_1 = (\#\,announcements)_t$$
$$i_2 = (\#\,announcements)_{t-1}$$
$$\vdots$$
$$i_n = (\#\,announcements)_{t-(n-1)}$$
$$i_{n+1} = (\#\,withdrawals)_t$$
$$i_{n+2} = (\#\,withdrawals)_{t-1}$$
$$\vdots$$
$$i_{2n} = (\#\,withdrawals)_{t-(n-1)}$$

where $i$ represents each of the inputs, ranging between $i$ and $2n$ inputs, with $n$ BGP announcement inputs as well as $n$ BGP withdrawal inputs. The *(#announcements)* and *(#withdrawals)* is the number of normalised *BGP announcements* and *withdrawals* respectively within the specified time interval. So, $(\#announcements)_t$ is the number of normalised *BGP announcements* for the current time period *t*, while $(\#announcements)_{t-(n-1)}$ is the $(n-1)^{th}$ previous time periods number of normalised BGP announcements.

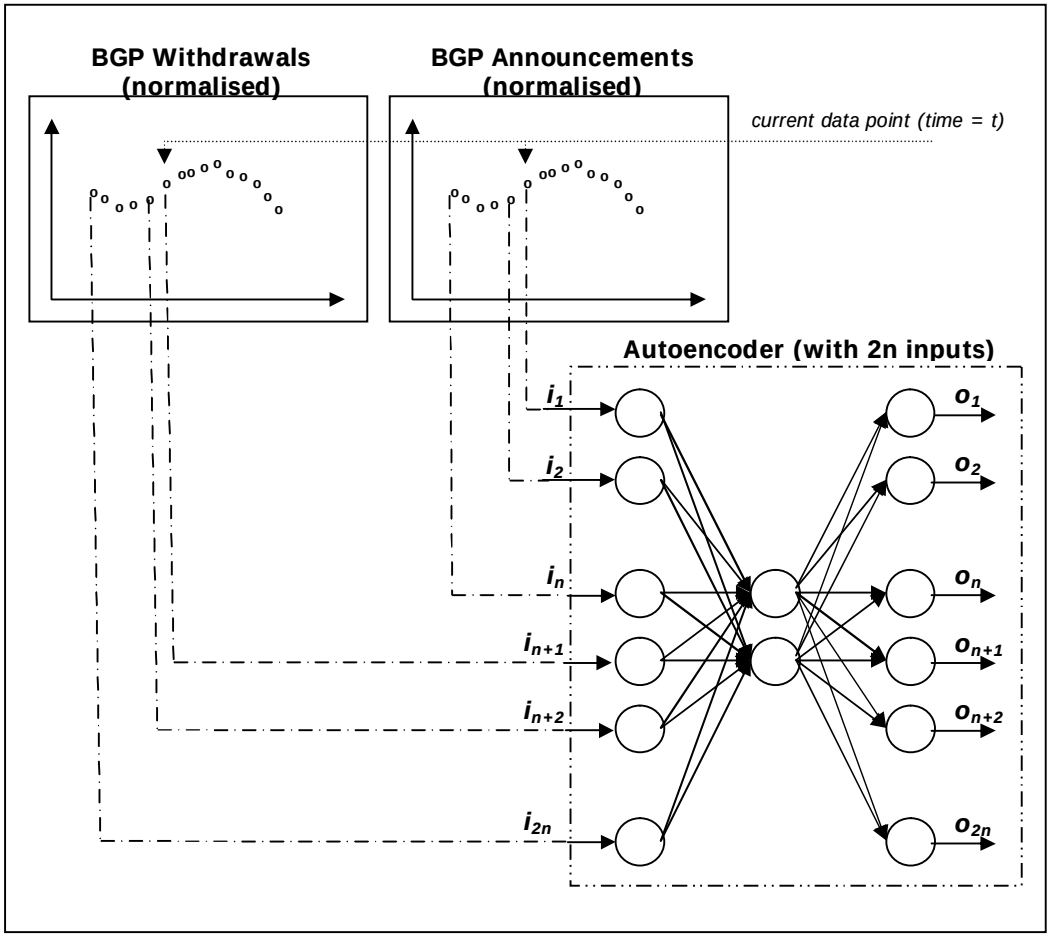


**Figure 14. Mapping of data-points to the autoencoder inputs. The first n inputs are mapped from the BGP announcements, and the next n inputs are from the BGP withdrawals. The current time interval's values are used, together with previous values providing historical data to the system.**

This structure for the inputs provides information about the current as well as previous number of announcement and withdrawal messages for the given interval. The separation between the number of announcements and withdrawals has been determined to allow the neural network to provide the network with as much information as possible, rather than using the sum of both which would provide the total number of BGP Updates for the interval.

The data gathered in the previous section is divided into datasets determined by the structure presented above. This creates the same number of datasets as the number of time intervals that there are time intervals for – with each dataset containing the current time interval's number of BGP announcements and withdrawals, as well as for the previous time periods. An un-normalised example dataset with $n=2$ (number of

BGP announcements as well as the number of withdrawals) is shown in Table 5. This sample dataset contains information for the current time period (at time *t*) and the previous time period (time *(t-1)*).

**Table 5. Sample un-normalised dataset containing current and historical BGP Update information. Each row represents a single dataset (excluding the timestamp), with one dataset for each time interval.**

| Start Timestamp | No. announcements (time = t) | No. announcements (time = t-1) | No. withdrawals (time = t) | No. withdrawals (time = t-1) |
|---|---|---|---|---|
| 2001-06-10 00:15:00 | 10772 | 9805 | 1582 | 1997 |
| 2001-06-10 00:30:00 | 16576 | 10772 | 1406 | 1582 |
| 2001-06-10 00:45:00 | 7424 | 16576 | 1385 | 1406 |
| 2001-06-10 01:00:00 | 10757 | 7424 | 1667 | 1385 |

Each individual input is then normalised using the technique described previously in section 3.2.3. The resultant normalised dataset example for the previous table is shown in Table 6. Note that all input columns contain a value ranging [0, 1]. Various numbers of inputs were used for testing, for which the entire dataset were recreated and normalised.

**Table 6. Sample normalised dataset used as input to the autoencoder, with four inputs to the autoencoder (*n=2*). Each row has been normalised to the range [0, 1].**

| Start Timestamp | Input 1 $(\#announcements)_t$ | Input 2 $(\#announcements)_{t-1}$ | Input 3 $(\#withdrawals)_t$ | Input 4 $(\#withdrawals)_{t-1}$ |
|---|---|---|---|---|
| 2001-06-10 00:15:00 | 0.177 | 0.260 | 0.699 | 1.000 |
| 2001-06-10 00:30:00 | 1.000 | 0.366 | 0.074 | 0.322 |
| 2001-06-10 00:45:00 | 0 | 1.000 | 1.000 | 0.034 |
| 2001-06-10 01:00:00 | 0.364 | 0 | 0 | 0 |

The graph shown in Figure 14 shows how a single normalised dataset (consisting of both announcements and withdrawals) is carried over to the inputs for the autoencoder, in the means described above.

## *4.3 Training the Autoencoder*

Once all datasets have been selected, the network is trained using the normal behaviour. This was found by observation from the entire data period for June to September 2001, avoiding time intervals around the times when worms or major

physical events that affect Internet stability were known to occur. Although more advanced techniques could be used to select this segment, the results proved similar regardless of the segment chosen (refer to section 6.5 for more information).

## 4.4 Tests Performed

The various parameters of the autoencoder were adjusted to determine the system's sensitivity to changes in configuration. Each parameter was varied to determine its individual effect on the system. Once these values were adjusted, the overall results were examined. The following parameters were tested for the neural network:

- The number of inputs was varied between 10 and 160 inputs.
- The number of hidden neurons was varied between 50 and 100.
- The time interval was changed using values of one-, five- and fifteen minute intervals.

The training data is also an important consideration for the neural network. Various training regions have been tested, to find how the choice of training data affects the system.

The primary goal for adjusting these parameters was for the autoencoder output to provide as distinct an indication of novelty as possible. This means that the autoencoder output should be adjusted to either be near-zero (no novelty) or high (indicating significant novelty), providing a clear indication of whether there is an anomaly.

The novelty value for each time interval is only useful as a relative measure of novelty, so the actual value is disregarded. Therefore, a novelty value is only useful when compared to previous values. In all cases the network was trained using the Scaled Conjugate Gradient optimisation technique. For all tests, a hundred training cycles were used, ensuring that the network error was less than $10^{-14}$. As described previously, the typical neural network concern of over-training is avoided when using an autoencoder.

## 4.5 Software

The raw data downloaded from the routers was extracted using the binary-to-ascii conversion provided by the MRT toolkit [42], which is written in the C programming language. It proved to be easier to compile and execute this code in Linux, since this is the language in which the original libraries were written. The code provided by the

MRT toolkit was modified to extract only the relevant BGP Update information, and an additional MySQL database plug-in for C was used to interface to the MySQL database [43] running on Linux.

The data in the database was indexed based on the timestamps for the raw data, due to the large amount of data as well as the need to search based on the various time intervals. Code was written and executed to rearrange the data into the various time intervals from June 2001 to September 2001, and rearranged into the structure required by the neural network.

The database toolkit from Matlab v6.5 [44] was used to extract the various datasets from the database. The Netlab toolkit [45] for Matlab was then used to train the autoencoder, and display the various results shown in the next section.

# 5 Results

This section describes the results obtained from running the various tests using the autoencoder. Based on the information presented in the literature, the important dates for global Internet instability in the selected time period were found and listed. Also, the display of the various sets of data for the *rrc0* router is shown in this chapter, which were used to train the neural network. Several variables for the autoencoder have been adjusted to determine the most effective methods to use, and each subsection shows the results of the change. The final results for the entire dataset are then presented. This is followed by a comparison of the autoencoder to a rule-based system, which is one of the objectives of this research.

## *5.1 Dates of Interest*

Data from the beginning of June 2001 to the end of September 2001 was used. Table 7 shows the dates of interest during this period in terms of Internet stability. The worm outbreak dates are when they were first detected, and there is an expected increase in autoencoder novelty around these dates. From the literature, it was also expected that the novelty during the Baltimore Tunnel Train Wreck and World Trade Centre attacks would increase, but not to the same levels as the that caused by the worms more global and long lasting effects.

The datasets used did not contain some values on 5 July 2001 from approximately 17:15 to 19:09, which were treated as zero for both announcements and withdrawals. There is therefore an expected increase in novelty for this time period. No reported global Internet instability could be found during June 2001.

**Table 7. Dates of interest in terms of global Internet instability from June 2001 to September 2001. These dates include both worms and significant physical occurrences that affected the Internet.**

| Date | Description |
|---|---|
| 13 July 2001 | Code-Red I v1 worm outbreak starts. |
| 18 July 2001 | Baltimore Tunnel Train Wreck |
| 19 July 2001 | Code-Red I v2 worm outbreak starts. |
| 4 August 2001 | Code-Red II worm outbreak starts. |
| 11 September 2001 | World Trade Centre Attacks |
| 18 September 2001 | Nimda worm outbreak starts. |

## 5.2 Extracted Data

The graphs in Figure 15 and Figure 16 show the combined BGP Update messages from the *rrc0* router. Each point represents the number of BGP Update messages (announcements and withdrawals combined) in a fifteen-minute time interval. The graph in Figure 16 shows the same information using a logarithmic axis. This graph is similar to Figure 9 since it is for an overlapping time period and is from the same router. The only difference is that the graph in Figure 9 used thirty-second time intervals per point. All times for the data shown and used in this section are in GMT/UTC time, and are for the RIPE NCC router only.

The graphs for the separate number of BGP announcements and withdrawals that make up the number of BGP Updates are shown in Appendix A.

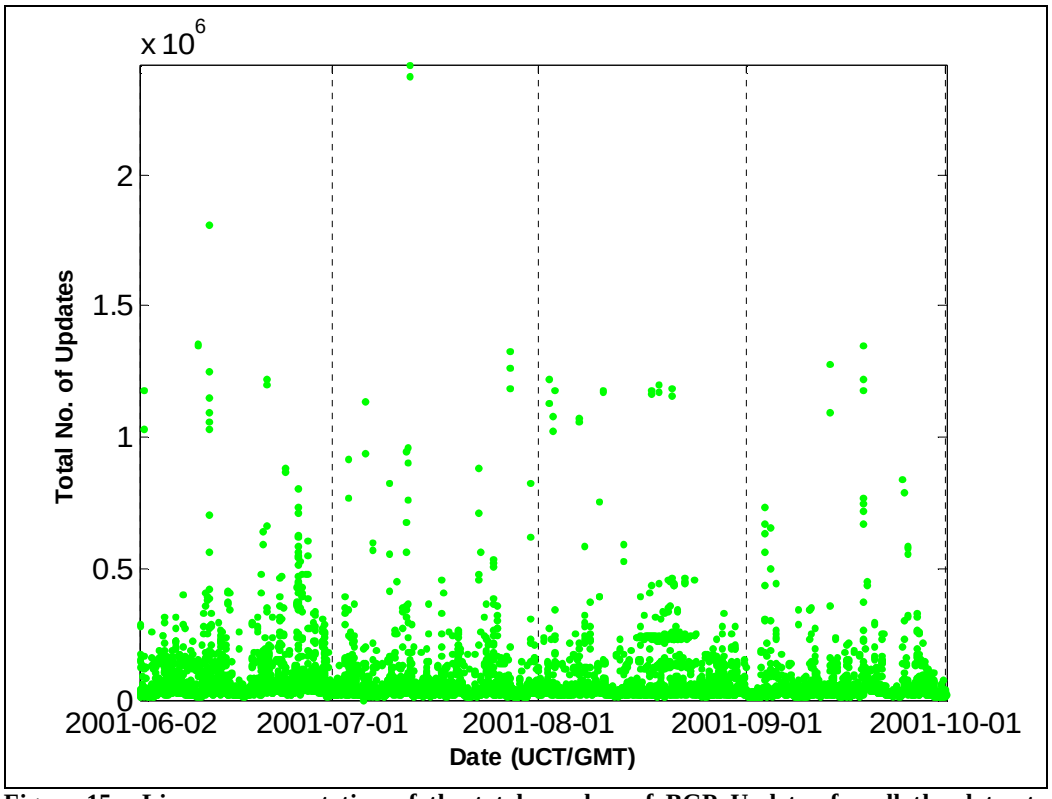


**Figure 15. Linear representation of the total number of BGP Updates for all the datasets between June and September 2001. These values were obtained from the rrc0 router data used for this project. Each point represents the sum of the BGP Announcements and Withdrawals for a fifteen-minute time interval.**

