Inferring Malicious Network Events In Commercial ISP Networks Using Traffic Summarisation

by

Peter Sandford

A Doctoral Thesis

Submitted in partial fulfilment
of the requirements for the award of

Doctor of Philosophy
of
Loughborough University

14th November 2011

Loughborough University

Thesis Access Form

Copy No……...…….**Location**…….……...…

Author……...…….……..…….

Title……..

Status of access OPEN / RESTRICTED / CONFIDENTIAL

Moratorium Period:……years, ending……../……200…….

Conditions of access approved by (CAPITALS):……

Supervisor (Signature)……...……...

School of……...……

Author's Declaration: *I agree the following conditions:*

Open access work shall be made available (in the University and externally) and reproduced as necessary at the discretion of the University Librarian or Dean of School. It may also be digitised by the British Library and made freely available on the Internet to registered users of the EThOS service subject to the EThOS supply agreements.

*The statement itself shall apply to **ALL** copies including electronic copies:*

This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

Restricted/confidential work: All access and any photocopying shall be strictly subject to written permission from the University Dean of School and any external sponsor, if any.

Author's signature…….Date……...……...……...

users declaration: for signature during any Moratorium period (Not Open work): ***I undertake to uphold the above conditions:***			
Date	Name (CAPITALS)	Signature	Address

4.1e

Research Student Office, Academic Registry
Loughborough University, Leicestershire, LE11 3TU, UK
Switchboard: +44 (0)1509 263171 Fax: +44 (0)1509 223938

Loughborough University

Certificate of Originality

This is to certify that I am responsible for the work submitted in this thesis, that the original work is my own except as specified in acknowledgments or in footnotes, and that neither the thesis nor the original work contained therein has been submitted to this or any other institution for a degree.

. . . . . .

Peter Sandford

14th November 2011

Abstract

With the recent increases in bandwidth available to home users, traffic rates for commercial national networks have also been increasing rapidly. This presents a problem for any network monitoring tool as the traffic rate they are expected to monitor is rising on a monthly basis. Security within these networks is paramount as they are now an accepted home of trade and commerce. Core networks have been demonstrably and repeatedly open to attack; these events have had significant material costs to high profile targets.

Network monitoring is an important part of network security, providing information about potential security breaches and in understanding their impact. Monitoring at high data rates is a significant problem; both in terms of processing the information at line rates, and in terms of presenting the relevant information to the appropriate persons or systems.

This thesis suggests that the use of summary statistics, gathered over a number of packets, is a sensible and effective way of coping with high data rates. A methodology for discovering which metrics are appropriate for classifying significant network events using statistical summaries is presented. It is shown that the statistical measures found with this methodology can be used effectively as a metric for defining periods of significant anomaly, and further classifying these anomalies as legitimate or otherwise. In a laboratory environment, these metrics were used to detect DoS traffic representing as little as 0.1% of the overall network traffic.

The metrics discovered were then analysed to demonstrate that they are appropriate and rational metrics for the detection of network level anomalies. These metrics were shown to have distinctive characteristics during DoS by the analysis of live network observations taken during DoS events.

This work was implemented and operated within a live system, at multiple sites within the core of a commercial ISP network. The statistical summaries are generated at city based points of presence and gathered centrally to allow for spacial and topological correlation of security events.

The architecture chosen was shown to be flexible in its application. The system was used to detect the level of VoIP traffic present on the network through the implementation of packet size distribution analysis in a multi-gigabit environment. It was also used to detect unsolicited SMTP generators injecting messages into the core.

Monitoring in a commercial network environment is subject to data protection legislation. Accordingly the system presented processed only network and transport layer headers, all other data being discarded at the capture interface.

The system described in this thesis was operational for a period of 6 months, during which a set of over 140 network anomalies, both malicious and benign were observed over a range of localities. The system design, example anomalies and metric analysis form the majority of this thesis.

Acknowledgements

There are many people I'd like to thank for their help and support over the years where this research has progressed;

Prof. David Parish for his patience, guidance and friendship

My wife Rachel, for her support and understanding

My brother Mark for galvanising my interest in the field of networks

John for his deciphering of my endless LaTeXissues

Everyone who has given a kind word or a well meant prod in the posterior!

List of Publications

"Understanding Increasing Traffic Levels for Internet Abuse Detection", P. Sandford, DJ Parish and JM Sandford, Security Journal, 202, 2007, pp 63-76, ISSN 0955-1662.

"Detecting security threats in the network core using Data Mining techniques", P. Sandford, DJ Parish and JM Sandford, in Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP page 1-4

"Identifying Internet Abuse in ISP Networks" P. Sandford, DJ Parish and JM Sandford, in Proceedings of Safety and Security in a Networked World: Balancing Cyber-Rights and Responsibilities. September 8-10, 2005, Oxford, United Kingdom

"Analysis of SMTP Connection Characteristics for Detecting Spam Relays", P. Sandford, DJ Parish and JM Sandford, in Proceedings of the International Multi-Conference on Computing in the Global Information Technology. Bucharest, Romania, 2006.

Contents

List of Figures

List of Tables

Acronyms

ANN Artificial Neural Network.

DNS Domain Name System.

DoS Denial of Service.

FPGA Field-Programmable Gate Array.

GBit/s Billion Bits Per Second.

HSN High Speed Networks.

IDS Intrusion Detection System.

IIS Internet Information Services.

IMS IP Multimedia Subsystem.

IPS Intrusion Prevention System.

MBit/s Million Bits Per Second.

MSS Maximum Segment Size.

MTU Maximum Transmission Unit.

NAPI New API (Application Programming Interface).

NAT Network Address Translation.

NIC Network Interface Card.

NIDS Network Intrusion Detection System.

OSI Open Systems Interconnection Reference.

POP Point of Presence.

SIP Session Initiation Protocol.

SMTP Simple Mail Transfer Protocol.

SOM Self Organising Map.

TCP Transmission Control Protocol.

TFN Tribe Flood Network.

TTL Time to Live.

VLAN Virtual Local Area Network.

VoIP Voice over IP.

WAN Wide Area Network.

Chapter 1

Introduction

1.1 Summary

Security over the Internet is becoming increasingly important. Consumers are relying on the Internet for much of their shopping [1], demonstrating increasing trust from the general populous in the technology. Financial institutions increasingly rely on the Internet for trading of equities and commodities, replacing the more traditional methods of dedicated leased lines.

There are many threats to the security of the Internet, some involving the privacy of data, some involving the security of hosts, others the infrastructure of the networks themselves.

Towards home user security, recent advances in most current operating systems allow the automatic patching of potential vulnerabilities, in many cases without any direct user intervention. The deployment of firewalls by default in operating systems has become common and the use of anti-virus software has become more pervasive. Recent operating systems deploy strategies such as making memory segments writable or executable, but not both. In the UK, consumer ISP connectivity is often achieved via routers which perform NAT for a household, preventing direct inbound Internet access.

Business users often employ layered sophisticated commercial firewalls, filtering all traffic which is present on their network. Most businesses employ administrators who will ensure that security on business hosts is in agreement with the appropriate policies.

While the security offered on single hosts and local networks is strengthened in line with their changing use, the security offered in core networks has not moved in line with this. The infrastructure the Internet employs has been demonstrably vulnerable to attack, even for major corporations hosting in distributed environments. Filtering of packets is often limited to simple metrics such as application

layer port numbers.

Response times to denial of service attacks on major hosts have led to changes in the way that major targets are hosted. There is a sizeable body of research on coping with, and mitigating denial of service; however, much of this research requires action from core routers, which ISPs are slow to implement. ISPs are generally reluctant to increase the load on core routers, as the increase in load will incur a cost in performance and therefore bandwidth. Attacks are still common, even for high profile sites [2].

Dealing with these attacks is problematic because:

- The data rates in core networks preclude stateful packet inspection
- Processing on core network devices is extremely limited
- Expert analysis is required to determine the sources of attacks
- Mitigation of these attacks is difficult to achieve at the receiver

The research described in this thesis attempts to provide a low cost (in processing terms), practical network anomaly detector. The potentially huge costs of processing packets and flows at full line rate were avoided, instead inferring activity from summarised data.

This research can be divided into several major activities as outlined here.

Initial studies were undertaken to investigate the viability of traffic capture and analysis at high data rates using relatively cheap hardware (discussed in section 3.2).

A fundamental issue with research of this nature, is the validation of results. In a live network situation, it is impossible[1] to ascertain the accuracy of classification. This is due to the lack of availability of any independent method of corroboration (traditional IDS systems were of limited use due to their dependence on the packet data, inability to classify at line rate, and inherent fallibility). To combat this, verification was attempted via two mechanisms; firstly a laboratory emulation was constructed in order to generate network misuse in a controlled environment (discussed in section 6.2.6), secondly results were compared (and shown to be consistent) with internal tools used by the ISP networking analysts, which do have access to higher levels of information.

Data mining tools and methodologies were used at several levels of abstraction to investigate relationships between network events and the laboratory data, allowing for careful feature selection for use with the live implementation.

[1] Given reasonable resource limitations

Once deployed, a database of training events was built from live data, classified by manual inspection and informed by the laboratory investigation, alongside general literature and experience in the field.

Data mining was again employed to investigate relationships in the data, leading to automated classifiers, which were run in real time to classify data on a pseudo real time basis.

1.2 Original Research Contribution

This work contributes to the network security monitoring field in several ways. It adds to the body of research proving the use of statistical summaries to be an effective and scalable way of monitoring for security purposes. It demonstrates a method for determining what these summaries should contain, and how they should be applied. This is a novel contribution, as it applies feature selection to live data taken from a core network, where previously only metrics based on domain knowledge had been used. This is applied in a data sensitive[2] environment, forcing the use of header values. Header values are used by the network at different layers to move messages about the network, and are therefore considered non-sensitive information. The header values were summarised into distributions for analysis purposes. These statistical summaries are combined with techniques to utilise them in a live and operational system. This research was conducted on a live national network, which gives it strength due to its practical nature, but also introduces challenges due to the lack of control over the data.

Given the fact the network that was operated on was data sensitive, fingerprinting of packet data is extremely difficult. To this end, the development of fingerprinting using the TCP checksum is also a novel contribution. This mechanism has applications for fast full packet fingerprinting via the checksum calculation being done on the NIC. This is particularly useful for its ability to operate without the data field, allowing for its application in environments where the data field is not available (such as the environment in which this research was conducted).

During the course of the work, there was a contribution to the field of unsolicited SMTP traffic detection, via IP profiling at the sender site. This concept breaks from the normal methods for dealing with unsolicited SMTP in that in does not filter at the receiver. The privileged data set which this research has access to, allows detection of spam at the source in a configurable manner. This research has already been published [3] and is a potential area for further future study; in particular the effort to distinguish further groups of SMTP generators,

[2]The core of an ISP network is subject to privacy laws, and therefore packet data may not be examined by a third party

such as legitimate mail servers and mailing lists. This work is not the focus of this thesis, but is discussed in Section 6.6. A copy of the published paper may be found in appendix A.

1.3 Structure of the Thesis

In Chapter 2 the general field of network monitoring is discussed in its varying scales and coverage. Chapter 3 presents the associated research from the most recent years in the area. This is grouped into three categories; network monitoring in a broad context, monitoring and prevention for security applications, and finally data mining in a network context.

Chapter 4 describes the system with which the data used in the research was captured and analysed. This section is split into hardware, software and system design sections.

Chapter 5 brings in the concept of data mining, and how it has been used in the research. This looks at both the development of appropriate metrics for the detection of criminal activity, and their application.

Chapters 6, 7 and 8 describe and discuss the data which has been collated and the patterns, relationships and types of activity that have been found in the data. Two example anomalies and their analysis are shown. The features which are selected are further investigated to demonstrate their appropriateness.

Chapter 9 draws some conclusions from the work, and discusses the possibility for further investigation.

Chapter 2

An Overview of Network Monitoring

Chapter 2 offers an overview of network monitoring with an emphasis on its application to network security. Of particular relevance to the research are the sections on monitoring at over one thousand Mbit/s and detection through anomaly.

Because of the security focus of the monitoring, this chapter begins by describing how the monitoring undertaken fits into an overall security model and within network monitoring as a whole.

Monitoring differing information, at differing data rates and in differing locations is discussed. The chapter then moves onto monitoring in a security context grouped by detection mechanisms, learning mechanisms and analysis techniques. It then describes the stimulus for network attackers using the honeynet project's 'MEECES' [4] acronym which provides a survey of motivating factors for the participation in illegal network activity.

This leads onto looking at mitigation techniques, grouped by topology and by the action taken. Finally it discusses data processing and characterisation.

Monitoring communication networks is a topic in which there has been considerable research over a considerable length of time. The networks being monitored have increased in size (both geographical and numerical) and data rate by orders of magnitude and the importance of accurate data has increased proportionately.

Network monitoring is a useful source of information for network managers, planners, security managers, marketing personnel and others. Information is gathered at many scales, from individual host usage to core networks with hundreds of thousands of hosts.

2.1 Monitoring as Part of Network Security and Information Assurance

Information Assurance is a wide ranging term, used to encompass confidentiality, integrity, authentication, availability and non-repudiation.

- Confidentiality

  The definition for confidentiality from ISO17799 [5] is "ensuring that information is accessible only to those authorised to have access". In computer security this is most often achieved through the use of cryptography.

- Integrity

  In terms of data integrity, this means that data is only modified with proper authorisation. This is not limited to the intentional alteration of data by a non-trusted party, but includes errors introduced through degradation of data. Examples would be a fire which caused the loss of data, or undetected network errors leading to corrupt data [5].

- Authentication

  This is the process by which users, or information is identified. Authentication allows this to be done in a way which prevents the information, or user from being forged or invented.

- Availability

  For data to be useful, it must be available in some form. This requires that the information, and a method to access it, is available to an authenticated and authorised user [5].

- Non-repudiation

  This is closely tied with Integrity, in that non-repudiation requires that all parties are aware of the state of the transaction. Once sent, it should not be possible to deny sending a transaction, nor possible to deny receiving a transaction, once received.

A reasonable example of these features in action would be an online banking transaction. In this example, authentication (via SSL certificates), confidentiality (via RSA cryptography) and integrity (via TCP's re transmission and checksum algorithms [1]) are all provided through the widespread use of the HTTPS protocol.

[1] These are not full-proof

Non-repudiation is handled at the application, and availability is handled by the individual infrastructure of the banking organisation.

The research undertaken here was primarily interested in DoS, which falls within the remit of availability. A threat to the network is a threat to the ability of a user to access information over it. A DoS attack is explicitly aimed at impairing the availability of information or of a service. For example, when a DoS attack is performed against a website, the intent is to prevent legitimate access to this resource, limiting its availability.

Network security is a very broad topic, and ranges from the physical security of devices through to vulnerabilities in protocols. In [6] the authors describe network security as divisible into three 'D's.

- Defense

  Defence is the most obvious of the forms of network security. Employing network defences reduces the chances of degradation of network performance or the compromise of other assets. Traditional network defences include devices such as firewalls, router access lists, spam and/or virus filters.

- Deterrence

  Deterrence is described as the second mode of security and involves reducing the frequency of security compromises by providing some unfavourable consequence resulting from a security compromise. Examples of deterrents would be national laws and company acceptable use policies.

- Detection

  The third 'D' is detection. Detection allows the severity of security compromises to be limited. In the absence of detection, security compromises could continue to do damage for an unlimited duration. Examples of detection based security would include IDS, log files and others. A very recent example of the importance of this type of security would be the compromise of a Sony Entertainment data base containing considerable amounts of customer data [7], which has been painstakingly analysed to identify potential breaches of privacy.

The research in this thesis is primarily concerned with detection, with the potential for expansion into defence. As such, most of the functionality falls within the remit of network monitoring, the next section of this chapter deals with network monitoring in differing categories.

2.2 Monitoring as Part of Network Management

FCAPS is the ISO Telecommunications Management Network model and framework for network management [8]. It defines five areas of interest, these are:

- Fault Management

  In the context of network management, a fault is an event which has a negative impact on the network. The role of fault management is to, where possible, predict these events. Where prediction and prevention is not possible, fault management locates and mitigates the cause of the fault.

- Configuration Management

  Configuration management involves auditing existing network devices, tracking changes made to configuration and to allow planning for future development.

- Accounting Management

  In the context of this research, accounting management refers to the management of users in terms of the provisioning of authorisation,authentication and backup.

- Performance Management

  Performance management involves measuring metrics such as utilisation, error rate and response times for existing infrastructure. Ongoing monitoring of metrics such as these allow for capacity planning for future requirements.

- Security Management

  Security management is primarily concerned with controlling access to the devices on the network, and to the network itself.

Packet capture can provide information for fault, configuration, performance and security management. However, the research presented here is primarily connected with fault and security management. DoS attacks and worm threats certainly affect performance but in an indirect manner.

2.3 Monitoring Grouped by Data Used

The research described in this thesis utilises data from packets collected on ISP networks. In this section the different types of data available to monitors and their uses are described.

Information can be gained from either data directly contained in the packets contents, or data about them. The data can also be gained from the routers, switches, firewalls, proxies, load balancers and other devices present on the network. In this thesis, the data will be split into two categories. It is possible to record the layer 3 protocol from the IP protocol field; it is also possible to record the time of the packet's arrival. This section has subdivided monitoring by these types of data, which are here named 'Packet' and 'Meta' data respectively.

2.3.1 Packet Data

Packet data can be used for identifying more detailed information such as which applications are present on a network. In a security context packet data is often used as a 'signature' to detect malicious packets.

Tools such as TCPDump [9] and Ethereal (and later Wireshark) [10] are popular for the capture and analysis of packet level data. At higher data rates tools like NetFlow [11] allow sub sampling of packets to provide detailed information without storing huge quantities of traffic. Packet data has the potential to reveal many details about communication on a network. This ranges from the obvious, such as the application flow data, through to the more abstract, such as the operating systems, load levels on the hosts involved etc., [2].

Packet data is often applied in scenarios such as intrusion detection, where particular packet signatures are potentially interesting. Monitoring packet level data can have large overheads, which is discussed later in this chapter.

2.3.2 Meta Data

The term 'meta data' is used in this thesis to define data which is inferred from data, rather than packet fields directly. Examples would include delay, loss and inter-arrival times, which while they are data about packets, they are not contained in the packets themselves.

One of the most obvious examples of meta data monitoring is the use of the classic 'ping' program to gauge delay across network paths using the ICMP echo request and reply mechanism. The application has become widespread in its distribution and application [12].

Routing information can be extracted from a network using tools such as traceroute and tcptraceroute, which use the TTL feature in IP to have packets dropped at each layer three device on the network[3].

[2]Operating systems can be inferred from TTL, and load levels can be inferred from IP identification numbers in many stack implementations

[3]Some administrators view this type of topology discovery as a security vulnerability in itself,

Other examples of the use of meta data would include bandwidth monitoring and loss statistics taken both passively and actively.

2.4 Monitors Grouped by Data Rate

The system developed in this research eventually monitored data rates in excess of 6GBit/s. In this section monitoring networks at varying rates is discussed.

The requirement for monitoring of links has meant that with the increase in bandwidth in the core, a commensurate increase in the capability to monitor bandwidth has been needed. Commercial products are available for the purposes of capturing traffic from links at 10Gbit and above. Monitoring at these varying data rates in more detail here is described.

2.4.1 Monitoring up to 100 Mbit/s

Monitoring at low data rates with current hardware allows a comprehensive level of processing to be undertaken on the traffic with even modest processing power. Full traffic records can be stored, containing all data within packets over significant periods of time.

Simply storing all of the packets at one hundred Mbit/s gives around forty-five gigabytes of data per hour. Standard PC architectures now house hundreds of gigabytes of hard disk space; therefore even on cheap hardware the whole data portion of packets can be stored for many hours at this data rate, however, even with large storage arrays, uncompressed data would be difficult to store for extended periods of time.

2.4.2 Monitoring up to 1000 Mbit/s

Monitoring at gigabit rates can cause problems for off the shelf PC hardware. The problems that IDS systems face at these rates has been discussed in [13], where the authors discuss the load in terms of memory associated with maintaining state on connections, the potential for packet drops due to network load and the potential for exhausting the available CPU resources. These problems are non-trivial to circumvent, and often result in compromises between system resources and detection rates. Common problems of interrupt rates are also discussed in section 4.2.2.

The issues of storing data increase by an order of magnitude moving from 100Mbit/s to 1Gbit/s. As presented in the previous section, storing all packets

and accordingly prevent layer 3 devices from sending the ICMP TTL exceeded message responses

at 100Mbit/s leads to forty-five gigabytes of data per hour. At 1000Mbit/s this figure is four hundred and fifty gigabytes per hour.

2.4.3 Monitoring at over 1000 Mbit/s

With the introduction of 10Gbit core network links, the requirement to monitor at this speed has developed. Monitoring at these rates is usually limited to meta data analysis except in specialised applications [14].

10Gbit NICs are commonly available even for consumer desktop computers, however, these cards are often not capable of coping with the full data rate on a 10Gbit link, and are instead simply compatible with the standard. Commercial specialist monitoring interfaces are available, which give the ability to perform large amounts of processing and filtering on the interface, reducing the load on the monitoring host. Companies such as Endace and Napatech offer products which give very strong capture performance at these data rates [15][16].

2.5 Monitors Grouped by Topology

The system described in this thesis monitored an ISP network in a distributed manner, gathering data and collating it at a centralised point. This section characterises different methods of data collection in terms of topology.

It is possible to monitor at one or more sites on a network. The topology of the monitoring system needs to be matched with the type of data which the system designer intends to extract. The types of possible topology and their prime uses are explained here.

2.5.1 Monitoring at a Single Point

In a situation where packets are broadcast at layer 2, as with a hub, it is possible to monitor multiple hosts on a network from a single point. However, most modern layer 2 devices switch packets, preventing passive monitors from intercepting traffic which is not directed to them.

In a switched environment where monitoring is being performed on the host which is of interest, no extra work is required; however, if this is not the case, it is often necessary to create a span port, mirror port or in the case of fibre, an optical tap [17].

Port spanning generally refers to traffic from multiple switch ports being copied to a signal 'span' port. Port mirroring generally refers to traffic from a single port being replicated to a 'mirror' port. Port spanning would generally be performed on

specific VLANs or ports. Consideration has to be given to the data rates involved, as spanning several 1Gbit ports to a single 1Gbit monitoring session may lead to dropped messages. Even in the case where a single port is spanned, the duplex nature of Ethernet may lead to frames being dropped. A port tap is a physical device which in the case of fibre, splits a proportion of the traffic to a secondary cable, leading to a reduction in signal strength on the original line.

2.5.2 Monitoring by Traffic Splitting

In situations where traffic rates are high (or where redundancy is desirable), it can be desirable to split traffic streams to allow multiple hosts to analyse the data. Often devices such as load balancers can be used. If configured intelligently, load balancers can ensure that related traffic is consistently fed to the relevant monitoring host. This is commonly achieved through 'sticky' session addresses (or SSL certificate, or other method), where a balancer will remember which way a particular host has been forwarded for a certain period of time. This ensures that there is some consistency in which devices receives the data from specific hosts [18].

2.5.3 Distributed Monitoring

On many networks, it is not sensible to monitor all the required traffic at any single point.

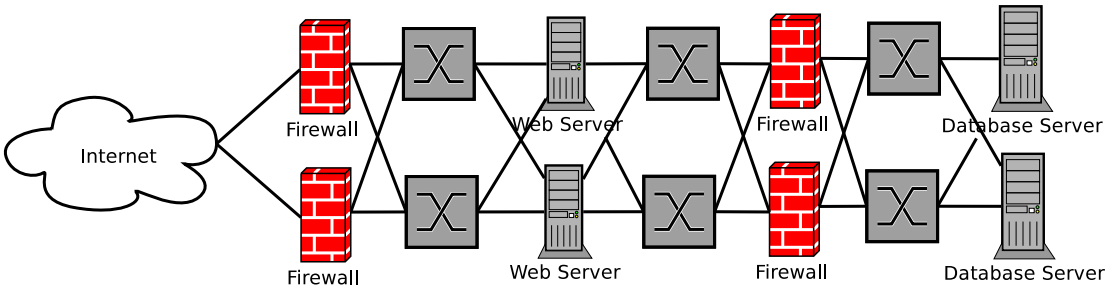


Figure 2.1: Example Network Topology

In figure 2.1 monitoring between the firewall and internet would give all traffic flowing on and off the network but not, for example, traffic from the web servers to the database servers. In this type of situation, it is often desirable to monitor at multiple points on the network, to provide coverage of relevant information.

Distributed monitoring comes with additional problems; multiple copies of the same packet may be captured, leading to confusion in analysis alongside the increase in the storage requirement.

2.6 Monitoring For Security Purposes

The data gathered in this research was analysed to attempt the detection of malicious activity across the network monitored. This was achieved via anomaly detection through statistical analysis of summarised data. In this section, network security monitoring in a broader context is outlined.

As usage of networks has become more focused on commercial interests, the importance of security has increased. The application of network monitoring to detect network misuse has formed a large part of the body of network monitoring research.

2.6.1 Security Monitoring Grouped by Detection Type

While the system eventually developed in this research was primarily anomaly based, there were several potential alternative options available. This section describes some of these options.

NIDSs are designed to monitor a network, host or both and detect events which it classifies as an intrusion. These systems are often closely linked with IPSs which provide some form of mitigation in addition to detection. Intrusion Detection systems are traditionally subdivided into two main categories, those of signature detectors and anomaly detectors [19].

2.6.1.1 Detection Through Signature

Signature based systems are built on knowledge of the behaviour of the intrusions they attempt to detect. This knowledge maybe about the types of process activity the intrusion will exhibit, or perhaps a particular string pattern present in a packet. A good example of this type of detector would be the open source IDS 'Snort' [20].

Signature based systems traditionally provide good detection rates, but require prior knowledge of the misuse they detect, making them weak at detecting new attacks. Certain types of attack, such as polymorphic worms, try to make this type of detection more difficult, though still possible [21].

2.6.1.2 Detection Through Anomaly

Anomaly based systems utilise an understanding of normal behaviour for a host, network or both, to define deviations from this normal as potential intrusions. The definition for normal activity used by the system can be gained via prior knowledge or via machine learning over time.

Analogies can be drawn with credit card processing, where spending patterns undergo similar analysis, and where anomalies are found, payments are stopped.

As with credit card processing, network anomaly detection is prone to false positives and negatives; however, unlike signature detection, it is less vulnerable to issues in detecting newer attacks [19].

2.6.1.3 Composite Detection

Composite detectors can provide more information than a purely signature or anomaly based system, and are often in practice systems which provide a better measure of the quality of the alarm. These detectors are in essence simply both anomaly and signature detector based systems. They suffer, and benefit from all the strengths and weaknesses of their component parts.

2.6.2 Security Monitoring Grouped by Learning Mechanism

The system developed in this research used a combination of self learning and taught mechanisms to classify network traffic. In this section describe these mechanisms are explained.

2.6.2.1 Self Learning

Self Learning systems are based on anomaly detection, and over time build a statistical model of normality, they then spot significant deviations from this normality. As these systems are self learning they have the potential for alerts to be generated for innocuous events.

In situations where systems are entirely unsupervised, extreme care must be taken with the input data, and the handling of alerts. In practice, few systems are entirely unsupervised in a network monitoring context.

2.6.2.2 Taught

Training of IDS systems can be achieved in several ways. In the case of anomaly based systems, this is often achieved through pre-classified data-sets, where the system is taught to classify along a similar line to the training set.

In the case of signature based detection, the teaching is often considerably more direct. Often exact patterns are specified to the system, which then attempts to find them and classify appropriately.

Systems which are programmed can positively generate alerts, as in the case of signature detectors, or negatively generate them in the case of anomaly based systems.

2.6.3 Security Monitoring Grouped by Analysis Technique

This research used summarised information to analyse the data present on the network, however, many systems use other mechanisms. This section describes some of the available options.

Where and what a system is designed to monitor is linked with the level of data available to a system. One of the more intensive tasks when monitoring network data is real-time flow reconstruction. This is both in terms of the memory required to store the connections and associated data, and in terms of the processing required to match packets to connections. The majority of traffic on the network monitored for this research was TCP/IP traffic, which is split into flows by IP:Port pairs. The server port is normally static, with a randomly chosen ephemeral client port.

Flow reconstruction is the process in which packet streams are turned into communication streams. Packets are assigned to particular flows, normally by IP address, Port (in the case of TCP or UDP traffic), sequence numbers and other variables. This can be hardware intensive in terms of both memory and processing requirements as the packet and flow rate increases.

There are two main types of monitor in this category, 'state aware' and 'stateless' detectors [19].

2.6.3.1 State Aware Detection

State aware detectors monitor the flow information contained within the network traffic. This allows a level of information suitable to find connection based anomalies. A good example of this would be the Synchronise Packet exploit within many TCP stack implementations [22], used as a mechanism within a Denial of Service attack. The state monitoring would then be aware of many half open connections.

State Aware detection can analyse entire flows of data, and is therefore able to detect sensitive information which is contained over several packets. A mechanism to avoid detection in a non-state aware system would be to set the MTU for a connection to something very small, forcing all data to be split into small sections, avoiding detection from non-connection aware systems. This could be achieved by settings within the operating system of the attacker's machine.

State aware systems may monitor meta data about connections; an example may be to monitor the inter arrival time of packets within a telnet session, which may enable the differentiation between human and scripted input.

It is also possible (and indeed common) to reconstruct connections at an even

higher level e.g., looking at http headers or ftp transfers.

2.6.3.2 Stateless Detection

Stateless detectors do not keep track of flows, and treat each packet as an individual entity. This allows for higher throughput, at the cost of information availability. In some cases this type of detection is unable to correctly identify attacks; these cases include polymorphic worms and encrypted connections.

2.7 Network Threats

Network Threats (which are also referred to as malicious network events) are the primary focus of the detection in this research. This section describes different types of network threat.

Network threats are varied and numerous, covering a range of activities, from password security on a single host to hundreds of thousands of hosts being involved in single attacks at a target.

To provide a bounded summary, the review of network security threats is limited here, discussions of physical site security at router housing would be inappropriate. Instead the emphasis on threats is with regards to networks rather than hosts. The relevant threats are placed into the following headings.

2.7.1 Worm Threats

A computer Virus refers to code which replicates itself. A worm is an extension of this, and contains a mechanism to spread between hosts.

Worms can be characterised by their ability to propagate from one host to another either directly or indirectly, through their own mechanism, or by means of an existing one such as e-mail [23]. Worms can be classified as having the following stages:

1. Target Discovery

   This can be done through means of active scanning, passively (where the host waits for contact from a vulnerable host) or pre-generated lists either on or off the infected host. It is worth noting that some worms have no target discovery mechanism, and simply transmit to randomly generated targets.

2. Transmission

   Transmission of the worm is achieved either by using some known vulnerability on a target host, using its own transmission mechanism, and gaining

some level of privilege on the target host or by some existing communication mechanism, such as email. Some worms use a secondary channel to transfer the worm code such as blaster [24].

3. Execution (Activation)

   Execution of the worm can be achieved via direct human action, indirect human action (such as a machine reboot), via scheduled process activity or via self execution.

Worm behaviour is sometimes classified as 'direct' or 'indirect'. Direct worms are defined by their use of their own transmission mechanism. They are normally self executing, leading to potentially high infection rates. Direct worms are perhaps the most threatening to a network as they can produce large traffic volumes with indirect DoS type effects on a network.

Indirect worms are in contrast defined by their use of a secondary communication channel such as email or a peer to peer file transfer, and their use of human based activation.

2.7.2 Trust Abuse Threats

Trust based abuse does not fall directly under the remit of this research. Due to its characteristics it does not have a great impact on network performance. Trust based attacks are in essence informational attacks. These attacks attempt to gain some information of value from a user and are often referred to as phishing. This is most commonly through email. Common target information includes credit card details and passwords for valuable accounts such as eBay [25].

2.7.3 Denial of Service

> In a denial of service attack, a malicious user exploits the connectivity of the Internet to cripple the services offered by a victim site [26].

DoS attacks can be costly to victims, perhaps the most famous example being an attack in February 2000, where a large scale Distributed Denial of Service attack was launched against several high profile targets, including www.Amazon.com and www.eBay.com.

The attacks can be varied in many ways; they can use genuine or spoofed addresses, any number of hosts in any control topology, they can use varying data rates and can last from minutes to weeks [22].

The attacks may be grouped by the method in which the DoS is achieved.

2.7.3.1 Flood Based Attacks

Flood based attacks rely on consuming resources through the transmission of large amounts traffic. The crudest form of this is simply sending enough UDP packets (or indeed any protocol, UDP floods were the most common form of attack observed during the research) that the target hosts bandwidth is completely consumed, preventing other legitimate traffic from arriving. This is depicted in figure 2.2, as can been seen, if the proportion of the total traffic represented by DoS is sufficiently high then legitimate traffic is statistically much less likely to be delivered [22]. In this first case, mitigation is relatively simple provided there is sufficient bandwidth available further up stream. Blocking all traffic from the attacker address destined for the victim would allow normal service levels to be resumed.

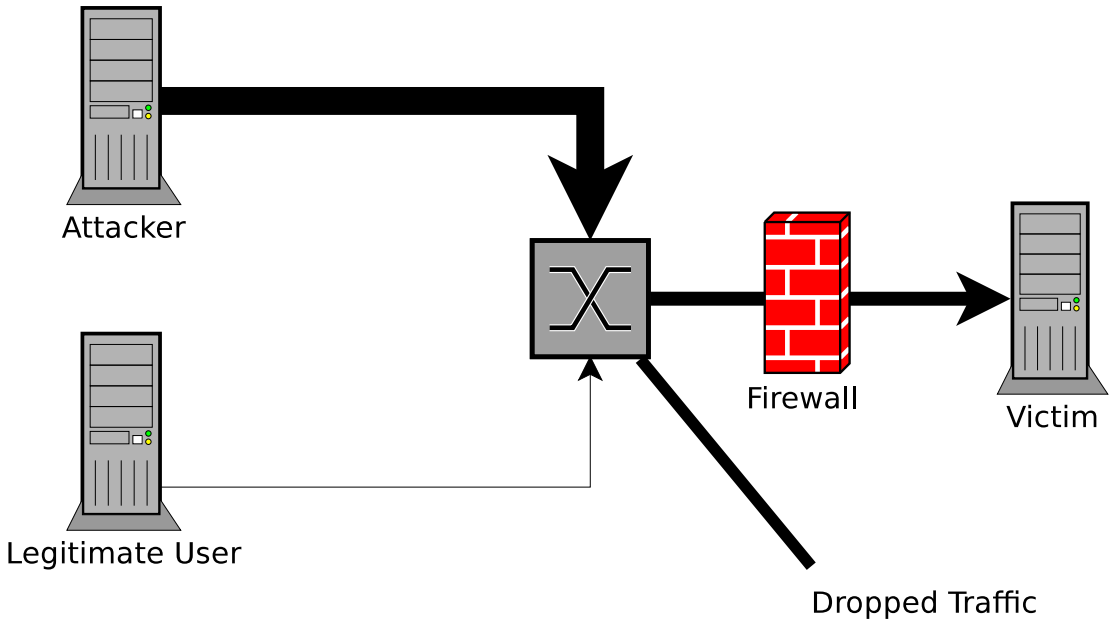


Figure 2.2: A simple flood based DoS attack

This attack can be made harder to mitigate through the use of IP spoofing. Spoofing is a colloquial term meaning forging. A host which spoofs its IP addresses, places another host's IP address in the source IP field of its outgoing IP packets, making them appear to have been sent by a different host. This is shown in figure 2.3.

2.7.3.2 Multiplier Based Attacks

Multiplier attacks are an extension of flooding attacks whereby the attacker utilises some method of increasing the load on the target via a secondary mechanism. An example of this would be to send a broadcast ICMP echo request with a source

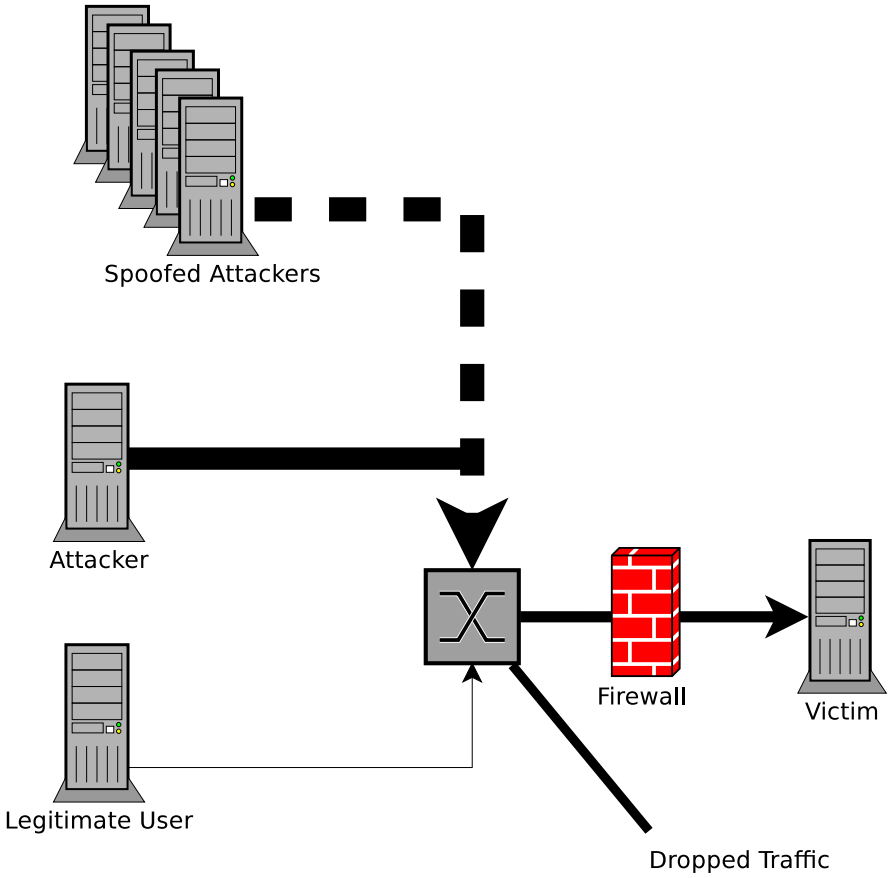


Figure 2.3: A spoofed flood based DoS attack

address spoofed to that of the target. All hosts receiving the echo request would (if not disabled) reply to the target, generating a potentially large multiplier. With a potentially modest amount of traffic sent by an attacker, huge volumes of traffic may be generated, flooding the available bandwidth for the victim.

2.7.3.3 Service Vulnerability Attacks

Service vulnerability attacks make use of some potential multiplier within a service. The classic example of this is the connection limit within most TCP based servers. If a TCP based server allows 15 concurrent connections, and each connection has a timeout of 3 seconds, then by sending 5 'synchronise' TCP packets per second at the target, and not responding past that packet, the connection limit will be permanently reached [22]. Another example of this would be repeatedly calling a processor intensive server side operation within a web server, effectively denying processor time to legitimate users.

2.8 Motivation

Network and host vulnerabilities can provide a wide range of possibilities for criminal individuals and groups. What they gain from this activity gives insight into their likely patterns and targets.

The Honeynet project was n group of people sharing a common interest, which originated in America, who used interactive network monitors to observe and track illegitimate activity on the internet.

From their research they determined a set of motivational factors for which they use the acronym 'MEECES' (a play on the FBI's MICE) [4]. The acronym is made up as follows:

- Money

  Financial gain, whether through extortion or though more direct means such as credit card details.

- Ego

  Hacking remote machines can give a feeling of self-importance to the offenders.