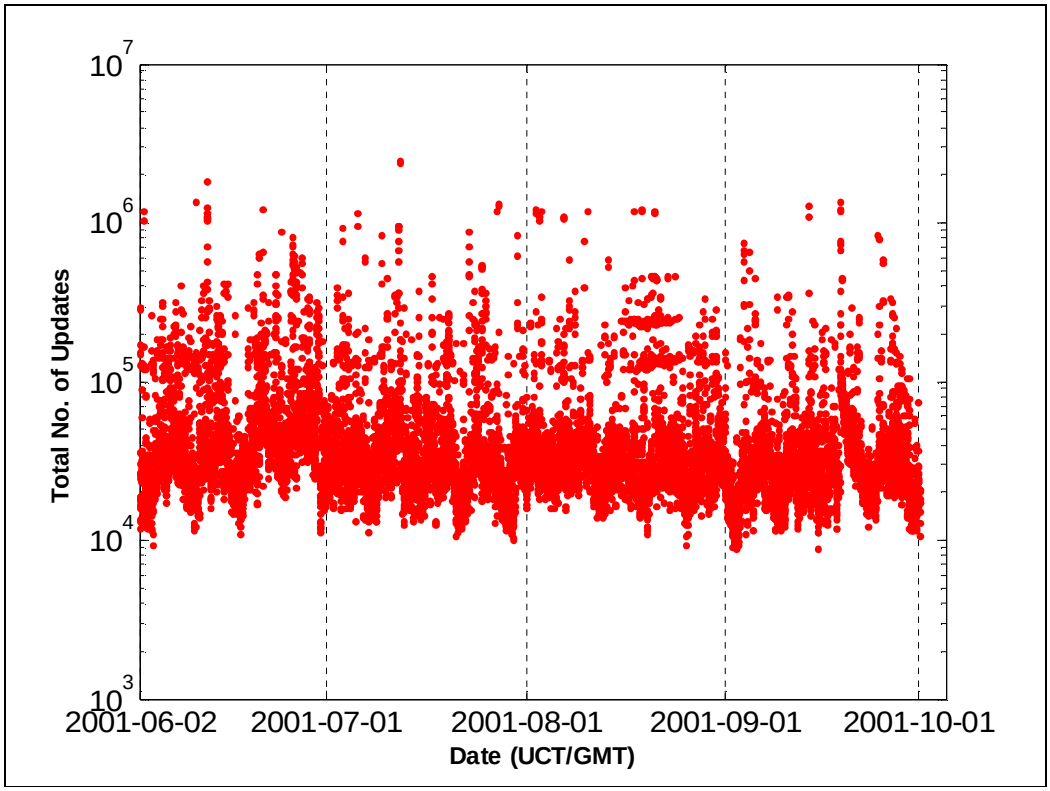


**Figure 16. Logarithmic representation of the total number of BGP Updates for all the datasets between June and September 2001. These values were obtained from the rrc0 router data used for this project.**

The graphs in Figure 17 and Figure 18 are a magnified view of the previous graphs, showing the effect of the Code-Red I v1 worm on this router. The number of BGP announcements reached a peak for the entire dataset from June to September 2001, with a total of 2,331,499 BGP announcement messages between 12:30 and 12:45 GMT on 12 July 2001. This is one day before the Code-Red I v1 worm was documented to have started.

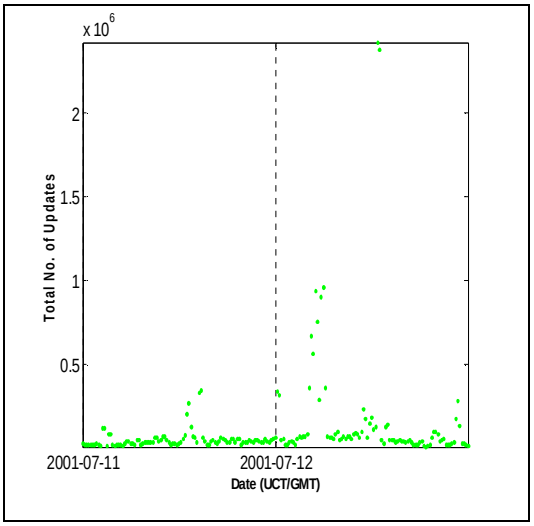


**Figure 17. Linear representation of the total number of BGP Update messages around the time of the Code-Red I v1 worm. The graph shows an increase in messages on 12 July 2001.**

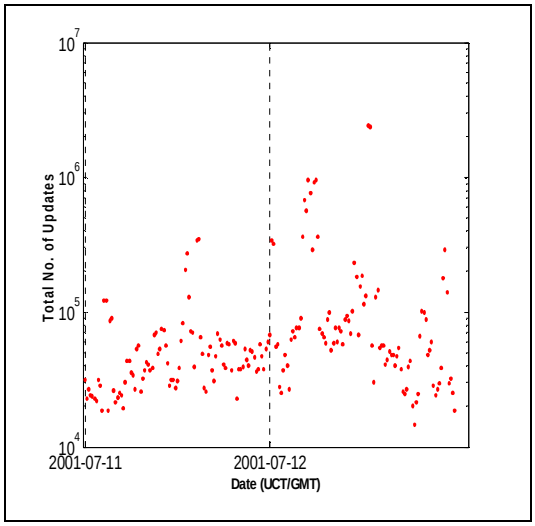


**Figure 18. Logarithmic representation of the total number of BGP Update messages around the time of the Code-Red I v1 worm.**

The increase in BGP update messages caused by the Nimda worm can be seen in both Figure 19 and Figure 20. There is a very sharp increase in the number of messages towards the end of the day on 18 September 2001. Again the graphical values have been grouped into fifteen-minute time intervals. The data peaks at 1,163,145 announcement messages between 20:30 and 20:45 GMT. The shape of the plot in Figure 20 corresponds to the previous data shown in Figure 11, with the only difference being the size of the time interval per point used.

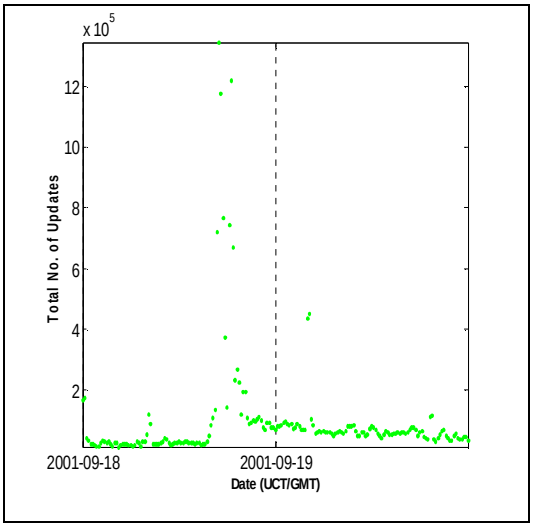


**Figure 19. Linear representation of the total number of BGP Update messages from the extracted dataset during the Nimda worm.**

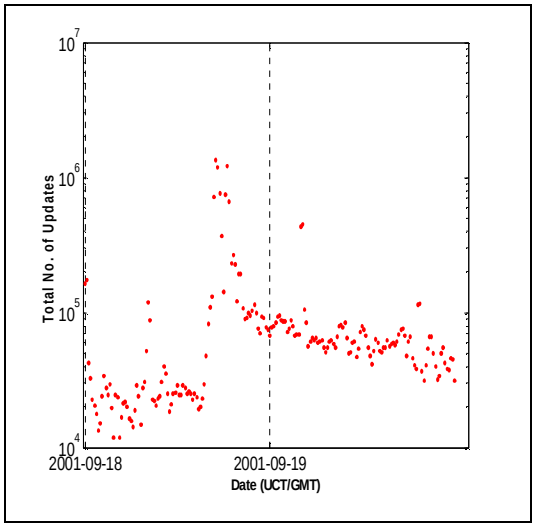


**Figure 20. Logarithmic representation of the total number of BGP Update messages from the extracted dataset during the Nimda worm.**

## *5.3 Test Datasets*

Several sets of data were used to determine how sensitive the autoencoder is to changes in the dataset changes. Based on the dates of interest, the autoencoder was trained during times where there were no reported worms or instances of Internet instability, and for which there was data available. The entire month of June 2001 had no known issues, and provided a sufficiently long time period for testing the autoencoder.

The autoencoder was trained with various combinations of the June 2001 data as training data. The graph in Figure 21 shows the autoencoder novelty output plotted for the entire dataset, when trained using all of the June 2001 data. The other autoencoder parameters are set to 100 inputs, 100 hidden neurons and each point represents the output for a fifteen-minute time period. The graph in Figure 22 shows the output from a similar autoencoder, except that the training data was reduced to only use data between 2-9 June 2001. The other system parameters were left unchanged.

The graphs show very similar results and trends, with no notable changes between the results from the two datasets. The autoencoder that used the whole of June's input took longer to train, since there was approximately four times as much data. When tested with various other training datasets, the system output showed minor changes in trends. These differences did not change the overall spikes of the system in any

discernable way. The training data for 2-9 June 2001 was therefore selected from these results for the remainder of the tests.

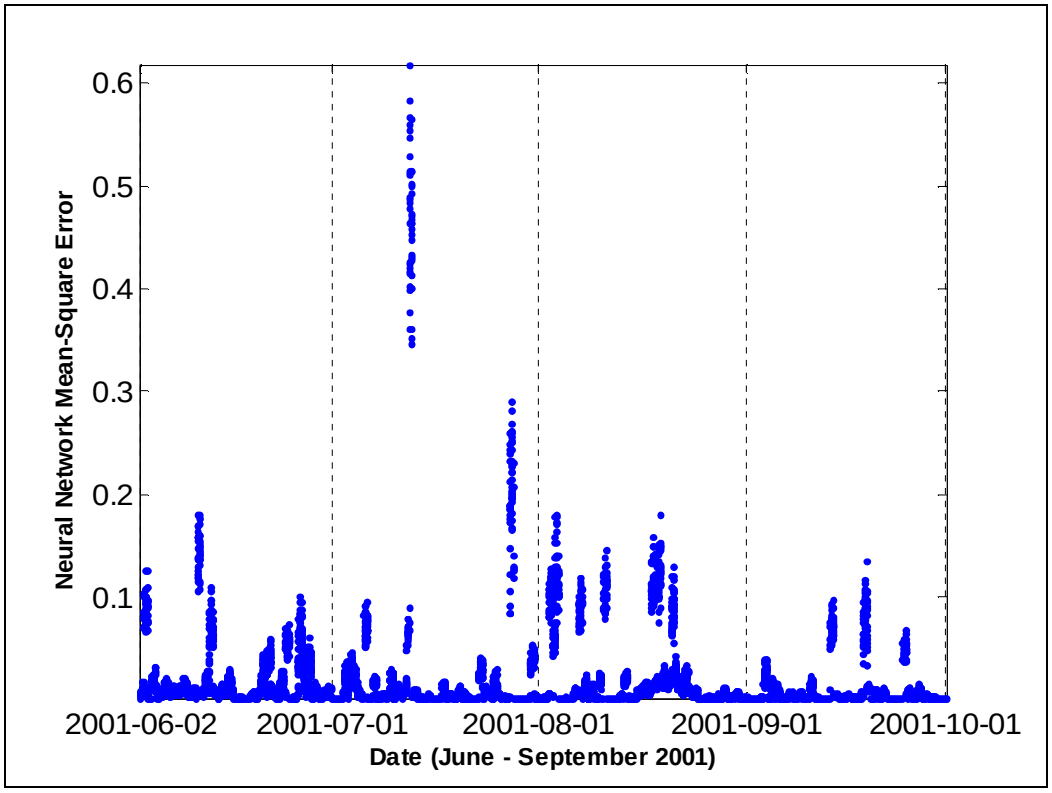


**Figure 21. Autoencoder novelty values when using the entire June 2001 dataset for training. Each point represents the novelty output for a fifteen-minute time interval.**

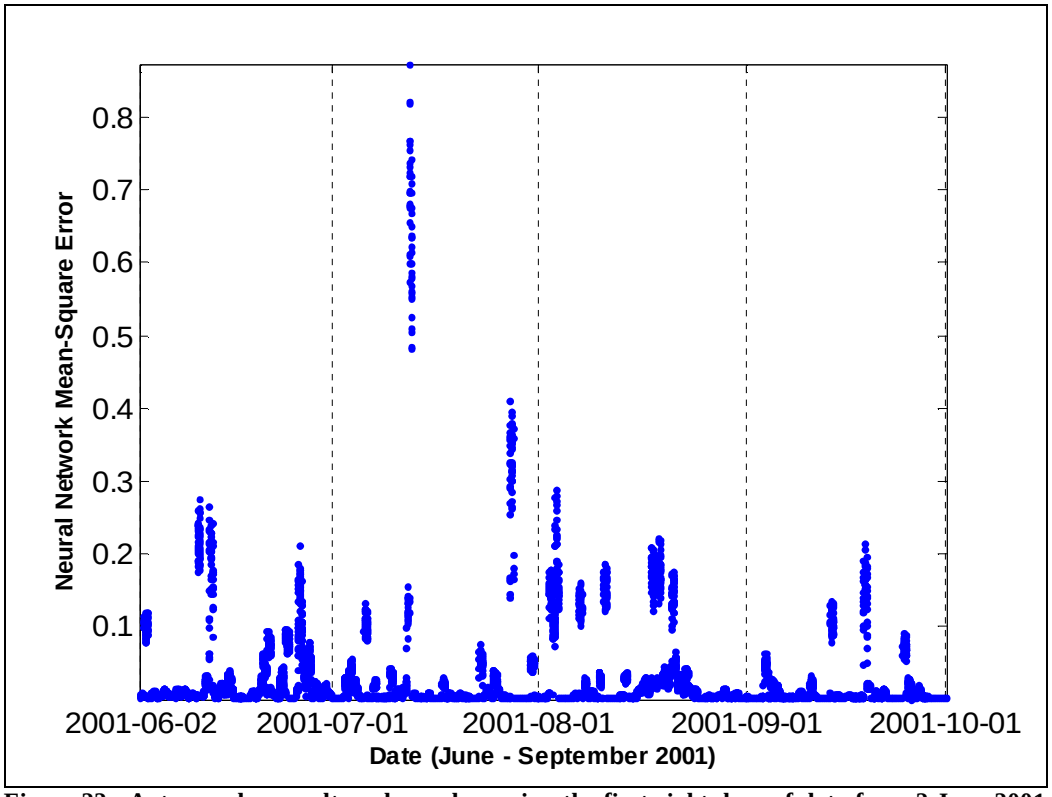


**Figure 22. Autoencoder novelty values when using the first eight days of data from 2 June 2001 to 9 June 2001 for training as the training dataset. This graph shows similar trends and spikes to the previous graph.**

## *5.4 Number of Input-Output Pairs*

The number of input-output pairs was varied to determine the optimal amount of historical data to provide to the system. By increasing the number of inputs, more data from previous time intervals are introduced to the system. It was found that by increasing the number of inputs, and therefore increasing the length of time that a single time interval will be used in the set of inputs, the novelty values continued to be affected in subsequent time periods. An indication of this effect was evident when a high number of announcements or withdrawals occurred. The raised values caused an increase in novelty as soon as large values were included in the dataset, and the novelty remained high for as long as these values were included in the subsequent datasets.