- Entertainment

  Individuals sometimes gain satisfaction by causing embarrassment or turmoil which others (often system administrators) have to attempt to recover.

- Cause

  Some groups have particular interests to advance. There have been attacks directed toward corporations, governments and individuals.

- Entrance to social group

  Most hacker communities are meritocracies and therefore hacking a difficult target is a primary source of social capital.

- Status

  Similar to the entrance to social group motivator, in that social capital is gain through the demonstration of technical skill. While this set may be somewhat limited, and only briefly discussed here, it is more than sufficient for the purposes of this thesis.

While this set may be somewhat limited, and only briefly discussed here, it is more than sufficient for the purposes of this thesis.

2.9 Mitigation Techniques

There are various means by which security threats may be detected and mitigated against. Mitigation techniques are particularly successful where the behaviour of malicious activities are clearly defined, consistent, and filterable. Where attacks are polymorphic (in the case of worms), large scale or widely distributed mitigation is more difficult. This thesis will progress to discussion on how data mining and packet summaries may be used to inform packet filters, allowing useful dropping of DoS traffic.

A primary concern with mitigating against malicious activity on the Internet lies in the changing threat landscape. At the instantiation of this research, direct worms were considered by many to be the biggest threat to the Internet as a whole. Since this date the main sources of computer infections have moved toward indirect threats such as mass mailing worms[27]. A botnet is a series of compromised computers, with a mechanism for disseminating control between them. Many botnets now exist, some comprising of millions of individual computers[28][29]. Some of these botnets are used for generating spam email. A description of the Statcheldraht botnet tool is given in section 5.2.6.1.

2.9.1 Mitigation Grouped by Action

There are a multitude of mechanisms to mitigate against a multitude of threats, here the techniques are split into ‘proactive’ and ‘reactive’.

2.9.1.1 Proactive Mitigation

A good example of proactive DoS mitigation was highlighted in the national press[30]. In this case a business website was targeted through DoS for extortion purposes. The owner of the site refused to pay the demanded fee, and risked further DoS attacks. Following a series of emails, the ransom was increased to $50,000 (£28,000) Instead of paying the ransom, the owner of the site had his traffic routed through a third party, specialising in the filtering of DoS traffic.

Spoofed based attacks can be actively prevented via ingress (sometimes called egress) filtering[31]. This technique prevents traffic with a source address which does not belong to a network from leaving that network. It requires an altruistic stance from network operators, and is not universally deployed.

Another example of proactive mitigation occurred when the blaster worm targeted Microsoft's automatic updates server. The worm was captured and the target noted. Microsoft removed the DNS record for windowsupdate.com on August 15th, 2003 [32] preventing the worm from being able to resolve the intended target to a reachable network address.

2.9.2 Reactive Mitigation

The traditional methods for dealing with DoS often involve tracing the sources manually and administrators filtering the traffic out at source. This can be both time consuming and costly, and therefore is only a solution for corporate scale organisations.

Port blocking is a common method of preventing the spread of worm traffic, though it is not always possible[4]. The network that was monitored in this research has blocked inbound traffic on the following ports 135 (TCP), 137 (UDP), 138 (UDP), 139 (TCP), 445 (UDP and TCP), 593 (TCP), 1433 (TCP), 1434 (UDP) and 27374 (TCP); all in response to vulnerabilities and potential vulnerabilities in services which by default host on these ports [5]. It is perhaps worth noting that some of these ports were blocked pro-actively, an example being ports used for Windows local file sharing, which then later prevented worm spread when exploitation techniques were discovered on those services.

2.9.3 Mitigation Grouped by Location

Mitigation of certain activities requires the technique to be applied at a particular position in the network. In this section some common mitigation locations are

[4] Blocking port 80 on a commercial broadband network would in all probability be unpopular amongst the customer base.

[5] For a full description of the monitored network, see Section 4.2.1

detailed.

2.9.3.1 Single Host Mitigation

Mitigation of DoS attacks such as simple flooding can be impossible on-host. Some protection from service vulnerability attacks can be possible through tuning options in software. As an example, it is possible to reduce the default TCP connection time-out, making the host less vulnerable to Syn floods.

Mitigation of worm traffic is normally achieved via a combination of host based firewalls and anti-virus software. The most readily available host based firewalls function by blocking unused ports, which is of limited use defending against attacks against a service.

More sophisticated firewalls use techniques as described in section 2.9 to block specific packets or connections.

2.9.3.2 Local Area Network Mitigation

Local networks were traditionally designed with a model of '*crunchy on the outside, chewy in the middle*' [33]. This is in reference to the fact most local area mitigation is achieved at a gateway. While it may seem a fallacy to only protect a network at a single point, the traditional view of network threats originates outside the local network. It made sense historically to defend a network at the single point of access, thereby protecting all machines while only having to secure at one point.

With the increase of laptop use and the increase of secondary attacks (attackers entering networks via email and other conduits) this method of network defence is becoming less effective.

2.9.3.3 Wide Area Network (WAN) Mitigation

Due to the larger data rates in WAN networks, many carriers consider themselves simply carriers. This means they do not take responsibility for the data which is present on their network. Consequently, their interests lie in mitigating potential threats to the network itself. WAN mitigation is extremely limited.

2.10 Summary

This chapter has presented an overview of network monitoring, describing the application of network monitoring to management and security. It has discussed the various topologies and data rates which may be encountered, and discussed the challenges associated with monitoring at high bandwidth sites. Various network threats have been described alongside some of the common methods for detecting

and mitigating against them. This chapter described how DoS attacks vary in terms of target, vulnerability and mechanism.

The next chapter will present research into the problems associated with network security monitoring, with a focus on the detection and mitigation of DoS.

Chapter 3

Related Work

In this section other related research in the field is examined, and the implications are discussed. Security network monitoring has become a broad research field, drawing from many related subjects. This chapter presents some of the research most relevant to the thesis.

Much of the research can be allocated into one of four categories; denial of service, feature detection, traffic capture / processing and mitigation.

This research was heavily involved with capturing traffic from a high data rate network. There is a large body of research in this field, section 2.4 outlines some of the major concepts.

3.1 Denial of Service

Denial of Service is one of the most common threats to network resources. Research into Denial of Service became focussed in February 2000, when several high profile targets such as Yahoo, Amazon, Buy, CNN and others were hit with traffic rates of around 1GB/sec.

In [22] the authors classify DoS attack using several metrics. These classifications are shown in figure 3.1.

In 2006 Moore et al. investigated how common DoS activity was on the Internet[34], this was achieved by monitoring IP space for responses from DoS victims to spoofed DoS attacks. They show that the number of DoS attacks with short durations is significant which corroborates the results presented in this thesis; the number of attacks observed by Moore et al. was around 20-40 unique victim IP addresses per hour. The research was limited to detecting spoofed DoS, which becomes less common as ingress (sometimes egress) filtering is applied by more network devices.

Flood based attacks are becoming more sophisticated; In [35] the authors de-

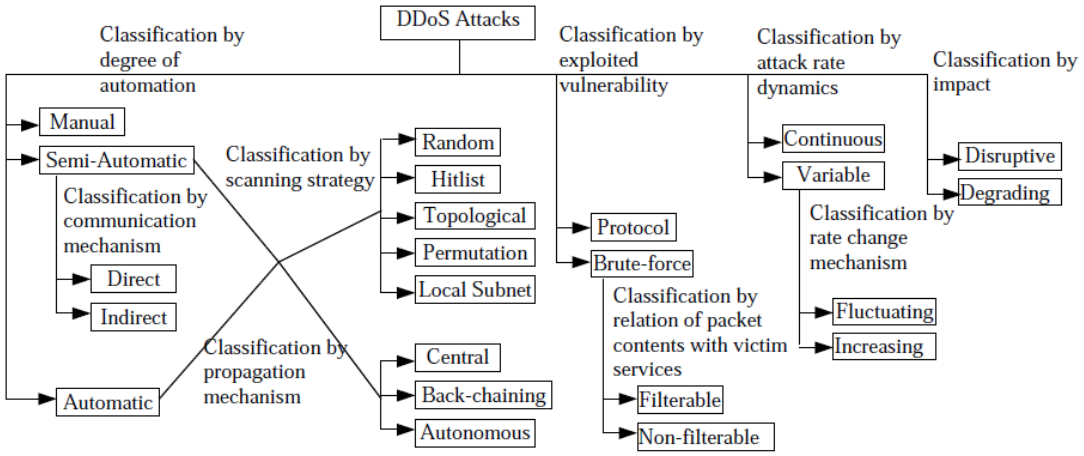


Figure 3.1: DoS Classifications. Figure taken from [22]

scribe the use of intermediary, unwitting hosts to generate traffic directed at a victim. This was achieved by spoofing the address of the victim, and sending traffic to a 'reflector' host, which in turn replies to the victim's IP. This can been seen in 3.2.

Ingress filtering is a good counter measure to this type of attack, as spoofing of the source addresses becomes much more limited. The authors note that there may be the possibility of application level reflectors, but that these would be easier to filter.

Wireless networks present a particular challenge due to their inherent broadcast nature. In [36] the authors present a survey of DoS attacks on common public WiFi locations such as airports, coffee shops and university campuses. They also discuss some of the countermeasures available.

Denial of service is not limited to the Internet. There has been interest in the potential damage caused by DoS attacks in sensor networks [37] and the potential risks specific to them. More recently D. Raymond et al examined the attacks specific to sensor networks, which target their susceptibility to denial of sleep style attacks [38]. Sensor networks have limited resources, as with all systems, but unlike many networks they are battery powered. This leads to unique vulnerabilities.

DoS provides differing challenges for differing areas of networking. Voice Over IP (VOIP) has become a recent area of interest, and has its own vulnerabilities to resource based DoS. VOIP uses the Session Initiation Protocol (SIP), and in [39] the authors examine the potential attack vectors present in the protocol.

DoS is not limited to traditional computer networks. Attacks have been discovered utilising vulnerabilities in mobile phone 3G technology. In [40] the author describes an attack against the IMS (IP Multimedia Subsystem) framework, ef-

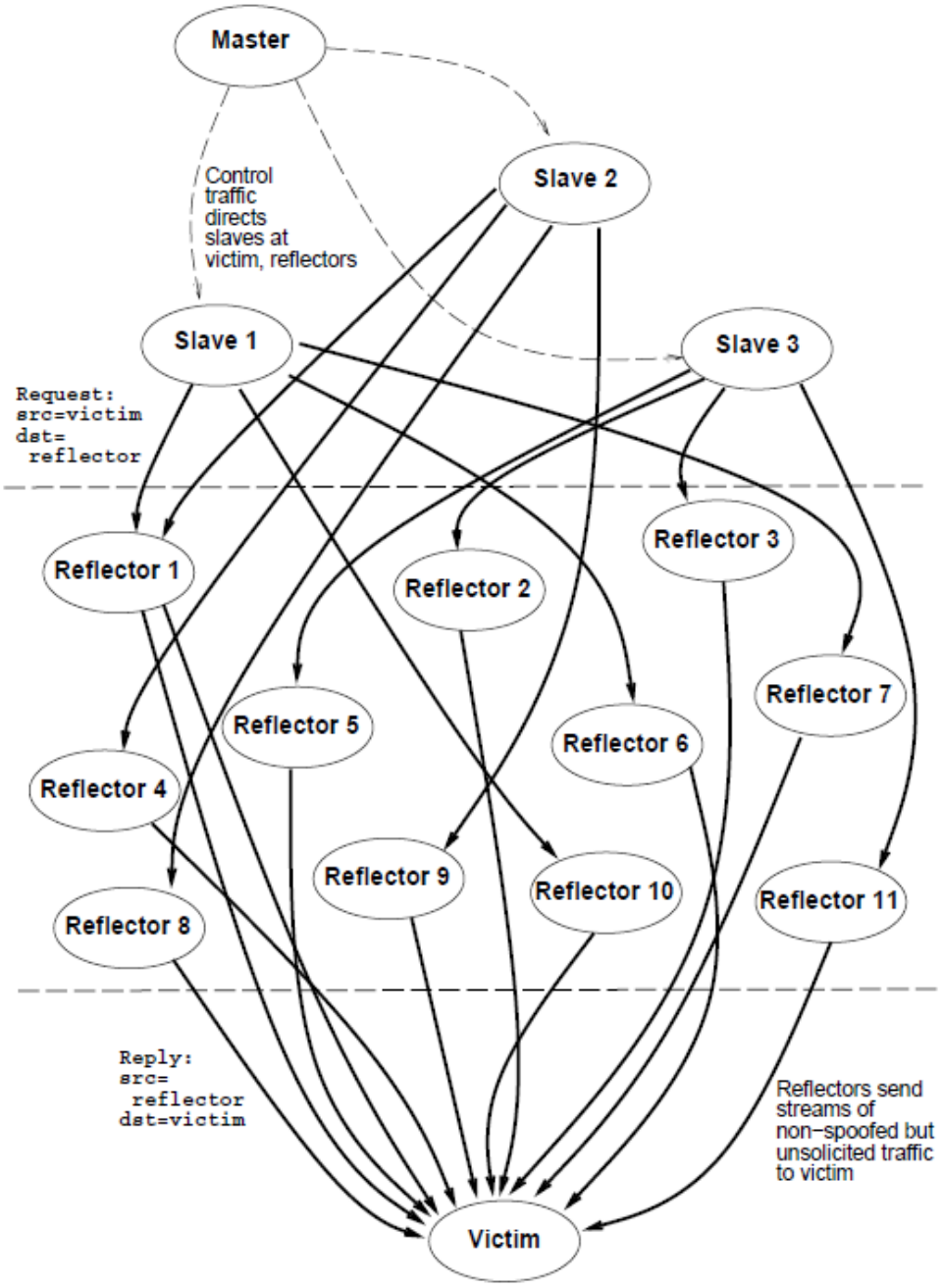


Figure 3.2: Reflector Attacks. Figure taken from [35]

fectively rendering networks using it alongside SIP (Session Initiation Protocol) unusable, with relatively small volumes of input traffic.

Tools for generating DoS are many and varied. The tool used to generate DoS traffic in this research was Stacheldraht, the authors of [41] compare several distributed DoS (or DDOS) generating tools. The tools examined were Trinoo, TFN (Tribe Flood Network), Stacheldraht and Mstream. Each tool uses a hierarchical topology, containing one or more attackers, a sent of handlers and potentially large amounts of agents. The agents are responsible for generating the attack traffic. Control of the handlers is achieved by a variety of methods, including telnet style terminals through to blowfish [42] encrypted ICMP commands. A more thorough examination of the DDOS tool used in this research can be found in [43].

3.2 Traffic Capture and Processing

There are many topologies at many scales present in the Internet today. The desired data defines where in this set of topologies we should monitor traffic. The differing locations for monitoring can be thought of as following one of two branches, Core monitoring and Local monitoring. The main distinctions between core and local branches are data rates and heterogeneity.

The problems of dealing with high data rates, especially when in relation to IDS type applications have been at length by several sources; Holger Dreger et al. describe trading off resource against detection rates, and the problems with memory and processor management in differing applications [13]. They discuss difficulties in several areas; firstly in dealing with the large number of packets per second. Secondly, in coping with the traffic diversity found in high volume networks, and thirdly, attempting to maintain state for the monitored connections. They name the primary difficulty for stateless IDS systems as the processor consumption, and for stateful IDS systems, the memory management. They further present several strategies for compromising detection rates to improve the ability to cope with the rate of data.

These problems have been addressed in many different ways. Sometimes through summarisation, sometimes, as with [44] it is done by splitting the traffic intelligently to allow processing by current hardware. The key research in the traffic splitting methodology is to maintain the consistency of the data between ‘slices’. This technology is similar to that of a load balancer, which are often deployed in other environments.

H. Song et al. develop an FGPA based solution to signature detection, and apply the novel use of a bit-vector algorithm [45]. The signatures were taken from the snort database [20], and therefore classification accuracy was commensurate with

snort's performance, more importantly though, signature detection was achieve at rates of a gigabit per second; the authors suggest that through improved hardware 10Gbit/s could be achieved.

In [46] the authors use a variety of capture mechanisms (Endace cards, standard NICs and FPGA based solutions) which allow for sampling or filtering of the data. This approach has been used in applications such as IDS and probabilistic IP traceback. Network cards produced by Endace and Napatech allow for high data rate capture by mapping the network card memory directly into user space. This technique has also been implemented by the open source driver modification pf_ring [47], where the community have achieved competitive packet capture rates with commodity hardware.

3.3 Feature Selection and Intrusion Detection

In local monitoring, signature detection is used as an effective way of generating very high classification rates, with low false positive results. The strength of signature detection rests on the consistency and the accuracy of the definition of the signature. Developing signatures can be a time consuming tasks, even for the most skilled network analysts. This problem has been addressed by the authors in[48] through the use of honeypot data and associated processing, to generate accurate signatures without interaction from network administrators.

K. Wang et al. use the probabilities of specific payloads occurring for a given application to derive an effective measure for detecting network abuse [49]. They show impressive detection rates when applied to the DARPA IDS evaluation data set.

The theme of feature reduction is taken up by the authors in [50] where various statistical methods are used to determine which features should be used for classification.

T. Lekie et al. have examined the use of metadata to detect misuse in an encrypted environment [51]. This shows the possibility of using abstract data to infer misuse in a network. It is primarily aimed at detection on a local network level. Similarly, Seong Soo Kim et al. have described the use of wavelet analysis as a tool for the detection of denial of service activity from aggregate packet headers [52]. This research again shows the potential for using non-traditional algorithms, dependent on higher level information to detect misuse. Recently significant research has made use of PCA (principle component analysis) to detect anomalies in network traffic statistics. This has been applied to NetFlow statistics to detect various types of anomalies including DoS, Flash crowds and worm activity [53].

In 2008 the authors of [54] discussed the issues with using traditional Netflow for the detection of events such as DoS. They propose changes to the Netflow software to allow the graceful degradation of sampling levels to mitigate the increasing flow rates associated with such events.

S. Zanero et al. develop a system using clustering purely based on unsupervised learning. This is unusual as it considers the payload [55]. Related to this research is an overview of feature selection given by I. Guyion, where she discusses the problems and solutions for handling data mining in fields where the number of variables is very large [56].

Seong Soo Kim et al. present a system analysing egress traffic on a network. They use develop several metrics which indicate the presence of DoS traffic. They use destination IP address and port numbers as potential indicators, and discuss the possibility of using other header variables [52].

A significant problem for network analysts is dealing with the potentially large set of alarms generated by IDS systems deployed on the network. Some of these alarms require action, others do not. The authors in [57] use data mining principles applied to the alarms generated by IDS systems from previous investigations to determine the appropriate actions needed for new ones. This was primarily used to filter false positive results.

The authors in [58] use Kolmogorov complexity as a metric for DoS detection. They conclude that metrics such as this are favourable to packet counting as they are insensitive to background legitimate traffic.

G. Carl et al examine the effectiveness of several methods for DoS detection in [59]. They evaluate common methods, but of most interest to us here, they look at wavelet-based signal analysis. They conclude that the methods in isolation are insufficient to completely address the issues.

3.4 Intrusion Prevention

Maintaining service during a DoS attack is still a challenge; many approach the problem from the receiver end, filtering the traffic from legitimate requests. This allows the host to sustain availability providing the bandwidth is available to prevent excessive loss of legitimate traffic. In the case of [60], this is done via identifying legitimate traffic via an http redirect and authentication code (formed from the client IP) and then providing a QoS system implemented upstream of the victim firewalls.

A method for determining the source of spoofed DoS traffic was suggested by S. Savage et al [61]. In this system packets are statistically marked by routers. Over time spoofed traffic can be traced back to the nearest router to the source.

This is useful as it allows the offending traffic to be blocked closer to the source, meaning less legitimate traffic would be affected.

Jelena Mirkovic et al. present a system to filter DoS at the source, via informing router policies to drop DoS before it reaches the target host [62]. In [63] the authors take a similar approach using a congestion policing feedback mechanism to flow the filtering of traffic to access routers.

A example of a concept is employed by R. Mahajan [64] where routers may detect offending traffic and limit its flow. The routers may inform upstream routers of this, and those routers can then apply a similar rule, leading to the traffic being dropped closer to source, and therefore with less impact on the target.

Some researchers have used overlay networking principles to help filter denial of service attacks as with [65] and more recently [66]. In this research the authors use an overlay to abstract the location of the victim, such that the attacker cannot directly access it. Instead they must go through one of several entry points onto the network. These entry points filter traffic by validating the incoming messages. Due to the distributed nature of the access points onto the overlay, an effective DoS attack would either need to locate the actual IP address of the victim, or flood all access points onto the network, a potentially difficult undertaking.

Some researchers have advocated an approach using micro payment topologies as a defence for DoS [67]. In this paper they describe the use of Client Brokers and Server Brokers sitting between Attacker and Victim. These brokers pass a micro payment transaction when the attacker wishes to connect with the victim, effectively acting as an authentication proxy. The aim of this research was more social engineering than software, in that the aim would be to give the owners a vested interest in ensuring their machines are not taking part in DoS or other unwanted activity by linking all use to a financial cost.

3.5 Summary

This chapter has presented a selection of research connected with network attacks, the processing and capture of data at high bandwidths, the application of intrusion detection to network traffic and the mitigation of attacks on a network. The research related to traffic capture and processing covers the challenges associated with memory and processor consumption along with potential ways to address these problems such as FPGA based hardware and traffic coalescence.

The research presented in this thesis takes the concept of traffic summerisation, and attempts to provide a mechanism for determining appropriate summerisation metrics. The next chapter will describe the architecture deployed for the purposes of this research, and discuss the various choices made in order to address the

significant and varied problems encountered.

Chapter 4

An Infrastructure for Data Capture

As discussed in Chapter 2, gathering data in high bandwidth environments is a challenge. In order to provide statistics in a distributed, scalable manner an architecture needed to be developed. This system would need to capture up to 12Gbit of data per second, across 6 remote points of presence. There would also need to be a mechanism for grouping and analysing data from disparate sites.

In this section the hardware, software and system implementation used to capture and process data is described. The system has been implemented within the core of a National network and began operating in June 2005. This thesis considers data in the period from September 2005 and April 2006.

4.1 System Design

4.1.1 Introduction

This work presents a novel approach to detecting significant network anomalies using summarised data gathered from the header portions of packets. There are two distinct difficulties when monitoring network traffic in the core. The first relates to the hardware and system structure required to monitor at the high data rates found in Wide Area Networks. This issue has been discussed in chapters 2 and 3, and there are many approaches to addressing the issue of high data rates.

The second concerns legal issues that may be encountered when monitoring communications. This area of law is largely untested in the courts and is a complex issue because of the international nature of the Internet. A discussion of these issues can be found in [68]. The guidelines followed by the researcher for the purposes of this work, suggested that privacy laws may preclude the use of the data portion of packets,

> "The United Kingdom, however, took a more restrained view and examined whether the data user could actually link the information to a specific person."

[68]

4.1.2 Architecture Overview

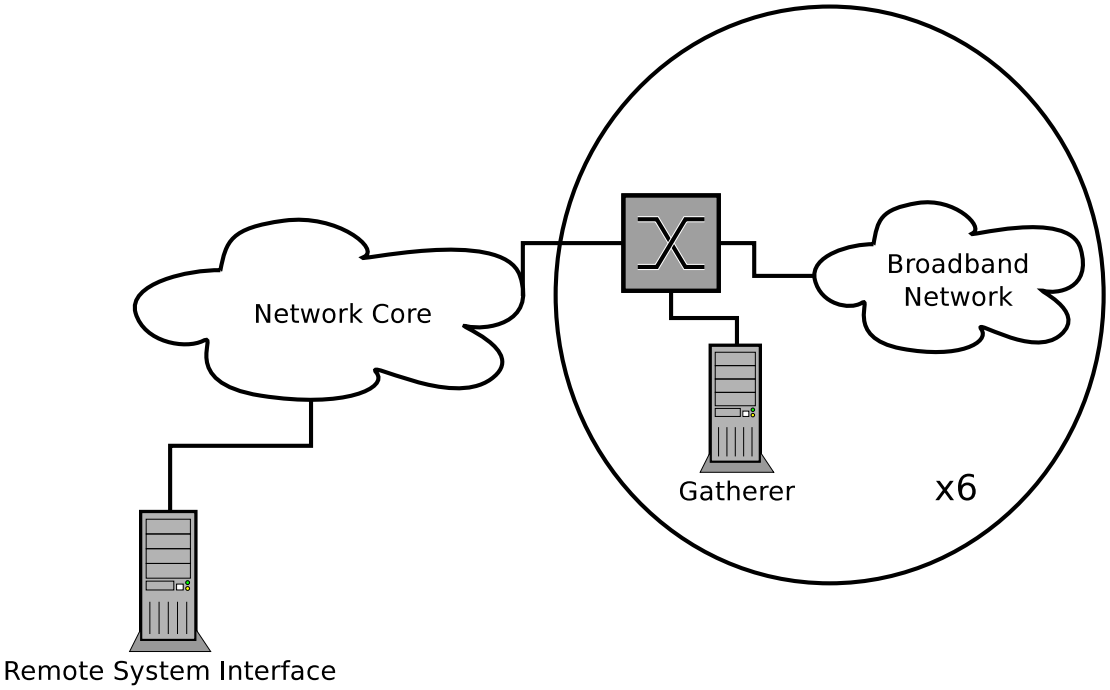


Figure 4.1: System Topology

Figure 4.1 shows an abstract topological view of the system. It can be split into two major components; gatherers and a central processor. The gatherers are spread throughout the network, while the central processor sits at a single site. The system data gathering is distributed to access the data in its natural path. This is to avoid unnecessary network load in re-routing the data elsewhere.

The physical locations of these devices (as shown in Figure 4.2) were

- Bromley
- Brighton
- Colchester
- Northampton
- Oxford

- Southampton

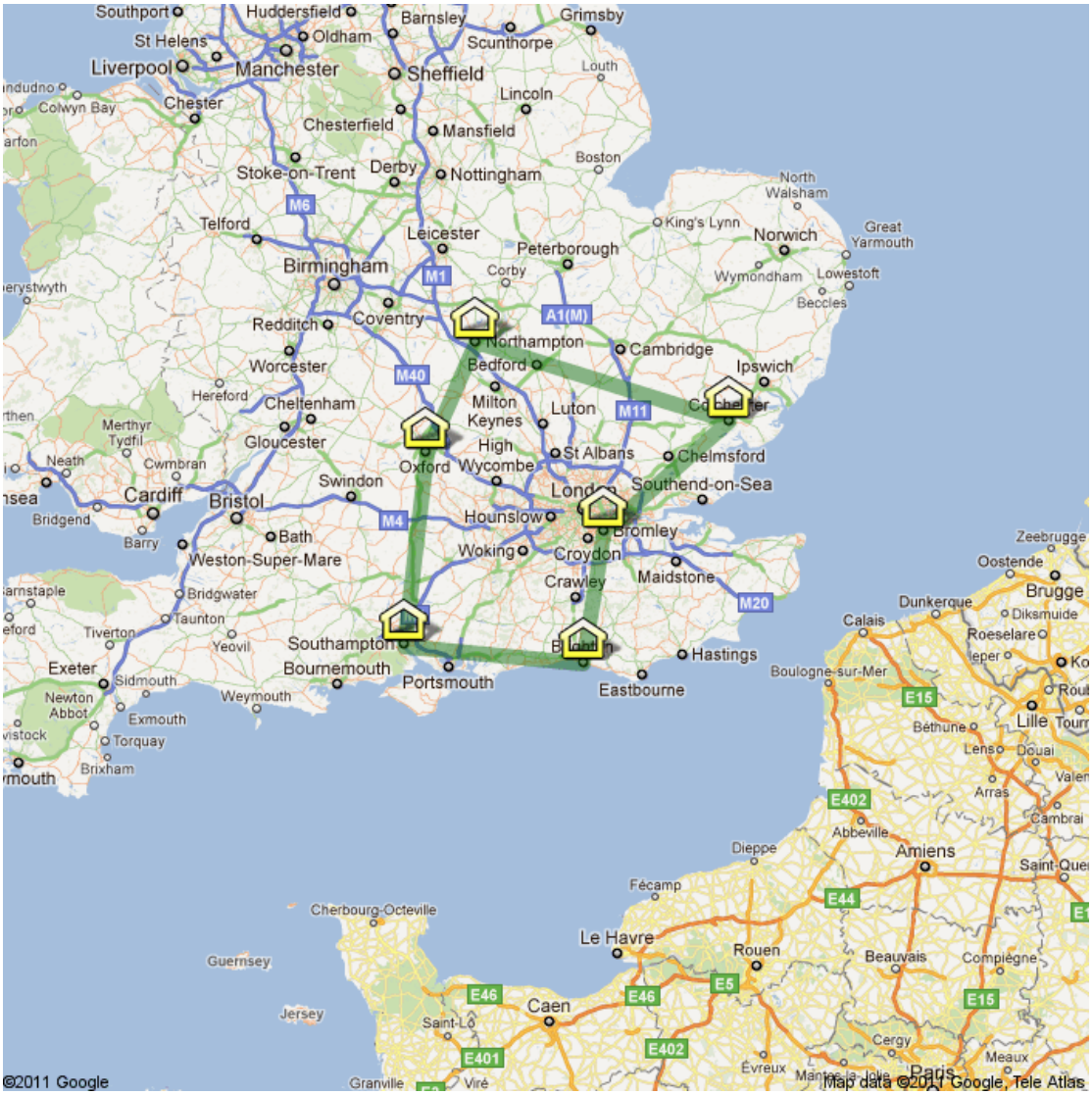


Figure 4.2: Geographic System Topology

To effectively monitor in a core network the traffic needs to be directed to the interface where the capture will take place. In some cases this can be non-trivial, especially when dealing with optical links[1].

The capture of traffic for the purposes of this project was achieved via port spanning mechanisms. Specific VLANs were spanned to a switch port which our gatherers monitored. This allowed us some control over the level of traffic observed in the early stages of the project. Once the gatherers had been fully established, all non-management traffic VLANs were spanned.

[1]When splitting an optical line, there is an inherent loss of signal strength (this can be compensated for by boosting the signal), which may, or may not be significant for a specific path. Any optical splitting would also require a link to be unavailable for a time.

The software on the gatherers' pipes summarised statistical data back to the central processor where it was stored in a database and classified. It was essential that the summerisation be carried out on the local gatherers, as the load of transferring traffic, and the load of processing it, is an unrealistic overhead for both the network, and the central processor.

4.1.3 Data Rate

The research described in this thesis involved monitoring at significant data rates, incorporating a large numbers of hosts. At the inception of the project, monitoring at 1Gbit rates was a technical challenge in itself. Therefore careful consideration was necessary to determine what hardware and software was necessary to monitor potentially up to 12Gbit/s distributed across a network[2].

4.1.4 Data Availability and Permissibility

While it was technically possible to capture the entire contents of a packet within the core a wide area network[3], there were restrictions that needed to be taken into account. A key consideration was the legality of capturing the data portion of the packet. The data contained in the packets traversing the network are subject to UK data protection and privacy laws. Although this is a relatively grey area as no case law exists on the use of this data, advice given by the ISP involved was that for third party monitoring, only the fields of a packet required by the network to route it are legally acceptable for inspection by a third party. In this case, the third party refers to a body who is not the user, the recipient or the network operator. The information this describes would include all IP header information, TCP header information, and for instance the SMTP 'RCPT TO' field, but not the SMTP 'Subject field'[4]. This is enough information to allow the various devices involved in routing the messages to deliver them to the intended service on the intended host. Due to obvious sensitivity in monitoring a commercial network, it was perceived as necessary to store the minimum information possible, while still meeting the requirements of the research. This minimum was taken to be the headers at layers 2, 3 and 4 (OSI). The metrics chosen are covered in detail in Chapter 5.

[2]There were 6 sites, each with two gigabit links, leading to a maximum rate of 12GBit/s

[3]Disk space may require this to be stored only for short periods of time.

[4]Interestingly it is currently considered acceptable to store the 'Mail From' field even though technically it is not needed to deliver the mail.

4.2 Hardware Design and Testing

In this section we look at the requirements and solutions found in dealing with the potentially high data rates, and their effect on the hardware used. This research was based on a relationship with a major UK ISP, and a modest budget (considering the scale of the task). With this in mind the architecture was based on standard server PC hardware.

4.2.1 Network Context

There were 6 points of presence (PoPs) within the monitored network, with data rates ranging from 500Mbits/s per second to 1.4Gbits/s with around 20-30 TB, and 100,000,000 connection attempts being processed per site daily. When the project began these data rates were considerably lower; some sites had peak rates of 200Mbit/s. It is a testament to the changing landscape of network monitoring that the data rates have increase by as much as a factor of 3 within a 2 year period.

The network monitored was a commercial home broadband network with around 30,000 hosts per site visible on the network at any one time. Under DoS conditions the number of concurrent[5] separate IP addresses reached hundreds of thousands.

4.2.2 Data Rates

Monitoring at rates of a Gigabit per second can require specialised hardware, as with many of the systems described in section 3.2. Given that the focus of this research was to use lightweight summarisation, the hardware requirements needed to be examined. The main problems for hardware involves bus transfer rates and interrupt coalescence. The software problems lay in handling memory and processing requirements in a feasible way. Data rates of one Gigabit per second equate to approximately 160,000 packets per second, assuming an average size of 750 bytes[6]. Consequently, any inline per packet processing must be carefully considered. One of the first activities undertaken was to determine the capture rates that could be achieved with off-the-shelf server PC architectures. Some simple testing was done in the laboratory using a number of 3GHz Intel Xeon servers with Intel Pro 1000 network interface cards. To discover what packet rates the system would be capable of dealing with, a machine was set up to generate

[5]In this context concurrent is taken to mean within the current summary window, which is usually 2 to 3 seconds.

[6]A poor assumption given the operation of TCP, however, for the purposes of 'ball park' calculations, not unreasonable.

UDP packets of a fixed size while another machine ran the libpcap based program TCPDump to capture the packets. TCPDump had been compiled with the PCAP_FRAMES = max option to give it the maximum buffer available in the kernel, this had been found to give the optimal results in previous tests. The IP packet sizes ranged from 1500 Bytes (1472 Bytes of UDP Data, with 28 Bytes between UDP and IP headers, this does not include the Ethernet Header) to 250[7] Bytes, and the number sent and received were counted.

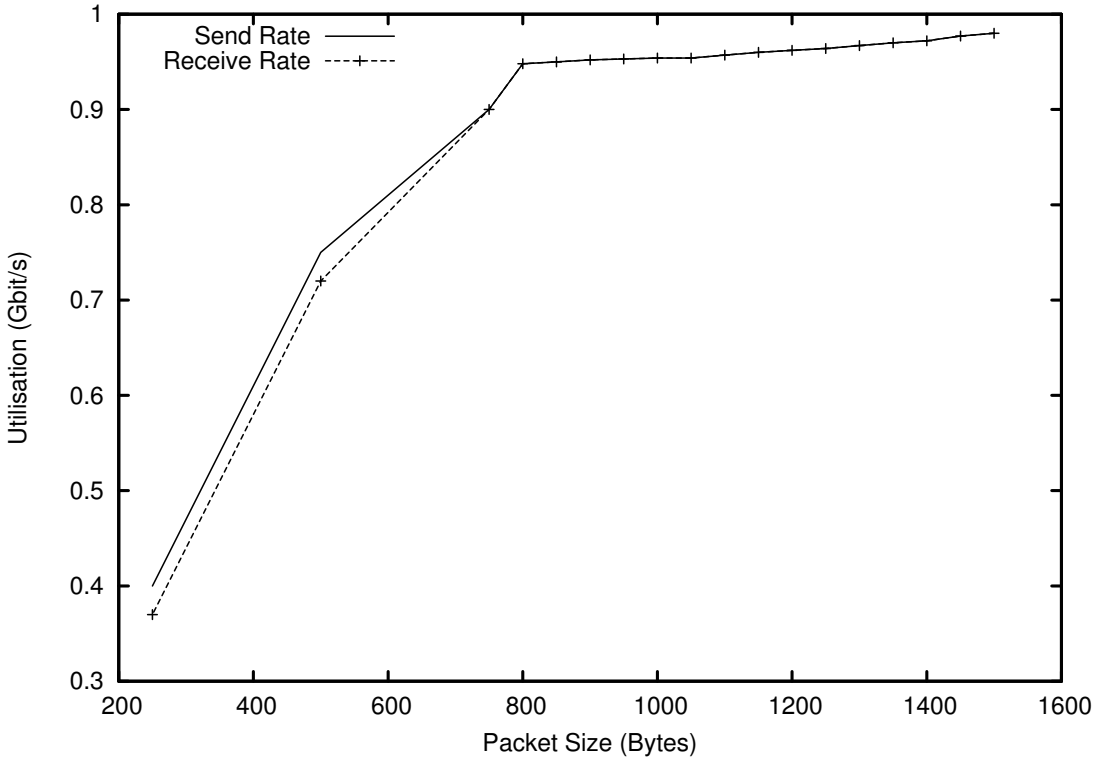


Figure 4.3: Data rate against packet size

As can be seen from figure 4.3, at higher packet sizes the send and receive rates are close to the one Gigabit mark, and are equal. When the packet sizes reduce, and therefore the number of packets increases, the sending and receiving machines struggle to maintain the high traffic rates (Figures 4.3 and 4.4).

Further tests were done using equal numbers of smaller and larger packets, averaging to defined rates. These tests showed favourable results when compared with the single packet size ones, a brief table of these is shown below.

The issues on the network card are clearly caused by packet rate, rather than data rate. Each time a packet enters the network card an interrupt message is generated, requiring the processor to spend time copying the packet from memory

[7]Smaller packet sizes were produced, but lead to much lower data transmission rates (I.e., the machine generating the traffic could not produce packets quickly enough) and therefore were excluded from further analysis.

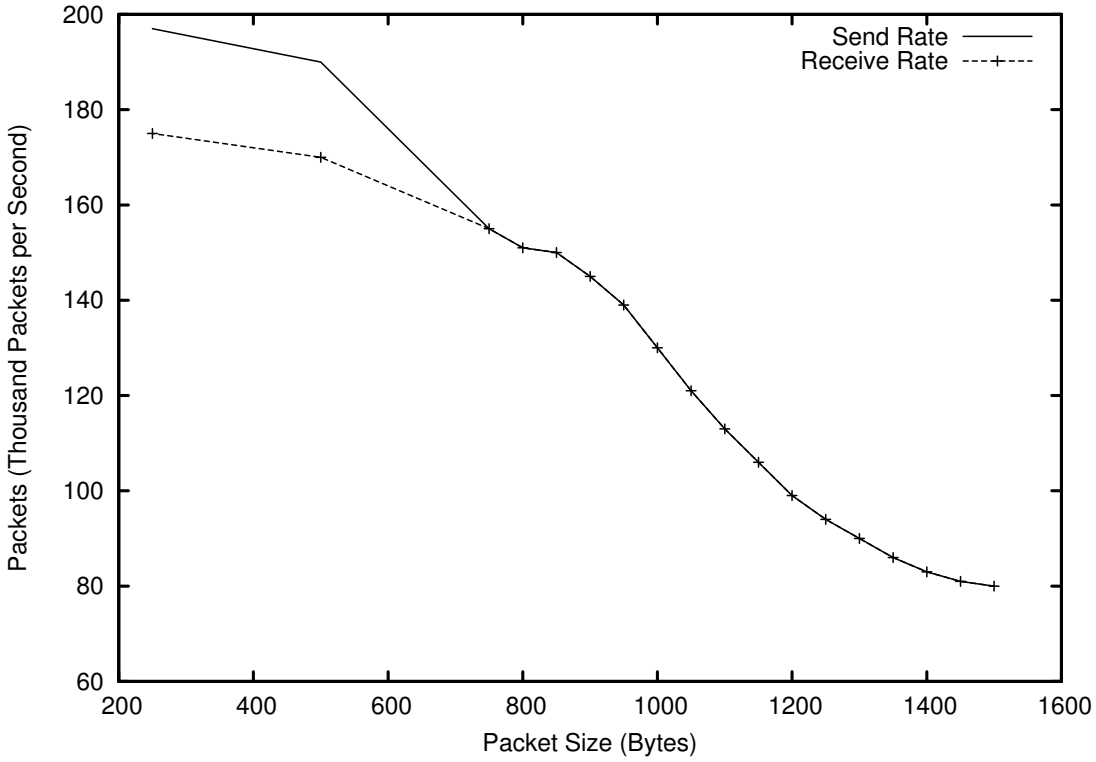


Figure 4.4: Packet count against packet size

Table 4.1: Data Transmit and Receive Rate

Average Packet Size	Sent	Received	Send Rate	Receive Rate
850	1460395	1460395	0.993069	0.993069
600	1678911	1678855	0.805877	0.80585
450	1797681	1782958	0.647165	0.641865

on the card into kernel space. In situations where there are many packets arriving per second, the processor can enter a state sometimes known as interrupt thrashing.

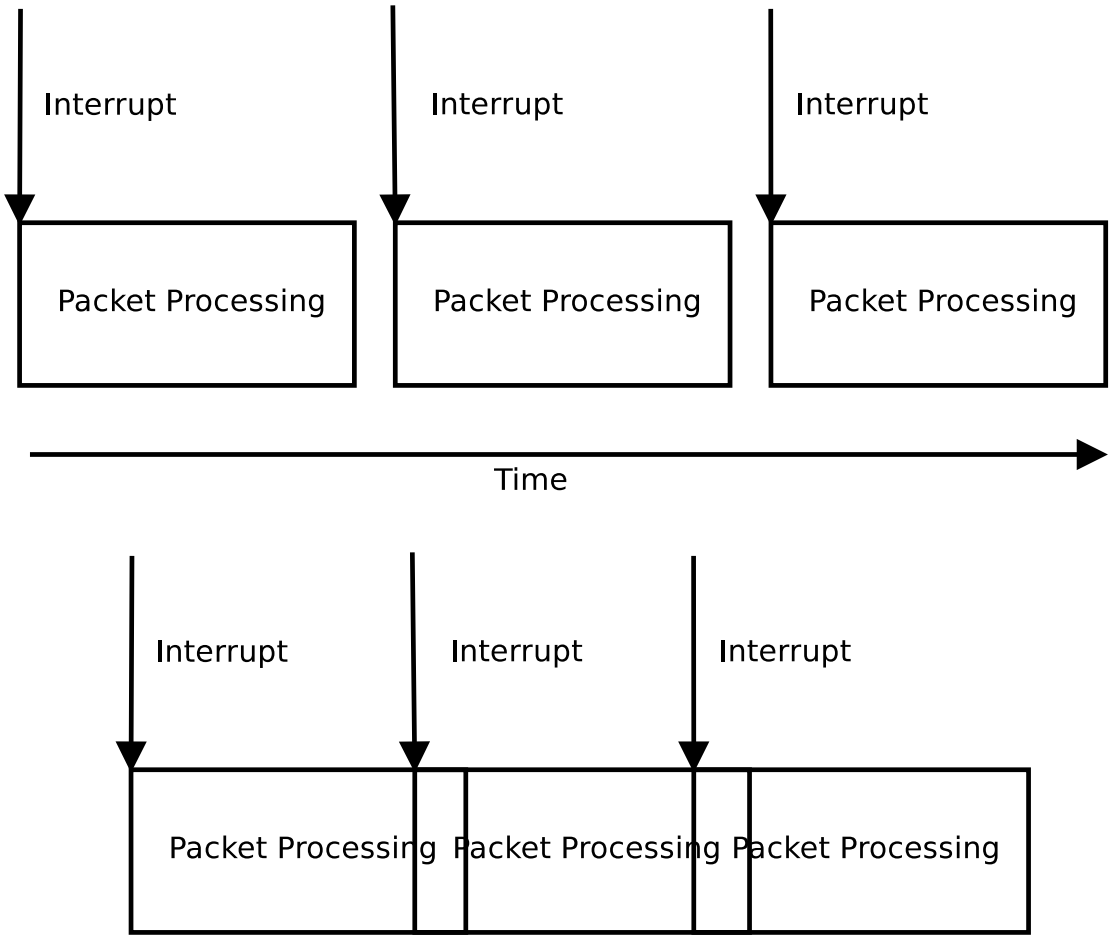


Figure 4.5: Interrupt Thrashing

Figure 4.5 shows the concept of interrupt thrashing pictorially. In the first situation the packet inter arrival time is sufficiently great that the processor has time to make the memory copy before the next packet arrives. In the second scenario the packet inter arrival time is too small to make this copy, and therefore the processor does not complete the handling of the first packet before the second arrives. If this packet rate continues the processor never has the time to recover and process any packets at all.

It is possible to mitigate this type of scenario. Many network card drivers utilise coalescence of packets on the network card when packet rates pass certain thresholds. This allows single interrupts to be generated for several packets, giving a single copy operation. For the purposes of this research, the latency between a packet's arrival and processing was not as critical as in most applications. Modifying the RxInt parameter provided as part of the Intel e1000g driver family allowed

for the tuning of the interrupts to reduce the context switches required to move the packets from the network card into user space.

Since the release of the 2.4.20 Linux kernel, the network subsystem now uses NAPI - a system where data is handled per device, rather than per packet. Through the use of this and the open source pf_ring library it is now possible to use direct NIC memory mapping to bypass kernel intervention. These recent developments allow for considerable performance increases over the technology available at the time this research was instigated.

There is a clear plateau region in both Figures 4.3 and 4.4. This appears to be due to automatic interrupt coalescence present in the card driver. In the live implementation of this system, the values for interrupt coalescence were adjusted to match the live data rate. As latency was not an issue here, the values in the live system for the maximum wait time, and buffer values were matched to the data rates.

It is important to match the packet inter arrival time, and the size of the network interface card buffer so as to avoid unnecessary packet loss.

It was believed from the above simple experimentation that standard PC hardware would be capable of providing the processing that was needed, and therefore a more expensive solution would be unnecessary.

The system implemented was based on 6 Dual 3GHz Xeon servers with 1GB of RAM 300GB IDE hard drives, with 1 dual 3GHz Xeon server with 6GB of RAM and dual raid 0 configured 360GB SCSI hard drive. Each machine had two one gigabit streams spanned from an ISP router, giving a potential two gigabits per second data rate per machine.

These machines generated statistical summaries of all the header information collected within each summary period, and transmitted these to a centralised database where it was classified and stored.

4.3 Software Design

In this final section of the design analysis, the software capture implementation is described, including the issues faced by it and the solutions found. The research made use of the Clementine data mining package (now IBM SPSS Modeller) as both a tool for investigating data relationships and as a run-time classifier. For more information about Clementine, see [69].

4.3.1 Proprietary Software

As previously mentioned, discussions with the ISP and the potential legal issues associated with data portions of packets preclude the use of standard signature based packet analysis to signature detect misuse. Instead summaries are built using meta-data from the packets and misuse is detected by variances in these summaries. This is an anomaly based approach, which has the advantages and limitations described in section 2.6.1.2.

The use of summaries is advantageous for several reasons. Firstly, the per packet processing involved in generating summaries is trivial when compared with that of pattern matching techniques. Secondly, by definition, large scale network events will affect the network at large, and should therefore result in statistical variances in the summary data, provided the summaries are of appropriate information.

The architecture chosen relies on fast packet statistic gathering with a pseudo real time summary generator. These two processes run independently, allowing the program to cope with high load by the manual adjustment of the size of the summaries gathered. The process works as follows:

The gatherer has a set of counters, which are incremented in relation to incoming packet variables. This is a memory intensive, processor trivial task, allowing the system to cope with significant numbers of packets in real time. The summary generator uses the counters from the gatherer to create the statistical measures needed to detect misuse. The design, while heavy on memory usage, has statically assigned volumes. This prevents the system from becoming bound by available memory.

Taking port usage as an example of this; in memory there are two arrays of 2^{16} elements each, corresponding to source and destination port numbers. When a TCP or UDP packet is received, the array elements corresponding to the source and destination ports are incremented. Periodically the array is processed into a summarised form, and reset.

This array counter approach was applied to packet length, IP ID, IP Flags, IP TTL, IP Protocol, IP addresses, transport layer port amongst others.

It is worth mentioning that while this same concept is applied to the IP addresses, only the 24 least significant bits are monitored due to the memory demands of a 2^{32} element array. This could potentially lead to a loss in accuracy, as two separate IP addresses may appear identical within a summary period. However, this will rarely occur, having a negligible effect on the summary statistics. This may be calculated as

Let pA be the probability that at least one incorrect IP address collision

occurs

Let pB be the probability that a collision would not occur for a 2 given IP addresses

Let nIP be the number of available IP addresses in our reduction

Let nS be the number of packets observed

$$pB = 1 - \frac{1}{nIP}$$

Therefore the probability of having at least one collision for nS observations is

$$pA = (1 - pB)^{nS}$$

Substituting in the appropriate values gives

$$0.5 = \left(1 - \left(\frac{1}{2^{24}}\right)^{nS}\right)$$

Which resolves to around 11 million packets. That is to say, if we had 11 million packets with random addresses passing through the system, there is a 50% chance of having at least one collision.

The primary alternative of matching IP addresses without using indexes leads to, for a binary search style algorithm, at most a $\log_2 n$ [70] relationship between the number of stored IP addresses and the number of necessary comparison operations, assuming a sorted list.

It is conceivable that a partial hashing algorithm could be to trade off memory for processing efficiency. However, in our case the memory was not a significant limit, and therefore the minimum processing solution was the most appropriate.

As the arrays used by the gatherer are of fixed size, the increase in the processing required by the summary generator is not noticeable, increasing by negligible amounts compared with the increase in packets processed. This means that with an increase in data rate the summary generator, which may be processor bound, can include more packets in a summary to compensate.

A side effect of this method, is that a summary period is not a fixed amount of time, but rather a fixed number of packets. The means that during periods of high data rate, the summary periods may be shorter. It also means, that during a period of complete outage, a summary may not be generated.

The packet capture mechanism itself is achieved via the libpcap library with some minor modifications; the software is compiled with the PCAP_FRAMES option set to its maximum size. The code itself is written in C, compiled under

GCC with compiler optimisations applied, including processor specific options. For a full listing of the gatherer C code, please see appendix B.

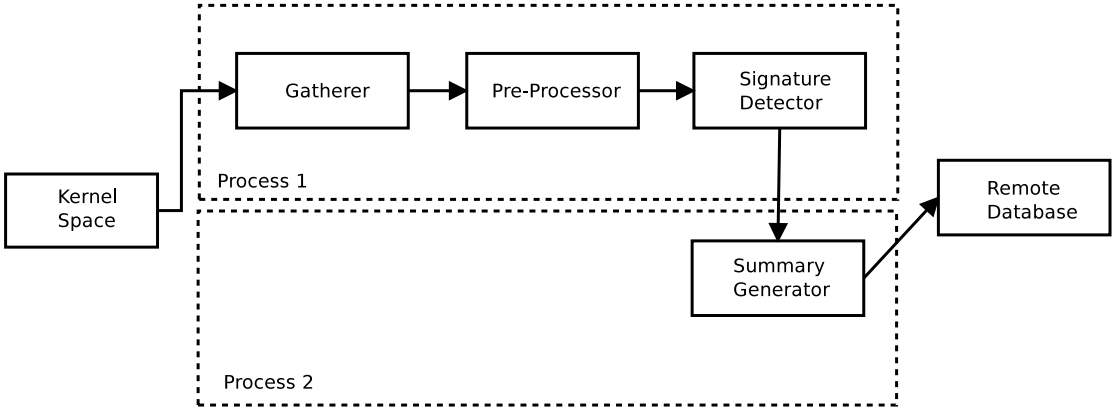


Figure 4.6: Software Block Diagram

The software pipes data from the summary generator to a remote database as shown in 4.6. The main block on the left shows the 3 tasks of the gatherer processes running as part of the gatherer machine software, a pointer to the data is the passed to the summary generator, before the summaries are transmitted. The piping of data to the remote database is achieved via a TCP socket connection.

From experience the major issues when dealing with high data rates are encountered on the capture interface, and may be overcome either through the use of specialist cards or through tuning options in the loadable driver modules.

Experience also tells us that while architectures based in on BSD perform quicker[8] than those based on older 2.4.x Linux kernels, the support for Linux solutions is stronger in many packages. Linux machines were therefore used for the gatherer machines.

The final part of the software is a centralised data processor. As each summary on each site is generated it is sent via a TCP socket[9] to a central machine. This machine uses the Clementine run-time engine to classify the data before storing it in a database. The run-time engine provided by the Clementine package is, in effect, a compiled neural classifier, which may be run on specified data files. This run-time engine is retrained periodically from new data to ensure that it is matching accurately, even in a constantly changing environment. Human interaction is necessary in the re-training process.

The server software at the central processor is written in Perl, which is perhaps a less than optimal performance choice. Perl is however extremely strong at text based processing, and as the data rates seen at the central processor are

[8]The memory copy from Kernel space to user space is more efficient
[9]On an independent management interface

comparably low, Perl's performance inefficiencies were not a considered to be a problem. The requirement to use Windows over a Unix solution comes from the use of the Clementine data mining package.

4.3.2 A Novel Signature Detector

In this section we describe a novel signature detection mechanism developed to allow certain payloads to be identified without recording the actual payload itself.

Signature detection is a common method for detecting security threats. It is employed by Intrusion Detection Systems to identify known threats with fixed payloads. While this is an effective approach it becomes difficult to utilise within the core of an ISP network for the reasons presented previously, namely hardware limitations and legal restraints.

While the focus of this work has been to use light weight statistical summaries as inputs to a detection system, some signature detection is beneficial for the accurate classification of individual packets and for the filtering of known attacks. To circumvent the legal restrictions in the UK preventing examination of the data portion of the packet, the checksum field in the header is used to determine whether the contents of the packet match a given signature.

4.3.2.1 Operation

The TCP checksum field is included in the protocol to allow for detection of transmission or reception errors in the data. The checksum field is given by taking the 16-bit one's complement of the one's complement sum of the 16-bit words in the TCP header and TCP data. As the checksum field is part of the data that is used to calculate the checksum, it is set to zero at the point of calculation. In the case of an uneven number of octets, the information is padded with zeros.

In this implementation, the checksum is calculated using header information from the packet which is being tested, but using the data portion of the signature that is to be matched for, which has been stored on the gatherer. Should this checksum match the original checksum of the incoming packet, it would indicate with a high probability that the data portion matches the signature.

For example, given a payload for a Code-red IIS overflow packet [71], we can filter all packets of appropriate size (from the IP length field in the header) and communicating on port 80 (from the port fields in the TCP header), and recalculate the TCP checksum. If the original checksum matches the new one, there is a high probability the packet is an overflow attempt. This is not possible for all worms. An infamous worm named 'Slammer' targeted MS SQL servers using

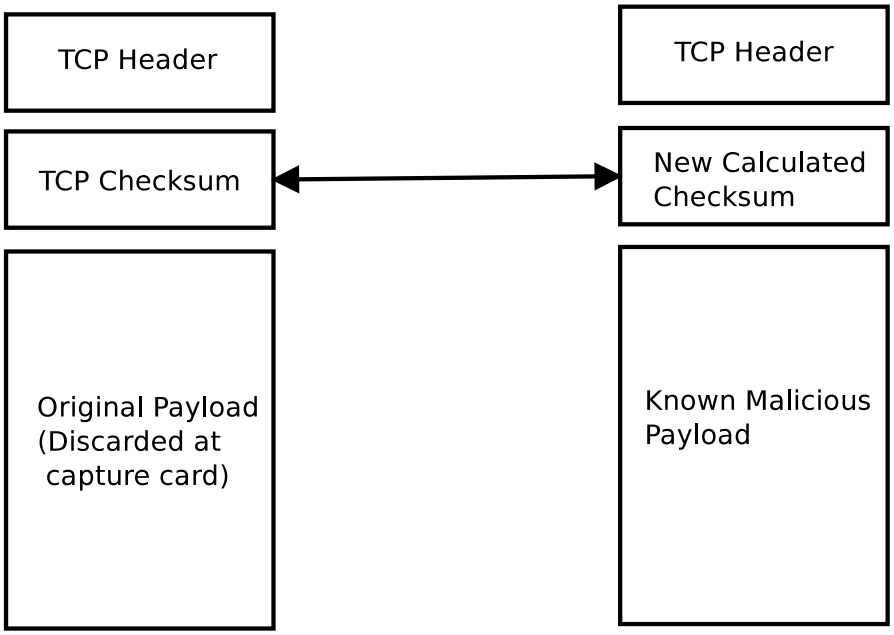


Figure 4.7: Signature Detection Mechanism

a UDP 420 byte payload to perform a buffer overflow [72]. The lack of reliable checksum would prevent this technique from detecting the attack.

To verify this technique, packets with known payloads were generated (sequential numbering as data fields). Their payloads had been entered into the detector, and were then detected with one hundred percent accuracy. For the technique to be more fully tested the number of false positives should be observed on an operational network. Unfortunately the only way to verify whether or not a packet has been correctly identified would mean inspecting the packet contents, breaking UK data privacy laws. It is however possible to approach the problem from a mathematical point of view by making some approximations about network traffic.

Assuming that the packets have an even distribution across ports and packet sizes, and that the checksum is evenly random in its mapping[10], the probability of a packet incorrectly matching the signature packet can simply be represented by

$$_pFailure = \frac{1}{MSS} \times \frac{1}{PortVariance} \times \frac{1}{ChecksumEntropy}$$

In the case of a TCP packet running over Ethernet this comes to approximately 1.5×10^{-13}, or one in six and a half million million packets. On the operational network monitored, depending on the time of day, there are around three hundred thousand packets per second seen on average. It would therefore observe approximately one incorrect labelling per year across all sites.

[10]All are favourable assumptions

The system has however not been used extensively in the live system, as there is a significant performance degradation when a large number of signatures are added. This could be somewhat alleviated by the calculation of the checksum on the interface card.

4.3.2.2 Limitations

For this mechanism to be effective, the exact packet signature must be available. This is a problem, particularly for TCP. It is not always possible to guarantee particular a packet will be present. Many TCP algorithms will alter which parts of payloads are present in which packets [11], based on a range of factors, such as packet loss and round trip times.

This indeterminacy will mean that either several definitions would need to be created for each potential TCP payload, or there will be a probabilistic miss-classification of data. In some cases (where the payload we are interested in is the first after the TCP handshake for instance) the data will be unaffected by TCP algorithms.

The behaviour of the TCP algorithms are consistent for given network conditions and the number of potential signatures is bounded, and in most cases very small.

Checksum reliability is, for uniform data, proportional to $1/2^{16}$, however on non-uniform data the reliability drops off very rapidly. Some estimates for the collision rate are as high as $1/2^{10}$ [73].

4.4 Summary

Coping with the high volume of data present in core networks is a significant research challenge. In this chapter a discussion of the architecture designed has been presented. Data rates present on the monitored network have been discussed and the legality of using this data described.

A novel signature detection mechanism has been presented which identifies known payloads in the absence of the payload data.

The architecture based on distributed summary gatherers with a central processor has been shown to be effective at dealing with high data rates. The signature detection mechanism has been shown to be capable of detecting payloads in the absence of payload data. This mechanism was not widely deployed in the system developed because of performance concerns.

[11] These algorithms include Nagle which will potentially coalesce multiple messages and MTU path discovery which may limit the size of the payload

The next chapter will introduce the concept of data mining, the various tools it encompasses and its application to this research.

Chapter 5

Applying Data Mining

This research was not intended to further the field of data mining itself. Instead existing data mining techniques were applied to the field of network monitoring. This thesis presents a novel combination of the use of traffic summaries at high data rates, with data mining techniques for feature selection and classification. This chapter introduces the concept of data mining, the tools used to perform data mining functions and the simulations used to determine appropriate features for the live system. Data mining was chosen as an appropriate set of tools for this research because the relationship between the measurements taken, and the classification of those measurements as representative of illegitimacy, is a complex one. Data mining tools are explicitly focussed on exploring complex data relationships making them an obvious choice for this type of analysis.

5.1 Detection Theory

The system described in this research was primarily anomaly based. The concept behind anomaly detection is that in some variable space, illegitimate traffic would be separate from legitimate traffic. However, it is not possible to gather all data associated with any traffic, for instance it is most likely to be impossible to know the sender's intention when creating traffic. With this in mind, an important part of the research was to investigate to what degree it is possible to separate legitimate and illegitimate traffic with the practically available variable space.

Figure 5.1 shows this concept, with the box representing multi-dimensional variable space, the triangle shape representing legitimate traffic, the square shape representing anomalous but legitimate traffic, and the circle shape representing illegitimate traffic.

It is worth noting at this point that while it may be that illegitimate traffic may be separate in a single axis, it is perhaps more likely that illegitimate traffic

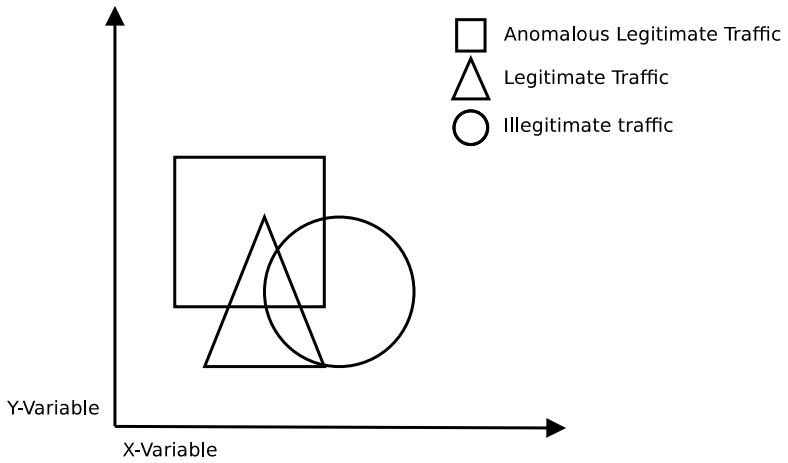


Figure 5.1: Anomaly Variable Space

will be separated from legitimate traffic through a more complex multidimensional function.

5.2 Data Mining Tools

Many differing techniques were used to analyse the data from both test and live networks. We describe their functionality and purpose here.

5.2.1 Clustering / Self Organising Maps

Clustering and Self Organising Map (SOM) algorithms are a subset of artificial neural networks. A SOM organises its output in a grid, the goal of the training is to associate similar areas of the grid with similar input patterns. The learning in a SOM is unsupervised learning meaning there is no defined output for the network to match.

This kind of algorithm is often used to find 'natural' groups in data. This type of algorithm is particularly useful for finding relationships between data. There are three separate algorithms used in this research for producing SOMs and Clusters.

- Kohoenen

  The Kohonen network is a SOM algorithm named after its inventor, Teuvo Kohonen. It simply arranges similar input patterns on similar areas of a grid [74].

- K-means Clustering

  This algorithm works through iterative refinement, and requires that the

number of desired output clusters is defined. This can be useful if the intention is to find a particular number of subsets within a data set [74].

- Two-step

  This algorithm is a SPSS (Clementine) proprietary one, which first determines the number of 'natural' clusters and then assigns data points to them. This can be informative as it does not require prior knowledge of the grouping present in the data.

5.2.2 Artificial Neural Networks (ANN)

Artificial neural networks are commonly called simply neural networks (which traditionally referred to biological neurons). These networks are a series of neurons interconnected via mathematical functions with weightings associated with them.

Most neural networks require training of some type; this is supervised learning. The goal of the training algorithm is to minimise the error in the output neurons given a corresponding set of inputs. Over time the weights applied to the various paths in the network are adjusted to minimise the output error. Some networks also adjust their topology including any number of layers comprising of any number neurons. There are in practice various topologies which are more effective than others [75], and therefore the search space is considerably smaller than it can appear.

Neural networks are often used to model complex relationships in an abstract manner where a formal representation is difficult.

5.2.3 Artificial Neural Networks & Weighting

Varying forms of machine learning were used to directly classify data, such things as decision trees, sequential models and logistical models [76] were all experimented with, if only briefly. ANNs on the whole provided the most accurate classification, therefore in order to discover which metrics were important, it was necessary to determine how the networks were reaching their classification.

The Clementine software package used for much of the data mining activity includes the functionality to extract the weights from an ANN and thereby calculate the relative importance of the inputs. This is useful as it allows us to see which variables are contributing most significantly to classifications.

As described above, neural networks have topologies made up of nodes, connected by a mathematical function. The network adjusts weights on these nodes to minimise the error on the output nodes for a given set of expected output data. Once trained, these weightings may be extracted for each input node by examining

the influence the input nodes have across the entire topology. This gives a numerical representation of the input importance which is invaluable to the research performed.

5.2.4 Graphing and Displaying the Data

Perhaps an overlooked method of mining data is to display it in various formats. One of the simplest methods for displaying data within this research was a two dimensional time axis graph.

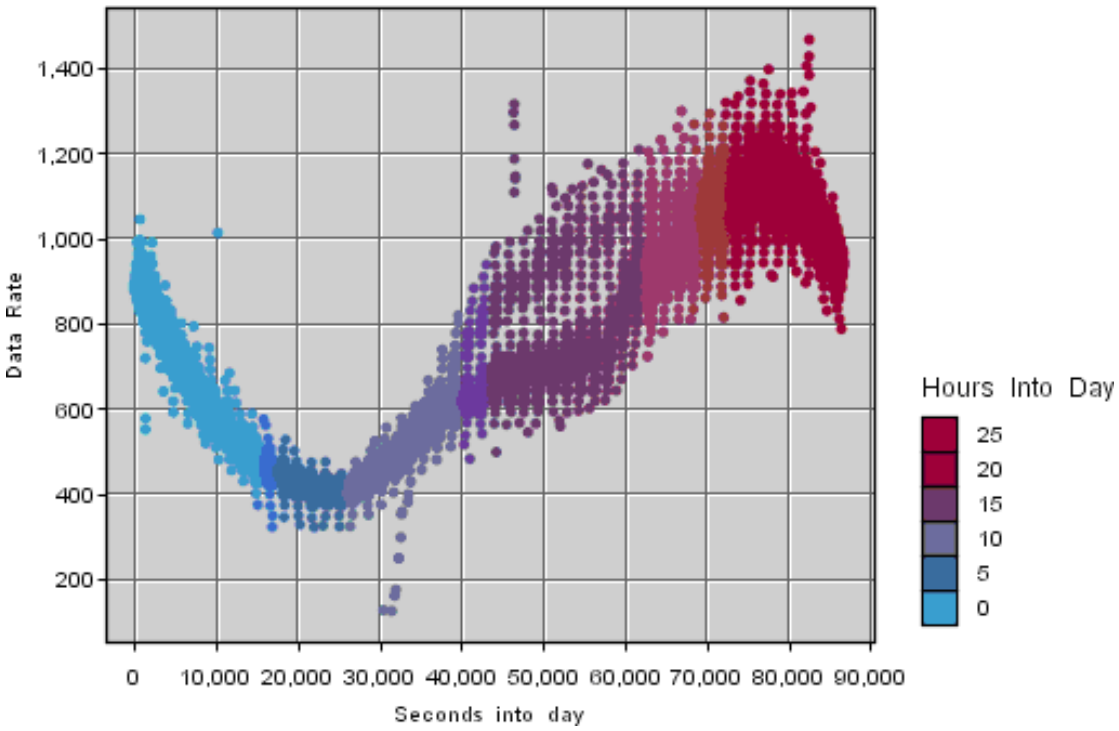


Figure 5.2: Monthly Data Rate

In Figure 5.2 a complex time of day based variation in the variable plotted can be seen (here one month's data is plotted onto a single 24 hour x-axis). This type of visual exploration of data can aid in the general understanding of how the data interrelate. This type of analysis is essential in the early stages of exploring a data set, as it provides the operator the ability to visualise large amounts of data in a simple manner.

5.2.5 Data Preparation

This section describes the principles with which data was prepared for use in ANN training. It is included here as it greatly influences the quality of the resultant model.

> "In computing this idea is expressed in the familiar acronym GIGO-Garbage in, Garbage Out" [76]p45

All data sets for training and testing were separate. This helps to prevent the machine learning techniques from latching on to specific, insignificant items in the test data, and preserves a more general model. Handling of data sets for neural training is notoriously difficult. The goal of the network is simply to minimise the error in the output, not to do this is a way that the operator considers sensible. If, for example, the classification could be made through the use of an ID field (i.e., all of classification X falls between ID Y and Z), then this is likely to be the input nodes most heavily used by the network, regardless of it's applicability to other data sets. By maintaining a separate training and testing data set, this type of scenario may be partially mitigated against.

All data were balanced in terms of number of examples of desired outcomes. If we imagine a situation where the data provided to the machine learning algorithm had a 99% positive output field, the machine learning algorithm can simply give a positive result for every prediction and gain a 99% prediction accuracy without actually finding any other relationships in the matrix.

For simplicity, all data used for training was kept in a matrix form schema. This means that for each measurement there are a set of variables, which can be expressed as a table. This data was stored in a MYSQL database present on the central processing machine.

5.2.6 Laboratory Emulation

As previously discussed, one of the research difficulties with classifying misuse on a network where the data field is not available, is determining the accuracy of the results.

As the detection mechanism eventually to be implemented would be working on a live network, with no control over the traffic, nor any method to independently label some classes of traffic, it was necessary to create an artificial environment to demonstrate the principles of the detection mechanisms, and the types of variation to later be observed in a live environment. This environment would allow for the simulation of DDoS attacks on a controlled network, where all DDoS traffic could be appropriately labelled and used for accurate training data.

5.2.6.1 Traffic Emulation

The purpose of the laboratory emulation activity was to provide a controlled environment with which to generate specific data to measure. The emulation was

built using sampled Netflow [11] measurements taken from a national network. The experiment emulated port numbers, size, IP address ranges and protocols found in the Netflow data. The traffic simulation was crude in the sense that application layer flows were not emulated. This was seen to be unnecessary as the system being proposed would not be performing flow reconstruction, and would therefore not be aware of interaction at that level of detail. The accuracy of the emulation directly impacts the validity of the modelling performed on the resulting data.

A proprietary network traffic emulator was used to create the appropriate traffic types and rates. The mix of traffic is shown in table 5.1, the network carrying 84% TCP traffic.

Table 5.1: Emulation Traffic Mix

Protocol	Percentage	Port	Size
TCP	25	Random	40
TCP	40	Random	MSS
TCP	11	80	40
TCP	11	80	MSS
UDP	12	Random	20-300
ICMP	1	N/A	64

As networks vary significantly in traffic profile, it was difficult to acquire an accurate picture of the commercial network without recording traffic from it, and therefore there were several areas in which the emulation was inaccurate. The exact operation of TCP was only loosely modelled (in terms of flags and options). Average packet sizes were modelled, but distributions were unknown, and therefore estimated [1].

The emulation assumes that TCP mainly operates in bulk mode, which holds given that statistically bulk connections handle more packets than interactive mode ones.

The packet headers were captured and entered into the Clementine data mining package. This data was analysed using a mixture of prior knowledge and Clementine's built in machine learning templates to derive some relationships between the packets.

Denial of service attack tools were then run with increasing traffic rates over the network in addition to the existing traffic. An infamous DoS tool at the time of the emulation was named 'Stacheldraht' [43]. The normal topology (represented

[1]This only applied to the UDP traffic, which was modelled as being normally distributed between 20 and 300 bytes in length.

in an abstract topological manner) is shown in Figure 5.3. The tool works in a 3-tier manner. A master-server is set-up, and several zombie[2] machines are controlled from it. The attacker logs into, and gives commands, to the master-server via an encrypted telnet like tool. The master-server communicates with the zombies through ICMP echo (ping) replies, which are not usually blocked by simple firewalls.

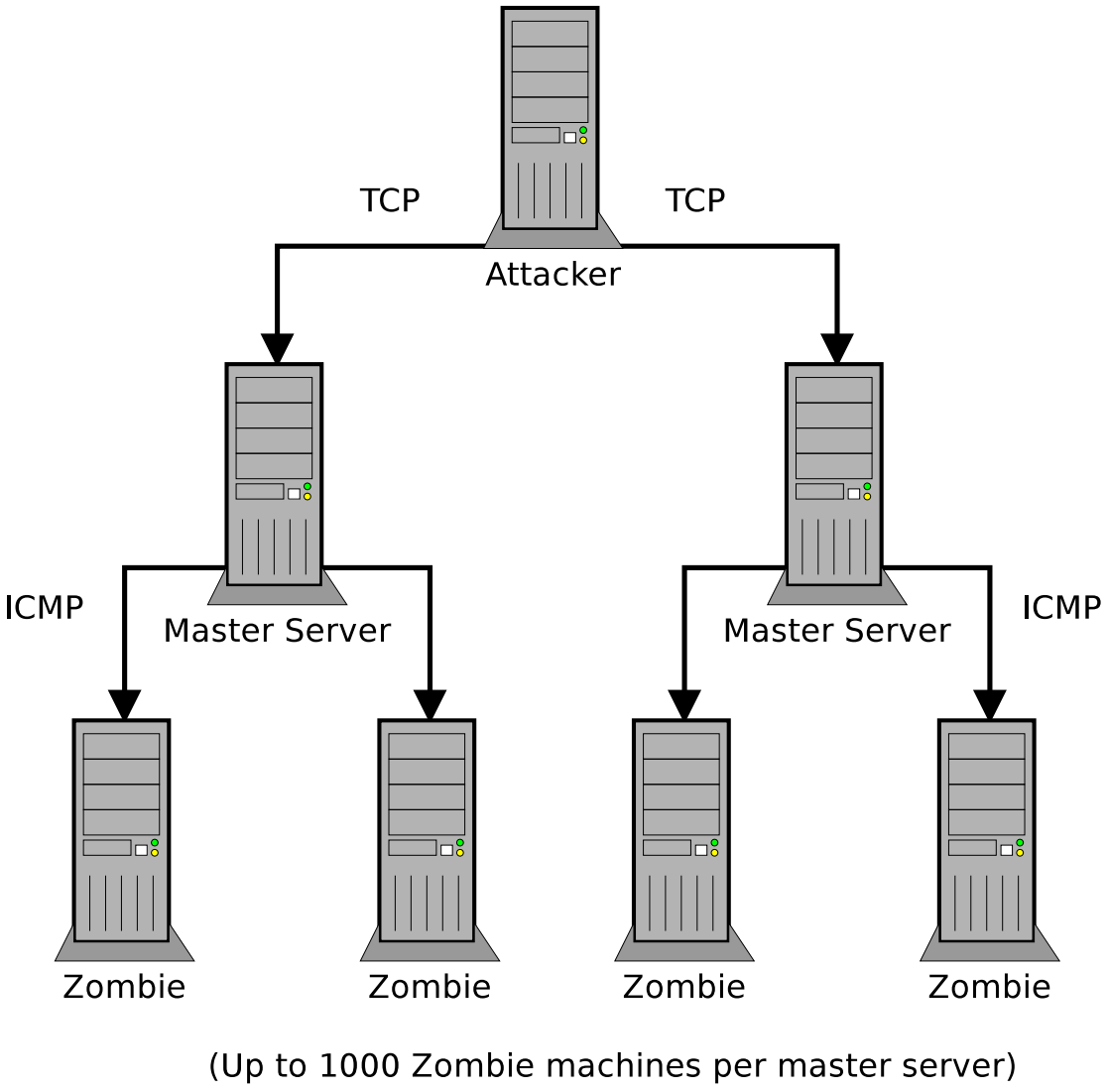


Figure 5.3: Stacheldraht Topology

The tool was configured with one controller, one handler and one zombie. The headers were then analysed to find relationships which would identify the DoS traffic and not the background.

It is clear that this experiment makes several key simplifications. Firstly the Netflow data is not a complete description of the normal network traffic. Whilst

[2]This term refers to a machine which has been previously compromised by an attacker and often, as in this case, forms part of a botnet.

clearly not representative of a live network, if the only data being analysed is that which has been given by the Netflow source or its derivatives, valid conclusions may still be reached.

Secondly, the traffic generation in this experiment is unaffected by the denial of service traffic added to the network. This is of course, a gross and incorrect assumption. However, for the purposes of initial investigation, given that the effects on a live network were to be later studied, it was taken to be acceptable.

ICMP, UDP and TCP attacks were added at one, ten, one hundred, one thousand and ten thousand packets per second. Stacheldraht allows for configuration of packet sizes (in the case of ICMP and UDP attacks), port ranges (in the case of SYN and UDP attacks), destination address(s), attack duration and a toggle for spoofing. Attacks in various forms were generated for each class, and the testing data used on the network. The traffic generators were run at a constant 40Mbits/s and had varying amounts of denial of service traffic added to them (see Table 5.2).

This set of DoS attacks is clearly limited in scope. No emulation of reflector attacks, or more advanced CPU or memory resource based attacks was attempted, nor were other DoS tools used at this point in the research. While this limits the applicability of the resultant data to a live national environment, the range and scope of potential attacks was considered too large to be evaluated exhaustively. The attacks which were selected give a strong subset of the possible attacks present on a live network, and were considered to be enough to provide solid data on which to evaluate attack features.

Table 5.2: Laboratory DoS Traffic Mixes

Experiment Number	Background Traffic Rate (MBit/s)	DoS Type	DoS Traffic Rate (MBit/s)
1	40	UDP	0.0056
2	40	UDP	0.056
3	40	UDP	0.56
4	40	UDP	5.6
5	40	UDP	56
6	40	TCP	0.0056
7	40	TCP	0.056
8	40	TCP	0.56
9	40	TCP	5.6
10	40	TCP	56
11	40	ICMP	0.0056
12	40	ICMP	0.056
13	40	ICMP	0.56
14	40	ICMP	5.6
15	40	ICMP	56

The resulting packet traces were converted into records with entries for header variables. The variable names can be found in Table 5.3.

These fields were used as inputs for ANN based investigations into the data.

Table 5.3: Packet Fields

Field Name
Time of packet Capture
IP Header Length
IP Type of Service
IP Total Length
IP Identification
IP Flags
IP Fragment offset
IP Time to live
IP Protocol
IP Source IP
IP Destination IP
UDP Source Port
UDP Destination Port
UDP length
TCP Source Port
TCP Destination Port
TCP Sequence Number
TCP Acknowledgement Number
TCP Header length
TCP Type
TCP Window Size
TCP Urgent Pointer
ICMP Type
ICMP Code
ICMP Checksum

The initial investigation of the laboratory data was achieved via Self organising maps.

Figure 5.4 and figure 5.5 show Kohonen networks of the data, with the differing shapes representing differing DoS states. Encouragingly there is some 'natural' differentiation between DoS and non-DoS traffic, this is particularly evident in ICMP and UDP based DoS (figure 5.5). The term natural is used here as this is unsupervised learning, and therefore any resultant patterning is purely a product of the algorithm. For these figures, and many others presented in this section, random noise was applied to the X-axis to provide some separate for the data points. As the clusters are discrete, this does not affect the reading of the charts.