An example of this effect was during the Code-Red I v1 worm, which had a very large novelty value when the high normalised input was included, as well as for the subsequent time intervals for which it was included. The graphs in Figure 23 and Figure 24 show the effect of the number inputs using extreme values of 10 and 160

inputs respectively, at the time at which Code-Red I v1 appeared. The large spike from the former graph quickly decreases in size after a few intervals when the high input is no longer included in the dataset. The latter graph has a much longer increase in novelty, since this high input remains present in more of the subsequent datasets. It is evident that the raised number of historical inputs used lengthens the time that the novelty values are increased.

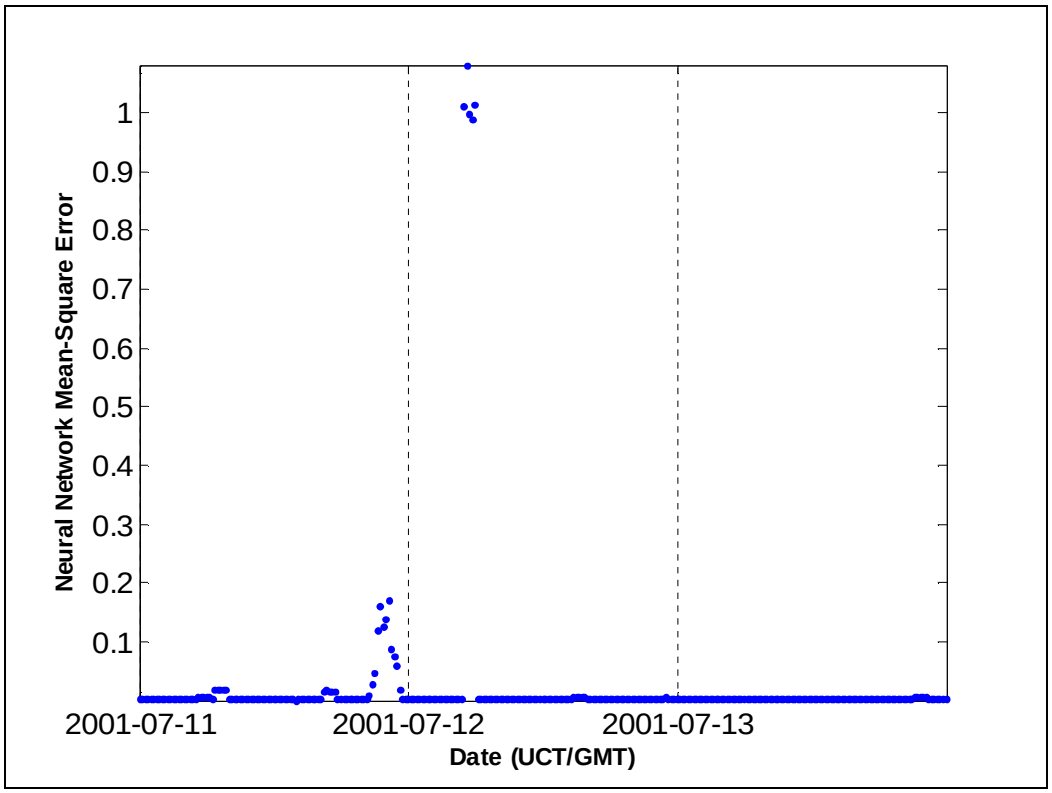


**Figure 23. Novelty output for Code-Red I v1 using only ten inputs (five for BGP announcements and five for BGP withdrawals). The graph shows how the low number of inputs decreases the number of time intervals that the system is affected by a single high input.**

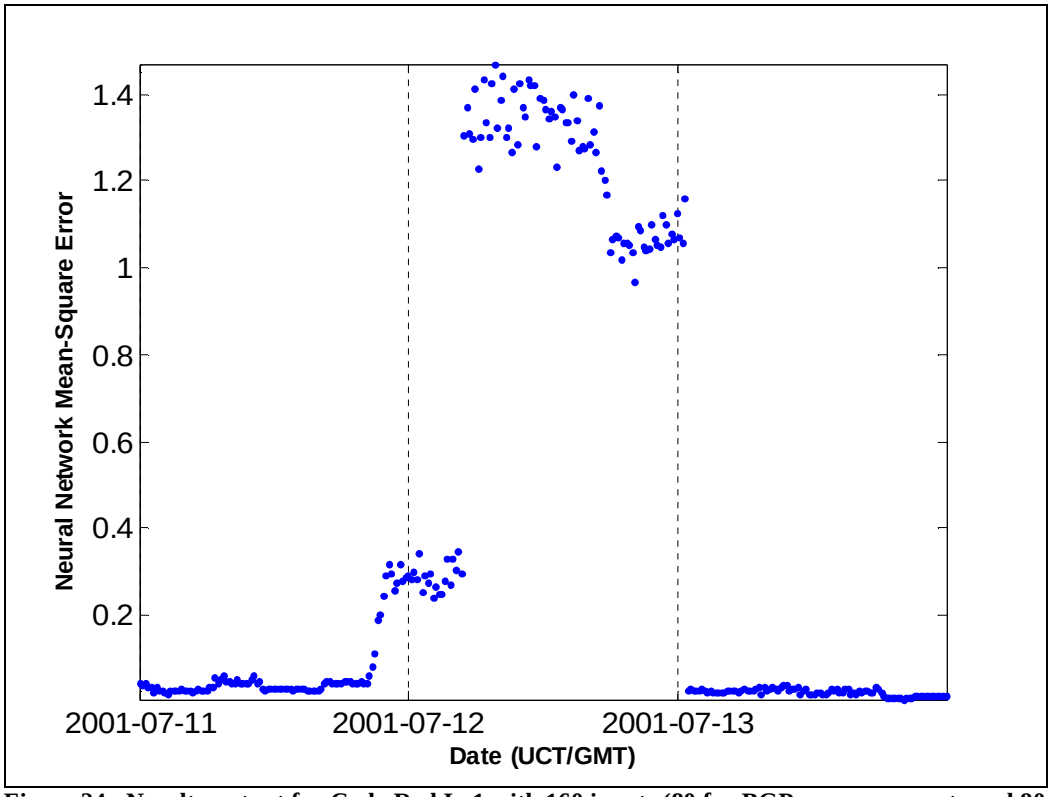


**Figure 24. Novelty output for Code-Red I v1 with 160 inputs (80 for BGP announcements and 80 for BGP withdrawals). The trend on this graph shows the opposite effect to the previous graph, which is for the same time period. A single high input now affects the novelty for a longer period of time due to the lengthened time that the high value is present in subsequent datasets.**

It is also evident from these figures that the number of inputs also affects the system's ability to learn from historical data. When very few inputs are used, the system is only able to respond to these inputs. In the extreme case with only one announcement and withdrawal input, the behaviour becomes similar to a rule-based system. But the trade-off is that by increasing the number of inputs, it takes longer to train and execute, and would require more computation for a real system.

There was a concern that by increasing the number of inputs, the effect of a single high value in the dataset would be dampened. If this were the case, then the autoencoder would respond slower to high inputs, and make the system less effective. This is discussed in section 5.8, which shows that this is not the case.

An optimal value of fifty announcements and withdrawal inputs was selected as a sufficient compromise between the training speed and the need for using the historical data. The other system parameters used were left unchanged from the previous section.

## 5.5 Hidden Neurons

In the literature, previous research has recommended using fewer hidden neurons than inputs. This ensures that the system adequately compresses the data, and does not become too closely linked to the training data. The autoencoder was therefore trained using 50, 75 and 100 hidden neurons. The graphs for 50 and 75 neurons are shown in Figure 25 and Figure 26 respectively. The graph for 100 neurons is shown previously in Figure 22.

As with the choice of training data, the output was not markedly affected by different numbers of hidden neurons. It was seen that reducing the hidden neurons caused slightly more erratic behaviour when a high input was presented. For this reason, the value was selected to be the upper limit of using the same number of hidden neurons as inputs (in this case 100 hidden neurons). The disadvantage is that more neurons require a longer time to train and run the autoencoder.

The other system parameters were left unchanged from the previous section.

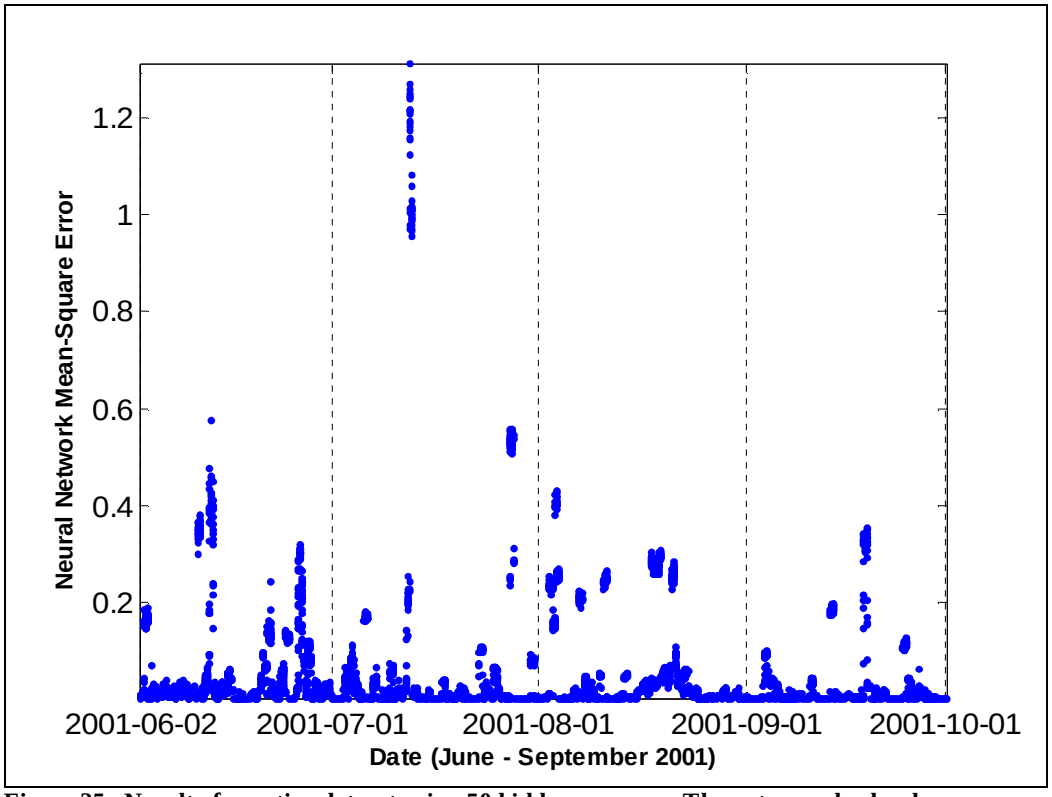


**Figure 25. Novelty for entire dataset using 50 hidden neurons. The autoencoder has less memory when the number of hidden neurons is low, and so tends to respond more erratically when high input values are used.**

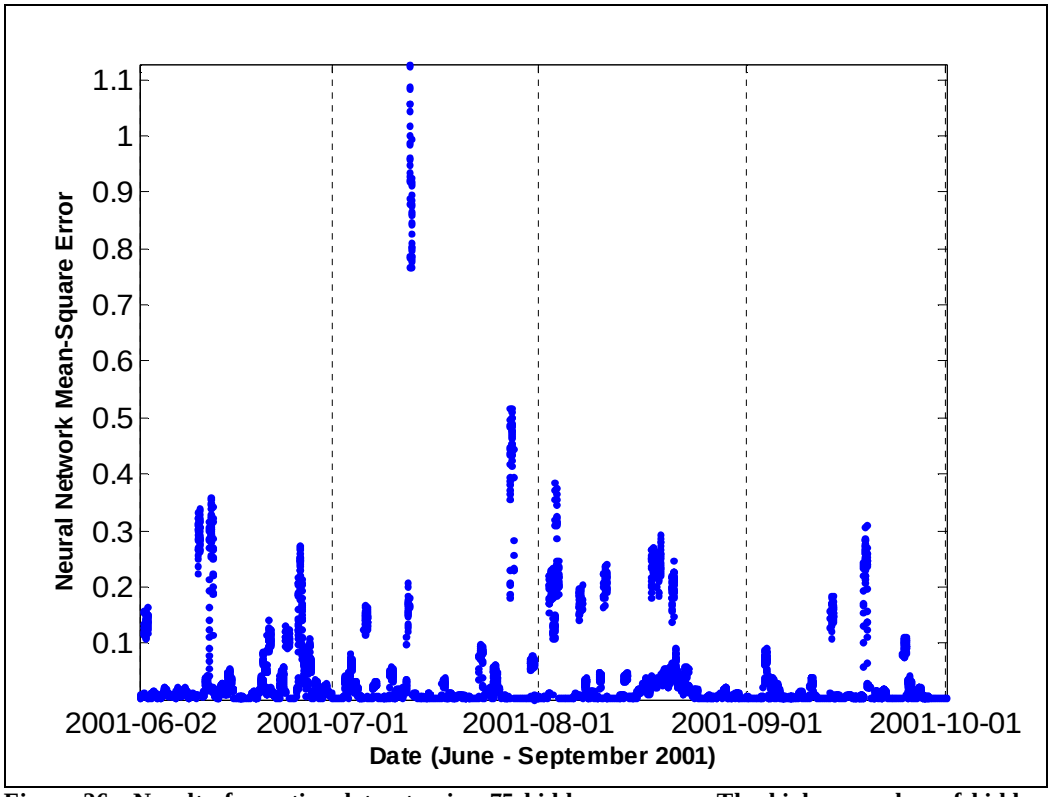


**Figure 26. Novelty for entire dataset using 75 hidden neurons. The higher number of hidden neurons adds more stability to the output. This is evident from the lower novelty values tending to be relatively closer to zero compared to the previous graph. This effect is marginal, and for the ranges tested does not fundamentally change the output trend.**

## *5.6 Time Interval Size*

The selection of the time interval size to group the number of BGP announcements and withdrawals used is the most important factor in detecting a worm earlier on. The system is only able to give a novelty once the number of BGP Updates for the current time period is known. A system configured using one-minute time intervals is able to detect a worm fourteen minutes ahead of a system using fifteen-minute intervals. The disadvantage of using shorter time periods is that the number of datasets increases, as well as the time taken to train the system. The computer memory requirements also increase dramatically, since fifteen times more memory is required in this example for the training and full datasets. The time to load, train and display the results was much longer for the one-minute time interval than for the longer time intervals.