Figure 5.6 shows another SOM algorithm output, in this case K-Means clus-

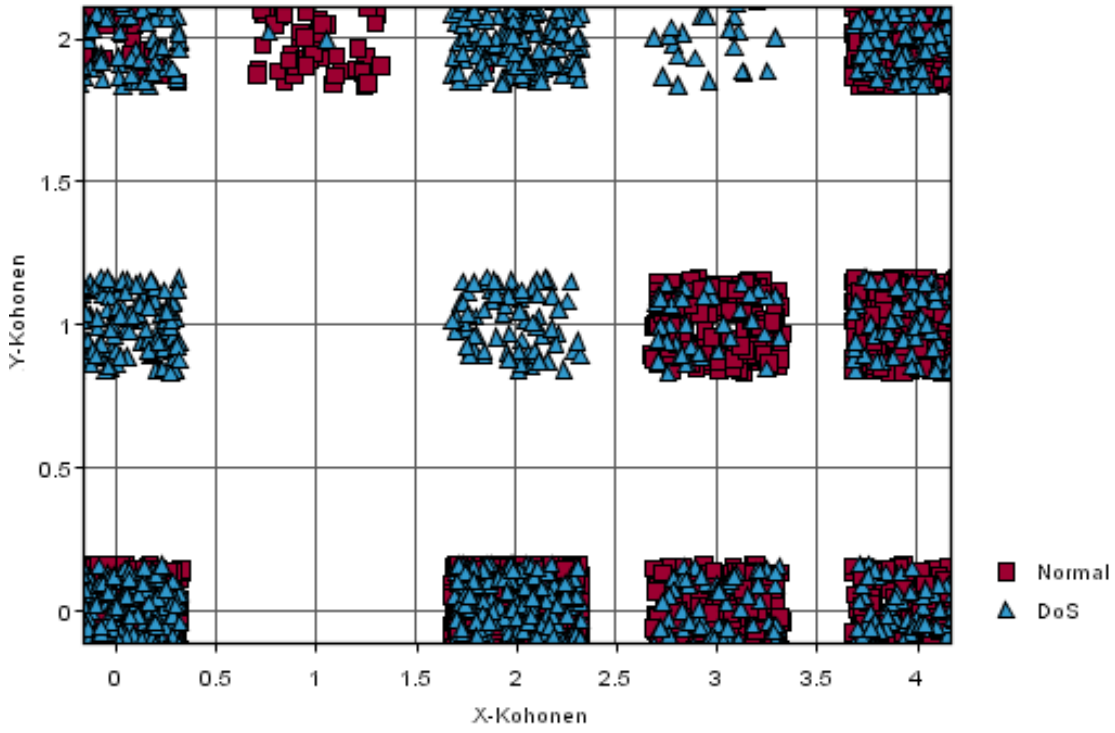


Figure 5.4: Laboratory DoS - Kohonen Network

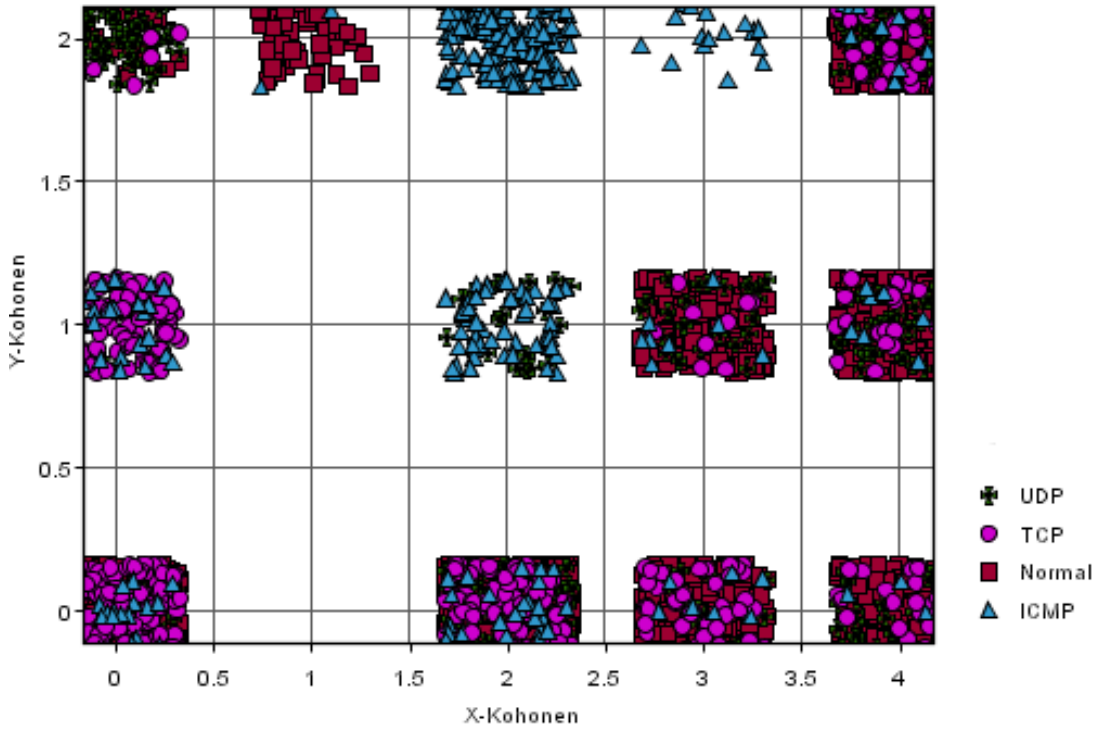


Figure 5.5: Laboratory DoS - Kohonen Network (2)

tering. The cluster is represented on the X-axis with the distance from the cluster centroid (mean) on the Y-axis. This algorithm requires the number of clusters as an input and was given 4, one for UDP, TCP, ICMP and normal traffic which were known to be present in the data. Cluster 3 contains exclusively UDP traffic, Cluster 4 contains ICMP traffic close to its centroid and Normal traffic at distance. The TCP DoS traffic is mainly present at the outer edges of Cluster 2.

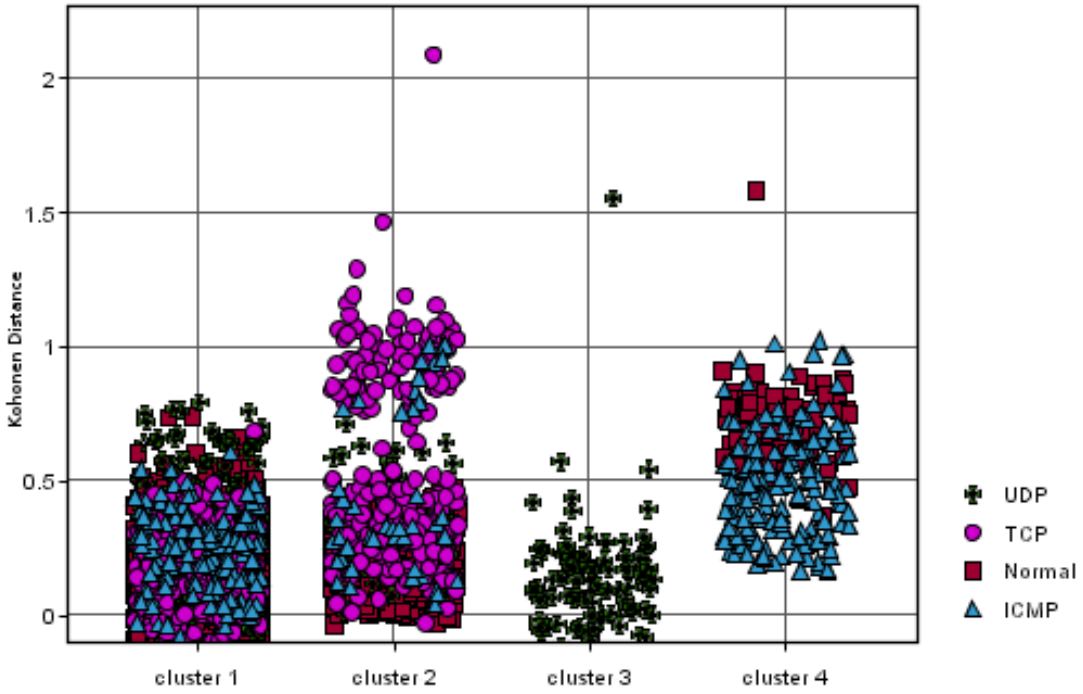


Figure 5.6: Laboratory DoS - K-Means Cluster

The K-means algorithm was then fed only clean traffic samples, and configured to form a single cluster. All records were then fed into the algorithm and the distances from the cluster calculated, this can be seen in Figure 5.7. Of note is that all DoS traffic appears on the outskirts of the cluster, again strengthening our confidence that the header values, and meta data derived from them could be a strong indicator of DoS.

We created summaries in various forms of header fields, and used these as inputs to a neural training algorithm.

Figure 5.8 shows the amount of bandwidth taken up by attack traffic, against the accuracy of labelling. ICMP DoS proved to be the easiest to detect, possibly due to the small amount of ICMP traffic naturally in the data. This separation of the ICMP traffic was also visible in the SoM outputs, where the ICMP based DoS traffic was generally speaking well separated from the background samples.

The network weightings were then analysed to determine the more useful met-

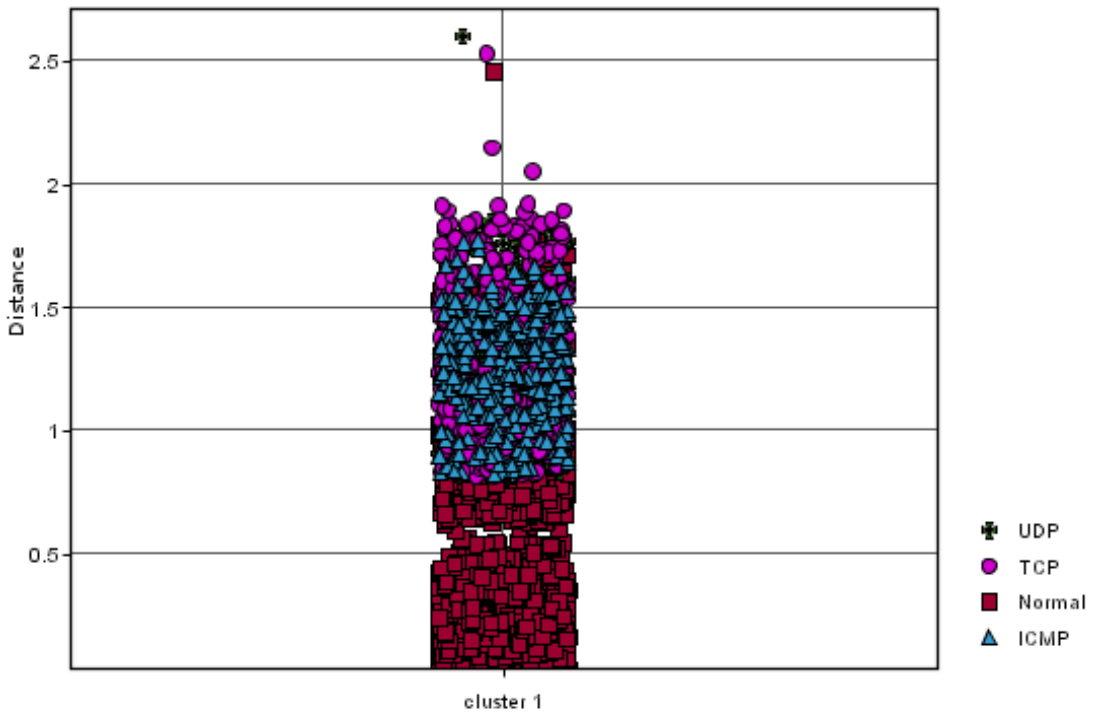


Figure 5.7: Laboratory DoS - K-Means Single Cluster

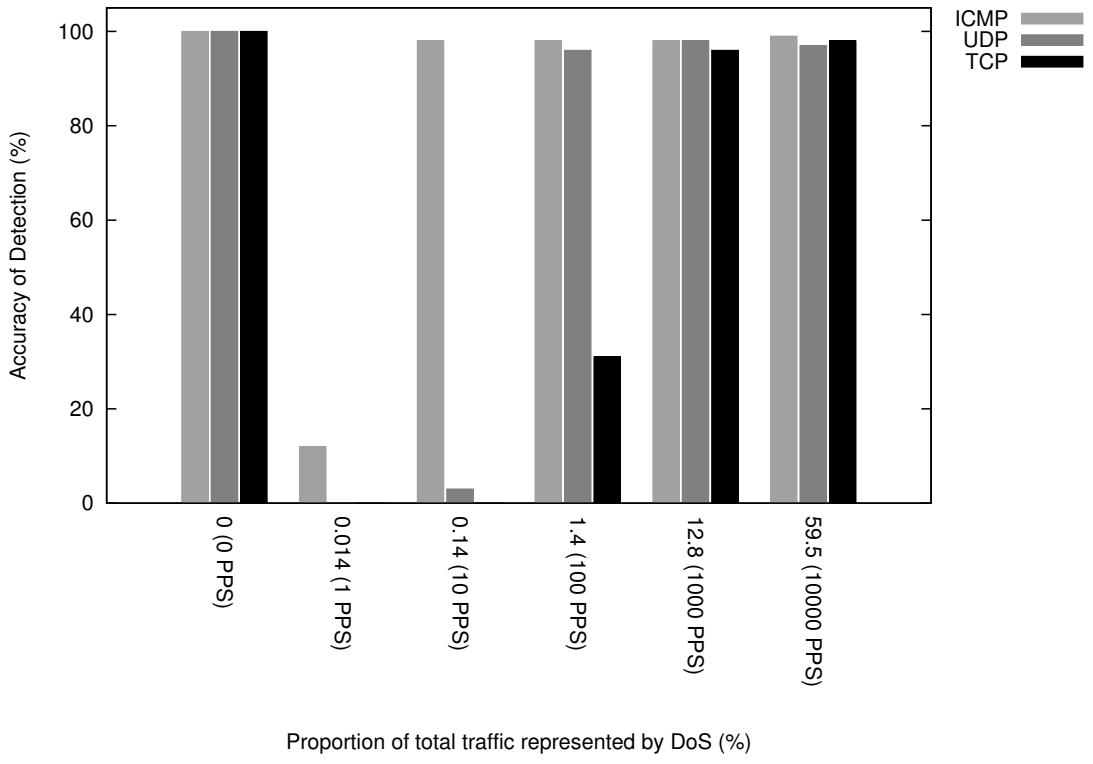


Figure 5.8: Detection Rates

rics. Table 5.4 shows the relative importance of the most significant input values to the ANN. These values are calculated by the weightings analysis described in section 5.2.3.

Table 5.4: DoS Relative Importance of Detection Metrics

Field Name	Relative Importance
Packet Sizes	0.3093
Transport Layer Flags	0.2739
Protocol Counts	0.1822
Transport Layer Ports	0.1235
IP Address Counts	0.1109
Data Rate	0.0002

Development of statistical triggers allowed the summaries containing DoS traffic to be detected at only 0.1% of the total traffic rate, and detected with no false positives at 1%.

The primary conclusion from this emulation was that there was enough variance in simple network measurements with increasing volumes of denial of service traffic to effectively detect its presence under simplified conditions. The emulation also pointed out which fields may be useful in the analysis of the live data which was to follow it.

It is worth noting that while TTL was ignored in this simulation [3] the denial of service attacks were identifiable from it successfully. The use of the TTL field is discussed in section 8.1.2.

5.2.7 Worm Traffic

To this point, there has been very little discussion of direct worm network threats. Direct worm activity can have an extremely detrimental effect on a network's performance. In January 2003 networks were flooded with traffic attributed to a worm exploiting vulnerabilities in Microsoft's SQL Server database, the worm was given the name Slammer [72].

Shortly after this in the summer of the same year a worm exploiting the DCOM RPC service on Windows 2000 and Windows XP hit many users. This worm was commonly referred to as blaster.

After a brief respite a worm named Sasser hit in May 2004, abusing vulnerabilities in the Local Security Authority Subsystem Service (LSSAS, hence 'Sasser').

[3] No network data for this was available at the time; therefore any analysis would have been purely speculative.

In August 2004 Microsoft released their Service pack 2 for windows XP. Crucially this was the first service pack to enable the firewall by default on the home user's machines. Our system was placed in a core network for the first time in June 2005.

Since this date there has been a distinct lack of direct worms using new vulnerabilities or services targeting these systems, in fact, Trojans were out numbering worms and viruses five to one in June of 2006 [27]. A worm named 'Zotob' was released targeting Windows 2000 machines, but as it functioned over port 445 (a port blocked on the monitored network) it did not appear significantly in the traffic monitored. The data presented in this research does not include anything more recent than April 2006, within this period of time, no significant direct worm threats were observed.

Instead many worms now use secondary communication methods such as email, peer-to-peer software, browser vulnerabilities and messenger clients.

This change in worm profiles has a significant impact on the network effect; events which are inherently dependent on users have significantly slower propagation times, lack synchronisation and generally have lower numbers of compromised hosts. Therefore they do not constitute as much of a threat to the network itself as direct worms.

During course of this research, no major worms were released to affect the network. Therefore to investigate their effects, test bed simulation was the only option.

From [77] we can discuss the general characteristics of large scale worm traffic. In this analysis the authors describe the target discovery as falling into one of three categories; Scanning, Target List and Passive. Of interest to this project are the scanning and target list variants, as they are most likely to cause significant disruption to a network.

In the case that a worm actively scans for potential targets, we should expect to see a significant increase in the number of packets of a particular size, potentially accompanied by an increase in the number of packets on a particular transport layer port.

Once the worm has compromised its target it may spread in one of two ways; as part of the exploit or through a secondary channel, such as FTP. At this stage we would expect to see an increase in the number of packets on a particular application layer port associated with the worm transfer.

To give some validation to these predictions traffic generators were set up, as with the DoS investigation, to generate a roughly representative background traffic sample. A machine was set up to reply [4] on all transport layer ports for the

[4]The machine replies with a SYN/ACK, and does not emulate services further

network 192.0.0.1/8 [5]. A machine was infected with the 'sasser' worm and allowed to generate traffic on the network, which was recorded by a fourth machine, which was running our summarisation software. This topology is shown in Figure 5.9.

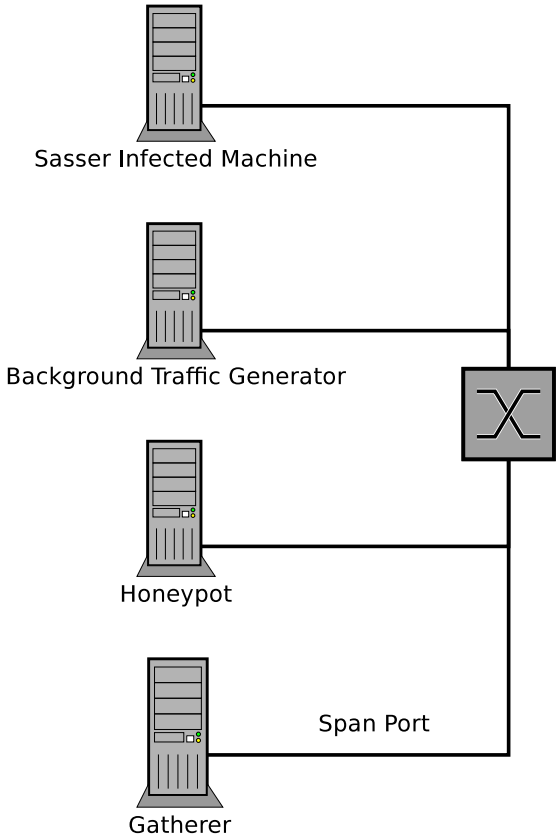


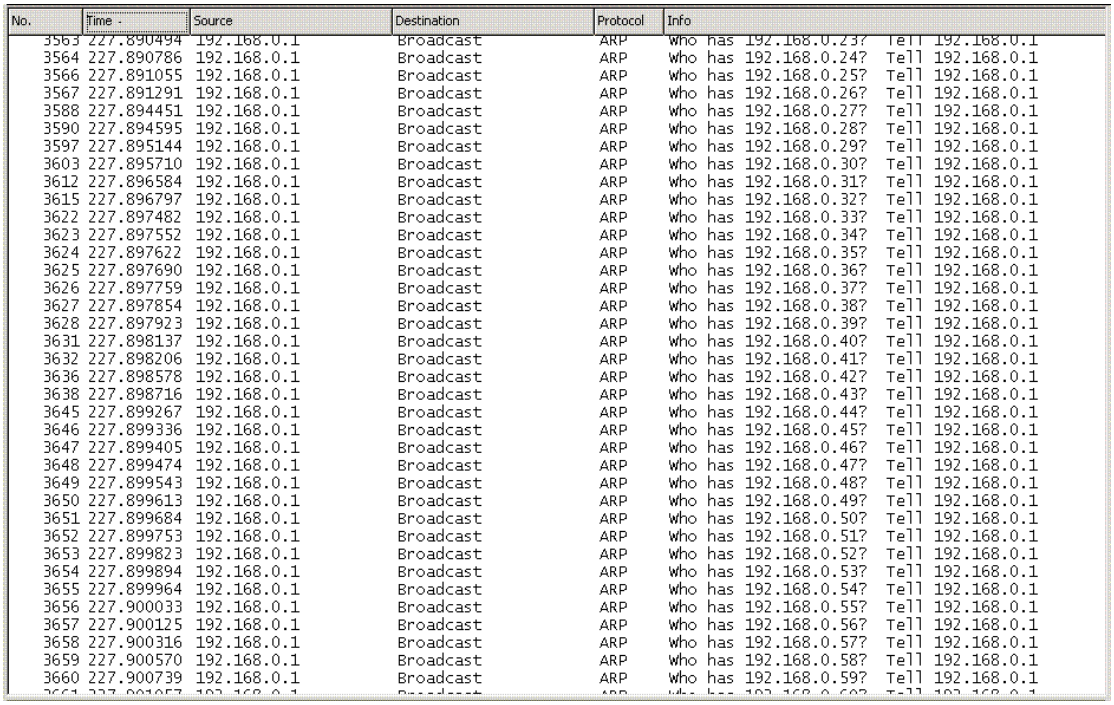
Figure 5.9: Worm infection topology

In this topology there are no hosts available for exploitation. This is intentional as modelling the speed of propagation and infection success rates would be arbitrary. These parameters are variable, depending on the specific worm in question, the network configuration (in terms of ratio of local an wide area host availability), and a large number of other factors [23].

As can be seen in Figure 5.10 the worm scanning technique is simply a sequential search through the IP sub-network it inhabits. Scanning the network locally is an effective strategy for worm target discovery [23], though it can limit the speed of Internet propagation.

The scanning and infection mechanism is in this case flooding the network. This type of activity leads to a significant change in port distribution. This stage allows the worm to build a list of susceptible hosts. In our simulation, there are

[5]Ideally the whole IPv4 address space would have been emulated, however the number of IP addresses emulated is a limitation of the software and associated PC hardware used to in providing the services.



No.	Time	Source	Destination	Protocol	Info
3563	227.890494	192.168.0.1	Broadcast	ARP	Who has 192.168.0.23? Tell 192.168.0.1
3564	227.890786	192.168.0.1	Broadcast	ARP	Who has 192.168.0.24? Tell 192.168.0.1
3566	227.891055	192.168.0.1	Broadcast	ARP	Who has 192.168.0.25? Tell 192.168.0.1
3567	227.891291	192.168.0.1	Broadcast	ARP	Who has 192.168.0.26? Tell 192.168.0.1
3588	227.894451	192.168.0.1	Broadcast	ARP	Who has 192.168.0.27? Tell 192.168.0.1
3590	227.894595	192.168.0.1	Broadcast	ARP	Who has 192.168.0.28? Tell 192.168.0.1
3597	227.895144	192.168.0.1	Broadcast	ARP	Who has 192.168.0.29? Tell 192.168.0.1
3603	227.895710	192.168.0.1	Broadcast	ARP	Who has 192.168.0.30? Tell 192.168.0.1
3612	227.896584	192.168.0.1	Broadcast	ARP	Who has 192.168.0.31? Tell 192.168.0.1
3615	227.896797	192.168.0.1	Broadcast	ARP	Who has 192.168.0.32? Tell 192.168.0.1
3622	227.897482	192.168.0.1	Broadcast	ARP	Who has 192.168.0.33? Tell 192.168.0.1
3623	227.897552	192.168.0.1	Broadcast	ARP	Who has 192.168.0.34? Tell 192.168.0.1
3624	227.897622	192.168.0.1	Broadcast	ARP	Who has 192.168.0.35? Tell 192.168.0.1
3625	227.897690	192.168.0.1	Broadcast	ARP	Who has 192.168.0.36? Tell 192.168.0.1
3626	227.897759	192.168.0.1	Broadcast	ARP	Who has 192.168.0.37? Tell 192.168.0.1
3627	227.897854	192.168.0.1	Broadcast	ARP	Who has 192.168.0.38? Tell 192.168.0.1
3628	227.897923	192.168.0.1	Broadcast	ARP	Who has 192.168.0.39? Tell 192.168.0.1
3631	227.898137	192.168.0.1	Broadcast	ARP	Who has 192.168.0.40? Tell 192.168.0.1
3632	227.898206	192.168.0.1	Broadcast	ARP	Who has 192.168.0.41? Tell 192.168.0.1
3636	227.898578	192.168.0.1	Broadcast	ARP	Who has 192.168.0.42? Tell 192.168.0.1
3638	227.898716	192.168.0.1	Broadcast	ARP	Who has 192.168.0.43? Tell 192.168.0.1
3645	227.899267	192.168.0.1	Broadcast	ARP	Who has 192.168.0.44? Tell 192.168.0.1
3646	227.899336	192.168.0.1	Broadcast	ARP	Who has 192.168.0.45? Tell 192.168.0.1
3647	227.899405	192.168.0.1	Broadcast	ARP	Who has 192.168.0.46? Tell 192.168.0.1
3648	227.899474	192.168.0.1	Broadcast	ARP	Who has 192.168.0.47? Tell 192.168.0.1
3649	227.899543	192.168.0.1	Broadcast	ARP	Who has 192.168.0.48? Tell 192.168.0.1
3650	227.899613	192.168.0.1	Broadcast	ARP	Who has 192.168.0.49? Tell 192.168.0.1
3651	227.899684	192.168.0.1	Broadcast	ARP	Who has 192.168.0.50? Tell 192.168.0.1
3652	227.899753	192.168.0.1	Broadcast	ARP	Who has 192.168.0.51? Tell 192.168.0.1
3653	227.899823	192.168.0.1	Broadcast	ARP	Who has 192.168.0.52? Tell 192.168.0.1
3654	227.899894	192.168.0.1	Broadcast	ARP	Who has 192.168.0.53? Tell 192.168.0.1
3655	227.899964	192.168.0.1	Broadcast	ARP	Who has 192.168.0.54? Tell 192.168.0.1
3656	227.900033	192.168.0.1	Broadcast	ARP	Who has 192.168.0.55? Tell 192.168.0.1
3657	227.900125	192.168.0.1	Broadcast	ARP	Who has 192.168.0.56? Tell 192.168.0.1
3658	227.900316	192.168.0.1	Broadcast	ARP	Who has 192.168.0.57? Tell 192.168.0.1
3659	227.900570	192.168.0.1	Broadcast	ARP	Who has 192.168.0.58? Tell 192.168.0.1
3660	227.900739	192.168.0.1	Broadcast	ARP	Who has 192.168.0.59? Tell 192.168.0.1

Figure 5.10: Arp Packets

no vulnerable hosts, therefore a target list taken from a genuine infected machine was used and a honeypot configured to respond to that range[6].

The worm connects to machines on port 445, with the intention of compromising them. The honeypot does not respond, as it is not programmed to emulate TCP connections past the 'synchronise and acknowledge' phase.

The worm steps through its pre-generated list of machines, attempting to compromise each machine in order.

Of note is that the connections are identical in terms of packet sizes, port number and source address.

In the live system, wide spread worms of this type would lead to a change in the variance of IP address counts, Port usage distributions and packet size distributions. Though the transfer of the worm was not recorded, it may be safe to assume that it would also take place over a fixed port [7], and would therefore effect the relevant distributions, leading to detection.

Chapter 8 shows the effects of DoS on some of these distributions. In many ways the effect of a mass worm on a network is similar to that of a bandwidth based DoS attack. The worm traffic however, is distributed across the address space, where as most DoS attacks target a small set of hosts.

[6]In this case the list refers to the Loughborough University Campus network

[7]This has historically been the case, as the services the worms have exploited have been hosted on static ports in line with the IANA assignment

While there were no worms observed on the live system, the laboratory recording of worm activity has allowed speculation that the traffic profiles of mass worm traffic will be similar to that of DoS traffic, but with an more even spread destination IP addresses.

5.3 Summary

Separating legitimate and illegitimate traffic is a significant challenge. Determining appropriate metrics for the purposes of detecting malicious activities traditionally requires expert domain knowledge.

In this chapter the various data mining methods utilised in the research were discussed, its application in a laboratory environment has been described, and the theoretical effect of a worm on the live implementation shown.

The exploration of data through various clustering algorithms has been presented, and shown to be an effective method of separating laboratory controlled DoS traffic from simulated legitimate network background. The use of ANN weightings as a metric of the importance of input has been introduced, and shown to be useful for evaluating the relative strengths of particular statistics in the detection of illegitimate traffic. An example of worm behaviour has been examined and allowed speculation of the expected effects on the distribution of key metrics in a live environment.

The next chapter will cover the data and analysis from the live network.

Chapter 6

Operation of the System

This research relies on relationships between certain metrics in the data being present. In order to understand the models and data relationships presented later in the thesis, this chapter presents some of the patterns and characteristics of the network being monitored. The work relies of a baseline network state, which while it may contain a level of illegitimate traffic, this is not believed to be of concern.

An analogous situation may be that of human health. People encounter germs on a constant basis, without significant illness. Only if the system becomes significantly ill are we made aware of a specific infection through the immune system's response. In the context of a network, there is an ongoing level of illegitimate traffic which may be present at any given time. This level of traffic is not of concern unless it poses a threat to the network itself. That is to say, a degree of illegitimate traffic may be considered to be part of the normal state of the network.

The system described in Chapter 4 was operation between September 2005 and April 2006. During this period, over 140 separate events were detected and classified ranging from routing changes to DoS. The data was collected from 6 sites distributed across the UK.

Sites were set to generate summaries from 1,000,000 packets, which in most sites was around 5 seconds, depending on the time of day [1]. All data presented in this chapter is taken from the live national network.

6.1 Data Rate

The data rates of national sites were of different magnitudes, however, all followed a broadly similar pattern of variance in line with the time of day. This is to be expected as the network utilisation is a result of the user's routine.

Figure 6.1 shows one week of data rates plotted against time of day. The peak

[1] see figure 6.1 for a graph showing time of day variation on data rate

utilisation occurs around 19:00 with the lowest utilisation occurring at 04:00. The plateau seen in the figure occurs at 13:00.

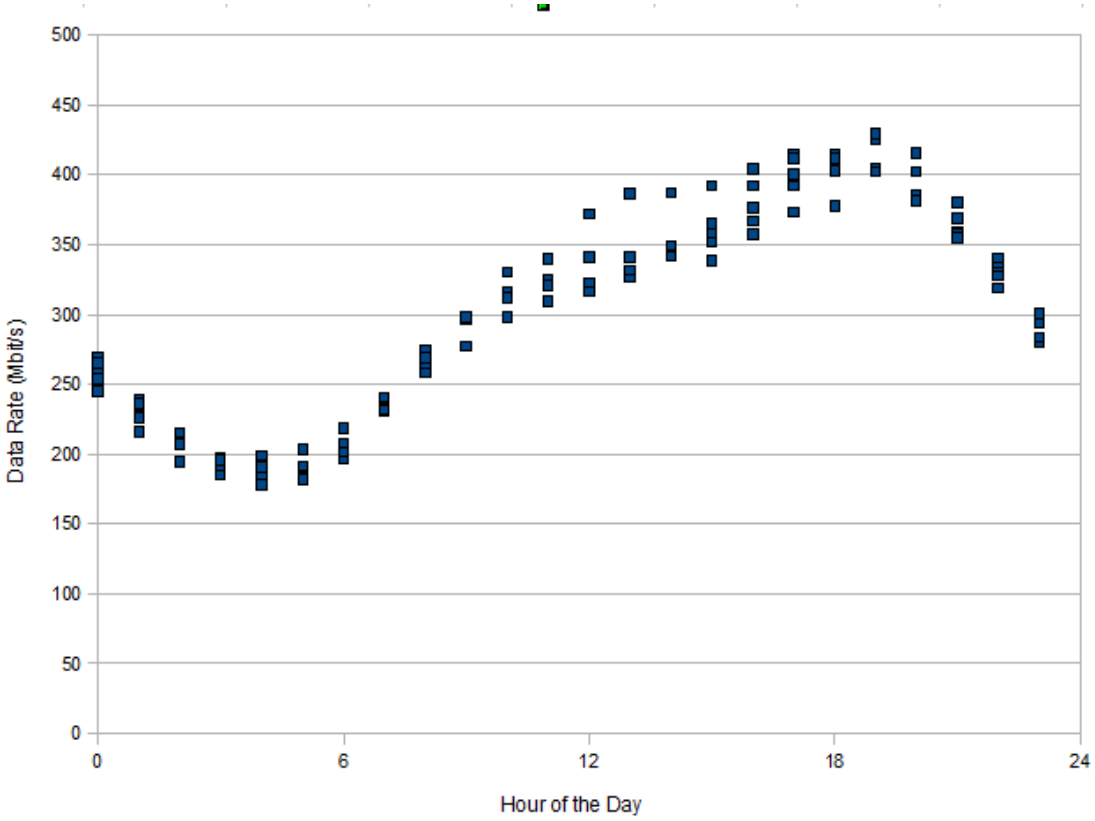


Figure 6.1: Data Rate Variation

This would suggest that the user's usage of the network closely mirrors their sleep patterns, but that there is a substantial amount of network traffic that is unaffected by this pattern. The data rate remains elevated during traditional working hours, but peaks at 19:00 when we may reasonably expect users to be returning from work[2].

6.2 Protocols

When considering protocols, we examine the application layer only. All traffic which concerns us in this research was spanned to our monitors over 1Gbit Ethernet. The spanned traffic is almost entirely IP at layer 3. In this section, only TCP, UDP and ICMP are examined; between them they constitute over 99% of the total traffic.

[2]The monitored network was UK based and therefore is dominated by a UK time of day variation

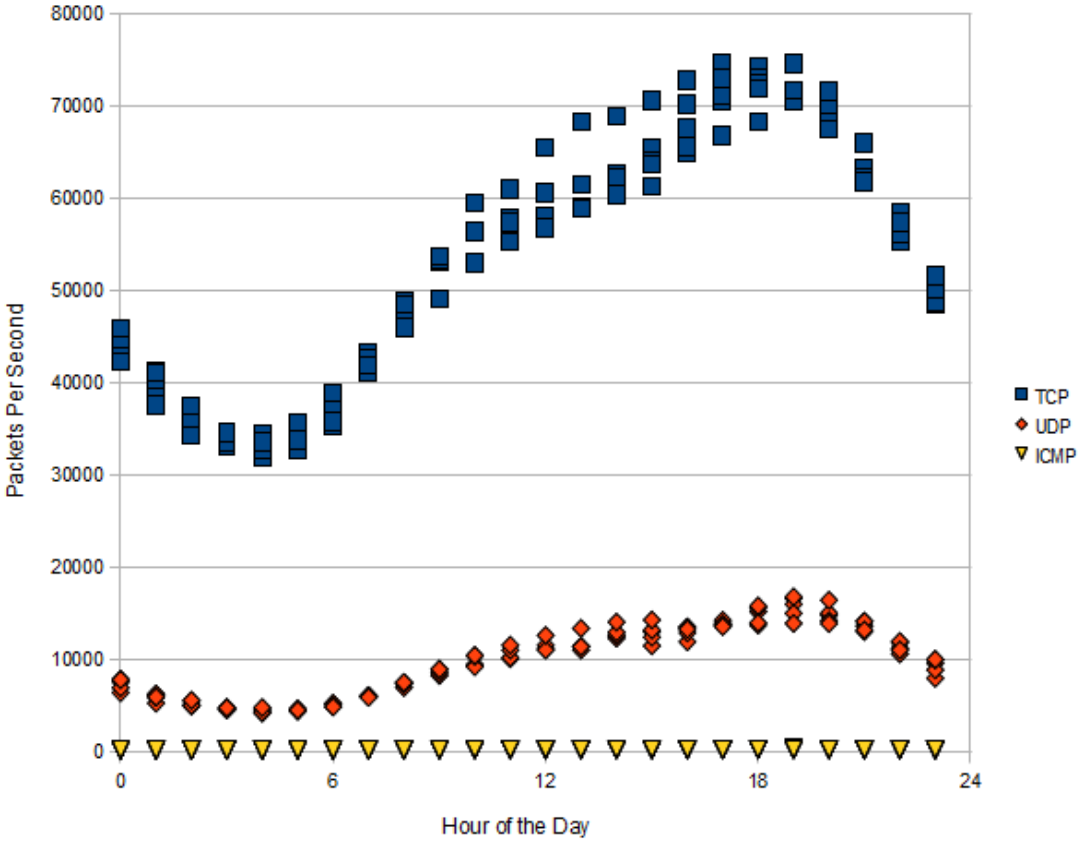


Figure 6.2: Protocol Variation

Figure 6.2 shows a similar pattern for protocol usage. The proportion of total traffic for each stays relatively constant, with a slightly higher proportion of TCP traffic in the early hours of the morning (see figure 6.3).

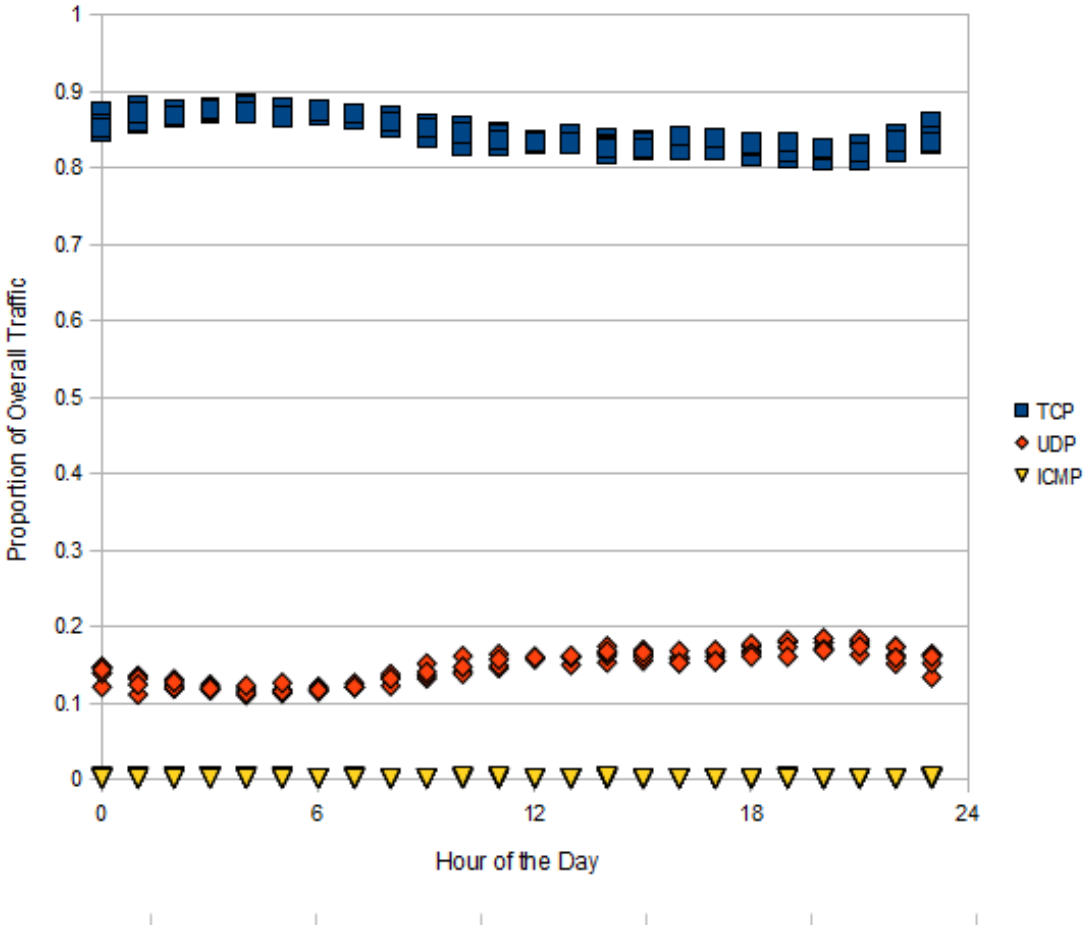


Figure 6.3: Protocol Variation as a Proportion of Overall Traffic

6.3 TTL

The Time to Live field is a header variable in the IP header which indicates the number of hosts[3] that may transmit the packet before it should be discarded. Differing operating systems have differing starting values for this field, the full range is 0-255. Chapter 8 contains a description of the TTL values and their significance. In this section the most common values for the monitored network are evaluated, these values are 125 and 61, which correspond to Microsoft Windows and most Unix variants respectively. Windows defaults to a TTL of 128, meaning 125 is 3 'hops' away from the monitor, Unix defaults to 64 which again, is 3 hops away from the monitor.