The graphs in Figure 27 and Figure 28 show the results for one- and five-minute time intervals. The representation of the fifteen-minute time interval is shown in Figure 22. The configuration parameters for the neural network have been left unchanged, using the values selected from the previous sections. It is clear from the

shorter time interval graphs that the novelty spikes for anomalies are relatively larger than the spikes for the longer time intervals. This is beneficial, since it allows for a clearer indication of a problem.

From these results, it is evident that novelty spikes for the fifteen-minute time interval graph are also reflected in the shorter five- and one-minute time intervals. No information is lost when reducing the time intervals. For these reasons, the one-minute time period was used for the rest of the analysis.

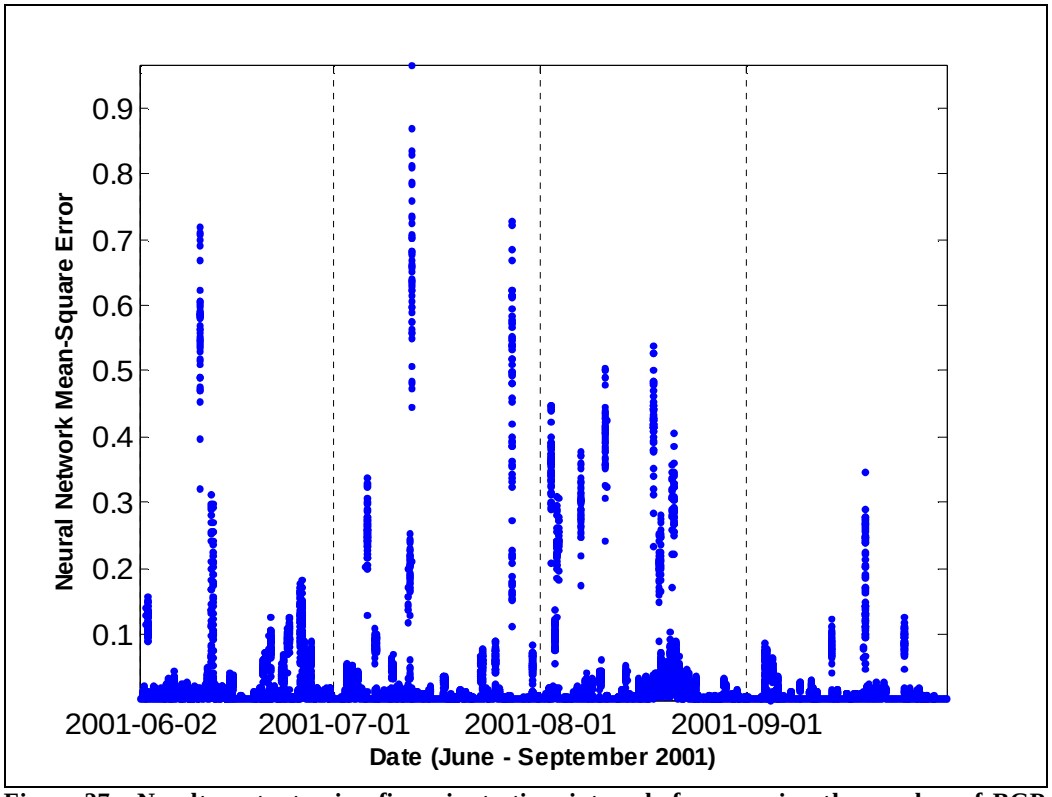


**Figure 27. Novelty output using five-minute time intervals for grouping the number of BGP announcements and withdrawals. The neural network was configured with 100 input-output pairs and 100 hidden neurons. The peaks are more pronounced than for the fifteen-minute time interval neural networks.**

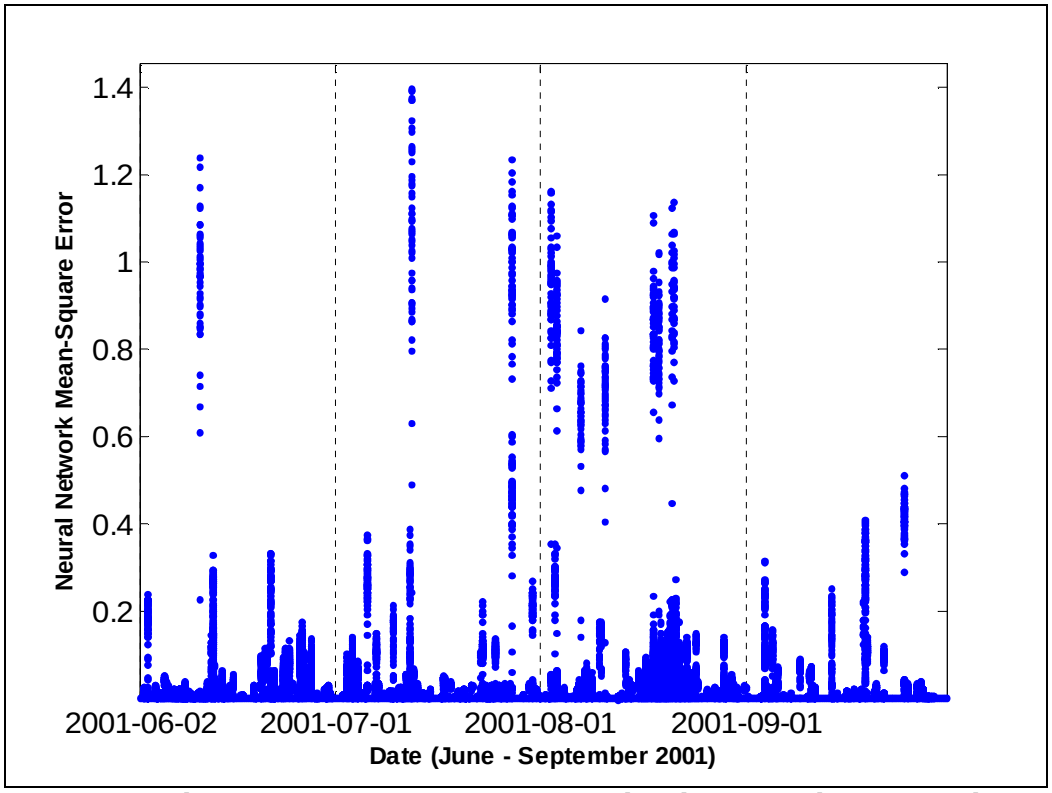


**Figure 28. Novelty output using one-minute time intervals. The autoencoder was trained using 100 input-output pairs. The novelty spikes are more evident than for the five- and fifteen-minute time intervals.**

## *5.7 Final Results*

The previous sections demonstrated the selection of each neural network parameter to provide the best indication of novelty. From these results, a final trained network was configured, using the following parameters:

- 100 input-output pairs (50 BGP announcements and 50 BGP withdrawals)
- 100 hidden neurons
- 1 minute time intervals

The resultant autoencoder novelty for the entire time period between June and September 2001 is shown in Figure 29. Each of the significant events is shown on the graph, and is discussed in section 6.1. These events show both local and global anomalies. This graph represents the effectiveness of the method proposed in this study for determining the presence of global instability, especially in the presence of worms. It also forms the basis of most of the discussion in the following chapter.

A horizontal threshold line has been added to the graph. Any novelty values above this threshold indicate a warning that there may be global Internet instability caused

by a worm. The height of the threshold has been chosen to include some of the highest peaks, specifically that of the Nimda worm. The choice of this threshold is also discussed in the next chapter.

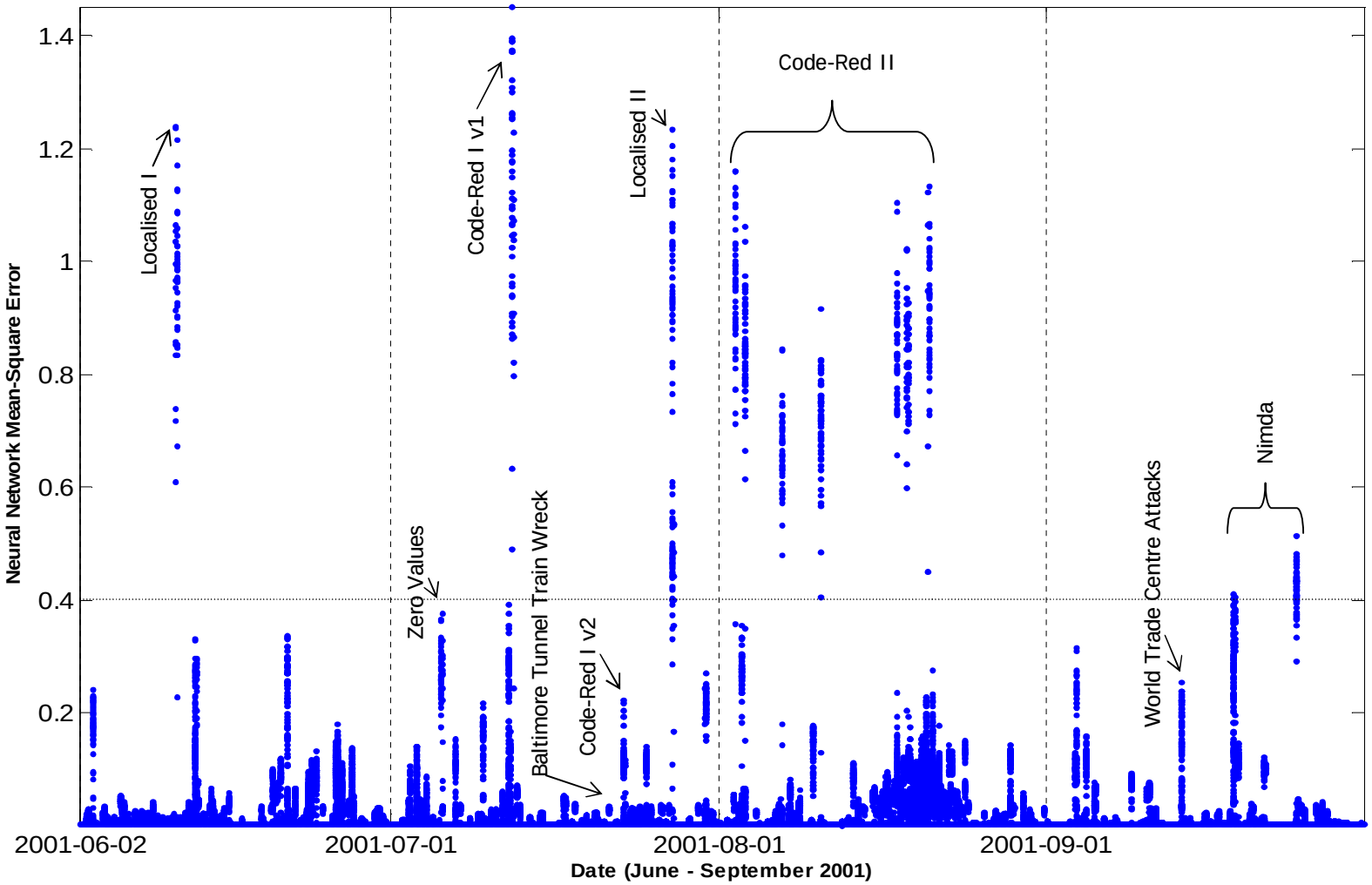


**Figure 29. Novelty values from the final trained autoencoder for the entire dataset from June to September 2001. The network has been trained using 100 input-output pairs, 100 hidden neurons and the time interval for each point is one minute. A horizontal novelty threshold has been manually inserted.**

## 5.8 Rule-Based Comparison

A rule-based prediction system is a simpler alternative to the autoencoder proposed in this project. Instead of using a neural network, the rule-based system would directly use the BGP Update data to provide warnings of instability. This section shows the results of a comparison of the two possible prediction methods. A basic rule-based system is proposed, which generates warnings when the total number of BGP Updates exceeds a pre-specified threshold.

The following table lists the highest number of BGP Updates, which would most likely be the cause of an alarm being triggered for a rule-based system. The data is the number of BGP Updates, taken from the same source in the database as for the autoencoder. The highest available granularity of one-minute time intervals was used. The values represent the fifteen highest BGP Updates for the entire dataset from June to September 2001, and are ordered from highest to lowest number of BGP Updates. The *Event* column indicates which physical or worm event most likely caused the peak (refer to section 5.1 for the full list of significant events). Any localised router events seen in the autoencoder are also correlated.

Similar to the autoencoder, the rule-based system would also require a threshold for the number of BGP Updates that would trigger an alarm, but since only fifteen values have been listed in the table, each of these values would most likely be above the threshold and cause an alarm to be triggered. It is interesting to note that the table does not include an event for the Nimda worm, which had a peak of 293,079 messages at 2001-09-18 18:01. As explained in the literature, significant damage was caused by Nimda, and was therefore expected to feature in the table. In order for this rule-based system to have generated a warning for Nimda, the threshold would have had to be set to include the top 25 peaks. By comparison, the autoencoder shows only twelve groups of data that exceed the threshold for the entire dataset (seven of which are caused by the extended effects of the Code-Red II worm).

From the table it can be seen that this rule-based system would be able to detect instability caused by global effects such as worms, as well as localised problems that were also shown by the autoencoder. But, two of the highest values were not caused by any known event, and would trigger false-alarms.