[3]Such as a router

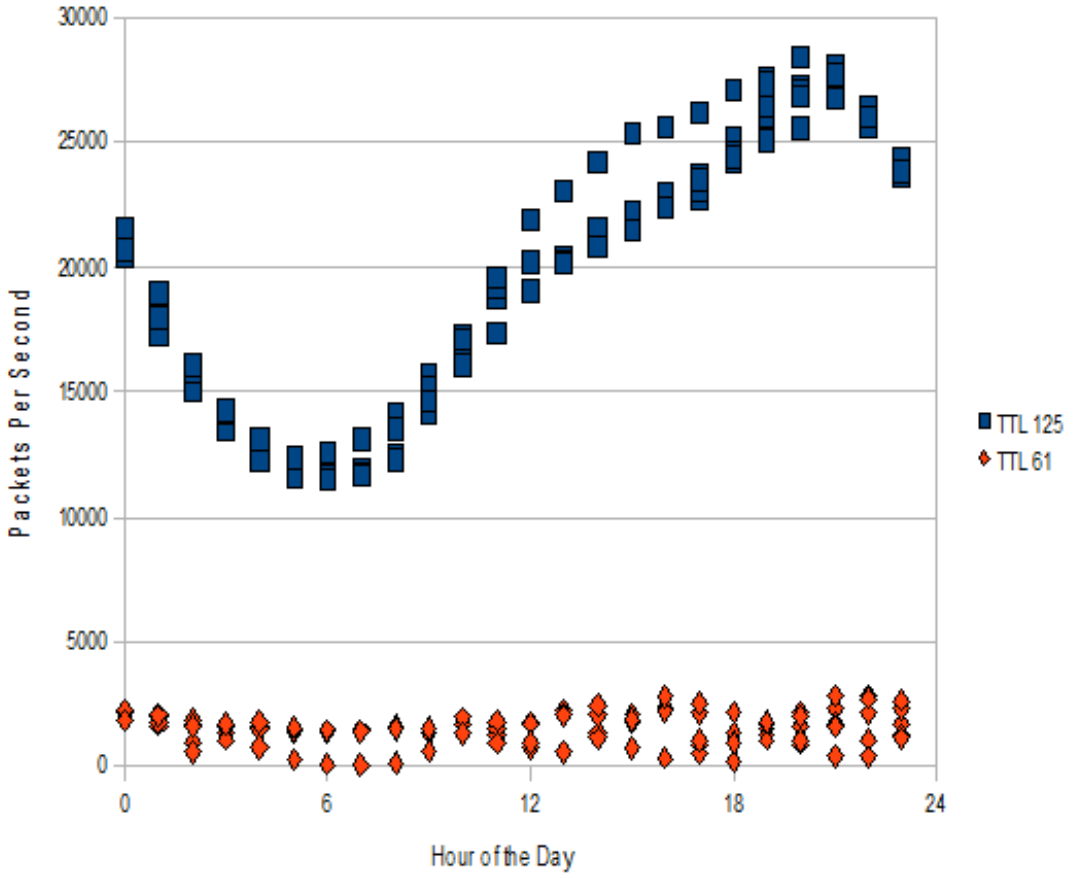


Figure 6.4: TTL Packets Per Second against Hour of the Day

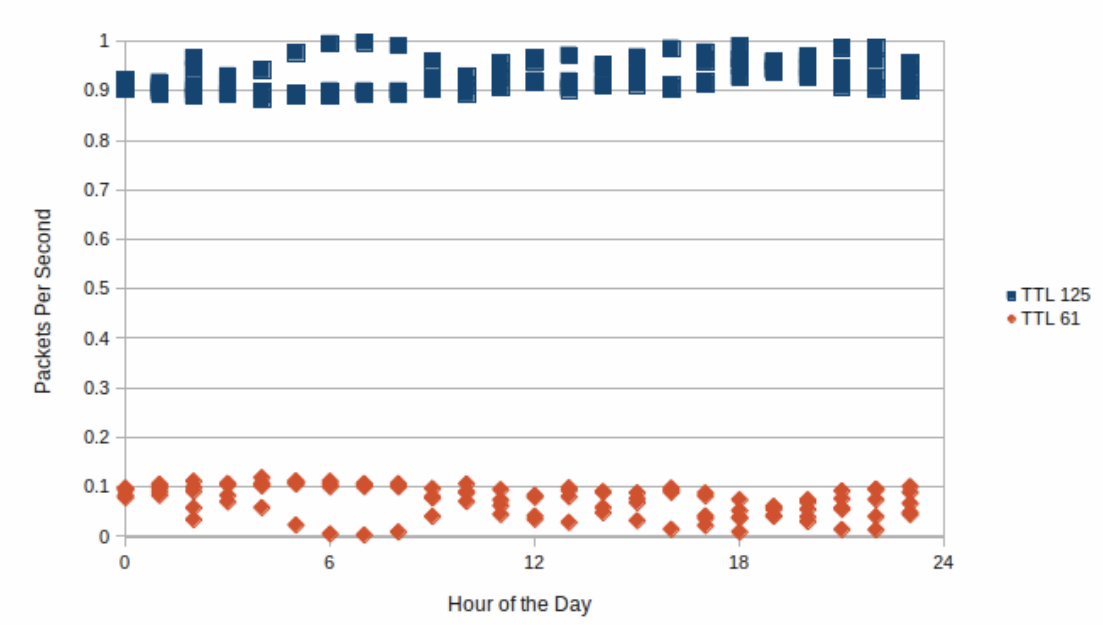


Figure 6.5: TTL Proportion against Hour of the Day

Figure 6.4 shows the number of packets per second against hour of the day. Of note, is that the TTL 125 (Windows) data appears to show a more patterned variation than the TTL 61 (Unix) data. However, inspection of figure 6.5 shows that the two TTL values have a fairly flat relationship compared with the time of day. This would suggest that there is no major variation in the times that the users of these systems are active.

6.4 Packet Sizes

In figure 6.6 the packet sizes are shown by protocol. ICMP and UDP packet sizes show no direct relationship with time of day. TCP packet sizes however appear to be higher during periods which are dominated by file sharing applications (see above). This is a result of TCP bulk transfer mode, where larger quantities of data are being transmitted, TCP will use MSS packets. On arrival of the data, the recipient will acknowledge this data. Over Ethernet a MSS packet is 1500 bytes [4], and an acknowledgement packet (no options) will be 40 bytes. Therefore, if all traffic were TCP bulk-mode traffic, the average packet size would be 770 bytes.

6.5 Port Usage

One of the most heavily utilised ports on many of the PoPs on the network monitored was port 80. This is commonly associated with HTTP traffic. Figure 6.7 shows the count of port 80 packets plotted for several days against the hour of the day[5].

As might be expected the volume of HTTP traffic drops off rapidly in the early morning, and peaks between 6 and 11pm. Figure 6.8 shows a similar plot for port 4662, which is commonly associated with eMule, a popular [6] peer-to-peer file sharing application. In contrast to port 80 traffic, this shows very little time-of-day variation. It could be speculated that this is because the peer-to-peer applications do not require user interaction, and therefore traffic generated by them is somewhat independent of user activity patterns. It may be common practice to leave computers downloading or uploading content with these tools while the user is asleep.

Another interesting view of the data is shown in figure 6.9 which shows the

[4] including IP / TCP headers

[5] This graph shows packets which were either sent to, or came from port 80, this differs from the data shown in Chapter 8.

[6] In 2005

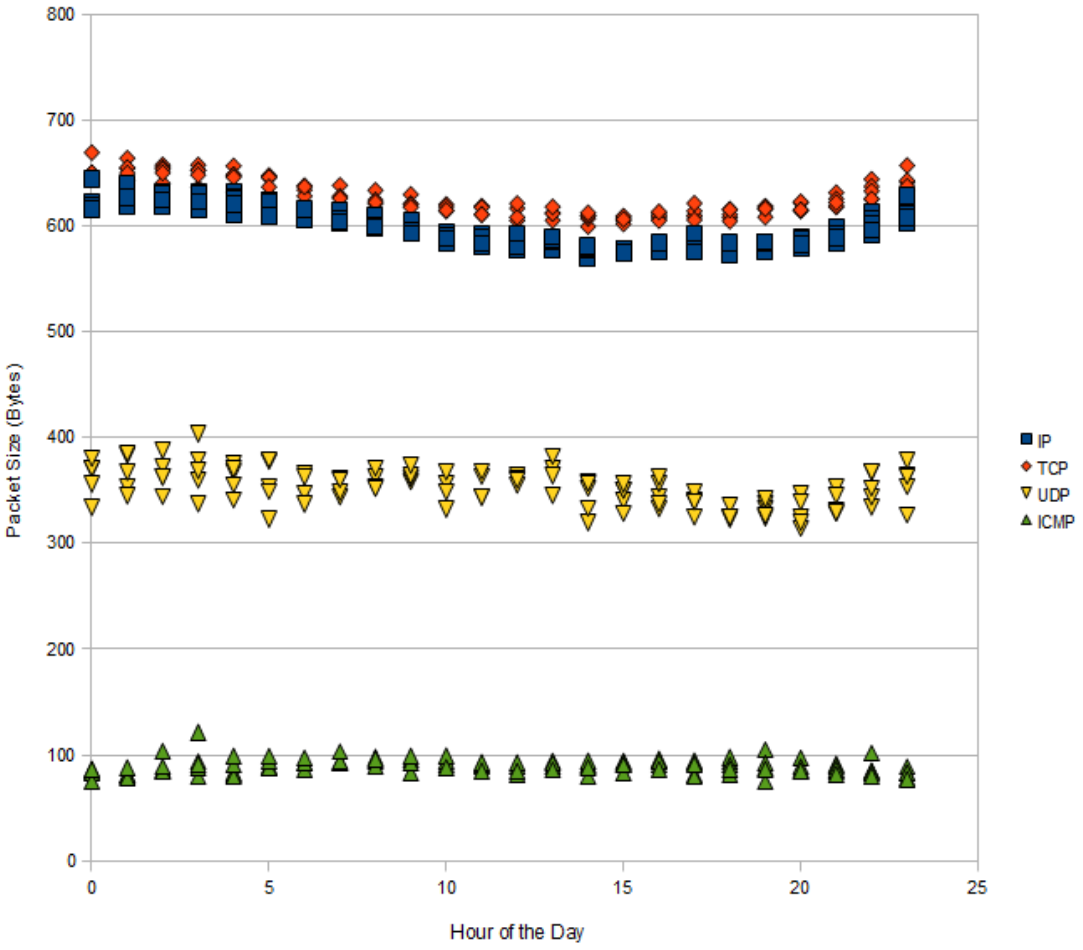


Figure 6.6: Protocol Variation as a Proportion of Overall Traffic

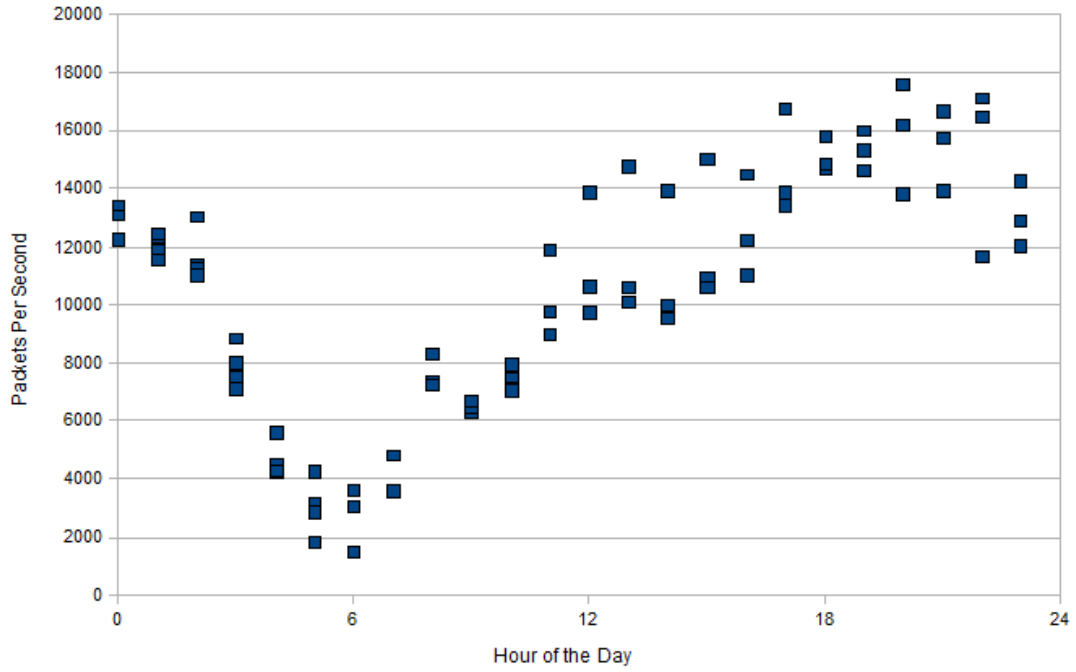


Figure 6.7: Port 80 Packets Per Second against Hour of the Day

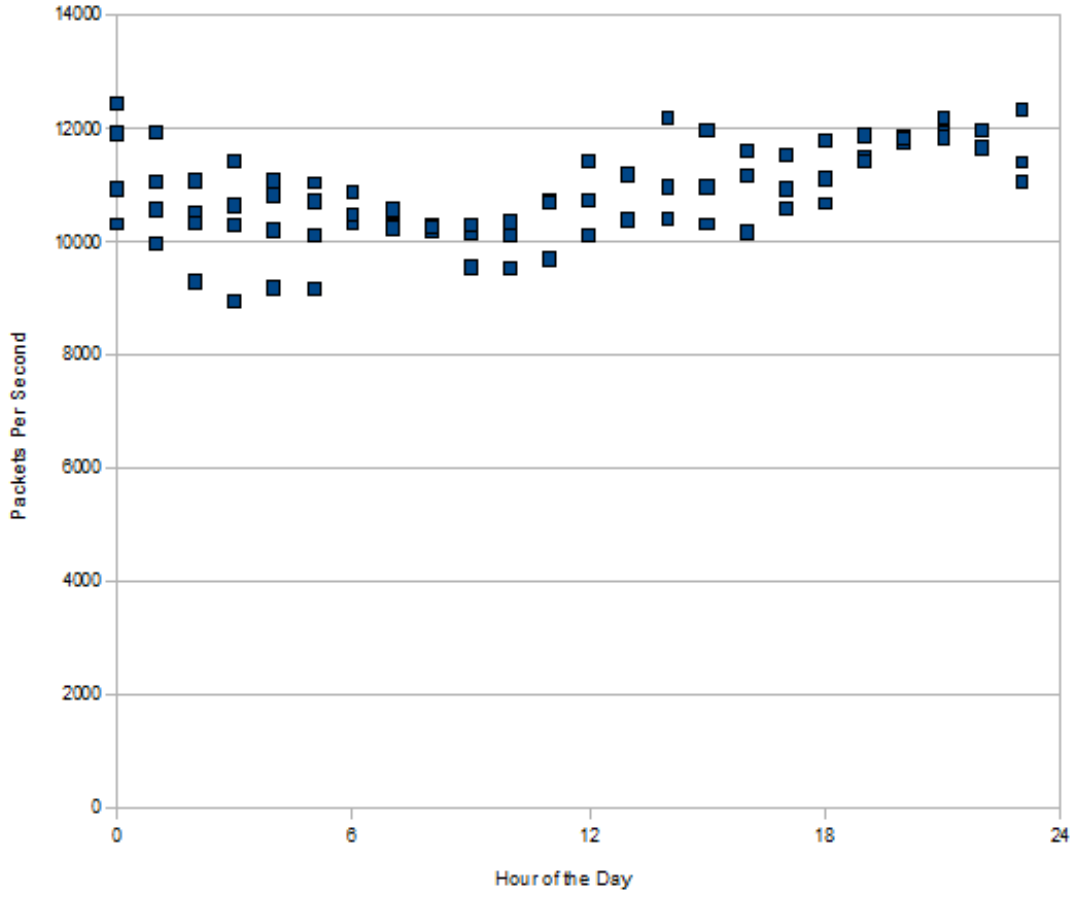


Figure 6.8: Port 4662 Packets Per Second against Hour of the Day

contribution to overall traffic made by several of the highest utilised ports. There is a trend which shows the use of the network becoming more diverse (smaller contribution from the major ports), and dramatic drop off in use for particular applications. The cause in this case does not appear to be the application falling into disuse (Gnutella was reputedly the highest used file sharing software in 2007). We can only suppose the application protocol changed behaviour, and stopped relying on TCP port 6346 as heavily[7].

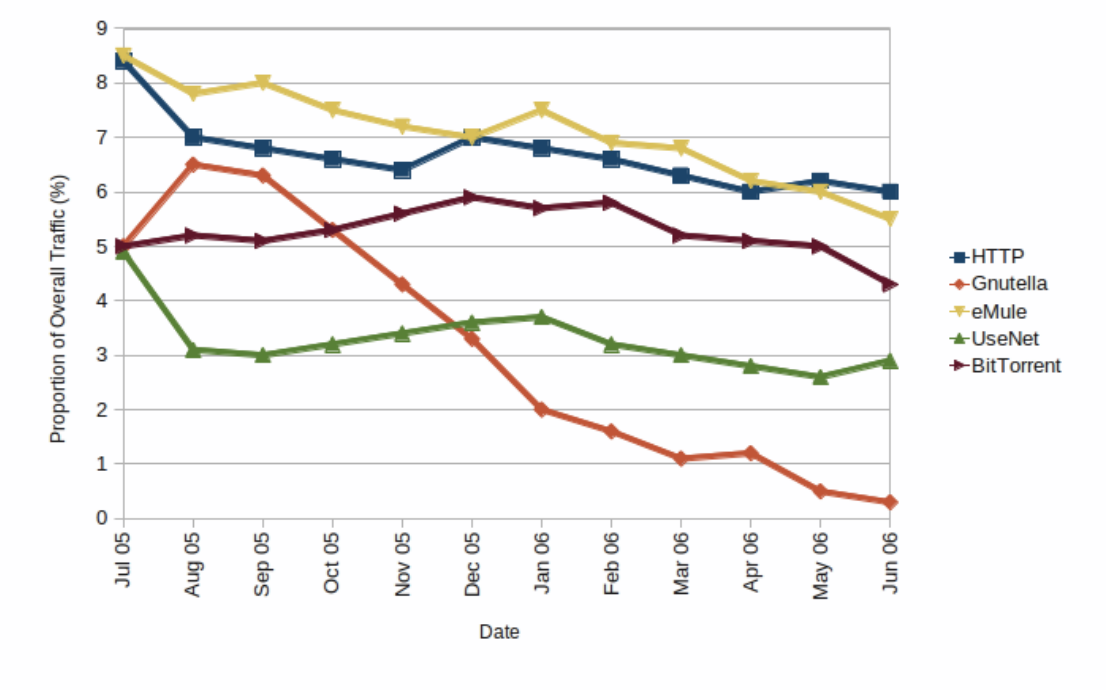


Figure 6.9: Proportion of Overall Traffic from Major Ports

6.6 SMTP Monitoring

During the period in which the system described in this thesis was operating, the issues of unsolicited email was raised. There was at the time an unknown volume of unsolicited email being sent over the monitored network, and a desire to understand the impact which this traffic had on the overall operation of the network.

It was thought that it should be possible to determine differences between certain categories of SMTP generators through the use of summary statistics. To differentiate between potential categories of SMTP generator, statistical summaries of the network traffic were recorded over a 24 hour period. A total of 89,748

[7]Gnutella introduced UDP support and other changes during its lifespan

hosts were observed sending SMTP traffic. Of those senders, 46 hosts were responsible for over 10,000 SMTP connections. The distribution of source addresses, destination addresses and connection numbers varied significantly between groups of senders. These groups were categorised theoretically as follows.

- Mail Server

  Mail servers are simply agents which receive and send email messages on behalf of users. They do not represent illegitimate use of the network. This group was characterised by a time of day variation in the number of email transfers alongside a relatively even distribution of sources and destinations.

- Mail Client

  A mail client is a program which will send and receive email from a mail server. Mail clients are a legitimate use of the network. This group was characterised by a time of day variation in the number of email transfers, a single source and a single destination.

- Open Mail Relay

  An open mail relay is a device which will forward email from any source to any destination. This is distinct from a closed mail relay in its indifference to the sender's identity. Open relays do not in and of themselves represent illegitimate use of the network, however they have been abused by users wishing to send unsolicited email [78]. This group was characterised by a time of day variation in the number of email transfers, a small number of sources and a large number of destinations.

- Bot

  A "Bot" is a machine which has been compromised by an attacker, with the potential to be used for various purposes. One potential use is to generate unsolicited email. Bot generated emails potentially represent illegitimate use of the network. This group was characterised by a constant number of connections throughout the day, a single source, and a large number of destinations.

This work was published, and a more complete discussion of the findings can be found in the paper attached to appendix A. It further demonstrates the potential for summarised data to provide insight into complex problems.

6.7 Summary

This chapter has described some of the characteristics of the monitored network under periods of relative normality. The time of day variation in data rates, port usage, protocols, TTL and packet sizes have all be examined and discussed.

Of particular note is the strong patterns in all parameters described. Each metric follows a tight distribution for a given time of day, allowing them to be modelled and predicted with some accuracy. This is encouraging as it suggests that deviation from these distributions should be marked.

The next chapter will present data taken from the live national network, the analysis of this data, and some examples of DoS attacks observed.

Chapter 7

Live Data

In the previous chapters various data mining techniques were discussed. In this chapter they are applied to live network data and used to classify live network events.

Once the laboratory based investigations into the statistical variances caused by malicious activity were complete, the focus moved onto recording and analysing a live data set.

The system described in chapter 4 was implemented in the core of a national network. The system recorded summarised statistics derived from packet headers continuously for a 6 month period. All summaries were stored in a MYSQL based database. The schema for the database was simply a flat table with rows inserted for each summary on each site. This was deemed adequate for the purposes of this research as the volume of data produced in the summaries, even over a large period of time, was small in database terms and quickly searched as desired.

The summaries recorded were investigated using the Clementine data mining package, with an emphasis placed on periods of significant deviation. This deviation was initially classified manually, informed by the laboratory experiences and, in specific examples, corroborated by the ISP's network team. The mechanism for achieving this was to use laboratory generated neural networks to help highlight periods of anomaly, then to visually inspect via graphs of particular fields. Later in the research as the neural classifiers became more sophisticated the system required less manual intervention, and could be used to classify most of the data in a semi-autonomous manner. As part of the research, reports were generated on a monthly basis, describing the activity on the network.

7.1 Gathering the data

Over a period of 6 months, a database was created with examples of anomaly, with classifications ranging from large DoS attacks to network routing changes. The data set contained over 140 non-contiguous events chosen to be representative of the differing types of activity observed. The DoS attacks in the database included both spoofed and non-spoofed examples of TCP-SYN flood based attacks, UDP flood attacks, from within and outside the monitored network. Several attacks were composite attacks, containing more than one type of traffic. The attacks were taken from all 6 monitored sites, which had mildly different data rates, host variance and general usage patterns. Along with the DoS samples, the surrounding 'normal' summaries were also included in the database.

In order to provide suitable training data, the normal samples were chosen to be of a similar duration and number to that of the DoS attack they surrounded. This ensured an evenly weighted data set with which to train automated models.

Different sections of this database, as well as the data in its entirety were used as inputs for different data mining models. These models were compared using the input weighting technique described in section 5.2.3.

7.2 Training

Initial investigation was done via self organising maps (SOMs). Two types of SOM were used, the first is a standard SOM based on the Kohonen algorithm. The output of this is shown in Figure 7.1. The X and Y axis represent the grid coordinates of the output matrix; the colour indicated the presence of DoS.

As can be seen, there is a clear divide between the regions containing DoS and Non-DoS samples. This is encouraging as it suggests there is significant variance in the two sets.

Further investigation was done with a sub-type of SOM in which the data passes through two phases. In the first phase, the algorithm determines the appropriate number of 'groups' (i.e., the X,Y pairs in the standard SOM model), and in the second phase it defines these groups.

In Figure 7.2 the Y axis shows the second of the day when the sample is taken, this is to provide a method to separate the points. As can be seen, firstly the algorithm selects that there should be two groups in the data, meaning that there are two clusters in variable space. Secondly these groups appear to be divided by DoS activity.

The research moved its focus from SOMs and Clustering algorithms to look for which variables in the summaries were providing accurate classification. This

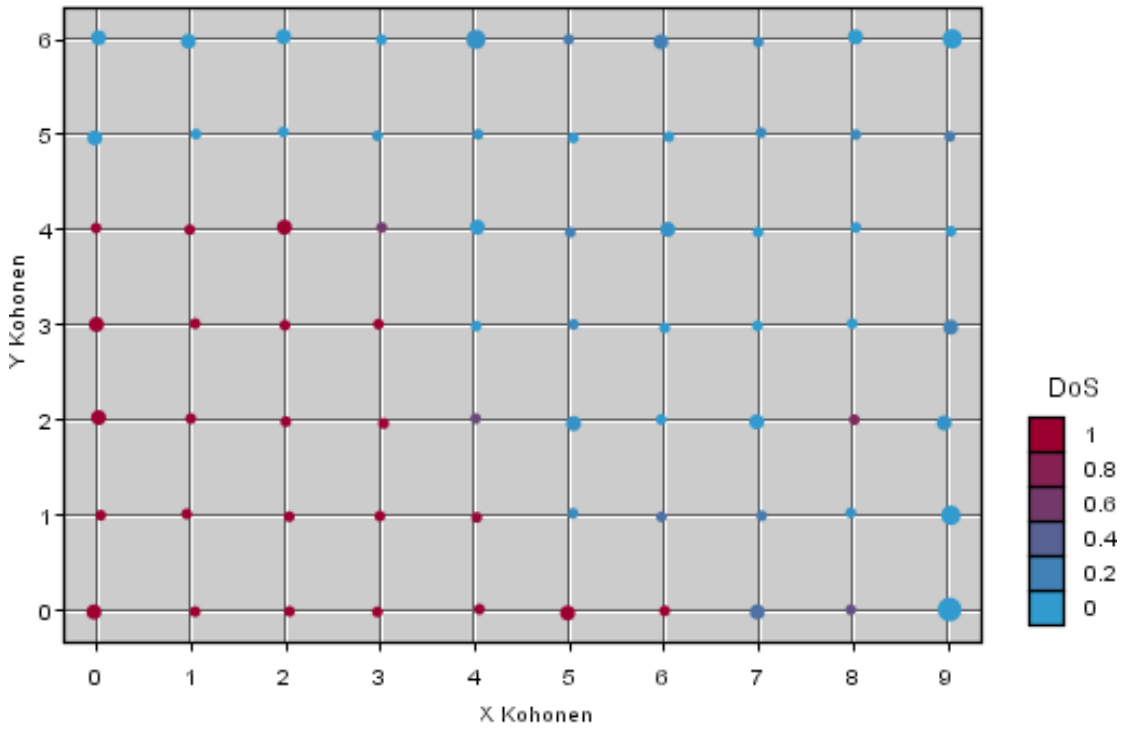


Figure 7.1: Self Organising Map

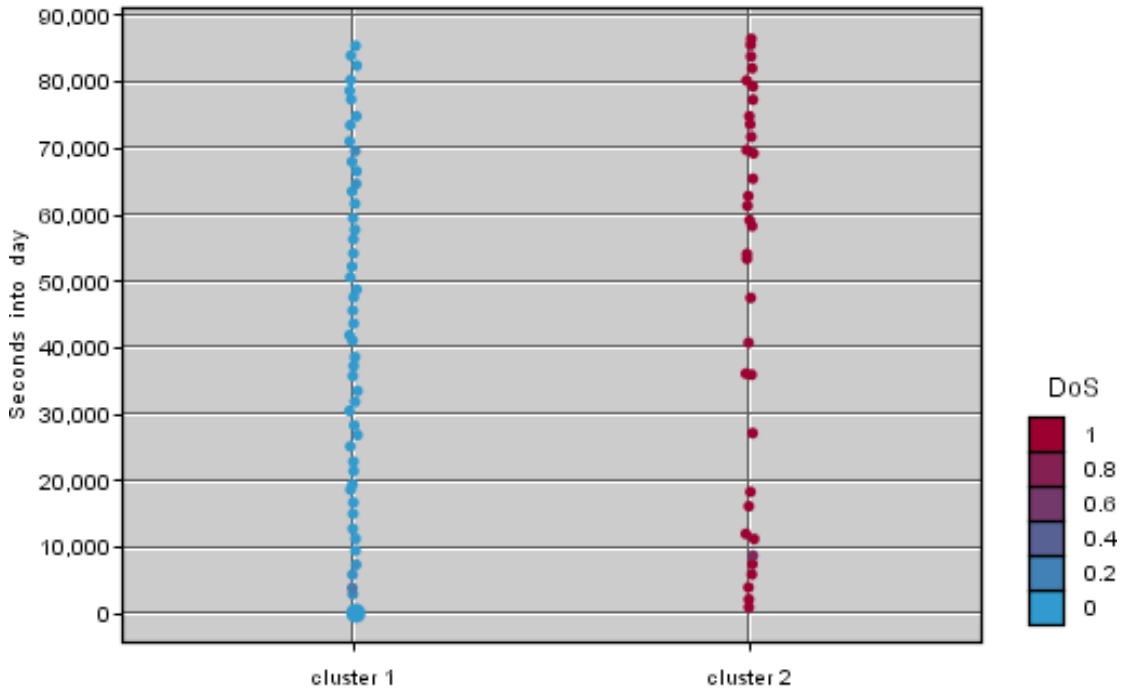


Figure 7.2: Clustering Algorithm

was achieved by analysing different classes of DoS for the highest weighted input nodes.

We applied this technique to individual DoS attacks, DoS attacks grouped by type and to the whole data set.

7.3 UDP DoS Example

To better illustrate the variances in the data under DoS conditions, two examples will be discussed. The two examples chosen are an IP address spoofed, bandwidth based UDP flood attack which existed sporadically over a one day period, and a non-spoofed TCP SYN flood based DoS attack, which lasted for seven minutes. The first example we examine is the UDP based attack

In the DoS database, the UDP denial of service attacks observed were exclusively bandwidth attacks. Mostly they are short lived[1], and of a smaller scale than those generally reported in the media.

Discussion with the network operator and with external information assurance bodies indicates that short lived DoS attacks are much more common than longer, sustained attacks on this type of network.

In this example there are a short series of UDP floods. Figure 7.3 shows the number of packets per second received overlaid with DoS Classification. As can been seen, there is a general level variation in packets per second over the 24 hour period, This is due to the fluctuating levels of load observed on the network based on consumer usage. The time of day variations seen are predictable, and consistent between sites. Under the periods of DoS the packet rate increases in a manner not consistent with the daily variation.

In this attack, IP address spoofing was used to hide the identity of the attacking hosts. Figure 7.4 Shows the variation in the number of IP subnets seen per packet sample overlaid with DoS classification. The DoS generator in this case was sending each packet with its own unique source IP address, leading to an almost one to one mapping of additional source addresses to additional packets.

7.4 TCP DoS Example

In this analysis an overview of the changes in the monitored summary statistics during a SYN attack is presented. The SYN attack is a well known DoS attack that aims to exhaust all the available connections at a particular server (see 2.3.3).

[1]It is possible to speculate on why this is; a home user network will have few attractive targets for the usual extortion type DoS activities, and often is subject to petty DoS vandalism between disgruntled users.

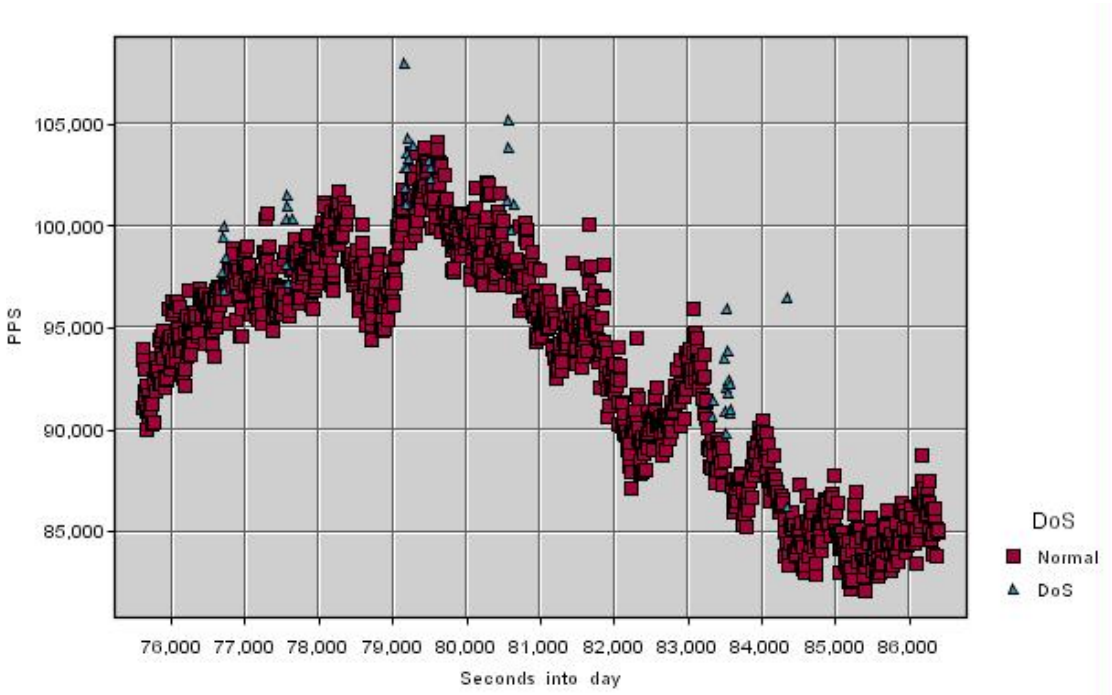


Figure 7.3: UDP DoS - Packets Per Second

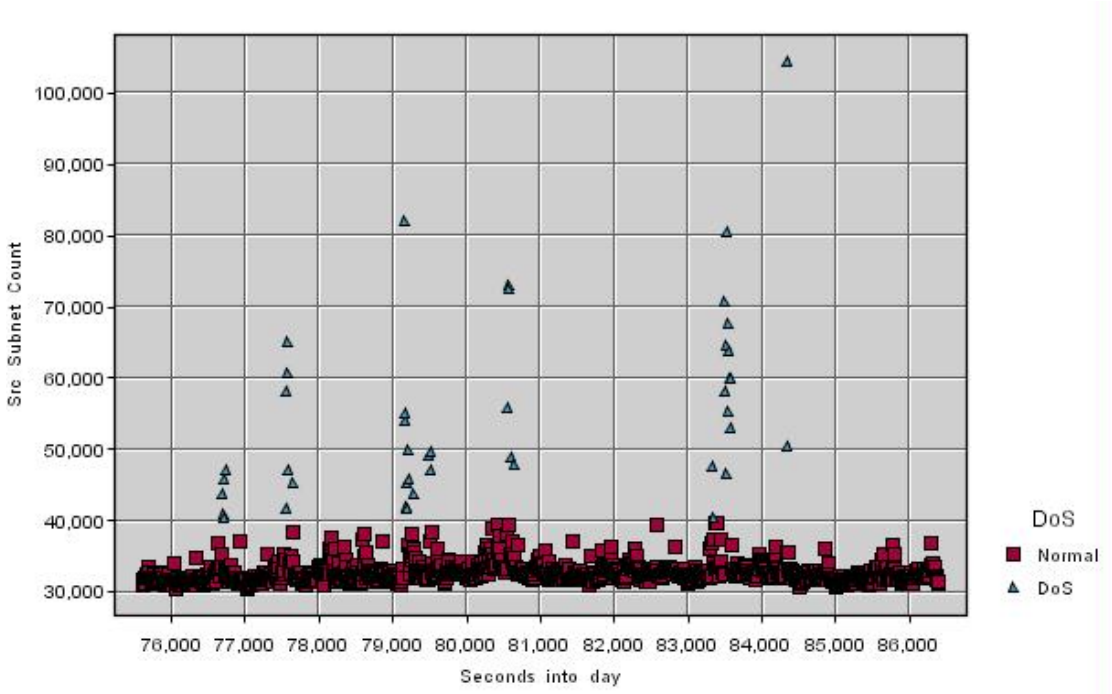


Figure 7.4: UDP DoS - IP Subnets Per Sample

Figure 7.5 shows a huge increase in the number of packets per second with the SYN flag set. This attack was relatively short lived, lasting only 7 minutes.

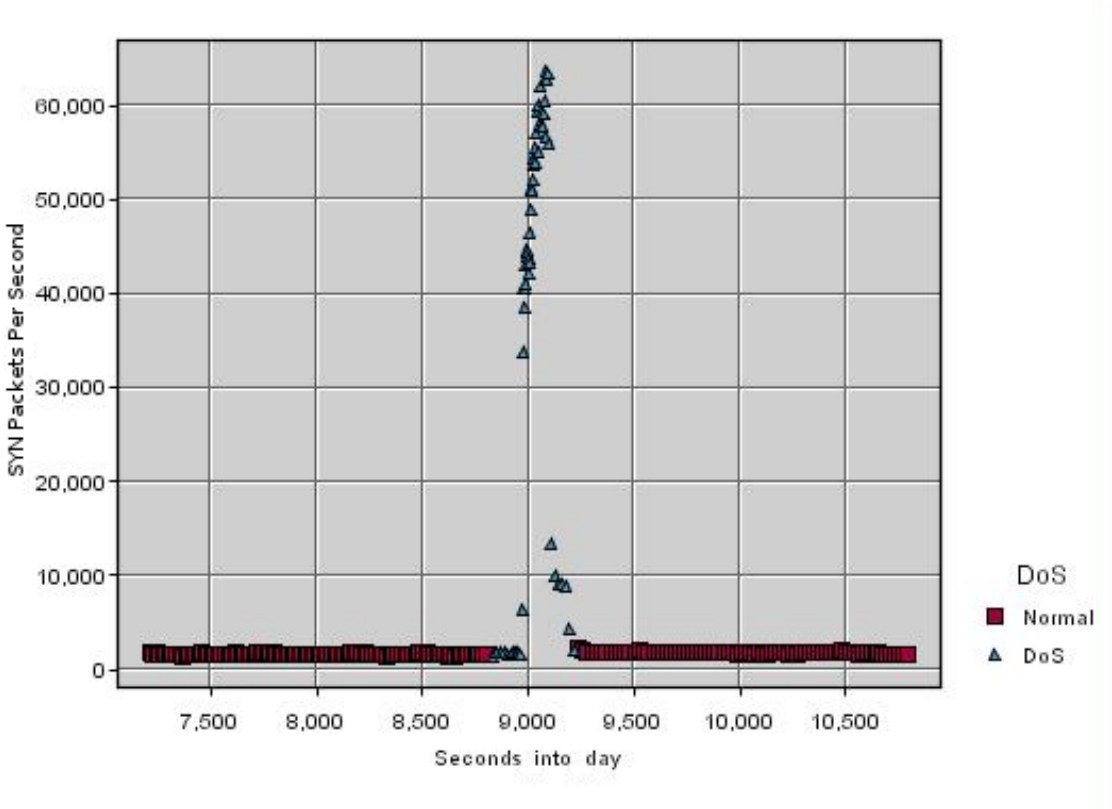


Figure 7.5: TCP DoS - SYN Flags Per Second

As should be expected, Figure 7.6 shows the corresponding reduction in the average packet size observed [2].

The IP Source addresses, per second and per sample, are shown in Figure 7.7 and Figure 7.8.

While there is an increase in source addresses per second, indicating that a larger number of hosts are sending traffic, there is a decrease in source addresses per sample, indicating that the majority of packets are being sent from a smaller set of hosts.

7.5 Feature Selection

This work sets out a methodology for recording lightweight statistics to detect network anomalies, and for determining which statistical measures should be used within a constantly changing environment.

[2]Syn Packets are around 40 bytes long (0 TCP data bytes), depending on which TCP options are selected

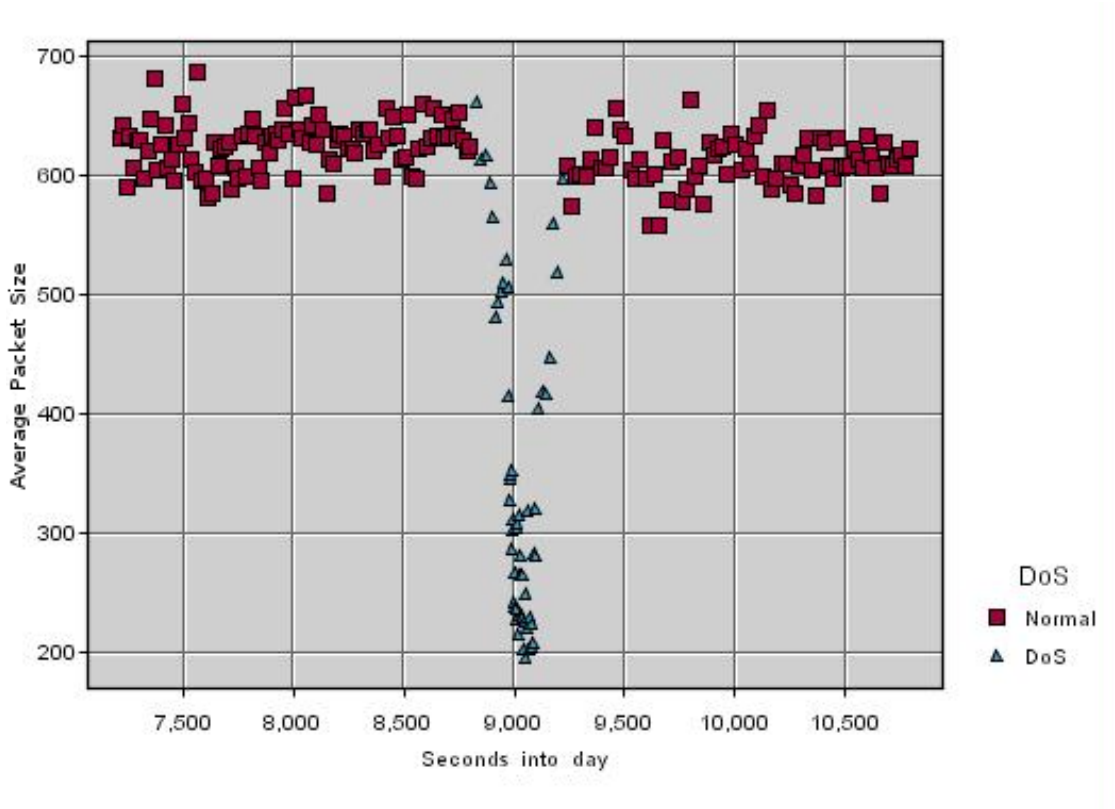


Figure 7.6: TCP DoS - Average Packet Size

With this in mind this section describes how data mining was used, not only to determine periods of anomaly, but also to identify which features were significant in this classification.

Using the DoS examples from the previous section, this analysis calculates the relative importance of the inputs to the data mining engine. This technique works by evaluating the ANN weightings for each input along the decision path.

Table 7.1 shows the relative importance of the inputs to the ANN used to classify data. In the UDP flood example, the most important input into the ANN was the IP address counts. This is no doubt due to the spoofed nature of this attack. UDP traffic under normal conditions has a low representation on the network (see figure 6.3), and therefore when large quantities of UDP traffic is observed it strongly affects the distribution of protocols. Finally of note is the variance in TTL. In the earlier laboratory based experiments, TTL had not been a significant contributor to detection [3]. However, in the live data examples TTL becomes a strong indicator of the presence of DoS. This is discussed in chapter 7.

Moving to the TCP SYN flood example, the analysis suggests that the value of the most commonly seen packet size is a highly significant factor in determining

[3]Due to the limitations of the accuracy of the experiment

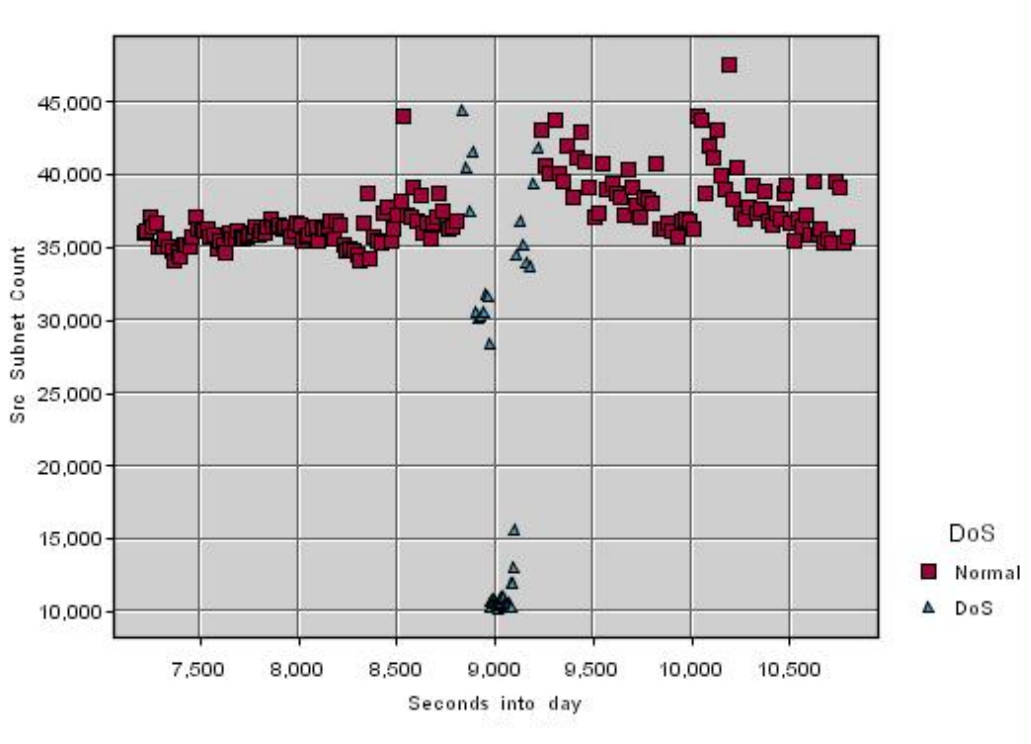


Figure 7.7: TCP DoS - Source Subnets Per Sample

Table 7.1: UDP DoS Relative Importance of Detection Metrics

Field Name	Relative Importance
IP Address Counts	0.571429
Packet Sizes	0.165713
Protocol Counts	0.148571
TTL Field	0.142857
ID Ratio	0.028571

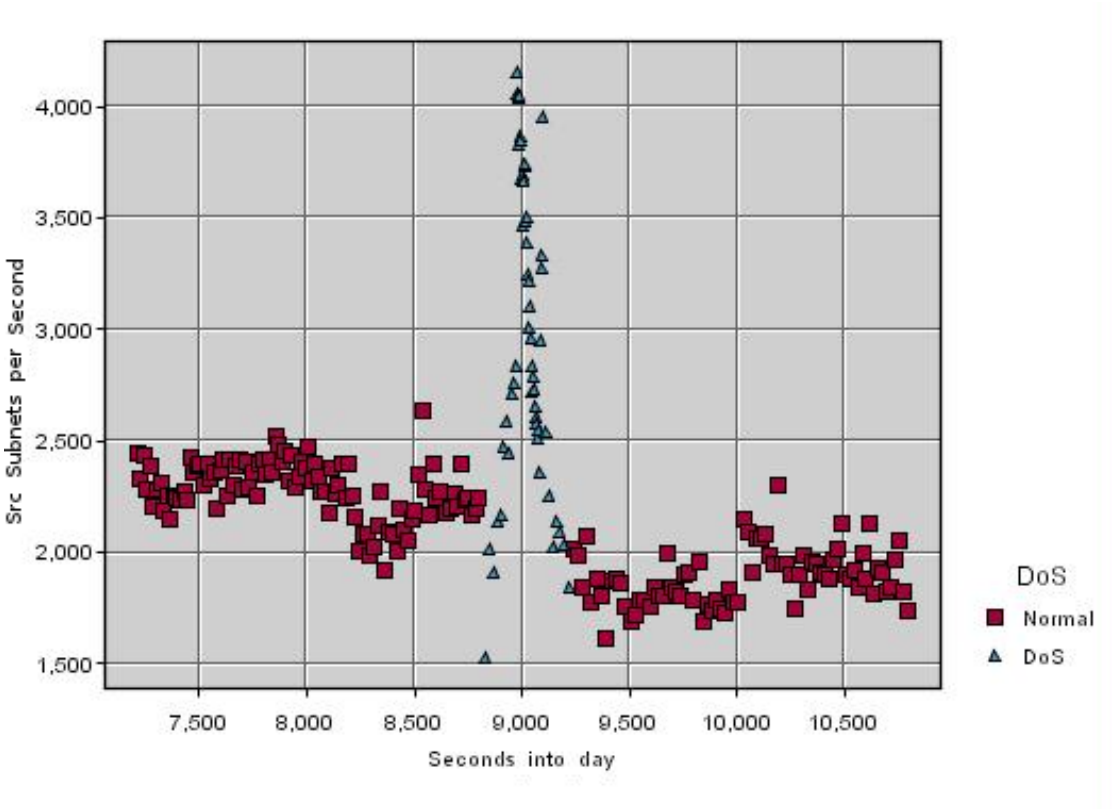


Figure 7.8: TCP DoS - Source Subnets Per Second

the presence of this type of attack. Perhaps surprisingly, TCP flag counts were considered only the 4th most significant input.

Since at the onset of the attack the number of packets with the SYN flag set was still relatively small, it may be that the changes in fields such as the ID header field, or the packet size, were more reliable indicators.

IP source address and destination address counts were also significant factors, which was corroborated in Figure 7.7 and Figure 7.8.

Table 7.2: TCP DoS Relative Importance of Detection Metrics

Field Name	Relative Importance
Packet Sizes	0.4375
ID Ratio	0.265625
TTL Field	0.140625
TCP Flags	0.09375
IP Address Counts	0.0625

The TTL field is again a prominent factor in the decision matrix here.

Figure 7.9 shows the relative importance of the inputs from the TCP, UDP and combination Laboratory emulated DoS. TTL is insignificant in the laboratory as it

was excluded from the analysis due to the lack of nTop data available to properly model it in the traffic generators.

The spoofing of traffic makes the IP Address Counts the primary factor in the UDP analysis, where as the fixed small packet sizes dominate the TCP analysis.

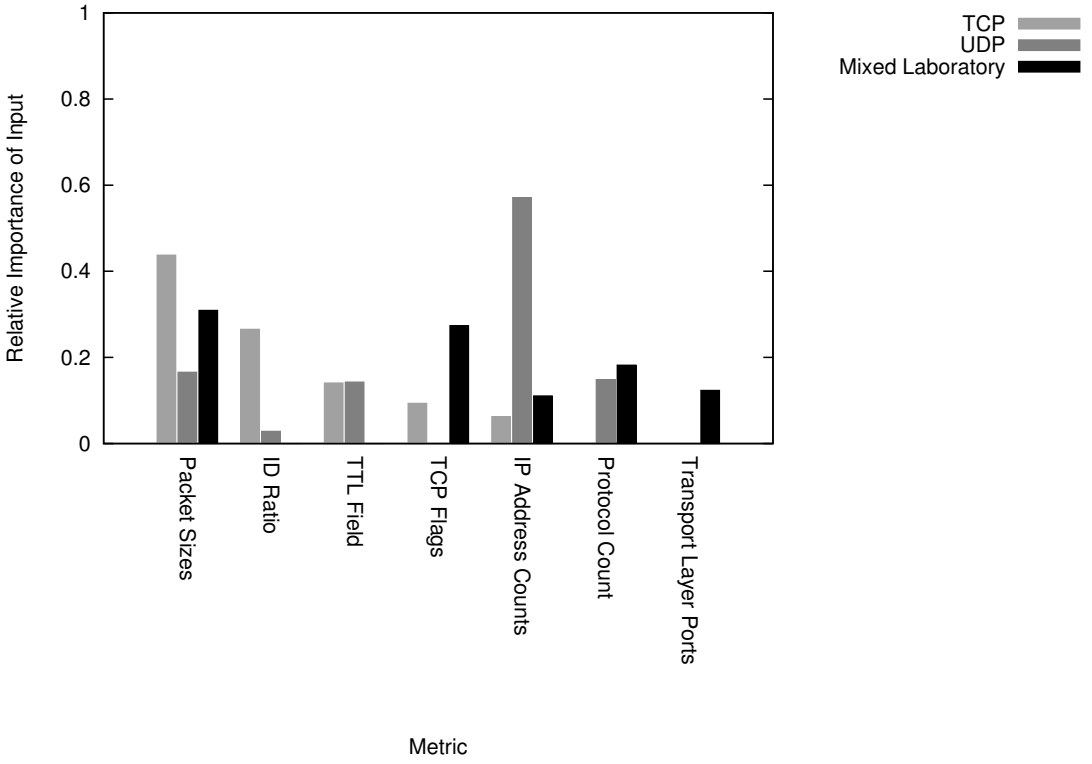


Figure 7.9: Comparison of Input Weightings

7.6 Malicious Attack Summary

The data used in this thesis was gathered between September 2005 and March 2006. A summary of the attacks analysed as part of this thesis is shown in 7.3.

Of the sites monitored, Oxford shows the most activity, particularly in November, December and January. It is unclear whether this is because of botnets present on this site, or whether there are hosts being targeted within it. In some periods, several similar attacks were observed per day for extended periods (as with Oxford in February 06). However, a more common picture for the system as a whole was for a small number of attacks to be observed in any given week.

Table 7.3: Network Attacks Observed, Grouped by Month

Date	Site	Attack Type	Count
September 05	Northampton	UDP Flood	3
	Oxford	UDP Flood	2
October 05	Brighton	UDP Flood	2
	Northampton	SYN Flood	1
	Oxford	UDP Flood	2
November 05	Colchester	UDP Flood	2
	Northampton	UDP Flood	3
	Oxford	UDP Flood	25
	Southampton	SYN Flood	1
December 05	Colchester	UDP Flood	2
	Northampton	UDP Flood	2
	Oxford	SYN Flood	10
		UDP Flood	22
January 06	Colchester	UDP Flood	1
	Oxford	SYN Flood	2
February 06	Colchester	UDP Flood	2
	Oxford	SYN Flood	52
March 06	Brighton	UDP Flood	3
	Northampton	UDP Flood	5

7.7 Summary

In this chapter the effect of attacks on the network were investigated using data mining techniques. Clustering techniques were shown to useful in organising the data into legitimate and illegitimate categories. A UDP based flood DoS attack, and a TCP based SYN flood DoS attack were examined and analysed using an ANN. Many of the attacks observed during the research show characteristics which are in common with attacks observed in [22] and [35]. The weightings of the inputs were calculated to give a relative importance value to the associated input fields.

This chapter has shown that separating DoS traffic from legitimate traffic is possible using a statistical metric based approach and a neural classifier. Further more, it has shown that analysis of the neural classifier allows for the evaluation of the relative importance of metrics.

In the next chapter the significant variables as selected by the data mining algorithms are examined to determine why they are considered significant, and to validate their selection.

Chapter 8

Discussion of the Data Mining Parameters

In the previous chapter the methodology for investigating the 'normal' network state was discussed, and the features for detecting DoS discovered. In this chapter the features discovered are discussed and their behaviours under anomalous conditions are examined. The events described in this chapter are all taken from the live national network.

8.1 Defining Normality (Feature Selection)

The research group at HSN Loughborough had previously shown how statistical measures for detecting change in distributions could be combined with a neural classifier to detect and classify changes in delay data [79]. The technique had been used to detect routing changes, network outages and time of day patterned changes. This concept was applied to the network in a much broader sense here.

The assumption made was that the persistent state of the network was normal. This is a matter of semantics; one could define normal as the activity which is non-malicious. This precludes the possible normal malicious traffic; an example of which would be worms such as 'Code Red' which has been present in network traffic for many years, and is now largely benign[1]. Instead we define a certain amount of illegal or malicious activity to be normal, and look for abnormal abnormalities.

This is a key definition, as instead of trying to classify what is happening on the network at all times, instead, we attempt to classify changes in the network state which are indicative of network based malicious activity.

It is clear that care must be taken in selecting which features of a data set to monitor. Which features, or fields, are selected directly affects which events and

[1]This virus attacks unpatched IIS servers, of which there are now very few uninfected

with what accuracy, are detected.

Taking the technique used on laboratory data in section 5.2.3, we took pre-classified live samples to train ANNs. Each for each event, the relative weightings of the inputs was analysed.

8.1.1 A Note on Distributions

Many of the distributions under DoS conditions presented in this chapter are samples of the overall distribution. For instance, in the case of the TTL field, only the predominant five peaks of the distribution were recorded. This is to save space in storage and does not prevent useful analysis of the data.

8.1.2 TTL Field Analysis

From the early analysis of the anomaly database, using the techniques described in the previous chapter, it was clear that certain fields were good indicators of general anomaly. The first example of this is the time to live field. To analyse this field in more detail a per-packet distribution was built over time and can be seen in figure 8.1.

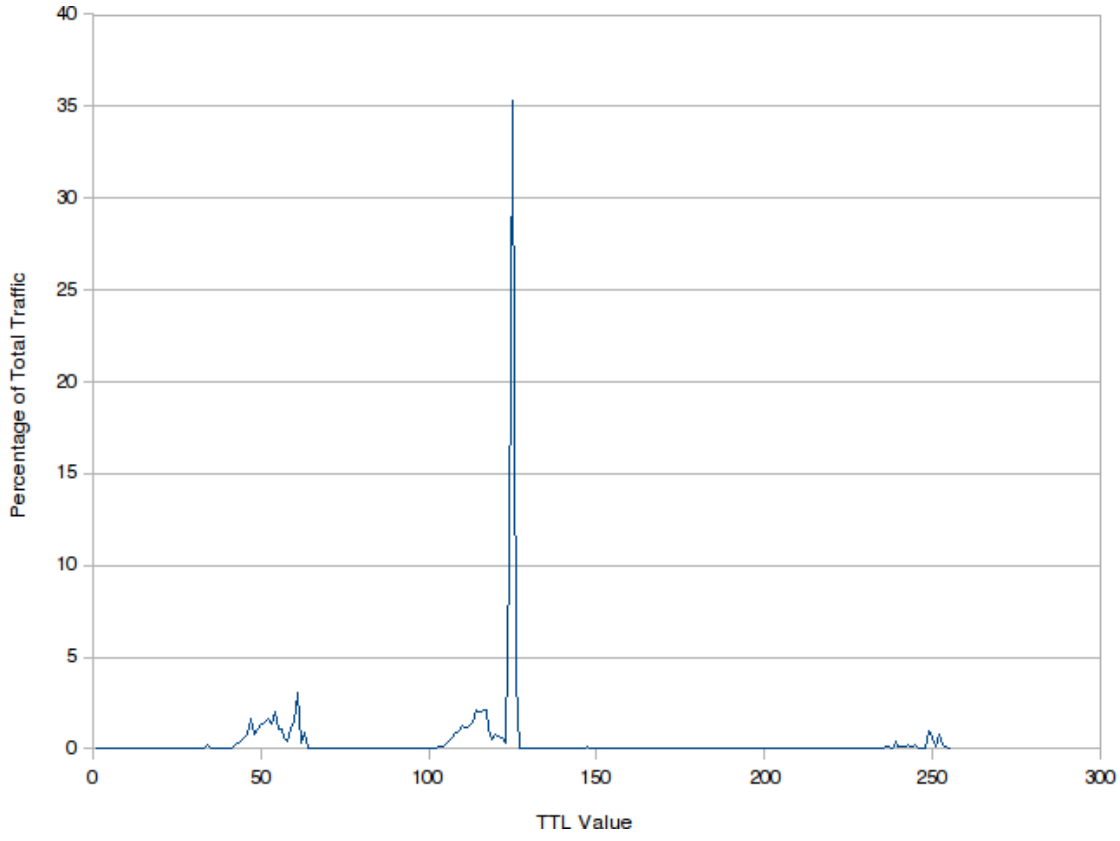


Figure 8.1: TTL Distribution Under Non-Attack Conditions

There are several observations and points of discussion to note from this graph. Most common operating systems have default TTL values set at the transmission of a packet. This value is often a configurable OS wide parameter, but is rarely changed from default settings. Table 8.1 shows common operating systems and the default TTL values associated with them.

Table 8.1: Operating System Default TTL Values

Operating System Name	Default TTL Value
Windows	128
Linux (2.4 Kernels)	64
OpenBSD	64
SOLARIS 8/10	64
AIX	64

When an IP packet leaves a machine, it will have the default time to live. Each hop[2] it passes will decrement the TTL. The TTL distribution of a monitoring point will depend on it's distance in hops from all hosts which it monitors.

The predominant peak seen in figure 8.1 is at 126 which would be consistent with representing all outgoing traffic from the ISP local POP from Windows based machines. We can also see a trail of IP addresses below this which are likely to represent incoming traffic from Windows based systems. This pattern is repeated around TTL 62 and below, representing the Unix hosts. There are also a scattering of maximum TTL packets (255 and a small trail) which can be associated with less common OSs such as Cisco IOS.

This distribution is a useful one as it describes the probability of packets originating from specific numbers of hops away from our monitor. This distribution is disturbed by some abnormalities in the network, such as routing changes or DoS. Should an additional router be placed between the local POP and the monitor, we would expect to see the peak at 126 move to 125 and so on.

Figure 8.2 shows the same distribution overlayed with the distribution from taken from a period of DoS. As can be seen, there is a new peak formed by packets with a TTL of 40. The fact that this is a single peak suggest that the majority of packets during this period originated either from a single host, or hosts on a single network.

From this, and the ANN weighting analysis, it is clear that not only is the TTL field a good indicator of the presence of DoS, but also that it is a potentially difficult metric to obscure. For an attacker to effectively match the distribution present on the network, they would need to have an in depth knowledge of the

[2]A hop in this context meaning a layer 3 device such as a router

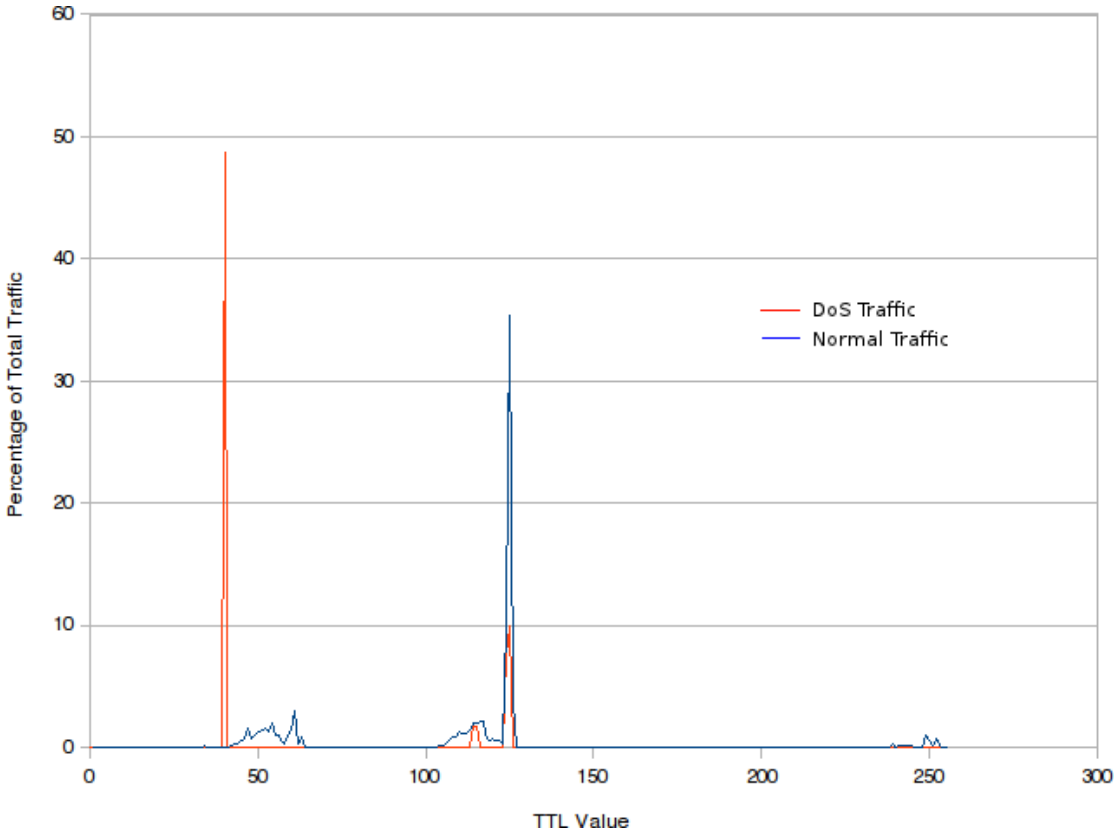


Figure 8.2: TTL Distribution Under DoS Attack Conditions

target network, the monitor's position in relation to the attacking machines and the position of the monitor in relation to the network it observes.

The combination of these factors make TTL not only an accurate indicator of DoS traffic, but also an extremely resilient one. This type of analysis is however, only effective for specific types of DoS or Worm attacks. In the case reflector style attacks minimal change in distribution would be observed. Assuming that the reflector attack used many well known web hosts, the TTL distribution entering the network will be very similar to its natural state.

8.1.3 Packet Size Analysis

The next metric we will examine will be IP packet size. Packet sizes are an interesting subject in and of themselves. The HSN research group at Loughborough have published work on how packets sizes may be used to perform application detection [80], and it follows that the packet size distribution on a network should have some correlation with the applications which are running over it. As a wider part of this work, packets size distributions were effectively used to create a snapshot of the applications present on the network at high data rate.

Figures 8.3 and 8.4 show the packet size distribution of the network under

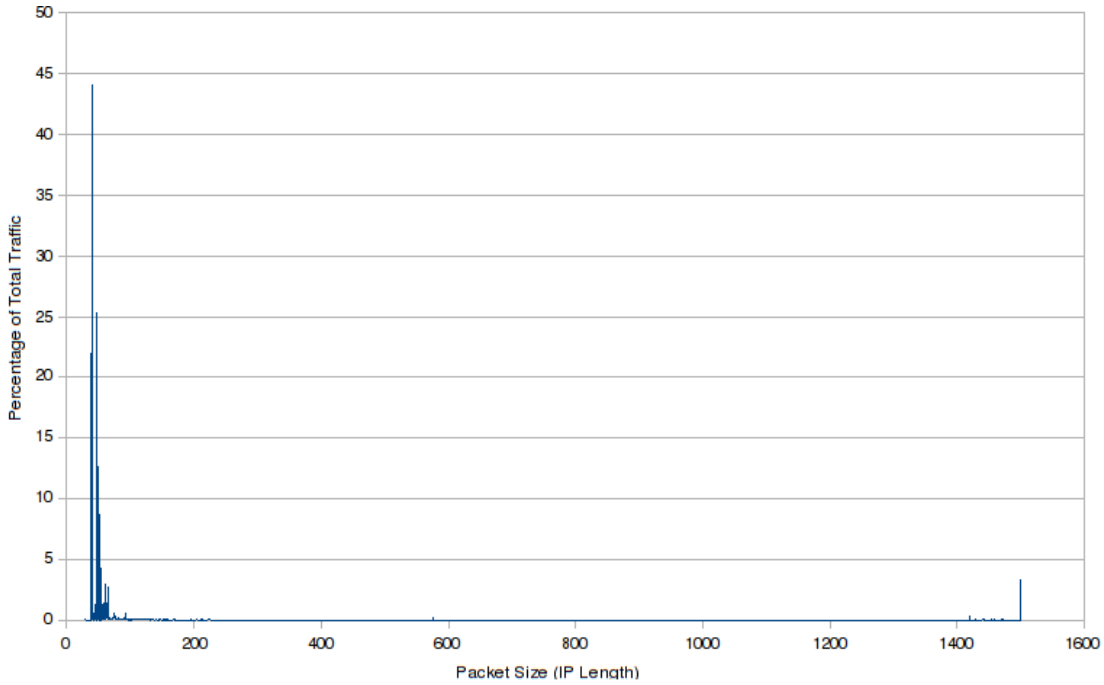


Figure 8.3: Packet Size Distribution Under Non-Attack Conditions(full scale)

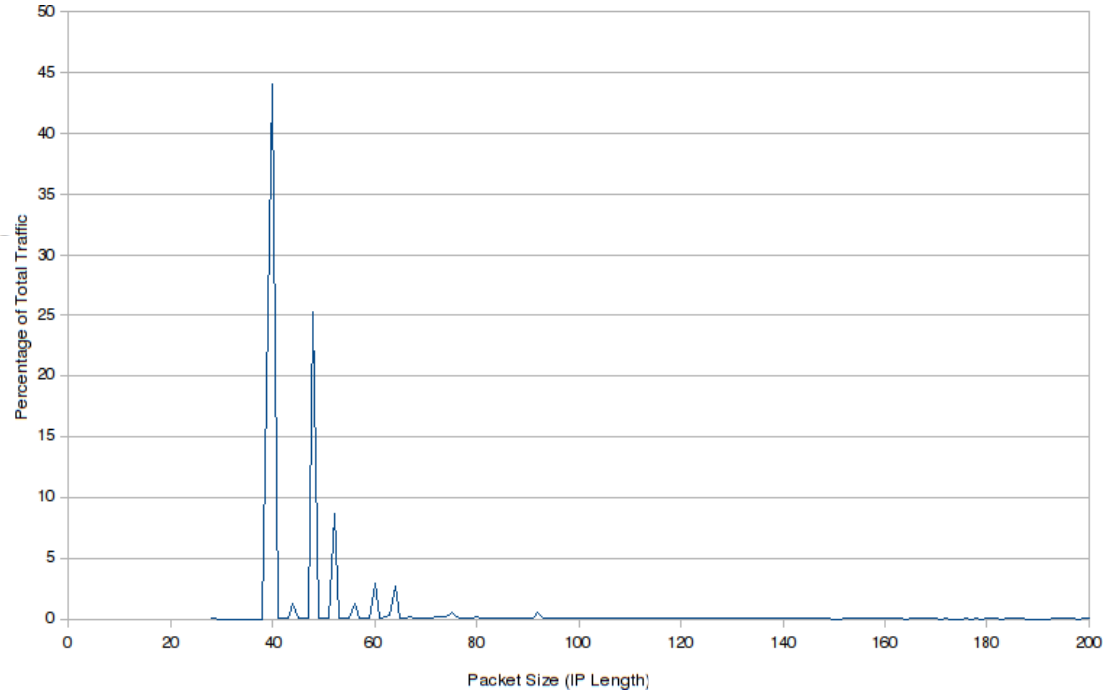


Figure 8.4: Packet Size Distribution Under Non-Attack Conditions(small scale)

normal conditions. As can be seen, there are significant points around 40-60 bytes and at 1500 bytes. The maximum transmission unit (MTU) of the network is 1500 as the network is using Ethernet v2.

TCP traffic makes up the majority of traffic on the network monitored, and is therefore the prime contributor to this distribution. At the smaller end of the distribution, we see peaks at 40, 44, 48, 52 and so on. The smallest TCP packet size is 40 bytes [3], this size would typically be seen on ACK, RST type packets. TCP also has a range of options such as Window size scaling, or time stamps. These options all [4] have a length of 4 bytes, which explains the 4 byte intervals in our packet size distribution.

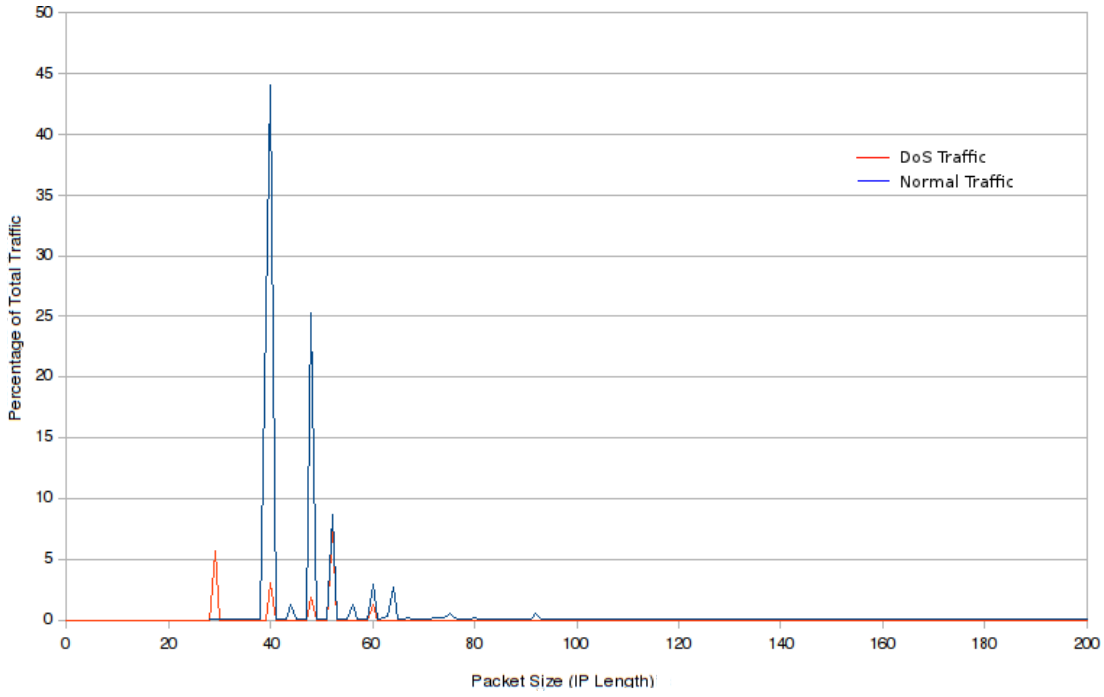


Figure 8.5: Packet Size Distribution Under DoS Attack Conditions(small scale)

Figure 8.5 shows the same packet size distribution, overlaid with one taken under a DoS attack. The major feature change here is a new peak at 29 bytes. In this example, the new peak was entirely made up of UDP packets, which were flooding the network.

Packet size distributions are determined by the types of traffic present on the network. As discussed earlier, the traffic has a predictable distribution, which may make it relatively simple for a potential attacker to replicate a similar distribution at the traffic generators, making this type of detection difficult. This would be more difficult in the case of a SYN style flood as the nature of the attack limits

[3] IP Length field of 40.

[4] With the exceptions of 'End of Option List' and 'No-Operation'.

the packet size distribution.

8.1.4 TCP Port Analysis

The final metric analysed in this section is the TCP port number. As with the other metrics, this distribution will be strongly related to the types of traffic present on the network. Networked applications have defined, or at least default, ports on which they are hosted [5]. The most obvious example of this would be TCP port 80, which is most commonly associated with HTTP traffic. As can be seen in figure 8.6 the most significant peak occurs at TCP port 80, which given the nature of the network in question, is predictable[6].

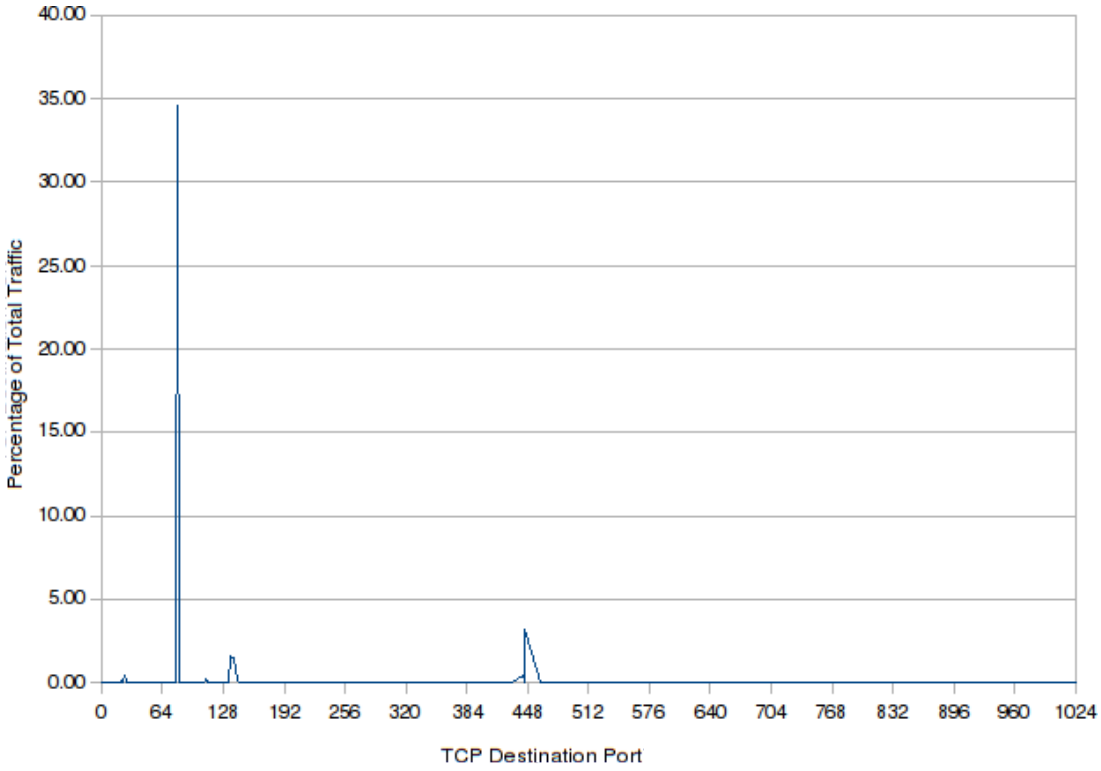


Figure 8.6: TCP Port Distribution Under Non-Attack Conditions

Table 8.2 shows the most common TCP port usage in order of significance. This distribution shows a time-of-day variation with HTTP usage showing a large decline in the later evening, with the ports associated with file sharing show less deterioration. It is perhaps worth emphasising that this data shows which TCP ports were in use, and that this is not a direct correlation to application usage.