**Table 8. Highest number of BGP Updates for the entire dataset from June to September 2001. The time buckets are grouped into one-minute intervals, and the significant events noted.**

| No. | Time (1min interval) | Number of BGP Updates | Event |
|---|---|---|---|
| 1 | 2001-07-27 14:50:00 | 595001 | Localised II (also seen with autoencoder). |
| 2 | 2001-08-02 13:30:00 | 592458 | Code-Red II |
| 3 | 2001-08-17 20:57:00 | 572124 | Code-Red II (extended effect) |
| 4 | 2001-08-18 20:39:00 | 556038 | Code-Red II (extended effect) |
| 5 | 2001-07-12 12:42:00 | 541756 | Code-Red I v1 |
| 6 | 2001-08-03 11:24:00 | 534271 | Code-Red II |
| 7 | 2001-08-20 20:44:00 | 526423 | Code-Red II (extended effect) |
| 8 | 2001-06-10 17:35:00 | 504463 | Localised I (also seen with autoencoder). |
| 9 | 2001-08-06 23:03:00 | 499349 | Code-Red II |
| 10 | 2001-06-10 17:36:00 | 486930 | Localised I |
| 11 | 2001-07-12 12:39:00 | 475865 | Code-Red I v1 |
| 12 | 2001-07-12 12:38:00 | 453161 | Code-Red I v1 |
| 13 | 2001-08-10 16:32:00 | 436326 | Code-Red II (extended effect) |
| 14 | 2001-08-10 16:33:00 | 432627 | Code-Red II (extended effect) |
| 15 | 2001-08-20 20:43:00 | 418252 | Code-Red II (extended effect) |

The next two graphs show the comparison of how quickly the rule-based and autoencoder systems would trigger an alarm due to the Code-Red I v1 and Nimda worms. The graph in Figure 30 shows the autoencoder novelty around the time of the Code-Red I v1 worm. The grid-line indicates the time that the rule-based system would trigger an alarm, based on the peak number of BGP Updates shown in the previous table. This grid-line corresponds to the time 2001-07-12 12:38, which was ranked $12^{th}$ in the table. The $5^{th}$ and $11^{th}$ values occurred in the following two minutes, and would also generate alarms, but this was the earliest time that the rule-based system would have triggered an alarm. It is evident from the graph that the autoencoder would trigger an alarm at roughly the same time as the rule-based system.

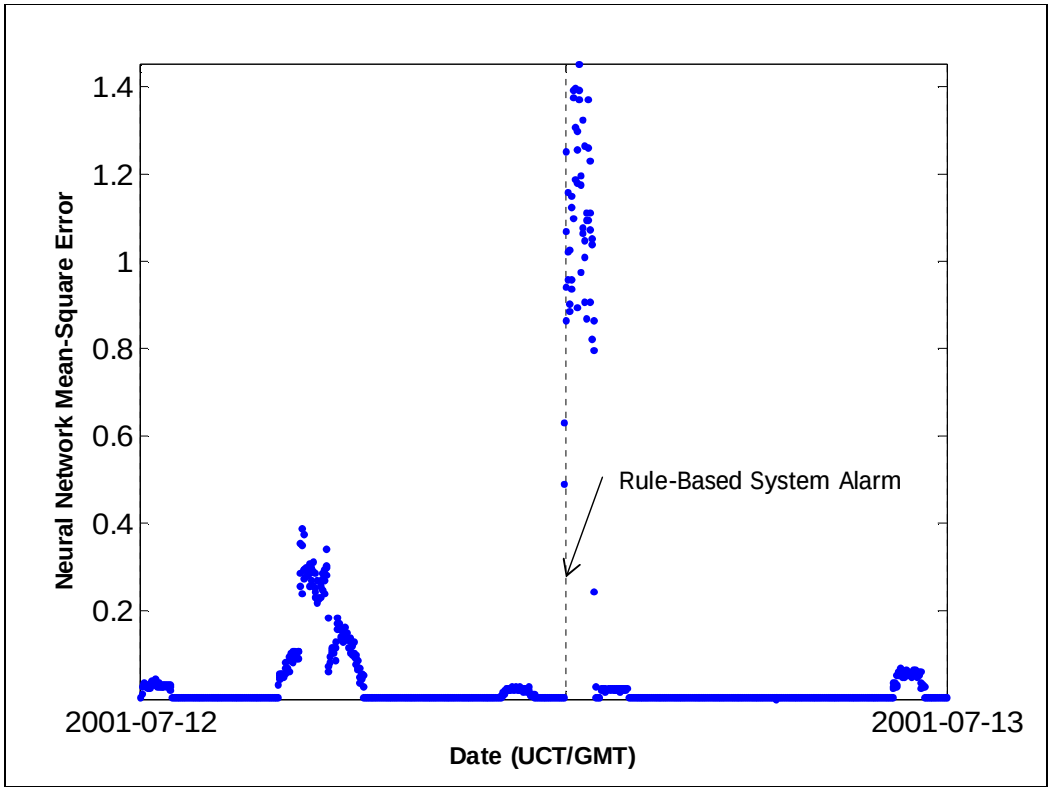


**Figure 30. Comparison between autoencoder and rule-based system for Code-Red I v1. The graph shows the autoencoder output, and the grid line indicates the point at which the rule-based system would trigger an alarm.**

The graph in Figure 31 shows the same type of information for the period the Nimda worm occurred. The grid-line shows when the rule-based system would trigger an alarm based on the highest number of BGP Updates at this time (corresponding to the 25$^{th}$ highest number of BGP Updates). It is interesting to note that in this case the autoencoder is able to generate an alarm more than an hour earlier than the rule-based system.

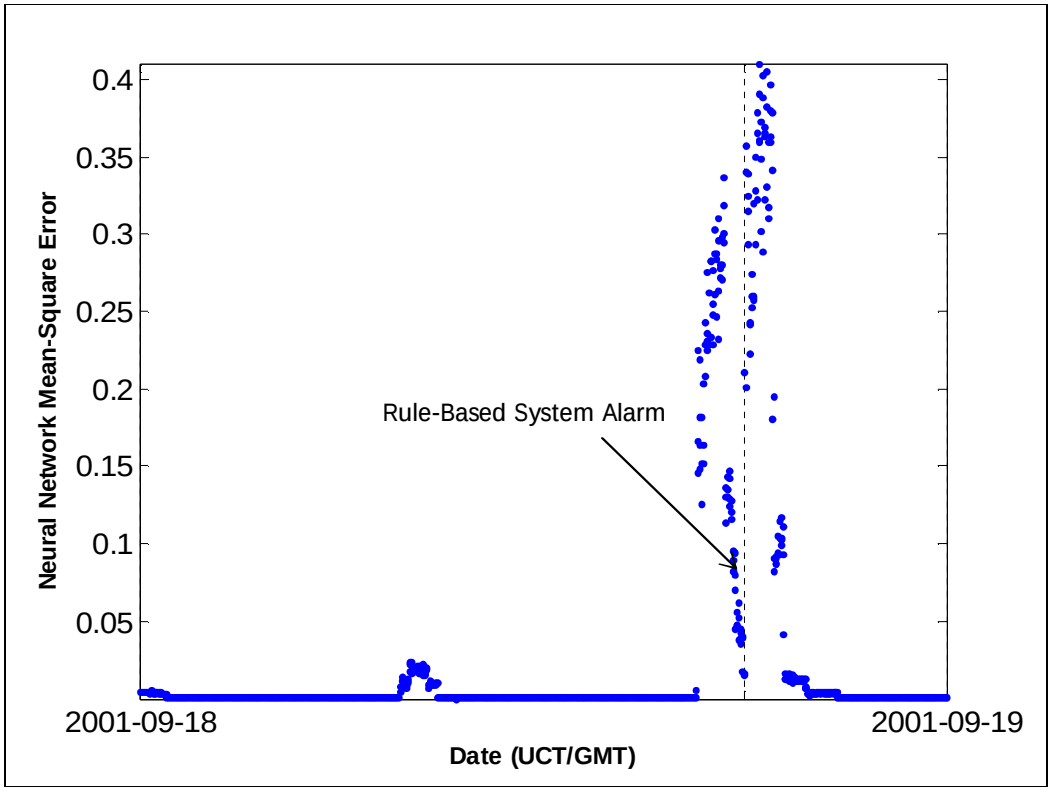


**Figure 31. Comparison between autoencoder and rule-based system for Nimda. The graph shows the autoencoder output, and the grid line indicates the point at which the rule-based system would trigger an alarm.**

# 6 Discussion

The following sections review the results from the previous chapter. Several of the key points and observations are discussed, followed by an analysis of the objectives of the project.

## *6.1 Output Interpretation*

This section discusses the final results obtained in section 5.7. The novelty values from Figure 29 shows several spikes that exceed the threshold. Each of these spikes as well as the dates of interest are discussed below:

- The first spike, *Localised I,* occurs during June 2001, which is not known to contain any global Internet instability. The data contains higher normalised values for the number of announcements and withdrawals at this time (reaching a peak of 504,463 BGP Updates in a one-minute time interval), which caused the high novelty values. The origin of the higher number of BGP Updates is suspected to be due to a localised problem. This is most likely due to an incorrectly configured router that is within the neighbourhood of the *rrc0* router. Another possibility is that *BGP* route flap damping has not been deployed on several neighbouring routers, and a local problem on one of the routers has caused route flapping in this localised area of the Internet.
- The second spike, *Zero Values,* is caused by the missing BGP Update data on 5 July 2001. The novelty value was expected to increase because of the zero number of announcements and withdrawals at this time, as mentioned in section 5.2. This spike does not reach the threshold, and would not raise an alarm.
- The third spike is caused by *Code-Red I v1,* which had a marked effect on global Internet instability, and was expected to cause a high novelty value. This had the highest number of announcement messages within the dataset, as well as the highest novelty caused by this. The spike clearly indicates a potential worm, and would generate an alarm.
- The *Baltimore Train Tunnel Wreck* showed a very small increase in novelty from the autoencoder. The novelty was far below the threshold which was not unexpected. This was a localised problem in another part of the Internet

that caused a brief burst of BGP Updates, after which the routing converged. The BGP protocol is able deal with this type of route change, which does not cause BGP route flapping or extended periods of high load.

- The *Code-Red I v2* worm which occurred on 19 July 2001 showed a small increase in novelty, caused by a slight increase in the number of BGP announcements and withdrawals. The value was far below the threshold, and would not be considered a threat to stability. From the literature, it was found that this worm was almost programmatically identical to the Code-Red I v1 worm, with one of the only difference being an improvement to the list of IP addresses used to attack other hosts. The worm was programmatically designed to stop spreading on the 20th day of the month (approximately a day after the worm first appeared), which seems to have severely limited the impact of the worm. The low novelty during this time is therefore not surprising.
- The next spike, *Localised II*, is again a high novelty value during a time when no large worms are known to occur. This is again suspected to be due to local instability in the neighbourhood of the *rrc0* router.
- The next spike occurred at the beginning of August 2001, and was caused by the appearance of *Code-Red II*. This worm had a more devastating effect than Code-Red I in terms of economic impact, but had a slightly lower novelty spike for the autoencoder. The peak is still well above the specified threshold, and can be regarded as a threat to stability. The novelty values throughout August remain high and this series of spike are most likely due to the long-lasting effects of the worm. The literature also noted that the two strains of Code-Red I were also still active and causing instability. As discussed in the section 2.3.2.1, these worms actively spread between the 1st and 20th of the month. The research also indicated that 32% of Code-Red I infected hosts had not been patched by the 1st of August, when the worm restarted its propagation phase. It therefore seems reasonable for the various Code-Red I strains to have caused some of these spikes, since the last of the spikes during August 2001 ended on about the 20th day of the month.
- The spike caused by the *World Trade Centre Attacks* on September 11th 2001 does not meet the threshold. Similar to the Baltimore Tunnel Train

Wreck, this spike is due to the sudden increase in BGP Updates which allows the network to readjust and find alternative routes. Once this has stabilised, the number of BGP routing messages returns to normal.

- The last two spikes are caused by the *Nimda* worm during September 2001. The peak is surprisingly low, considering the devastating effect that Nimda had on the Internet as a whole. But the spike is still considerable and would trigger an alarm.

From this discussion, it was shown that an autoencoder trained with the BGP Update data is able to correctly identify the presence of worms. The suspected localised effects are also detected by the autoencoder, which would also generate alarms. This may be useful for detecting any sort of instability, but would be a false alarm if the system were only used to detect worms.

## 6.2 BGP Behaviour

This section discusses the behaviour of BGP in the presence of a worm compared with localised outages such as the World Trade Centre Attacks.

The prolonged increase in load caused by worms' causes high latency times between routers. If the BGP Keep-Alive messages sent between routers are not received timeously because of these delays or router failures, the BGP sessions between the routers are closed. The routers then inform the surrounding routers of the change in routers using BGP announcements. Once routers are able to communicate, they attempt to re-establish sessions, which causes BGP withdrawal messages to be sent. This effect continues until human countermeasures are put into place, or until the worms stop spreading because of their programmed behaviour. The high number of BGP messages was evident from the spikes seen in the results.

The localised effects seen from the analysis differ in behaviour to the worms, but both caused increases in the autoencoder output. The effects of *Localised I* and *Localised II* are assumed to be caused by routing problems in the vicinity of the *rrc0* router. The novelty values are large during these times, and comparable to the effect of worms. This can be accounted for by the global Internet instability causing localised instability throughout the Internet. This implies that the autoencoder treats global and localised instabilities in the same way.

The effect of localised instabilities on the number of BGP Updates is less pronounced when originating from disparate areas of the Internet. This was evident in

the results for the spikes caused by the World Trade Centre Attacks and the Baltimore Tunnel Train Wreck, neither of which would have caused an alarm. The routers in the neighbourhood of these occurrences would have showed considerably higher novelty values than were seen at the *rrc0* router.

The localised problems do increase the number of BGP updates, but the high level of redundancy for Internet routes allows autonomous systems to connect via many routes. These events don't cause overall Internet instability, and any prolonged outage caused will likely be at a few endpoints which have had their only available routes disconnected.

The results also showed several cases of an increase in the number of BGP Updates prior to the time of the worm being reported in the literature. This was evident from Code-Red I, which was only documented to have started on the following day. The autoencoder correspondingly showed a peak earlier than documented, showing that the BGP impact occurred earlier than the instability noticed by humans. This further justifies the use of BGP information to detect instability.

## 6.3 Single Router Data

It has been determined that the global Internet instability caused by worms can be detected by monitoring a single router. This is important since the creation of an advanced warning system would require live data to be processed. If the data were not received timeously then the system would be less effective, since it would have to wait for the data before calculating the novelty. If this type of system is installed at a single router, there would be no problem with gathering a single set of data. If more than one router were required, it would be necessary to collate the data from these routers. This would involve transmitting data from scattered Internet routers to a central place, with the transmission medium most likely to be the Internet itself. Since the instability caused by the worm could delay or stop the reception of these messages, a system requiring more than one router could prove ineffective at the time that it is most required to detect a worm. Even in the event that the data could be transmitted within a guaranteed time, the system would still be required to wait for values from all the routers.