[5]There are some exceptions to this.

[6]This is an analysis of TCP destination port at a specific point in time. For a description of source and destination port usage over time, see Port Usage in the Operation of the System chapter

Table 8.2: Highest TCP Port Usage and Associated Applications

TCP Port	Percentage of Total Traffic	Associated Application
80	34.57	HTTP
4662	3.22	eMule (P2P File Sharing)
445	3.16	Microsoft Directory Services
6346	2.33	Gnutella (P2P File Sharing)
135	1.57	Microsoft DCOM
139	1.55	NetBios
6881	1.37	BitTorrent (P2P File Sharing)
8080	1.08	HTTP (Alternative)
6348	0.67	Gnutella (P2P File Sharing)
25	0.44	SMTP
443	0.42	HTTPS

It is reasonable to assume however that the majority of the traffic on a particular TCP port will be attributable to the associated TCP application.

Figure 8.7 shows the distribution of ports under a SYN based DoS. The flood was not significant enough to appear in its own right in the statistics, however, there is a significant drop in the proportion of total traffic contributed by the well known ports.

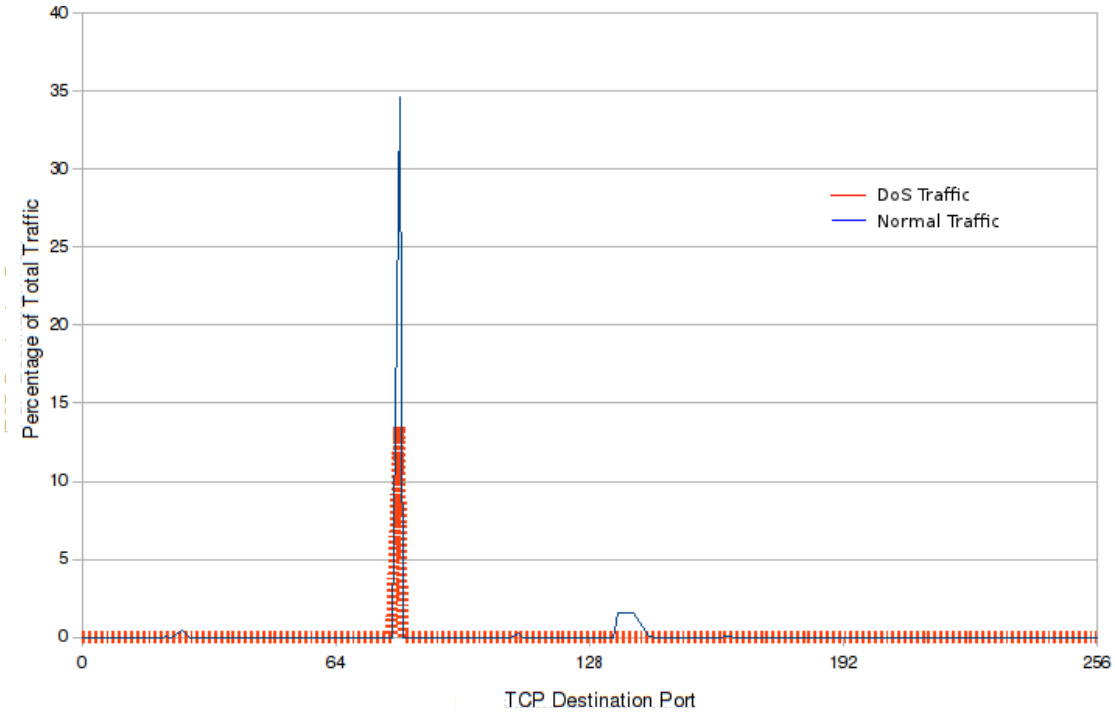


Figure 8.7: TCP Port Distribution Under DoS Attack Conditions

8.1.5 IP Address Counts and IP Identification

The majority of IP traffic is bidirectional. That is to say that a client would both send, and receive data. TCP requires that this be the case to allow data to be acknowledged. The network monitored in this research primarily contains TCP traffic (as can be seen in section 6.2. That nature means that we should expect to see a similar number of source IP addresses as destination addresses. This is because a packet travelling from a client to a server will have the client source address, and the server destination address. In the reverse path, those addresses would also be reversed (shown in figure 8.8), leading to 2 individual source and destination addresses being observed on the monitor.

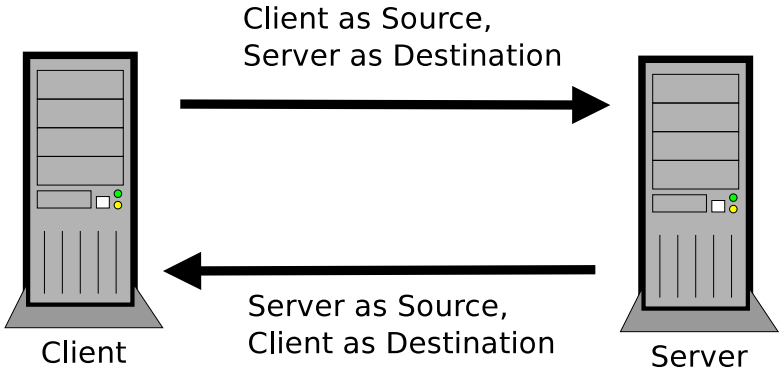


Figure 8.8: Bidirection Traffic

If this ratio between source and destination is not one, it means that the monitor is not seeing traffic in one direction, or that it is not being replied to. As can been seen in figure 8.9, the ratio between sources and destinations moves from being centred around 1:1, to a more spread distribution, centred around 1.5:1.

Spoofed DoS attacks move this ratio significantly away from 1:1, as does large scale network scanning. Spoofing source addresses will lead to a ratio between sources and destinations significantly higher than 1:1, and scanning a network is likely to lead to a ratio significantly smaller than 1:1.

IP identification numbers are used to allow fragmentation of IP packets if necessary on the path. It is valid, though not recommended to set this number to 0. In many of the attacks observed on the ISP network, an identification number of 0 was selected for the attack traffic, wildly moving the standard distribution of this parameter. In some cases, the IP ID for an attack was fixed to a non-zero number.

As can be seen in figure 8.10, the IP ID changes from the normal state where the most commonly seen ID is 0, to 256 under DoS conditions for this attack. This is because all of the packets which are part of the DoS attack have the IP ID set

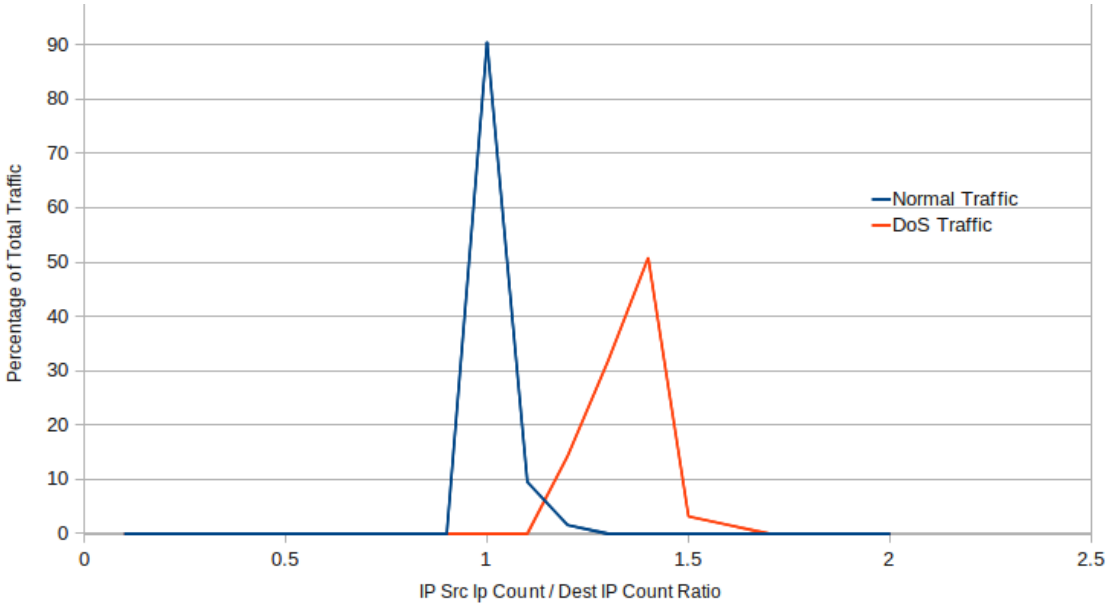


Figure 8.9: IP Source:Destination Ratio Under DoS Attack Conditions

to 256.

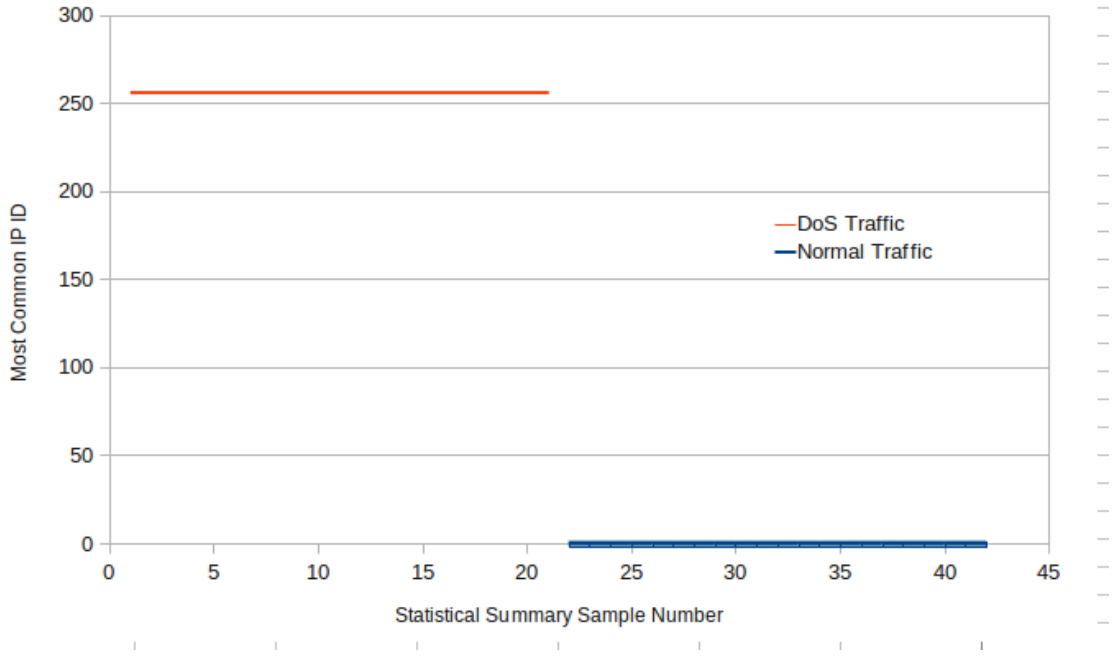


Figure 8.10: Most Common IP Identification Number Under DoS Attack Conditions

8.2 Summary

In this chapter we have examined variables which the neural network had selected as good indicators of the presence of DoS. For each example, an explanation for the type of change observed during DoS has been presented. This goes some way to demonstrating that not only are neural classifiers a potential tool for the identification of metrics, but also that the metrics selected here are robust. It has been demonstrated that models developed in a laboratory environment apply successfully in a live network. New metrics, such as TTL distribution, have been shown to be useful in the detection of malicious events which were not observed in the laboratory[7].

The next chapter will summarise the research presented throughout this thesis, draw conclusions from the data and present possibilities for future work in this area.

[7]Due to limitations in the emulation.

Chapter 9

Conclusion

In Chapter 3 some of the approaches used to address the difficulties involved with detecting malicious anomalies on large scale networks were introduced. These issues are largely based around processing the huge quantity of data in an efficient manner, dealing with the changing nature of network threats and providing a mechanism to protect the network against such attacks.

Chapter 4 described the network context for this research, the system that was implemented and the rational behind many of the design decisions made. It was demonstrated that simple PC based architectures could potentially deal with high speed network monitoring through the trading of memory for simplicity of processing. A signature detector based on the TCP checksum was devised and demonstrated to be effective in a controlled environment, this work was not taken extensively into the live network because of performance concerns and the lack of independent verification of its classifications. The signature detection mechanism did however provide a method for the detection of payloads in an environment where the payload was not available for examination.

Chapter 5 demonstrated that data mining techniques could be successfully applied to a laboratory emulation to detect the presence of DoS in traffic samples when the DoS traffic represented as little as 1% of the total bandwidth of the data. The concept of calculating relative importance of input fields through the comparative importance of their weightings within an ANN was introduced and applied to laboratory data.

Chapter 6 presented a summary the operational data recorded by the system. It was shown that the network monitored had strong time of day variation in metrics such as data rate and port usage. This cements the concept that the network behaviour should be predictable, and therefore possible to model.

Chapter 7 showed that data mining techniques could be applied to network data to successfully distinguish malicious network traffic from non-malicious data. Further more, it was demonstrated that these techniques could identify which

characteristics were the most significant in terms of divergence between the groups of data. Of particular interest was the selection of fields such as the IP TTL field distribution, IP Identification distribution and IP address counts as a strong indicators of the presence of DoS.

Chapter 8 examined the parameters selected through the ANN weighting analysis and evaluated them to demonstrate their suitability as detection metrics. It was shown that the data mining approaches selected extremely effective metrics, and further, metrics which may have been non-obvious to a casual observer. This is a particular strength of the research, as not only was a system developed for the detection of illegitimate traffic, a process by which the system could be trained and adapted to new threats has been proven. The IP TTL field in particular may prove a difficult metric to mask for a potential attacker due to the requirement for a large amount of information specific to the relative locations of the intended victim and monitoring station.

The internet forms an integral part of much of modern life. As people are becoming increasingly dependent on the Internet, the underlying infrastructure needs to becoming increasingly resilient to attack. DoS has consistently been a problem over an extended period of time, showing that the mitigation of this type of attack is a non-trivial problem.

Analysis network traffic at core network data rates is a significant challenge. This research has shown that meta data in the form of statistical summaries is a powerful tool for the detection of network attacks such as DoS and direct Worm propagation. These events, due to the level of traffic required for them to be effective, have an impact on the statistical landscape of a network. Leveraging this has been shown to be an efficient way of detecting their presence. The system developed detected the presence of DoS constituting as little as 0.1% of the total traffic volume in traffic emulations.

Feature selection for network security has traditionally relied on expert domain knowledge. This work has implemented data mining techniques for the classification of malicious network events. Further more, it has shown that these techniques can be used to identify the key features in the summarised data. Features such as TTL distribution and the ratio of source and destination IP addresses have been shown to be effective indicators of the presence of DoS.

Validating laboratory work is often a difficult endeavour; this research took data mining from a laboratory environment into a multi-gigabit national network, where it detected many network security events. Areas of similarity and areas of divergence between the controlled emulation and the live network were shown. Several ‘real-world’ attacks were analysed, and their impact on the summaries shown to be consistent with the neural classifier’s relative importance.

The system developed as part of this research operated in a live national consumer network for an extended period of time, monitoring data rates of up to 5Gbit/s across 6 sites, with hundreds of millions of connections being processed daily on standard PC architectures. The system adapted to the increasing traffic volume observed through the increasing of summary periods, demonstrating the power of summary statistics in providing scalable detection. The system developed was also flexible enough to be applied to high-speed application detection of VoIP traffic and to the detection of unsolicited SMTP generators.

9.1 Further Work

The ultimate aim of this type of research was to mitigate against the impact of malicious activity on the network. It is the author's view that filtering traffic local to the target is only effective against certain types of attack. To fully protect a target the filtering of traffic should be done as close to the source as possible. With this in mind, when a system like the one described here detected DoS on the network, research into automatically identifying parameters of the traffic in question would allow a more advanced use of push-back[64] or equivalent techniques. This is advantageous as, while push-back can control the traffic upstream, it is still necessary to drop a proportion of legitimate traffic.

If it were possible to provide a DoS filter with a more specific description of the features of a specific DoS, it would allow more a targeted throttling of the upstream data. For instance, metrics such as UDP port, or IP address are likely to be subject to a large number of false positives, leading to a degraded performance for legitimate users. If that specification were to be extended to be throttling UDP packets of size 1500 bytes heading for a specific IP address with a TTL of 58, the number of false positives and therefore the amount of performance degradation for legitimate users would be reduced.

This avenue of research was not explored as part of this thesis; the research was carried out on a live commercial national network, and therefore any experimental mitigation techniques would have had to be laboratory based. It should be possible to demonstrate this type of filtering in a test-bed environment.

References

[1] BBC report on Internet shopping. 2006. At time of writing, this article can be found at `http://news.bbc.co.uk/1/hi/business/4630472.stm`.

[2] LulzSec hackers claim CIA website shutdown. 2011. At time of writing, this article can be found at `http://www.bbc.co.uk/news/technology-13787229`.

[3] P.J. Sandford, J.M. Sandford, and D.J. Parish. Analysis of SMTP connection characteristics for detecting spam relays. 2006.

[4] O. Arkin. Tracing hackers: Part 1. *Computer Fraud & Security*, 2002(4):12–17, 2002.

[5] International Organization for Standardization. Information Technology Security Techniques. Code of practice for information security management. iso/iec 17799:2005(e).

[6] R. Bragg, M. Rhodes-Ousley, and K. Strassberg. *Network security: the complete reference*. Osborne, 2004.

[7] How does Sony breach affect customers? . 2011. At time of writing, this article can be found at `http://www.bbc.co.uk/news/technology-13206687`.

[8] ISO/IEC 10040 Information technology – Open Systems Interconnection – Systems management overview. 1998.

[9] TCPDump. More information at `http://www.tcpdump.org/`.

[10] Ethereal Protocol analyser. More information at `http://www.ethereal.com/`.

[11] Netflow. More information at `http://www.cisco.com/warp/public/732/Tech/nmp/netflow/index.shtml`.

[12] Wikipedia Ping article. At time of writing, this article can be found at `http://en.wikipedia.org/wiki/Ping`.

[13] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer. Operational experiences with high-volume network intrusion detection. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 2–11. ACM, 2004.

[14] R.G. Clegg, M.S. Withall, A.W. Moore, I.W. Phillips, D.J. Parish, M. Rio, R. Landa, H. Haddadi, K. Kyriakopoulos, J. Augé, et al. Challenges in the capture and dissemination of measurements from high-speed networks. *Communications, IET*, 3(6):957–966, 2009.

[15] Napatech network capture. More information at `http://www.napatech.com/`.

[16] Endace network capture. More information at `http://www.endace.com/`.

[17] M. Kaeo. *Designing network security*. Cisco Systems, 2003.

[18] Wikipedia load balancing entry. Note: at time of writing, this article is available at: `http://en.wikipedia.org/wiki/Load_balancing_%28computing%29`.

[19] S. Axelsson. Intrusion detection systems: A survey and taxonomy. 2000.

[20] Snort. More information at `http://www.snort.org/`.

[21] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Security and Privacy, 2005 IEEE Symposium on*, pages 226–241. IEEE, 2005.

[22] J. Mirkovic and P. Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[23] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. A taxonomy of computer worms. In *Proceedings of the 2003 ACM workshop on Rapid Malcode*, pages 11–18. ACM, 2003.

[24] J. Miller, B Kostanecki J. Gough, J Talkbot, and K Roculan. Microsoft dcom rpc alert. 2003.

[25] A. Van Der Merwe, M. Loock, and M. Dabrowski. Characteristics and responsibilities involved in a phishing attack. In *Proceedings of the 4th international symposium on Information and communication technologies*, pages 249–254. Trinity College Dublin, 2005.

[26] A. Hussain, J. Heidemann, and C. Papadopoulos. A framework for classifying denial of service attacks. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 99–110. ACM, 2003.

[27] Sophos Threat Management Report June 2006. At time of writing, this article can be found at `http://www.sophos.com/sophos/docs/eng/papers/sophos-security-report-jun06-srus.pdf`.

[28] Breaking the butterfly botnet. Note: at time of writing, this article is available at: `http://www.bbc.co.uk/news/10240117`.

[29] Microsoft's foiling of botnet gets mixed response. Note: at time of writing, this article is available at: `http://news.bbc.co.uk/1/hi/technology/8537771.stm`.

[30] Blackmailers target $1m website. 2006. At time of writing, this article can be found at `http://news.bbc.co.uk/1/hi/technology/4621158.stm`.

[31] Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. At time of writing, this article can be found at `http://delivery.acm.org/10.1145/rfc_fulltext/RFC2267/rfc2267.txt?key1=RFC2267&key2=1658621611&coll=GUIDE&dl=GUIDE&CFID=3822917&CFTOKEN=94822887`.

[32] Blaster (Worm). 2003. At time of writing, this article can be found at `http://en.wikipedia.org/wiki/Blaster_(computer_worm)`.

[33] L. Miller and P.H. Gregory. *CISSP for Dummies*. For Dummies, 2009.

[34] D. Moore, C. Shannon, D.J. Brown, G.M. Voelker, and S. Savage. Inferring internet denial-of-service activity. *ACM Transactions on Computer Systems (TOCS)*, 24(2):115–139, 2006.

[35] V. Paxson. An analysis of using reflectors for distributed denial-of-service attacks. *ACM SIGCOMM Computer Communication Review*, 31(3):38–47, 2001.

[36] K. Bicakci and B. Tavli. Denial-of-service attacks and countermeasures in ieee 802.11 wireless networks. *Computer Standards & Interfaces*, 31(5):931–941, 2009.

[37] A.D. Wood and J.A. Stankovic. Denial of service in sensor networks. *Computer*, 35(10):54–62, 2002.

[38] D.R. Raymond and S.F. Midkiff. Denial-of-service in wireless sensor networks: Attacks and defenses. *Pervasive Computing, IEEE*, 7(1):74–81, 2008.

[39] D. Sisalem, J. Kuthan, and S. Ehlert. Denial of service attacks targeting a sip voip infrastructure: attack scenarios and prevention mechanisms. *Network, IEEE*, 20(5):26–31, 2006.

[40] B. Zhao, C. Chi, W. Gao, S. Zhu, and G. Cao. A chain reaction dos attack on 3g networks: analysis and defenses. In *INFOCOM 2009, IEEE*, pages 2455–2463. IEEE.

[41] S. Dietrich, N. Long, and D. Dittrich. Analyzing distributed denial of service tools: The shaft case. In *Proceedings of the 14th USENIX conference on System administration*, pages 329–340. USENIX Association, 2000.

[42] N. Smyth. *Security+ Essentials*. eBookFrenzy.

[43] D. Dittrich. The "stacheldraht" distributed denial of service attack tool. 1999. Note: at time of writing, this article is available at: `http://staff.washington.edu/dittrich/misc/stacheldraht.analysis`.

[44] C. Kruegel, F. Valeur, G. Vigna, and R. Kemmerer. Stateful intrusion detection for high-speed network's. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 285–293. IEEE, 2002.

[45] H. Song and J.W. Lockwood. Efficient packet classification for network intrusion detection using fpga. In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pages 238–245. ACM, 2005.

[46] F. Dressler and G. Carle. History-high speed network monitoring and analysis. In *Proceedings of 24th IEEE Conference on Computer Communications (IEEE INFOCOM 2005), Miami, FL, USA*. Citeseer, 2005.

[47] Pf_ring packet capture library. More information at `http://www.ntop.org/`.

[48] C. Kreibich and J. Crowcroft. Honeycomb: creating intrusion detection signatures using honeypots. *ACM SIGCOMM Computer Communication Review*, 34(1):51–56, 2004.

[49] K. Wang and S.J. Stolfo. Anomalous payload-based network intrusion detection. In *Recent Advances in Intrusion Detection*, pages 203–222. Springer, 2004.

[50] S. Chebrolu, A. Abraham, and J.P. Thomas. Feature deduction and ensemble design of intrusion detection systems. *Computers & Security*, 24(4):295–307, 2005.

[51] T. Leckie and A. Yasinsac. Metadata for anomaly-based security protocol attack deduction. *Knowledge and Data Engineering, IEEE Transactions on*, 16(9):1157–1168, 2004.

[52] S.S. Kim, A.L.N. Reddy, and M. Vannucci. Detecting traffic anomalies through aggregate analysis of packet header data. *NETWORKING 2004, Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications*, pages 1047–1059, 2004.

[53] A. Lakhina, M. Crovella, and C. Diot. Characterization of network-wide anomalies in traffic flows. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 201–206. ACM, 2004.

[54] W. Zhenqi and W. Xinyu. Netflow based intrusion detection system. In *2008 International Conference on MultiMedia and Information Technology*, pages 825–828. IEEE, 2008.

[55] S. Zanero and S.M. Savaresi. Unsupervised learning techniques for an intrusion detection system. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 412–419. ACM, 2004.

[56] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[57] K. Julisch and M. Dacier. Mining intrusion detection alarms for actionable knowledge. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 366–375. ACM, 2002.

[58] A. Kulkarni and S. Bush. Detecting distributed denial-of-service attacks using kolmogorov complexity metrics. *Journal of Network and Systems Management*, 14(1):69–80, 2006.

[59] G. Carl, G. Kesidis, R.R. Brooks, and S. Rai. Denial-of-service attack-detection techniques. *Internet Computing, IEEE*, 10(1):82–89, 2006.

[60] J. Xu and W. Lee. Sustaining availability of web services under distributed denial of service attacks. *IEEE Transactions on Computers*, pages 195–208, 2003.

[61] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for ip traceback. *ACM SIGCOMM Computer Communication Review*, 30(4):295–306, 2000.

[62] Jelena Mirkovic. D-WARD: Source-End Defense Against Dis-tributed Denial-of-Service Attacks (Ph.D. thesis, University ofCalifornia Los Angeles). 2003.

[63] X. Liu, X. Yang, and Y. Xia. Netfence: preventing internet denial of service from inside out. *ACM SIGCOMM Computer Communication Review*, 40(4):255–266, 2010.

[64] R. Mahajan, S.M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *ACM SIGCOMM Computer Communication Review*, 32(3):62–73, 2002.

[65] A.D. Keromytis, V. Misra, and D. Rubenstein. Sos: Secure overlay services. *ACM SIGCOMM Computer Communication Review*, 32(4):61–72, 2002.

[66] A.D. Keromytis, V. Misra, and D. Rubenstein. Sos: An architecture for mitigating ddos attacks. *Selected Areas in Communications, IEEE Journal on*, 22(1):176–188, 2004.

[67] D. Mankins, R. Krishnan, C. Boyd, J. Zao, and M. Frentz. Mitigating distributed denial of service attacks with dynamic resource pricing. In *Computer Security Applications Conference, 2001. ACSAC 2001. Proceedings 17th Annual*, pages 411–421. IEEE, 2001.

[68] J.R. Reidenberg. Resolving conflicting international data privacy rules in cyberspace. *Stan. L. Rev.*, 52:1315, 1999.

[69] Clementine data mining. Note: at time of writing, this article is available at: `http://www.spss.com/software/modeling/modeler/`.

[70] B. Kaiser and K. Knight. *Concrete abstractions: an introduction to computer science using Scheme*. Course Technology Ptr, 1999.

[71] D. Moore, C. Shannon, et al. Code-red: a case study on the spread and victims of an internet worm. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 273–284. ACM, 2002.

[72] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. *Security & Privacy, IEEE*, 1(4):33–39, 2003.

[73] J. Stone, M. Greenwald, C. Partridge, and J. Hughes. Performance of checksums and crcs over real data. *Networking, IEEE/ACM Transactions on*, 6(5):529–543, 1998.

[74] B.S. Everitt, S. Landau, M. Leese, and D.D. Stahl. *Cluster analysis 5e (series: wiley series in probability and statistics)(hardback)*. 2010.

[75] K.O. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. *Network (Phenotype)*, 1(2):3.

[76] D.J. Hand, H. Mannila, and P. Smyth. *Principles of data mining*. The MIT Press, 2001.

[77] S. Staniford, V. Paxson, and N. Weaver. How to own the internet in your spare time. In *Proceedings of the 11th USENIX Security symposium*, volume 8, pages 149–167, 2002.

[78] J. Jung and E. Sit. An empirical study of spam traffic and the use of dns black lists. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 370–375. ACM, 2004.

[79] M. Sandford, D. Parish, and I. Phillips. Neural approach to detecting communication network events. In *Communications, IEE Proceedings-*, volume 149, pages 257–264. IET, 2002.

[80] D.J. Parish, K. Bharadia, A. Larkum, I.W. Phillips, and M.A. Oliver. Using packet size distributions to identify real-time networked applications. In *Communications, IEE Proceedings-*, volume 150, pages 221–227. IET, 2003.

Appendix A

SMTP Investigations

Analysis of SMTP Connection Characteristics for Detecting Spam Relays

PJ Sandford, JM Sandford, DJ Parish
Electronic and Electrical Engineering Department,
Loughborough University
UK
{p.j.sandford, j.m.sandford, d.j.parish}@lboro.ac.uk

Abstract

Research into preventing spam is an ongoing concern. Much of the effort to date has focused on filtering email at the receiving end using techniques such as content filters. Techniques that could prevent spam from initially being sent would be beneficial and contribute to the preventative effort. In practice much spam is sent from compromised hosts. These can potentially be detected by ISPs using simple monitoring techniques. This monitoring takes place in the core of an ISP's network and provides a periodic summary of email activity. Relatively simple post-capture analysis can then identify the most significant spam relay machines.

Keywords

Spam, Spam Relays, Network Monitoring, Data Summaries

Introduction

In January 2004 Bill Gates claimed that, "Spam will soon be a thing of the past" [1]. Two years later and unsolicited email continues to proliferate, clogging up mailboxes and consuming bandwidth. The evils of spam range simply from unwanted advertising to propagating pornography or fraud attempts known as *phishing*. The extent of the problem has led some commentators to describe email as becoming 'almost unusable' [3].

Various techniques have been developed and deployed to try and prevent spam from being delivered. These techniques include the use of content filters, blacklists, whitelists and collaborative filters. Predominantly these approaches focus on spam mitigation at the receiving end. Pressure is growing however for ISPs to mitigate against spam at the sending end. This is difficult because senders can continually re-register, establishing multiple identities which make it difficult for ISPs to prevent users from sending spam. A further problem exists due to the use of spam relays. In addition to open relays, spam relays may be established on compromised hosts enabling spammers to continually change the IP address of the spam source.

This paper proposes a traffic monitoring based approach to identify hosts on an ISP network that are being used as spam relays. Importantly, this approach does not involve the capture or analysis of the email content itself, only of the packet header information. The technique involves recording the number of SMTP connections established by each host on the monitored network segment, and the number of mail servers each host connects to. Although at an early stage of the work, a prototype tool has been used to provide information to network operators, informing them of the hosts that are operating as spam relays. The approach could however, be incorporated in to an autonomous system that would prevent spam relays sending unwanted email from the monitored network.

The paper is organized as follows. First an overview of current spam prevention techniques is given. Our spam relay detection technique is then presented together with details of its experimental implementation within a national ISP's network. Some results of its use on this network are then presented.

Preventing Spam

Various strategies are employed to try and filter spam. Predominantly, these focus on identifying spam at the receiving end. An overview of these strategies is presented here. A more in depth treatment is available in [6].

One simple technique that may be used to combat spam functions by filtering email through the use of

Proceedings of the International Multi-Conference on Computing in the Global Information Technology (ICCGI'06)
0-7695-2629-2/06 $20.00 © 2006 IEEE

blacklists [11]. These are maintained either on a commercial or voluntary basis and contain the IP addresses of hosts that are known to have sent spam. Mail clients can use DNS style queries to see if the source IP address of incoming mail appears on the list. Whilst this technique can help to filter spam it is both limited and crude. Lists tend to go out of date very quickly due to constant changes in the source IP addresses of spam. A further limitation is that many blacklists do not distinguish between hosts in the same domain resulting in many legitimate emails being filtered. The use of blacklists has contributed to pressure on ISPs to prevent spam being sent from their IP address space, as in one scenario 900,000 addresses from a single ISP network were listed [2].

A related concept to blacklists is that of a whitelist. As the name implies this works in exactly the opposite way to a blacklist. Instead of listing hosts that are known to send spam, a whitelist contains hosts that are trusted. The obvious disadvantage of this approach is that it presents greater complexity for the many legitimate cases where an email is sent to a new contact. A solution to this requires the sender to confirm their address. This can lead to a number of problems however as the scheme requires the automatic generation of a reply to email from source addresses not on the users whitelist (for a fuller explanation of why this is bad practice see [6]).

The use of heuristic methods for identifying spam is a mature research area. Various analysis techniques have been proposed to evaluate the content of an email (e.g. [5][8]). An advantage of this type of content filtering is the potential for transparent deployment. Unfortunately, while impressive accuracy rates have been reported, too many false positives prevent email from being discarded automatically after having been flagged as spam. Furthermore, such a large quantity of spam is propagated that even accuracy rates of 99.9% still miss a significant amount of spam. SpamAssassin [10] is a widely used open source tool that employs heuristic techniques to identify spam. Like other solutions, while SpamAssassin has been shown to effectively filter a high percentage of unsolicited email, it is still far from perfect. Studies have shown that not only does it miss a small percentage of spam; it may also incorrectly filter legitimate email [7].

A further technique that may be used in conjunction with other systems is termed collaborative filtering. This approach exploits the fact that multiple spam emails are generally sent unchanged (or with minor changes) to a large number of email addresses. Through the use of 'spam traps' (unused email addresses, deliberately placed on the internet to be harvested by spam software), the content of spam emails is recorded and then shared with participating spam filters. A commercial example of this type of approach is BrightMail [4].

The general consensus is that a single technical solution that is able to prevent the propagation of spam is unlikely to be found given the constraints of the current Internet architecture [6]. A composite approach, incorporating many of the techniques described above, can and does help alleviate the problem and reduce the amount of spam. Techniques to date tend to focus on filtering spam at the receiving end, although some of the techniques may be deployed by mail servers to prevent spam from being sent. A specific problem however, exists due to the presence of spam relays installed on compromised hosts. These allow the propagation of large quantities of spam from constantly changing sources. The ability to reliably identify machines that are acting as spam relays, often as the result of being infected by a worm or virus, is another approach with which to reduce the amount of spam distributed over the network. This paper presents such an approach which can be used by a network operator such as an ISP.

Detecting Spam Relays

In the approach to be described spam relay detection is conducted alongside other security tasks and makes use of a shared monitoring system. This system collects various network traffic statistics by passively monitoring core network links. The system consists of six gatherers (sometimes referred to as monitors or sniffers) and a central processor located within the ISP's network. A further analysis/development host is also located at a remote site, in this case Loughborough University. The gatherers are based on Dell Dual 3GHz Intel Xeon servers with Intel Pro 1000 network interface cards to allow monitoring at data rates of in excess of 1 Gigabit/second. The gatherers are located at six distributed PoP (Points of Presence) sites and monitor all network traffic either from or to the local broadband network. The monitoring software gathers light weight summary statistics that are used for further security purposes [9].

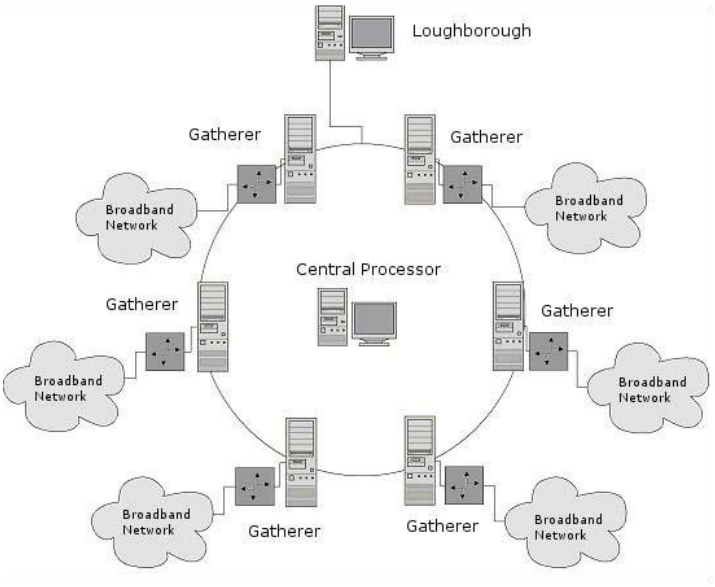


Figure 1: The Monitoring Architecture

To provide information regarding the location of spam relays, the gatherers log TCP/IP packet header summaries which have the SYN flag set and a destination port value of 25 (this indicates SMTP). These packets are closely correlated to TCP connections with mail servers. Home users on a commercial ISP network may quite legitimately contact a number of mail servers on which they have accounts, although many home users access email through an HTTP interface. The number of connections to mail servers expected through legitimate usage is however relatively small in comparison to the number of connections established by spam relays. Whereas a home user might connect to a mail server a few times an hour, spam relays send thousands of emails every hour to hundreds of mail servers. This large disparity between legitimate home users and machines that have been compromised and are being used as spam relays suggests simple traffic analysis may be a means of identifying those machines that are sending spam. The ISP can then take action to either block SMTP traffic from these machines, or, more realistically, ensure that the machines are 'cleaned up' and reconfigured with appropriate levels of security.

The marked difference in profile between compromised hosts acting as spam relays and legitimate users contacting mail servers is aided by the absence of legitimate mail servers hosted on the monitored network. Where customer service agreements allow mail servers to be present, the scenario does become more complex. In this instance it is still possible to distinguish between legitimate mail servers and spam relays by examining the distribution of port 25 connections rather than simply the quantity. For instance, legitimate mail servers would tend to exhibit a daily pattern that reflects the habits of the users of that mail server. Spam relays by contrast, normally do not exhibit this profile. Instead they may send at a constant rate 24 hours a day, or may send many thousands of emails over a short period of time and then nothing at all. Examples of these profiles are given in the following results section.

Of further note is that open relays are not permitted on the monitored ISP's network. In the case where open relays were used legitimately, distinguishing them from relays installed on compromised hosts may prove difficult. In this instance it may be necessary to maintain a list of the legitimate relays to feed in to the detection process. It should be noted however, that legitimate open relays are still used to send spam.

Results

Figures 2 – 7 provide examples of hosts sending unusual numbers of SMTP packets. In the current configuration logs show the number of SMTP packets seen in a 24 hour period, plotted in an hourly bar chart. This allows network operators to quickly assimilate which hosts have been acting as spam relays. The target destinations for each spam relay may be displayed by double clicking on the source

address of interest (e.g. Figure 4). These appear in order of frequency. Common targets typically include popular mail providers such as hotmail and yahoo.

All the examples shown come from a single 24 hour period during September 2005. Note that for reasons of confidentiality, the actual IP addresses have been obscured. Figure 2 shows a host which has established in excess of 58,000 SMTP connections within the 24 hour period. As can be seen from the plot, these have been established at a constant rate of around 2500 thousand connections an hour. The particular ISP network being monitored provides broadband and dialup services to home users. As this service does not provide a static IP address, hosting mail servers is impractical. On networks where mail servers are hosted it is clear that both the quantity and the profile of SMTP connections shown below would still distinguish this host as a spam relay.

A mail server would show a variation in the number of connections established per hour, consistent with a daily usage pattern whereas this host has sent at a consistent rate throughout the 24 hour monitoring period. A further demarcating factor would be the host name, which in this case, indicates that this host is a home user.