The results have shown that any localised problems at the router also produce spikes at the output of the autoencoder, which cannot be distinguished from novelty spikes caused by worms. This effect would mostly be resolved by using data from routers in

disparate areas of the Internet. Localised effects in one area will not markedly affect a separate area of the Internet, whereas the increase caused by the worm would be global across all routers. By combining this data, it would be apparent which spikes are caused by global instability, and which are local.

## 6.4 Threshold

In order to determine if a novelty spike is a threat, it is necessary to set a threshold. Any spikes above this threshold should indicate a potential problem, while values below this can be ignored. Methods for setting a threshold were beyond the scope of this study. The threshold for Figure 29 was chosen based on the selection of a value which would include only a few spikes across the entire data range. A method for setting this threshold could be done in several ways, such as manual adjustment based on previous data (similar to that used here), or using a trained learning algorithm. Further investigations into threshold assignments are required to resolve this issue.

## 6.5 Autoencoder Effectiveness

The autoencoder was not particularly sensitive to changes in the various system parameters, such as the number of hidden units and input-output pairs. The implication of this is that the system responds in a similar manner for a wide range of parameters, making the choice of parameters a straightforward matter that is unlikely to affect the systems overall performance. The selection of the training region had a more pronounced impact, but the novelty outputs were high for the significant events even when the autoencoder had been trained to include the events. This demonstrates the autoencoder's ability to perform PCA as well as data compression. This level of tolerance would be very useful in a real-world system, since the system could be trained easily and be less dependant on tweaking.

The choice of a training region is common across all types of neural networks. In the case of a neural network classifier, it would be necessary to break the data into "normal" and "worm" categories. The autoencoder is instead able to intrinsically determine novel data.

The autoencoder required very little information about the system, from which it was able to provide reasonably accurate results. That the system was able to identify the worms is interesting, since none of the training data included any data about worms. This demonstrates that the system was not biased toward any type of worm, and should be able to respond to new types of worms in a similar manner. This is

very important in the case of worms, since their means of propagation and infection are vastly different and continuously changing.

Another advantage of the autoencoder is that it was able to respond to a high number of BGP announcements or withdrawals quickly. This was shown in the comparison to the rule-based system in the results (see section 5.8), where the autoencoder provided a high novelty at the same time or sooner than the peak number of announcements. This demonstrates the autoencoder's ability to detect underlying trends in the data.

## *6.6 Meeting of Objectives*

The following subsections discuss the results with respect to each of the objectives for this project.

### 6.6.1 Autoencoder to provide a measure of instability

**Objective:** *To use the novelty detection of a trained autoencoder to provide an indication of the possibility of Internet instability, specifically that of worms.*

The autoencoder was shown to be able to detect instability, both localised and global. The Internet-wide instability was seen to be caused by worms. Events such as the World Trade Centre attacks did not feature to the same degree as for worms within the studied time period. The autoencoder required no worm data for training, ensuring that it would be able to detect new and different types of worms that cause Internet instability. The autoencoder was easy to train, and was able to determine the underlying system parameters from the data. It was also resilient to changes in the number of inputs, hidden neurons and the training data used, providing a high level of confidence in the novelty output values, without much concern that the configuration may be incorrect.

### 6.6.2 Novelty Detection vs. Rule-Based System

**Objective:** *To determine if novelty detection using an autoencoder is able to predict Internet-wide instability sooner than a rule-based system.*

It was shown that a simple rule-based system is able to detect many of the worms and localised events that the autoencoder was able to detect. Both the rule-based and autoencoder systems required a threshold to be set, which affected the number of alarms generated by each system. The autoencoder was shown to generate fewer

alarms for a similarly tested threshold, while still detecting the most important anomalies.

The autoencoder is able to use both current and previous data, from which it can determine novelty using trends in the data. The rule-based system is always reliant on the current data point, making it more susceptible to outlying points. In the cases of the Code-Red I v1 and Nimda worms, the autoencoder was able to respond either as fast as or faster than the simple rule-based system.

From the results, it can be seen that the trained autoencoder performs better than the simple rule-based system. A more accurate rule-based system will likely require more configuration parameters to be effective. A rule-based system does not have the ability to learn as in the case of the autoencoder, and would instead require that a network administrator adjust the system's behaviour using complex parameters. Although the autoencoder requires more computational power than a rule-based system, much of the complexity could be hidden from the administrator of such a system.

### 6.6.3 BGP Update Messages as Only Input

**Objective:** *To determine if using the BGP Update messages alone as input to the autoencoder is sufficient to predict instability.*

Although the autoencoder was able to determine a lot of the underlying information about the data, additional information could provide better results. The exact parameters that should be used can be debated, but additional routing information is already available in the data that was downloaded for this project. Non-routing information, such as a measure of the traffic load at any point would potentially be useful. This data is not as easily available, especially when analysing data from previous years and routers, as in the case for this project.

### 6.6.4 Single Router

**Objective:** *To determine if a single router can effectively represent Internet-wide instability.*

It was seen that a single router is able to show global Internet instability, but also includes any localised routing problems as well. These localised problems cause novelty spikes from the autoencoder, and cannot be distinguished from global instability. As mentioned earlier in this chapter, collating the data from multiple

disparate routers may solve this problem, since only global instability would be evident from the results of multiple routers.

# 7 Conclusion

Worms cause billions of dollars of damage yearly. Each new worm is designed to take advantage of new vulnerabilities; hence their signatures are vastly different. This project selected the common impact of these worms to use for input data to the autoencoder, to make the system able to respond to one of the side-effects of the worms spread – their effect on routing. Specifically, the BGP Update messages which increase in number in the presence of a worm on other Internet instability caused have provided a convincing input to the autoencoder.

The work performed in this project demonstrated that a trained neural network is able to detect both global and localised instability at a single router. The main focus was to detect global Internet instability caused by worms, for which the autoencoder was able to correctly show anomalies. The autoencoder is not able to distinguish between a global instability such as in the case of the worm, and a localised event that is in the vicinity of the router. Any world-wide event that affects Internet stability, such as the World Trade Centre Attacks, would not trigger an alarm from the output of the autoencoder, unless it is in the neighbourhood of the router being monitored. This work is the first known application of a neural network to predict Internet instability using BGP routing information.

The autoencoder was shown to be easy to train, and the selection of each neural network parameter and training data did not fundamentally change the output novelty for the system. The autoencoder was trained using only normal data, which makes it easier to train than a neural network classifier. It was able to detect anomalies such as worms without prior knowledge of the worm data, which makes it flexible for detecting new worms with different signatures.

A threshold was added to the autoencoder output, to indicate whether the novelty value indicates a threat to stability. For this project, the level was chosen by looking at the whole dataset, and choosing a value that would include the most significant spikes. The selection of this threshold was not analysed in detail, and could be improved in further work.

A comparison was performed between the output from the autoencoder and a simple rule-based system. From the results of this comparison, it was evident that both systems were able to detect both global and localised instability. The autoencoder

was able to use current as well as previous data to provide the novelty, which allowed it to respond at either the same speed or faster than the basic rule-based system. The neural network was also less sensitive to outlier points, since it had the ability to use previous data and to recognise trends in the data. The autoencoder is more complex internally than the rule-based system, but a rule-based system would require far more configuration changes for a network administrator to achieve similar performance to that of the autoencoder.

It is possible that additional inputs such as traffic load could improve the data. By correlating the output from disparate routers of the Internet, the localised effect seen on the routers could be eliminated, leaving only the global Internet instabilities. The problems associated with using more than one router were discussed. The most significant concern was that the transporting of the data between routers would most likely be via the Internet, which would cause a delay in the detection of global Internet instability.

A system based on the results of this project would not eliminate the threat of worms, since the prediction made by the neural network relies on the effects of a worm already circulating on the Internet, but it will provide forewarning of a potential attack. This in turn will allow for earlier analysis of the vulnerability exploited, making patches available sooner and alerting uninfected network administrators about the threat. In these cases the damage will be reduced, with a large saving in time for the Internet community.

# 8 Further Work

A true measure of the performance of this method would be to run the autoencoder on live data. This would demonstrate whether the autoencoder is able to detect new forms of worms that haven't been seen before. Another advantage of live testing would be to demonstrate a real worm prediction system prototype. An extension to this would be to retrain the network on the fly, allowing the autoencoder to adapt to the behaviour of new worms.

It is believed that the greatest improvement could be made by using additional inputs to the autoencoder. It is suggested that only inputs that are available from a single router be used, so that the solution would be practical for a real implementation. One input could be the number of unique routes that have been updated within the current time-period, which will provide more information as to how many individual routes are being affected at the current time. A low value of unique routes but with a high number of BGP Updates would most likely indicate instability of only a few routes, which is unlikely to be as a result of a worm. Another possible input would be the number of packets flowing through the router for the specified time interval. This would provide some information to the autoencoder that is not directly BGP based.

Another validation of this technique would be to use data from other routers. The router chosen for this project has been used in several studies that provided the basis for this work, so the choice of other routers could provide a broader verification of the results obtained here. It would also then be possible to compare the novelty of an edge router with that of a core Internet router. This would demonstrate if a core router is able to detect a worm earlier to a smaller router.

# 9 References


1. C. Hunt, *TCP/IP network administration*, O'Reilly, 2002.
2. K. Coffman and A. Odlyzko, *Growth of the internet*, Optical Fiber Telecommunications **IV** (2002), no. B: Systems and Impairments, pp. 17-56.
3. Internet domain survey, http://www.isc.org/index.pl?ops/ds/reports/2004-07/, Last accessed 8 February 2005.
4. How many Online?, http://www.nua.ie/surveys/how_many_online/index.html, Last accessed 8 February 2005.
5. S. Savage, A. Collins, E. Hoffman, J. Snell and A. T., "The end-to-end effects of internet path selection", in *SIGCOMM*, 1999, pp. 289-299.
6. C. C. Zou, W. Gong and D. Towsley, "Code Red worm propagation modelling and analysis", in *9th ACM Conference on Computer and Communications Security*, 2002, pp. 137-147.
7. T. Chen and J. M. Robert, "The evolution of viruses and worms," *Statistical methods in computer security*, W. Chen (Editor), Marcel Dekker, 2004.
8. D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford and N. Weaver, "The spread of the Sapphire/Slammer worm," CAIDA, Technical Report, 2003.
9. D. Moore, C. Shannon, G. Voelker and S. Savage, "Internet quarantine: Requirements for containing self-propagating code", in *2003 IEEE Infocom Conference* **3**, 2003, pp. 1901-1910.
10. I. Dubrawsky, Effects of worms on internet routing stability, http://www.securityfocus.net/infocus/1702, Last accessed 5 January 2005.
11. J. Cowie, A. Ogielski, B. J. Premore and Y. Yuan, Global routing instabilities during Code Red II and Nimda worm propagation, http://www.renesys.com/projects/bgp_instability, Last accessed 6 January 2005.
12. C. Labovitz, A. Ahuja, A. Bose and F. Jahanian, "Delayed internet routing convergence", in *ACM SIGCOMM*, 2000.
13. A. Khanna and J. Zinky, "The revised ARPANET routing metric", in *ACM SIGCOMM '88*, 1988, pp. 45-56.
14. Y. Rekhter and T. Li, "A Border Gateway Protocol 4 (BGP-4)," RFC 1771, http://www.ietf.org/rfc/rfc1771.txt, Last accessed 15 August 2005.

15. L. Subramanian, S. Agarwal, J. Rexford and R. H. Katz, "Characterizing the internet hierarchy from multiple vantage points", 2002.
16. L. Wang, "Observation and analysis of BGP behaviour under stress", in *ACM SIGCOMM Internet Measuring Workshop*, 2002.
17. Y. Rekhter and Y. Li, "A border gateway protocol 4 (BGP-4)," *RFC-1771*, 1995.
18. E. Chen, "Route refresh capability for BGP-4," *RFC-2918*, 2000.
19. C. Villamizar, R. Chandra and R. Govindan, "BGP route flap damping," *RFC-2439*, 1998.
20. Z. Mao, R. Govindan, G. Varghese and R. Katz, "Route flap damping exacerbates internet routing convergence", in *ACM SIGCOMM conference*, 2002, pp. 221-233.
21. M. Lad, X. Zhao, B. Zhang, D. Massey and L. Zhang, "Analysis of BGP update surge during Slammer worm attack", in *Fifth International Workshop on Distributed Computing*, 2003.
22. J. Nazario, J. Anderson, R. Wash and C. Connelly, "The future of internet worms", in *Blackhat Briefings*, 2001.
23. E. H. Spafford, "The internet worm incident", in *ESEC `89 2nd European Software Engineering Conference*, Springer, 1989.
24. P. J. Denning, *Computer viruses*, American Scientist **76** (1988).
25. D. Moore, C. Shannon and J. Brown, "Code-Red: A case study on the spread and victims of an internet worm", in *Internet Measurement Workshop (IMW)*, 2002.
26. Z. Chen, L. Gao and K. Kwiat, "Modeling the spread of active worms", in *IEEE INFOCOM*, 2003.
27. S. Staniford, V. Paxson and N. Weaver, "How to own the internet in your spare time", in *11th USENIX Security Symposium (Security '02)*, 2002.
28. P. Boutin, 'Slammed!', *Wired Online Article*, http://www.wired.com/wired/archive/11.07/slammer_pr.html, Last accessed 9 July 2004.
29. D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford and N. Weaver, *Inside the Slammer worm*, IEEE Security and Privacy. **1** (2003), no. 4, pp. 33-39.

30. N. Joukov and T. Chiueh, "Internet worms as internet-wide threat," RPE Report, 2003.
31. R. Russell and A. Mackie, "Code Red II worm: Incident analysis report," Attack Registry and Intelligence Service (ARIS), 2001.
32. Netcraft web server survey, http://www.netcraft.com, Last accessed 3 February 2005.
33. N. K. Kasabov, *Foundations of neural networks, fuzzy systems, and knowledge engineering*, Massachusetts Institute of Technology, 1998.
34. B. B. Thompson, J. Robert, J. Choi Jai and M. A. El-Sharkawi, "Implicit learning in autoencoder novelty assessment", in *International Joint Conference on Neural Networks, 2002 IEEE World Congress on Computational Intelligence*, 2002.
35. L. Tarassenko, A. Nairac, N. W. Townsend, I. Buxton and B. Cowley, *Novelty detection for the identification of abnormalities*, International Journal of Systems Science **31** (2000), no. 11, pp. 1427-1439.
36. M. E. Tripping and C. M. Bishop, *Probabilistic principal component analysis*, Journal of the Royal Statistical Society **B** (1997), no. 61, pp. 611-622.
37. RIS raw data, http://abcoude.ripe.net/ris/rawdata/, Last accessed 6 January 2005.
38. T. Griffin, BGP impact of sql worm, http://www.research.att.com/~griffin/bgp_monitor/sql_worm.html.
39. C. Labovitz, G. R. Malan and F. Jahanian, *Internet routing instability*, IEEE/ACM Transactions on Networking **6** (1998), no. 5, pp. 515-528.
40. RIS raw data, http://www.ripe.net/projects/ris/rawdata.html, Last accessed 24 January 2005.
41. RIPE network coordination centre, http://www.ripe.net, Last accessed 24 January 2005.
42. Multi-threaded routing toolkit, http://www.mrtd.net, Last accessed 24 January 2005.
43. *Mysql database*, http://www.mysql.org
44. MathWorks,Matlab, http://www.mathworks.com.
45. I. Nabney and C. Bishop, "Netlab neural network software," 2001, http://www.ncrg.aston.ac.uk/netlab/.

# Appendix A: BGP Announcements and Withdrawals

The following graphs show the number of BGP announcement and BGP withdrawal messages individually for the dates from June to September 2001. The total number of BGP Update messages, which is the sum of the announcements and withdrawals, has been displayed in the main text. The number of BGP Update messages (plotted linearly in Figure 32 and logarithmically in Figure 33) closely follows that of the announcements, which is expected since the announcements dominate the results. Each point represents the number of BGP Updates during a fifteen-minute time period. The graphs in Figure 34 and Figure 35 similarly show the total number of BGP withdrawal messages for the same time period.

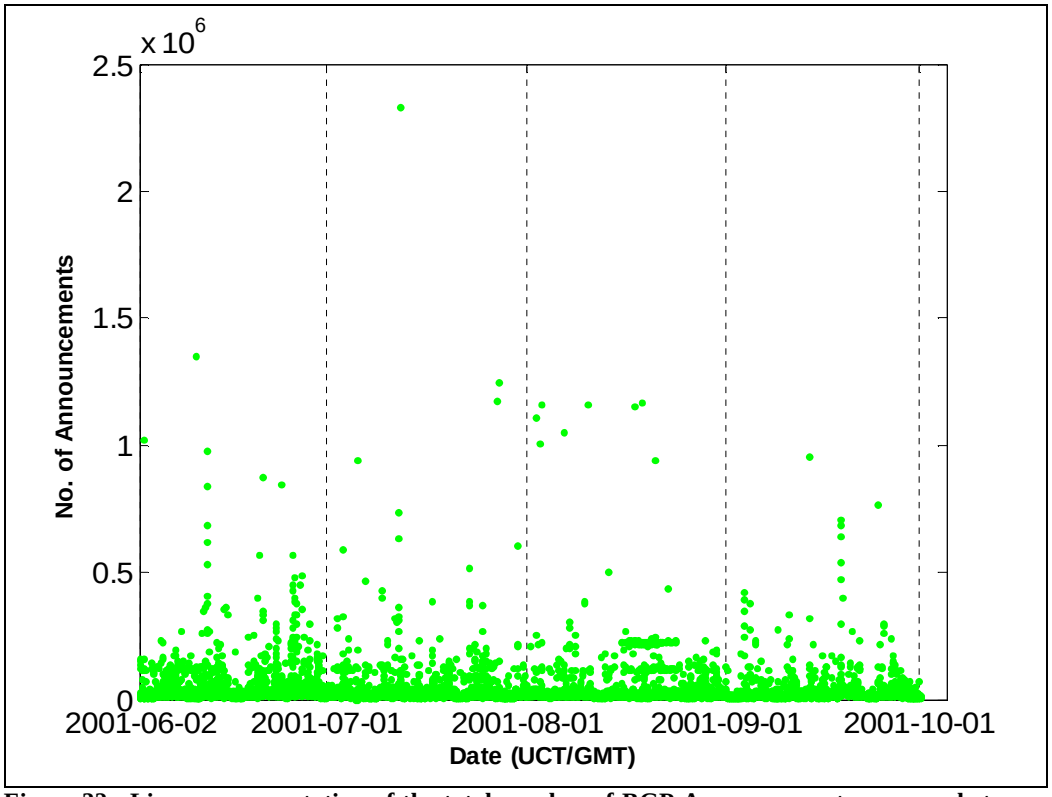


**Figure 32. Linear representation of the total number of BGP Announcement messages between June and September 2001.**

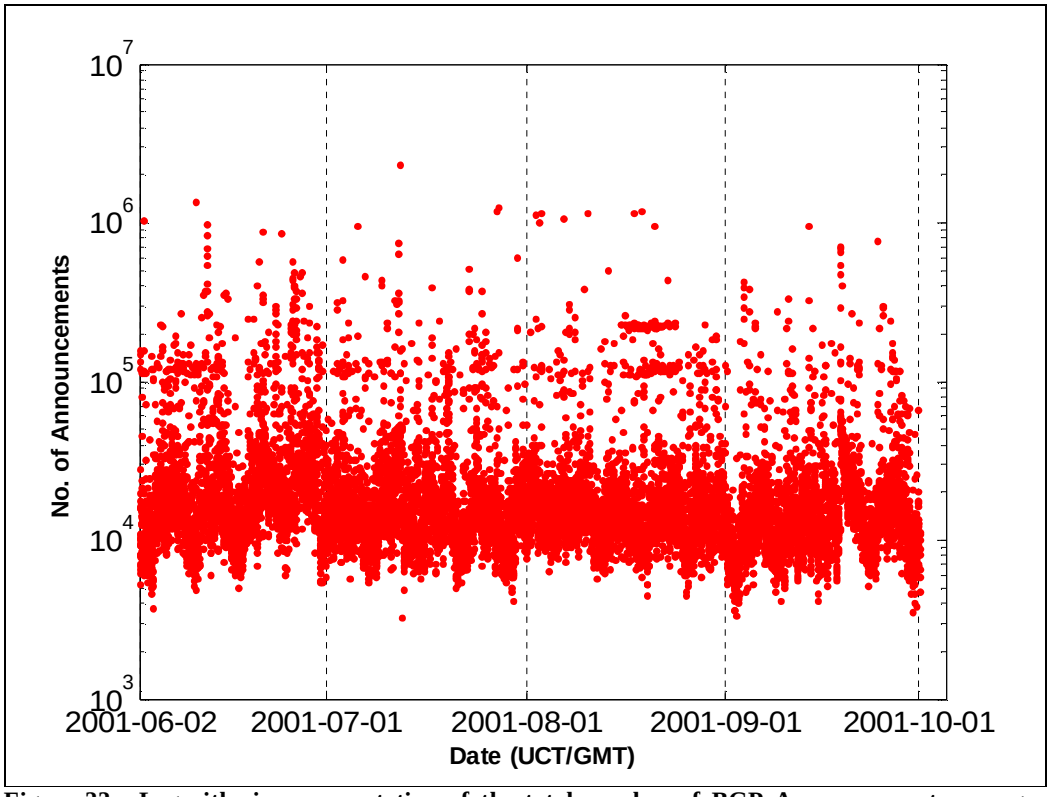


**Figure 33. Logarithmic representation of the total number of BGP Announcement messages between June and September 2001.**

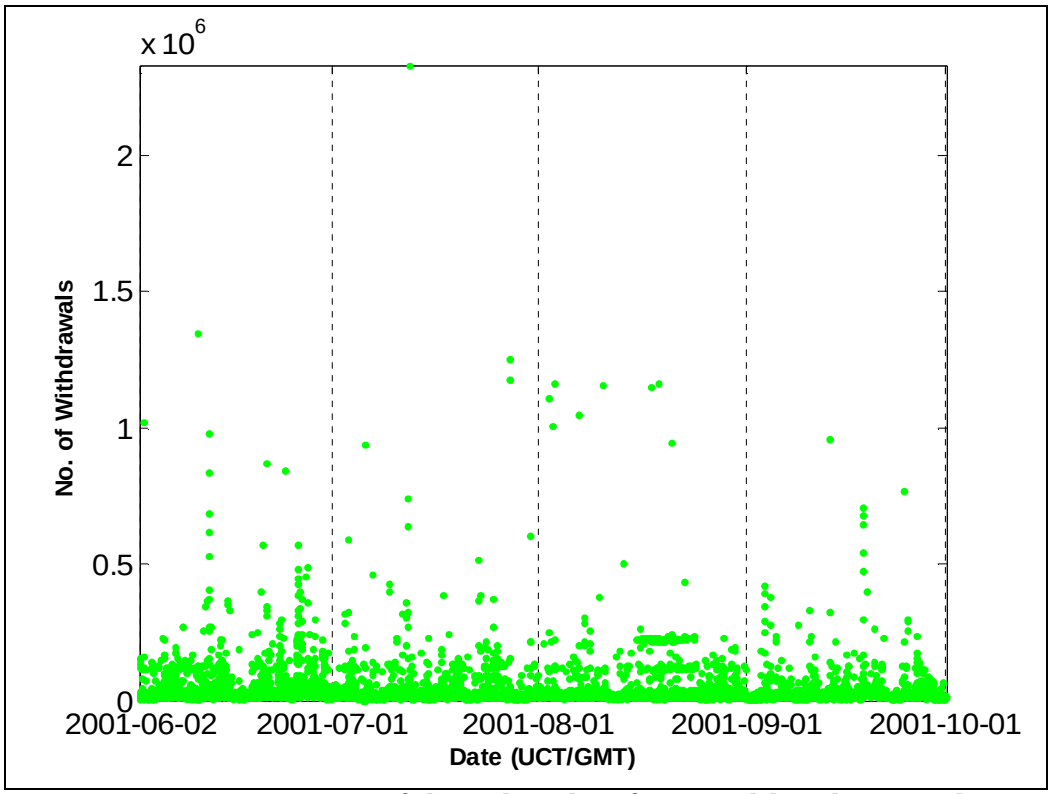


**Figure 34. Linear representation of the total number of BGP Withdrawal messages between June and September 2001.**

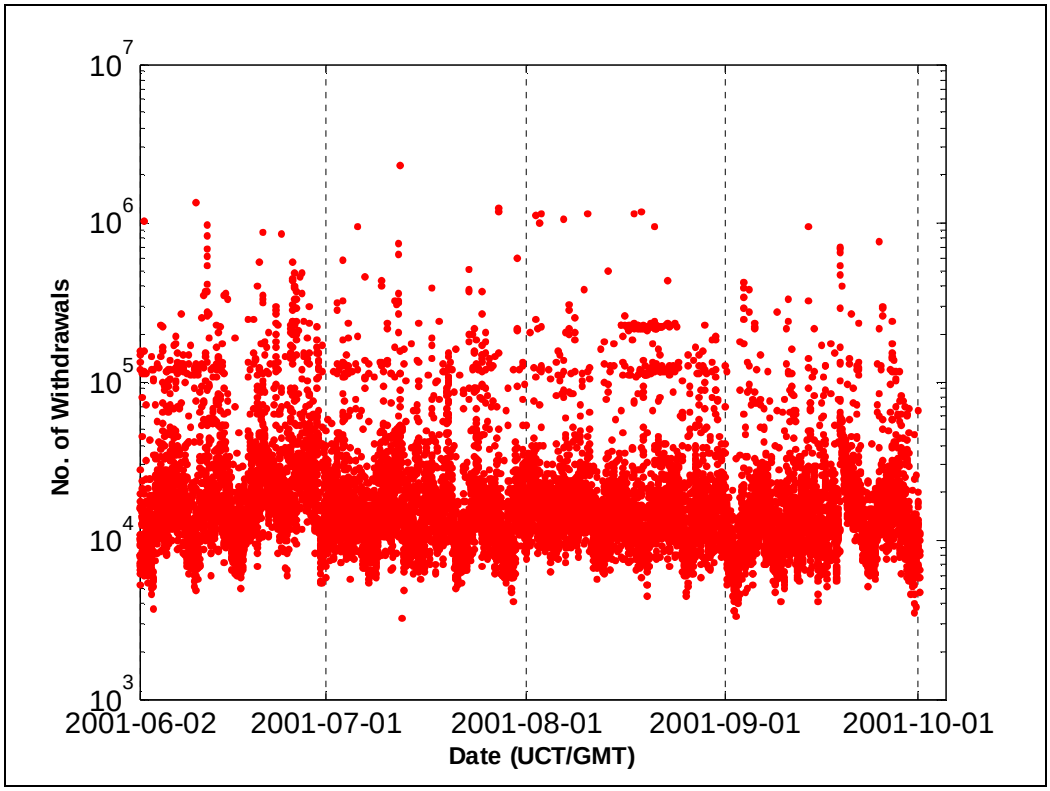


**Figure 35. Logarithmic representation of the total number of BGP Withdrawal messages between June and September 2001.**

# Appendix B: Paper Submitted to PRASA

The following paper was submitted to the *Pattern Recognition Association of South Africa* (PRASA) and accepted for presentation in November 2004. The conference was held in Grabouw, South Africa. The paper appeared in pages 81-86 of the conference proceedings.

# Predicting Global Internet Instability Caused by Worms using Neural Networks

*Elbert Marais, Tshilidzi Marwala*

School of Electrical and Information Engineering
University of the Witwatersrand
Private Bag 3, WITS, 2050, South Africa

e.marais@ee.wits.ac.za
t.marwala@ee.wits.ac.za

## Abstract

Internet *worms* are capable of quickly propagating themselves by exploiting vulnerabilities of software running on hosts that have access to the Internet. Once these worms have infected a computer, they have access to sensitive information on the computer, and are able to corrupt or retransmit this information. This paper describes a method of predicting Internet instability due to the presence of a worm on the Internet, based on information currently available from global Internet routers. A neural network is trained to predict anomalies in the number of router messages received by a router. The output from the network can help to provide an early warning for the presence of a worm.

## 1. Introduction

The rapid spread of worms leads to a sudden increase in Internet network traffic, which can have a direct impact on the stability of the Internet as a whole. Due to the widespread use and reliance by many businesses on the Internet for generating revenue, a worm is able to cause extensive downtime of infected hosts, as well as making the Internet inaccessible even for those hosts that are not infected.

Many tools are available to prevent the spread of viruses and worms. This project assumes that worms will continue to be written and effective regardless of these tools, and deals with the early detection of a worm that is already causing Internet instability.

The Border Gateway Protocol (BGP) is the current Internet standard for routers to exchange routing information, which relies on the TCP protocol for network transport of these routing messages. From existing research [1, 2, 3, 4], it has been shown that this routing information can be used to distinguish traffic caused by a worm from that caused by large-scale network outages (such as the World Trade Centre attacks on 11 September 2001).

A neural network has been trained to predict the presence of a worm on the Internet, by observing the behaviour of a global Internet router. By training the network to recognise normal routing behaviour, it is able to distinguish unusual behaviour, specifically that of an active worm. This forewarning could significantly limit the impact of a worm, reducing the impact to productivity caused by Internet instability.

This document provides background on the BGP protocol, worms and novelty detection using neural networks. This is followed by the method taken to train the network, and the results that have been obtained.

## 2. Worms

*A worm is a program that can run independently and can propagate a fully working version of itself to other machines* [5 p.3]. The main difference between a virus and a worm is that a virus attaches itself to a file or program, whereas a worm is capable of running independently. A worm spreads complete but possibly mutated versions of itself to other computers. It achieves this by exploiting holes in an operating system [6], or relies on users to infect the computers (such as email worms which rely on users to open attachments).

Even with countermeasures such as anti-virus and intrusion detection software in place, viruses continue to exist. New versions of operating systems and software are continuously being released, each possibly providing new ways for a virus or worm to spread. Additionally, worms and viruses that rely on a user for propagation will have a certain level of success. Even though anti-viruses do control the spread of worms, they are only effective after some spread and possible damage has been done. Not all computers have anti-virus software installed, or have the latest virus patterns. In the case of the *CodeRed II* worm, it took sixteen weeks for most hosts to fix the vulnerability, and thousands of systems were not patched six weeks later [7].

The cost effect of worms and viruses is difficult to estimate, since it is a combination of the time spent analysing and patching the problem, as well as the loss of productivity by end-users who are unable to use their computers or the Internet. The effect of worms in the public and private sector has been estimated to cost millions of dollars. The *CodeRed* worm which spread on 19 July 2001, together with subsequent strains, caused approximately $2.6 billion [8]. The estimated cost of SQL Slammer was estimated at $1 billion [9].

The rate at which worms spread is largely dependant on their means of propagation and the number of possible hosts that are susceptible to the worm. A worm relying on users to open attachments will spread slower than a worm that immediately replicates itself once it is present on a computer. Also, since the worms exploit vulnerabilities in specific software or operating systems, only these systems are susceptible to being infected by the worms. Other systems that can't be infected are still able to transmit the worm, as in the case of a worm that affects only Windows systems using email, but can be relayed by other operating systems which forward the emails. The following is an indication of the speed at which worms are able to spread:

- The *CodeRed II* worm required less than 14 hours to reach its saturation of more than 359,000 infections, with the worms spread peaking at just over 2,000 infections per minute [7].
- The *SQL Slammer* worm targeted a buffer overflow vulnerability in computers running Microsoft's SQL Server. The worm infected at least 75,000 hosts over a period of approximately 10 minutes [10].

## 3. Routing and BGP

Routing provides a means of transporting data packets from a source to a destination. A router is a physical device with many network connections, used for transferring data packets as well as determining which path to use for sending these packets. Each router has several routing tables, which are used to determine the next device to send packets to, based on each packets final destination. Routing protocols are used for inter-router communication to establish these routing tables. This project is focussed on global Internet routers, which relay traffic for the entire Internet.

Due to the dynamic state of the Internet (routers are continuously added, routers go offline temporarily), the routes between endpoints within the network are continuously changing. These route changes are propagated using control messages sent between the routers, so the more IP addresses available the more the overhead load on the network. For these reasons, the Border Gateway Protocol (BGP) is

used as the standard for routing throughout the Internet [11].

It is infeasible for a router to store the routes between all source and destination and IP addresses for the Internet, due to the number of IP addresses available on the Internet. Instead, BGP stores only the routes between Autonomous Systems.

*Autonomous Systems* are made up of a network or group of networks that have a common administration and common routing policies [12]. This grouping allows for several IP addresses to be represented by a single *Autonomous System*, which greatly simplifies the number of routing entries that each router must hold, since groups of IP addresses are stored together rather than individual values.

Four types of messages are used by BGP for exchanging information between routers [12]:

- *Open*
- *Update*
- *Notification*
- *Keep-Alive*

These messages are relevant only for routing status and update information, and are independent of the data being sent between the routers. The message that is most relevant for this research is the *Update* message. BGP routers send *Update* messages to exchange network accessibility information, such as when a new router is present that causes the router to change its routing paths. These changes are what are propagated to the surrounding networks. Two specific types of *Update* messages are important for the neural network:

- *Announcement* messages which inform a router that a new route is available.
- *Withdrawal* messages indicate that a previously available route has become unavailable.

Routers use these messages combined with several configuration parameters to determine the routing table entry used between the *Autonomous Systems*.

# 4. Neural Network for Novelty Detection

*Artificial neural networks* are expert systems based on modelling of the human brain [13]. These networks consist of an input and output layer which is specific to the system, as well as a hidden layer consisting of several *neurons* that are trained using a set of training data. Once a network has been trained it can be used to generate outputs for new input datasets. All data applied to the input layer of a neural network should be normalized to the range of either [-1:1] or [0:1], to ensure that the network uses data within the same range.

One method of distinguishing normal behaviour from novel behaviour is the use of an *autoencoder* [14]. An *autoencoder* consists of an input and output layer with the same number of inputs and outputs, combined with a narrow hidden layer. The network is trained using only positive examples (i.e. standard behaviour), with the training data using the same inputs and outputs. The hidden layer attempts to reconstruct the inputs to match the outputs, which minimizes the error between the inputs and the outputs when non-novel data is presented to the system. The network uses a narrow hidden layer, which forces the network to reduce any redundancies, but still allows the network to detect non-redundant data. An example autoencoder is shown in Figure 1.

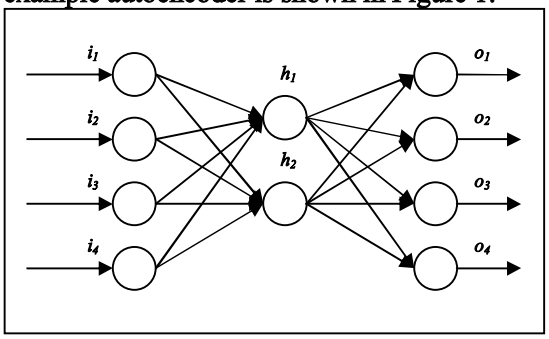


Figure 1. Example of an autoencoder neural network, with four input and output nodes, and two hidden nodes. The inputs and outputs used for training are identical.

The test data that is to be searched for novel behaviour is presented to the trained network.

The measure of novelty for each input set is then given by the mean-square error:

$$e = \frac{1}{n}\sum_{1}^{n}\left((t_n - i_n)^2\right) \quad (1)$$

where $n$ is the number of inputs and outputs, $t_n$ is the $n^{th}$ output and $i_n$ is the $n^{th}$ input.

The network gives a low error for input datasets similar to the training data, but gives a high error for data that is significantly different to the original training data. It is then possible to set a threshold for the maximum error before the data can be classified as novel.

## 5. Method

Existing research on the effect of worms on global Internet routers has shown that worms cause Internet routing instability [1, 2]. Based on the analysis in [1, 3, 4], it is evident that worms have a distinguishable effect on the number of *Update* messages sent between BGP routers. Figure 2 and Figure 3 shows a marked increase in the number of announcements (BGP Update messages) caused by the *Nimda* virus. It is interesting to note that the Baltimore train wreck which severed several large fibre links on 18 July 2001, and the wide scale internet outages from the World Trade Centre attack on 11 September 2001 don't feature on this graph.

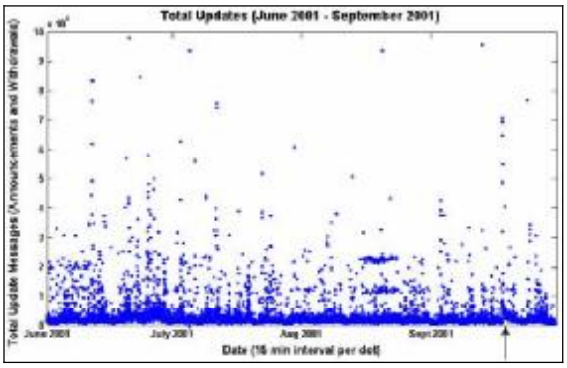


Figure 2. Number of BGP *Update* messages received by a router from June – September 2001. The arrow the increased router messages caused by the *Nimda* worm (September 2001). This data has been extracted from archives of a router based in Amsterdam, and each dot represents the total number of *Update* messages (announcements and withdrawals) in 15 minute time period.

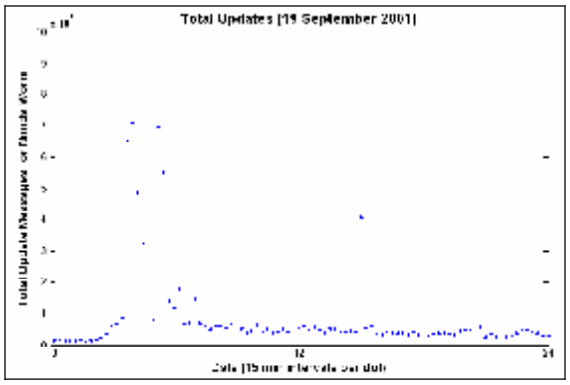


Figure 3. Number of BGP UPDATE messages received on 19 September 2001 (GMT). The sudden increase is due to Internet instability caused by the *Nimda* worm.

The neural network inputs were based on the data shown in Figure 2, but using the total number of *announcement* and *withdrawals* for each fifteen minute interval separately. Fifty inputs were used for the number of *announcement* messages, starting with the number of router announcements for the current time period, and then each subsequent input for the number of announcements for the fifteen minutes prior to the previous time period. This provides the network with the trend of the total *announcement* messages for the current and previous time periods. Another fifty inputs are used for the number of *withdrawals* for each fifteen minute time period, in the same manner as for the *announcement* inputs.

The network was trained using a period which showed no significant worm activity in the dataset shown in Figure 2. The network was configured for the novelty detection method described previously, using the *Scaled Conjugate Gradient* method to optimize the network weights, which was used due to the method's computational efficiency and expediency [15]. As expected for the *autoencoder* network configuration, the mean-square error between the training inputs and the actual network outputs was minimal.

The mean-square error between the network inputs and outputs for September 2001 are shown in Figure 4, for which it is known that the *Nimba* worm significantly impacted the

Internet. The graph clearly indicates when there were increases in network instability, with the largest spike corresponding to the instability caused by *Nimda*.

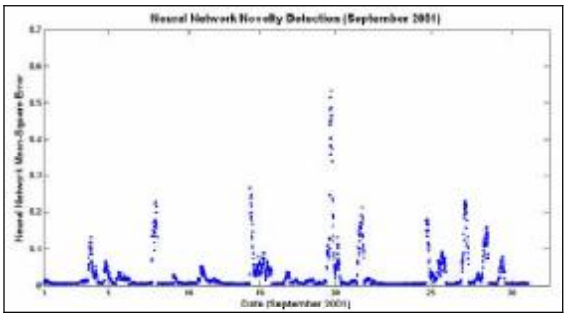

Figure 4. Graph of mean-square error for the trained neural network for September 2001. The largest spike corresponds to the effect of the *Nimda* worm, which appeared on the Internet on 19 September 2001.

## 6. Conclusion

This project extends from previous analysis which found that the additional load caused by worms corresponds to the number of BGP messages sent between routers. Based on initial findings, it appears that a single router can be used to train a neural network to predict the presence of Internet instability caused by a worm.

From previous incidents of worms, it has been seen that worms are capable of causing a large amount of damage simply through their propagation. Antivirus software and intrusion detection will continue to improve, but are limited in their ability to detect novel worms.

By providing an earlier warning of a potential worm, virus experts will be able to start analysing exploited system vulnerability earlier. This allows for reducing the effects of a worm, by making patches available earlier and alerting uninfected network administrators about the threat.

## 7. References


[1] I. Dubrawsky, "Effects of worms on Internet routing stability," *Security Focus Article*, June 2003. http://www.securityfocus.net/infocus/1702, Last accessed 30 September 2004.

[2] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S.F. Wu, L. Zhang, "Observation and analysis of BGP behavior under stress," in Proc. *ACM SIGCOMM Internet Measurement Workshop*, Marseille, France, November 2002.

[3] J. Cowie, A. Ogielski, B.J. Premore, Y. Yuan, "Global Routing Instabilities during Code Red II and Nimda Worm Propagation," Renesys Corporation, September 2001. http://www.renesys.com/projects/bgp_instability/, Last accessed 30 September 2004.

[4] M. Lad, X. Zhao, B. Zhang, D. Massey and L. Zhang, "An analysis of BGP update burst during slammer attack," in *Proceedings of the 5th International Workshop on Distributed Computing (IWDC),* December 2003.

[5] E. Spafford, "The Internet worm incident," Tech. Rep. CSD-TR-933, Department of Computer Sciences, Purdue University, September 19, 1991.

[6] S. Staniford, V. Paxson, and N. Weaver. "How to Own the Internet in your spare time," in *USENIX Security Symposium*, August 2002.

[7] D. Moore and C. Shannon, "Code-Red: a case study on the spread and Victims of an Internet worm," in proceedings of the *2002 ACM SICGOMM Internet Measurement Workshop*, Marseille, France, pp. 273–284, November 2002.

[8] D. Moore, C. Shannon, G. Voelker and S. Savage, "Internet quarantine: requirements for containing self-propagating code," in *Proceedings of the 2003 IEEE Infocom Conference*, San Francisco, CA, April 2003.

[9] P. Boutin. "Slammed!," Wired Online Article, Issue 11.07, July 2003,

http://www.wired.com/wired/archive-/11.07/slammer_pr.html, Last accessed 30 September 2004.

[10] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the slammer worm," in *IEEE Security and Privacy*, vol. 1, issue 4, pp. 33-39, Aug 2003.

[11] C. Labovitz, G.R. Malan and F. Jahanian, "Internet routing instability," *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 515-528, 1998.

[12] Y. Rekhter, Y. Li. "A border gateway protocol 4 (BGP-4)," Tech. Rep. RFC-1771, March 1995.

[13] N. K. Kasabov, *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering*, MIT Press, London, 1998.

[14] N. Japkowicz, C. Myers, M. A. Gluck, "A Novelty Detection Approach to Classification" in IJCAI, pp. 518-52, 1995.

[15] T. T. Jervis and W. J. Fitzgerald. "Optimization Schemes for Neural Networks". Technical Report CUED/F-INFENG/TR 144, Cambridge University Engineering Department, Cambridge, England, 1993.