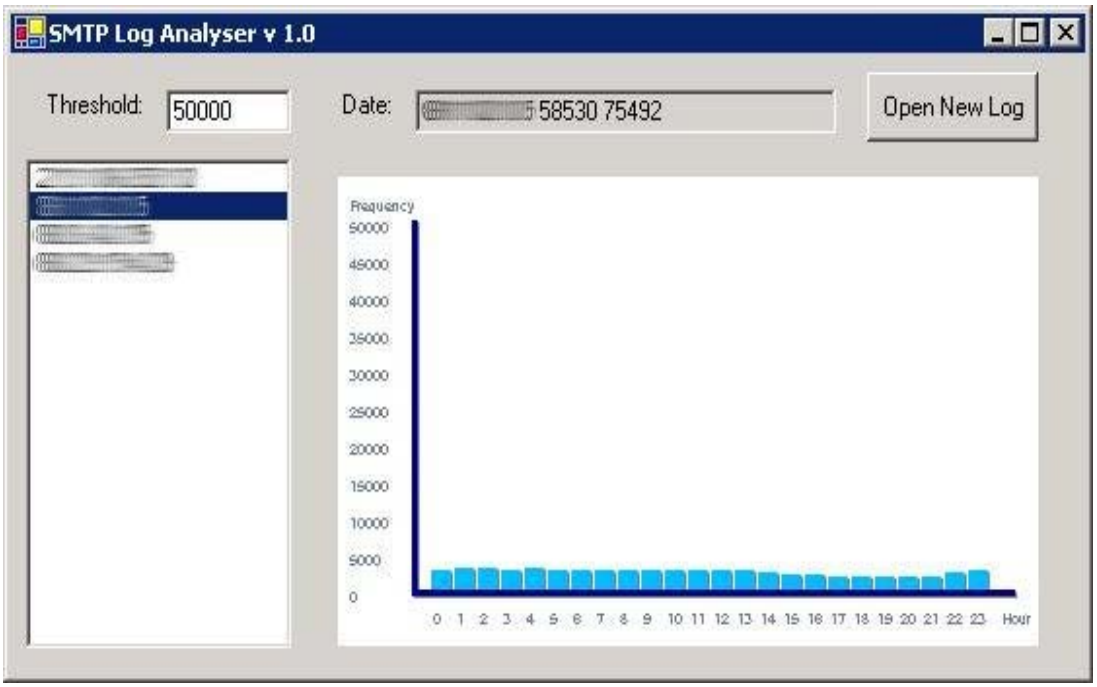


Figure 2

In contrast to Figure 2, Figure 3 shows a host that has established close to 25,000 SMTP connections in a single hour but none at any other time during the day. Although this host exhibits quite different characteristics to the one previously shown, it is again immediately obvious that this profile does not fit that of a legitimate mail server. Compromised hosts may, in addition to being used as a spam relay, be used to send 'mail bombs' or to scan for vulnerable hosts. 'Mail bombs' occur where very large quantities of email are sent to the same address rendering the address unusable. In this instance however, the plot is of a host being used to scan multiple addresses.

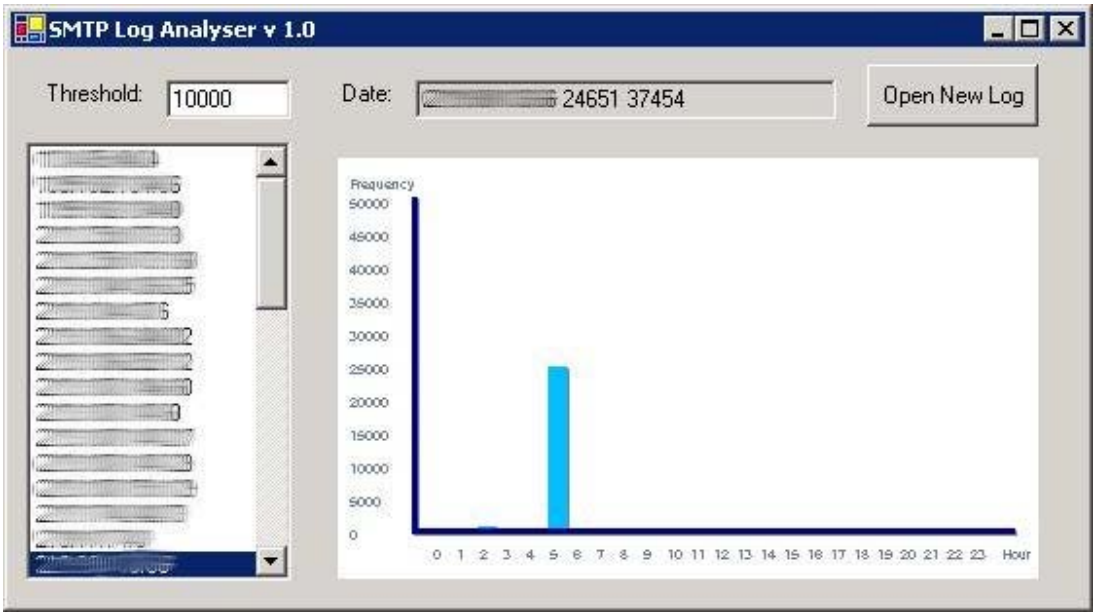


Figure 3

Figure 4 shows a section of a table which relates the destination addresses to the source address for the traffic displayed in Figure 3. As in the other Figures, the IP addresses are obfuscated but in this case sufficient text is left to show the source has systematically scanned hosts, possibly across a number of other ports.

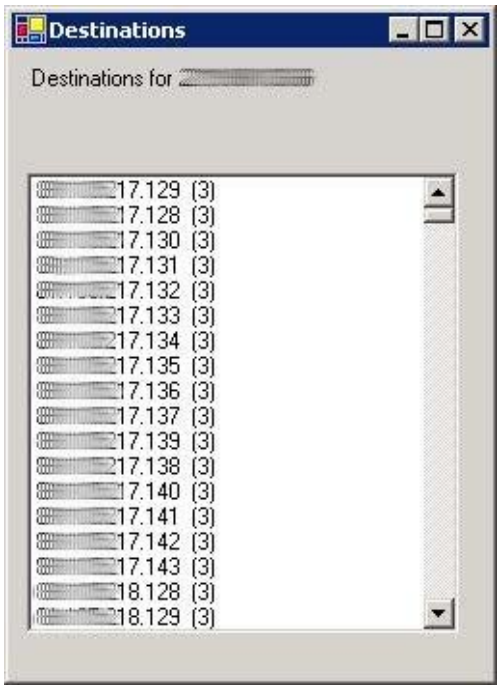


Figure 4

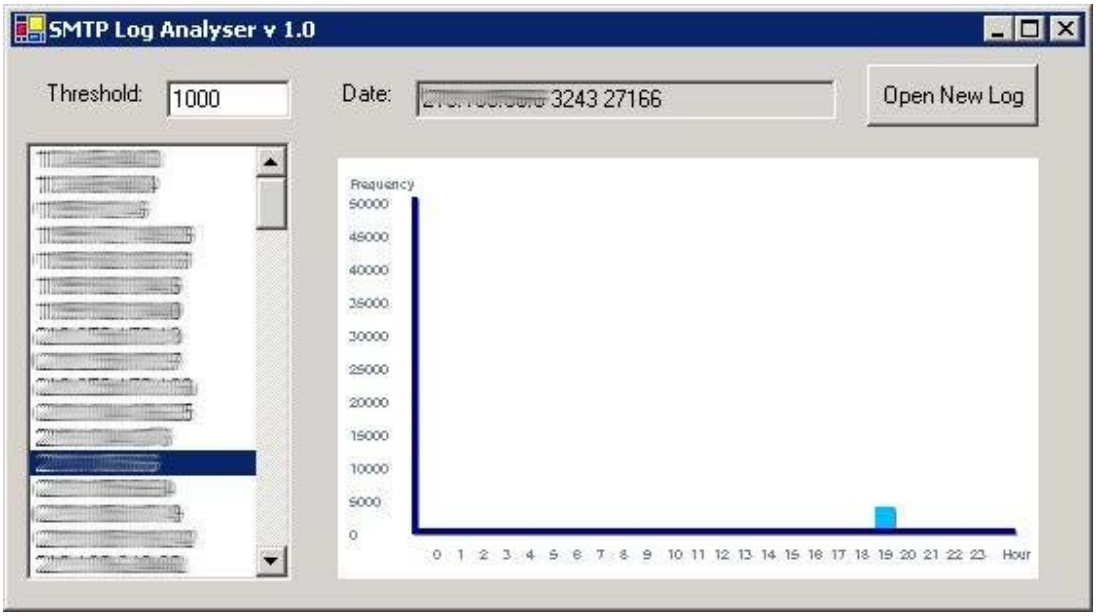


Figure 5

Figure 5 shows a host that has made over 3000 SMTP connections in a single hour but none at any other point in the 24 hour period. A profile of this type does not fit that expected of either a mail client or a mail server.

Whereas a mail client might be expected to make a few connections an hour to one or two mail servers, this host has established a new connection every second to over one thousand mail servers. The profile is also clearly different from that of a mail server. This host appears to have been compromised and used to send spam. A possible explanation for the host only sending email during one hour of the day is that the host may have been switched off at other times.

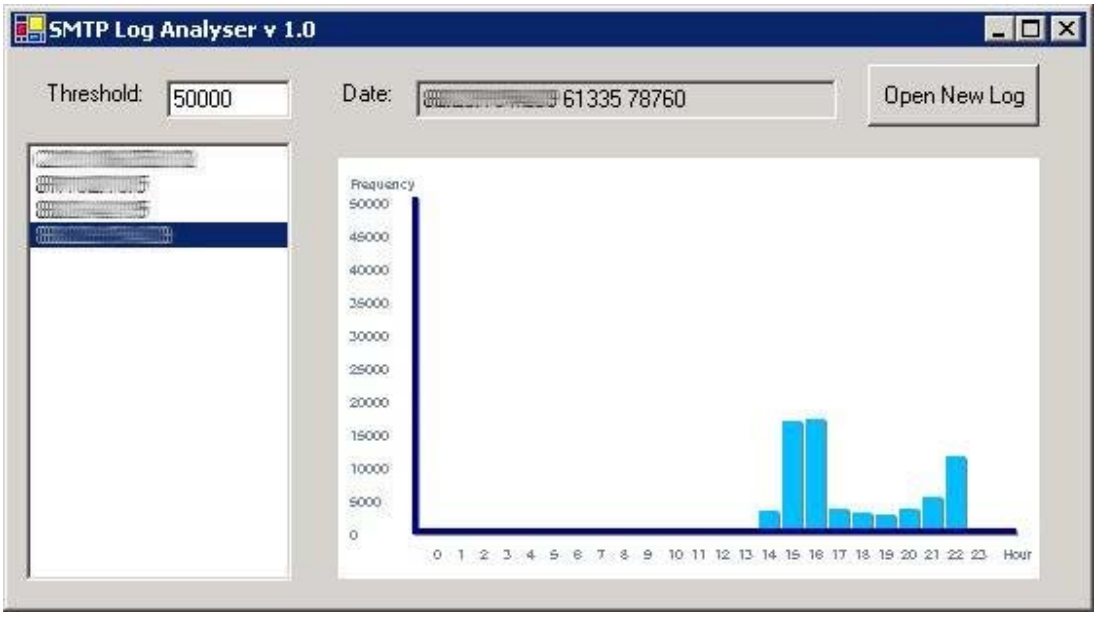


Figure 6

In figures 6 & 7 hosts are shown that have sent large quantities of email to multiple mail servers. The distribution of SMTP connections per hour is not constant, but is markedly different to that which would be expected of a mail server. In figure 7, high numbers of connections are established throughout the 24 hour monitoring period. In figure 6, the host appears to have been switched off for part of the day although such a profile could also be attributed to an IP address that is reassigned to a spam relay. In both figures 6 & 7 the quantity of connections established is very large. The sending host shown in figure 7 established over 160,000 connections during the 24 hour period, with almost 30,000 connections established in one hour alone.

In total 89,748 hosts were observed sending SMTP traffic on the monitored network segment. 46 hosts had established over 10,000 thousand SMTP connections per host during the day. 4 hosts had each established over 50,000 connections.

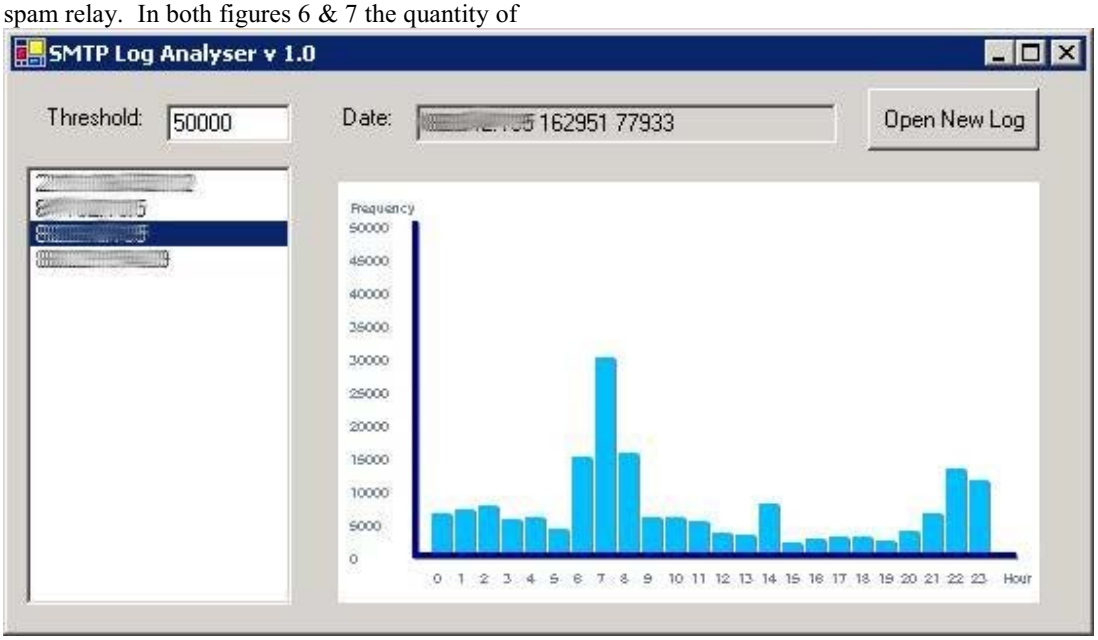


Figure 7

Conclusions

It seems unlikely, given the restraints of the current Internet architecture, that a single approach or technology will be found that is able to totally eliminate unsolicited email. Several technologies exist that contribute to the reduction of the quantity of spam that users receive. These tend to focus on filtering mechanisms as the email is received. The ability to prevent spam from initially being sent from compromised hosts used as spam relays would contribute to the effort to reduce the quantity of spam seen in the Internet. This paper has shown how spam relays installed on compromised hosts could be identified by the ISP networks on which they are hosted. At present, the identification process is achieved by visual inspection of the data. However, given the large disparity between the SMTP connection profiles of legitimate mail clients and servers and spam relays, an automated process could easily be developed to detect spam relays based on the quantity and profile of SMTP connections, and the destination addresses.

References

1. BBC News Article, Published 24th Jan 2004, available from http://news.bbc.co.uk/go/pr/fr/-/1/hi/business/3426367.stm
2. BBC News Article, Published 9th May 2005, available from http://news.bbc.co.uk/go/pr/fr/-/1/hi/technology/4528927.stm
3. S.M. Bellovin, "Inside risks: Spamming, phishing, authentication, and privacy", Communications of the ACM,

Volume 47, Number 12, Page 144, 2004
4. http://www.brightmail.com
5. H. Drucker, D. Wu, V.N. Vapnik, "Support Vector Machines for Spam Categorization", IEEE Transactions on Neural Networks, Vol. 10, No. 5, September 1999
6. S. Hird, "Technical Solutions for Controlling Spam", in Proceedings of AUUG2002, Melbourne, September, 2002
7. C. O'Brien, C. Vogel, "Comparing SpamAssassin with CBDF email filtering", In Proceedings of 7th CLUK Research Colloquium, 2004
8. M. Sahami, S. Dumais, D. Heckerman, E. Horvitz, "A Bayesian Approach to Filtering Junk E-Mail", in AAAI'98 Wkshp. Learning for Text Categorization, Madison, WI, July, 1998
9. P.J. Sandford, D.J. Parish, J.M. Sandford, "Identify Internet Abuse in ISP Networks: Practical, Technical and Legal Issues", Safety and Security in a Networked World: Balancing Cyber-Rights and Responsibilities, September 2005
10. http://spamassassin.apache.org/
11. A comprehensive list of DNS based blacklists is available from http://www.declude.com/JunkMail/Support/ip4r.htm

Appendix B

Gatherer C Code

```
1  /* Ethernet header */
2  struct sniff_ethernet
3  {
4          uint16_t packet_type;           /* Packet Type: 0 (to us), 1 (broadcast), 2 (multicast
               ), 3 (else to else), 4 sent by us */
5      uint16_t ARPHRD;               /* Linux value for the layer device type. */
6      uint16_t slink_length;
7      uint16_t bytes[4];
8          uint16_t ether_type;                        /* IP? ARP? RARP? etc */
9  };
10
11 /* IP header */
12 struct sniff_ip
13 {
14         #if BYTE_ORDER == LITTLE_ENDIAN
15                 uint8_t    ip_hl:4,         /* header length */
16                            ip_v:4;          /* version */
17         #if BYTE_ORDER == BIG_ENDIAN
18                 uint8_t    ip_v:4,          /* version */
19                            ip_hl:4;         /* header length */
20         #endif
21         #endif /* not _IP_VHL */
22         uint8_t  ip_tos;                    /* type of service */
23         uint16_t ip_len;                    /* total length */
24         uint16_t ip_id;                     /* identification */
25         uint16_t ip_off;                    /* fragment offset field */
26         #define IP_RF 0x8000                /* reserved fragment flag */
27         #define IP_DF 0x4000                /* dont fragment flag */
28         #define IP_MF 0x2000                /* more fragments flag */
29         #define IP_OFFMASK 0x1fff           /* mask for fragmenting bits */
30         uint8_t  ip_ttl;                    /* time to live */
31         uint8_t  ip_p;                      /* protocol */
32         uint16_t ip_sum;                    /* checksum */
33         struct  in_addr ip_src,ip_dst;  /* source and dest address */
34 };
35
36 /* TCP header */
37 struct sniff_tcp
38 {
39         uint16_t th_sport;                          /* source port */
40         uint16_t th_dport;                          /* destination port */
41     uint32_t th_seq;                           /* sequence number */
42     uint32_t th_ack;                           /* acknowledgement number */
43         #if BYTE_ORDER == LITTLE_ENDIAN
44                 uint8_t    th_x2:4,                 /* (unused) */
45                            th_off:4;                /* data offset */
46         #endif
47         #if BYTE_ORDER == BIG_ENDIAN
48                 uint8_t    th_off:4,                /* data offset */
49                            th_x2:4;                 /* (unused) */
50         #endif
51         uint8_t  th_flags;
52         #define TH_FIN  0x01
53         #define TH_SYN  0x02
54         #define TH_RST  0x04
55         #define TH_PUSH 0x08
56         #define TH_ACK  0x10
```

```
57          #define TH_URG  0x20
58          #define TH_ECE  0x40
59          #define TH_CWR  0x80
60          #define TH_FLAGS        (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
61          uint16_t th_win;                 /* window */
62          uint16_t th_sum;                 /* checksum */
63          uint16_t th_urp;                 /* urgent pointer */
64 };
65
66 /* UDP header */
67 struct sniff_udp
68 {
69     uint16_t uh_sport; /* Source Port */
70     uint16_t uh_dport; /* Destination Port */
71     uint16_t uh_len;   /* Udp length */
72     uint16_t uh_check; /* Checksum */
73 };
74
75
76 /* """""""""""""""""""""""""""""""""""""""""""""""""""""""""""
77  Name: nethackdetect
78  Version: 2.80 Beta
79  Author: Pete Sandford
80  Date: 26th of November 2003
81  Last Rev: 3 Feb 2006
82
83  Description: This program is designed to be run
84  on a single machine using linux. The
85  program has 2 processes.
86
87  One process works on the captured packets,
88  storing data in real time. The other process
89  works in a batch-style manner creating
90  statisctial summaries in a quasi-realtime
91  manner.
92
93  The first process is memory intensive, but
94  effort has been made to ensure the processing
95  per packet is kept to a minimum.
96
97
98  """""""""""""""""""""""""""""""""""""""""""""""""""""""""""" */
99 #define LIBNET_BIG_ENDIAN 1
100
101 #include <pcap.h>
102 #include <stdio.h>
103 #include <stdlib.h>
104 #include <errno.h>
105
106 #include <sys/socket.h>
107 #include <netinet/in.h>
108 #include <arpa/inet.h>
109 #include <signal.h>
110 #include <net/if.h>
111 #include <netinet/if_ether.h>
112 #include <netinet/tcp.h>
113 #include <math.h>
114 #include <time.h>
115 #include <sys/shm.h>
116 #include "nethack_types.h"
117 #include <sys/types.h>
118 #include <unistd.h>
119 #include <string.h>
120 #include <getopt.h>
121 #include <libnet.h>
122
123 #define _BSD_SOURCE 1
124 #define FILE_SIZE 10000
125 #define SMTP_FILE_SIZE 10000000
126 /* Smtp trigger level (0-1000) */
127 #define SMTP_TRIGGER 600
128 /* Interfaces */
129 #define INTERFACE1 "eth1"
130 #define INTERFACE2 "eth2"
131 /* Define the header sizes */
132 #define eth_size 14
133 #define ip_size 20
134 /* IP_RANGE is the maximum number of subnets (this is a loose definition, simply 2^24, xxx.xxx
        .xxx.fixed) */
135 #define IP_RANGE 16777216
```

```
136
137 /* Function definitions */
138 void process(u_char *useless,const struct pcap_pkthdr* pkthdr,const u_char* packet);
139 void userexit(int);
140 void report(void);
141 int alert(void);
142 float ks(float *,float *,int,int);
143
144 /* Alerting Module Limits */
145 float alerting_vars[30];
146
147 /* Switches to control which set of memory is being addressed by the write / read processes */
148 int memswitchread=0;
149 int memswitchwrite=0;
150
151 /*site number */
152 int site=1;
153
154 /* Shared memory id */
155 int shmid;
156
157 /* File outputs */
158 FILE* smtp;
159 FILE* alert_file;
160
161 /* Shared memory segment */
162 struct shmid_ds* shmid_ds;
163
164 /* The process ID of the child process */
165 int pid;
166
167 /* The Size of the sample of packets to take before reporting */
168 uint32_t packet_sample = 0;
169
170 /* The total number of packets captured by the program */
171 uint32_t packet_count = 0;
172
173 /* SMTP Counter */
174 uint32_t smtp_file_count=0;
175 uint32_t smtp_packet_count=0;
176
177 /* The log number to start with */
178 uint32_t file_count=0;
179 /* Structure to hold the input signatures */
180 struct sig
181 {
182   /* List of ports with signatures on them (Maximum of 10 on any 1 packet size, could be more
        than 1 instance on a port) */
183   int port_list[10];
184   /* List of the signatures for each of the port's fingerprints */
185   char sig_list[10][1600];
186   /* The number of signatures for this port */
187   short number;
188 };
189
190 struct share_mem
191 {
192   /* An array of flags, indicating whether an IP has been seen by the application or not. */
193   char ip_src_array[IP_RANGE];
194   char ip_dst_array[IP_RANGE];
195
196   /* An array, with an element for each packet size. */
197   uint16_t ip_packet_size[1538];
198   uint16_t udp_packet_size[1538];
199   uint16_t tcp_packet_size[1538];
200   uint16_t icmp_packet_size[1538];
201
202   /* An array with an element for each TCP port number */
203   uint16_t tcp_sport[65535];
204   uint16_t tcp_dport[65535];
205   uint16_t identification[65535];
206   uint16_t checksum[65535];
207
208   /* An array with an element for each TTL field */
209   long ttl[256];
210   long udp_ttl[256];
211   long tcp_ttl[256];
212   long icmp_ttl[256];
213
214   /* SMTP variables */
```

```
215   long smtp_count;
216     long smtp_total_bits;
217   long Non_NTL_smtp_count;
218     long Non_NTL_smtp_total_bits;
219
220     /* Skype variables */
221     long skype_count;
222     long skype_total_bits;
223
224   /* An array with an element for each UDP port number */
225   uint16_t udp_sport[65535];
226   uint16_t udp_dport[65535];
227
228   /* Counters for the various protocols */
229   long cTCP,cUDP,cTotalIP,cICMP,cUnknownIP;
230
231   /* Counters for the TCP flags */
232   long FIN, SYN, RST, PUSH, ACK,URG;
233
234   /* Average packet sizes for the various protocols */
235   long saTotal, saTCP,saUDP,saICMP;
236
237   /* Flags to indicate whether the shared memory has been written / read (0 for no, 1 for yes)
          */
238   short written;
239   short read;
240 };
241
242 /* A pointer to the shared memory */
243 struct share_mem *shareptr;
244
245 /* Nasty Fudge:- There is a bug somewhere in the program which causes a free() problem. This
       workaround allows the memory at the shareptr location to be freed. */
246 struct share_mem *shareptrfree;
247
248 /* A set of Signatures, stored for each packet size */
249 struct sig detect_matrix[1538];
250
251 /* Signature input and output files */
252 FILE* sig_output;
253 FILE* sig_input;
254
255 /* Mode variables */
256 int l=0;
257 int a=0;
258 int p=0;
259 int s=0;
260 int f=0;
261 int c=0;
262 int v=0;
263 int h=0;
264 int n=0;
265
266 int main(int argc,char **argv)
267 {
268     /* Variable Declarations */
269   char *dev;  /* Device to be listened on */
270     char errbuf[PCAP_ERRBUF_SIZE];
271     pcap_t* descr;
272   pcap_t* descr1;
273   pcap_t* descr2;
274   short packets_lost=0;
275   int i, rtrn;
276   FILE* alert_in;
277     uint32_t packet_total;
278     int opt;
279     char currentline[1000];
280     int pos;
281
282     /*options
283       a - spam
284       f - file
285       c - capture sample size
286       v - verbose
287       p - packets total to capture
288       l - Log start number
289       h - help
290       n - trigger
291    r - site number         */
292   while((opt = getopt(argc,argv,"adsf:c:vihp:l:nr:"))!=EOF){
```

```
293       switch(opt){
294         case 'a':
295             a=1;
296             break;
297         case 's':
298             s=1;
299             break;
300         case 'f':
301             f=1;
302                 sig_input = fopen(optarg,"r");
303             break;
304         case 'c':
305             c=1;
306                 packet_sample = atoi(optarg);
307             break;
308         case 'v':
309             v=1;
310             break;
311         case 'h':
312             printf("Options:\n-n Enable Alerting Module\n-f <file> Enable Checksum module with
                      specified file\n-a Enable Spam Module\n-c <sample> Set sample Size\n-v Enable
                      Verbose output\n-p <packets> Set the number of Packets to capture\n-a SMTP
                     logger \n-r Site number\n-h Help - You're looking at me baby!\n");
313                 exit(0);
314             break;
315         case 'p':
316             p=1;
317                 packet_total = atol(optarg);
318             break;
319             case 'l':
320                 l=1;
321                 file_count = atol(optarg);
322                 break;
323             case 'n':
324                 n=1;
325                 break;
326             case 'r':
327                 site = atoi(optarg);
328                 break;
329         }
330     }
331
332   /* Attempt to allocate the memory for the shared memory array (~80 MB) */
333   if((shareptr = malloc(2*sizeof(struct share_mem))) == NULL)
334   {
335     printf("Error Allocating Memory\n");
336     exit(1);
337   }
338
339   /* Copy the shared memory pointer to allow us to fudge it later */
340   shareptrfree = shareptr;
341
342   /* Create a shared memory segment */
343   shmid = shmget((key_t) IPC_PRIVATE, 2*sizeof(struct share_mem), IPC_CREAT);
344   if(shmid < 0)
345   {
346     perror("shared memory allocation failure\n");
347     exit(1);
348   }
349
350
351   /* Attach the parent process to the memory (child will inherit)*/
352   if((shareptr = (struct share_mem *)shmat(shmid,0,0)) == (struct share_mem *)-1)
353   perror("Can't attach to shared memory\n");
354
355   /* Handle User Exit */
356   signal(SIGINT, userexit);
357
358   printf("************************************\n*        Network Traffic Analyser     *\n*
            Version: 3.10 Beta          *\n*        Author: Peter Sandford       *\n*       Date:
       March 2005              *\n************************************\n\n");
359
360
361   /* Set the sample size */
362     if(!c)
363     {
364         packet_sample = 300000;
365         printf("Default packet sample size used: %u\n", packet_sample);
366     }
367
```

```
368   /* If we are in alerting mode */
369   if(n){
370     i=0;
371         /* Open the files */
372     alert_in = fopen("alert_variables.txt", "r");
373     /* If we have an error opening */
374     if(alert_in==NULL)
375     {
376       printf("Error Alerting file\n");
377       exit(1);
378     }
379     fgets(currentline, 200, alert_in);
380     while(!feof(alert_in))
381     {
382       alerting_vars[i]=atof(strtok(currentline, "\n"));
383       fgets(currentline, 200, alert_in);
384       i++;
385     }
386   }
387
388   /* If we are compiling the Checksum version */
389   if(f){
390         /* Open the files */
391     printf("Opening output and input files\n");
392     sig_output = fopen("Attacks.txt", "w");
393     fprintf(sig_output,"This is the Attack Log\n");
394     /* If we have an error opening */
395     if(sig_input==NULL)
396     {
397       printf("Error opening sig_input file\n");
398       exit(1);
399     }
400     /* Get a line */
401     fgets(currentline, 200, sig_input);
402     /* Read in all variables and store in the detection matrix */
403     while(!feof(sig_input))
404     {
405       pos = atoi(strtok(currentline, " \n"));
406       printf("%d\n",pos);
407       detect_matrix[pos].port_list[detect_matrix[pos].number] = atoi(strtok(NULL, " \n"));
408       strcpy(detect_matrix[pos].sig_list[detect_matrix[pos].number],strtok(NULL, " \n"));
409       printf("Added '%s' at %d to library at reference %d\n", detect_matrix[pos].sig_list[
              detect_matrix[pos].number], pos, detect_matrix[pos].number);
410       detect_matrix[pos].number++;
411       fgets(currentline, 200, sig_input);
412
413     }
414   }
415     dev="any";
416   /* Check to see if we found a device */
417     if(dev == NULL)
418     {
419     printf("%s\n",errbuf); exit(1);
420     }
421
422   descr1 = pcap_open_live(INTERFACE1,64,1,-1,errbuf);
423   descr2 = pcap_open_live(INTERFACE2,64,1,-1,errbuf);
424     /* Open the device */
425     descr = pcap_open_live("any",64,1,-1,errbuf);
426   if(descr == NULL)
427     {
428     printf("pcap_open_live(): %s\n",errbuf); exit(1);
429   }
430
431   printf("Entering Capture, listening on %s\n\n", dev);
432
433   /* Allow the process to write to the shared memory */
434   shareptr[0].written = 0;  /* [0] indicates the first block of memory 0 indicates NOT written
          */
435   shareptr[0].read = 1;    /* [0] indicates the first block of memory 1 indicates read process
         has finished */
436   shareptr[1].written = 0;  /* [1] indicates the second block of memory 0 indicates NOT
         written */
437   shareptr[1].read = 1;    /* [1] indicates the second block of memory 1 indicates read process
          has finished */
438
439   /* Create a new process */
440   if((pid = fork()) < 0)
441   {
442     perror("fork");
```

```
443     exit(1);
444   }
445
446   /* If we are the child */
447   if(pid == 0)
448   {
449     report();
450   }
451
452     if(!p)
453     {
454         packet_total=-1;
455     }
456     if(a){
457
458       smtp=fopen("smtp_log000000.csv","w");
459     }
460     if(v)
461     {
462         printf("Sample size:%d\n",packet_sample);
463     }
464     /* Main loop. Calls process(packet) when a packets is intercepted */
465   if(packet_total != -1)
466   {
467         if(v){
468             printf("Capturing for %u packets\n",packet_total);
469         }
470     for(i=0;i<(packet_total/packet_sample)+1;i++)
471     {
472       /* If the memory is not currently being read */
473       if (shareptr[memswitchwrite].read == 1)
474       {
475         /* Set the writing flag to prevent the child reading */
476         shareptr[memswitchwrite].written=0;
477         /* Check for lost packets */
478         if(packets_lost ==1)
479         {
480           printf("Potentially dropped 1 or more Packets\n");
481                     packets_lost = 0;
482         }
483         /* Capture packets */
484         pcap_loop(descr,packet_sample,process,NULL);
485         /* Finished writing, so allow the child to read */
486         shareptr[memswitchwrite].written=1;
487         /* Move to the second block of memory */
488         memswitchwrite = !memswitchwrite;
489         if(v){printf("Memswitchwrite = %d\n",memswitchwrite);}
490       }
491       else
492       {
493         /* If we are unable to read currently */
494         packets_lost = 1;
495         i--; /* Decrement to ensure the required number of packets is still captured */
496       }
497     }
498   }
499   else
500   {
501     while(1)
502     {
503             if(v){
504                 printf("Capturing Continuously\n");
505             }
506
507         /* If the memory is not currently being read */
508       if (shareptr[memswitchwrite].read == 1)
509       {
510         /* Set the writing flag to prevent the child reading */
511         shareptr[memswitchwrite].written=0;
512         /* Check for lost packets */
513         if(packets_lost ==1)
514         {
515           printf("Potentially dropped 1 or more Packets\n");
516                     packets_lost = 0;
517         }
518         /* Capture packets */
519         pcap_loop(descr,packet_sample,process,NULL);
520         /* Finished writing, so allow the child to read */
521         shareptr[memswitchwrite].written=1;
522         /* Move to the second block of memory */
```

```
523           memswitchwrite = !memswitchwrite;
524                   if(v){printf("Memswitchwrite = %d\n",memswitchwrite);}
525         }
526         else
527         {
528           /* If we are unable to read currently */
529           packets_lost = 1;
530         }
531       }
532     }
533
534     printf("Leaving Capture\n");
535
536     /* Close capture */
537     pcap_close(descr);
538     pcap_close(descr1);
539     pcap_close(descr2);
540
541     /* Ensure the child recieves the exit command */
542     shareptr[1].written = -1;
543     shareptr[0].written = -1;
544     /* Call function to show all statistics */
545     if(shmdt(shareptr)<0)
546     {
547       perror("Detach Failed\n");
548     }
549
550     printf("Closing Capture\n\n");
551     printf("Exiting Normally\n\n");
552     /* Wait for the child process to exit */
553     sleep(10);
554     /* Free the memory using a separate pointer */
555     printf("Free Memory\n");
556     free(shareptrfree);
557     printf("Memory Free\n");
558     fclose(sig_input);
559     fclose(sig_output);
560     /* Release the shared memory segment */
561     if ((rtrn = shmctl(shmid, IPC_RMID, shmid_ds)) == -1)
562     {
563       perror("shmctl: shmctl failed");
564       exit(1);
565     }
566     else
567     {
568       printf("Shared Memory Released\n");
569     }
570
571
572
573     return 0;
574 }
575
576
577 void process(u_char *useless,const struct pcap_pkthdr* pkthdr,const u_char* packet)
578 {
579   const struct sniff_ethernet *ethernet;  /* The ethernet header */
580   const struct sniff_ip *ip;         /* The IP header */
581   const struct sniff_tcp *tcp;       /* The TCP header */
582   const struct sniff_udp *udp;       /* The TCP header */
583     int sum;                /* Checksum calculated */
584   char *data;            /* A pointer to the data field of the packet */
585   char new_sig[3000];        /* Fingerprint data */
586   int data_length,i,j,k,l,sig_number;
587   char smtp_file[20];
588   struct timeval  smtp_time;
589   struct timezone smtp_zone;
590   l=0;
591
592   /* Map the ethernet and IP headers over the packet */
593   int size_ip = sizeof(struct sniff_ip);
594     int size_ethernet = sizeof(struct sniff_ethernet);
595   /* uint32_t count, ip_count=0;
596    Check we have a valid sized packet */
597   if(strlen((char*)packet)>1600)
598   {
599     printf("ERROR: Packet Exceeded Maximum Size: %d bytes", strlen((char*)packet));
600   }
601
602   /* Use a structure mask to develope variables */
```

```
603   ethernet = (struct sniff_ethernet*)(packet);
604   ip = (struct sniff_ip*)(packet + size_ethernet);
605   tcp = (struct sniff_tcp*)(packet + size_ethernet+size_ip);
606   udp = (struct sniff_udp*)(packet + size_ethernet+size_ip);
607
608   /* If we have an IP ethernet packet */
609   if(ethernet->ether_type == (uint16_t)8)
610   {
611
612     /* If we have not seen this source IP before */
613     if(!shareptr[memswitchwrite].ip_src_array[(uint32_t)(ntohl(ip->ip_src.s_addr)/256)])
614     {
615       shareptr[memswitchwrite].ip_src_array[(uint32_t)(ntohl(ip->ip_src.s_addr)/256)]=1;
616     }
617
618     /* If we have not seen this destination IP before */
619     if(!shareptr[memswitchwrite].ip_dst_array[(uint32_t)(ntohl(ip->ip_dst.s_addr)/256)])
620     {
621       shareptr[memswitchwrite].ip_dst_array[(uint32_t)(ntohl(ip->ip_dst.s_addr)/256)]=1;
622     }
623
624     /* Increment the TTL for the value in the packet */
625     shareptr[memswitchwrite].ttl[ip->ip_ttl]++;
626
627     /* Increment the counter which corresponds to the new packet size */
628     shareptr[memswitchwrite].ip_packet_size[ntohs(ip->ip_len)]++;
629
630     /* Increment the total IP packet counter */
631     shareptr[memswitchwrite].cTotalIP++;
632
633     /* Increment the counter which corresponds to the identification */
634     shareptr[memswitchwrite].identification[ntohs(ip->ip_id)]++;
635
636     /* Increment the counter which corresponds to the checksum */
637     shareptr[memswitchwrite].checksum[ntohs(ip->ip_sum)]++;
638
639     /* Determine the protocol of the new packet */
640     switch(ip->ip_p)
641     {
642       /* TCP */
643       case (char)6:
644           if(f){
645                   if (detect_matrix[ntohs(ip->ip_len)].number != 0)
646               {
647                 data_length = ntohs(ip->ip_len) - (4*(int)ip->ip_hl);
648                   data = (char*)(packet + size_ethernet+size_ip + 8);
649                 sum = ntohs(udp->uh_check);
650
651                 for(j=0;j<detect_matrix[ntohs(ip->ip_len)].number;j++)
652                 {
653                   if(detect_matrix[ntohs(ip->ip_len)].port_list[j]==ntohs(udp->uh_dport))
654                   {
655                     fprintf(sig_output,"Checking for '");
656
657                     for(i=0;i<(data_length - 8);i++)
658                     {
659                       for(k=0;k<2;k++)
660                       {
661                         if((char)*(detect_matrix[ntohs(ip->ip_len)].sig_list[j]+l)!=(char)0)
662                         {
663                         if((int)*(detect_matrix[ntohs(ip->ip_len)].sig_list[j]+l)<59)
664                           {
665                             sig_number += ((int)(*(detect_matrix[ntohs(ip->ip_len)].sig_list[j
                                  ]+l))-48)*pow(16,(1-k));
666                           }
667                           else
668                           {
669                             sig_number += ((int)(*(detect_matrix[ntohs(ip->ip_len)].sig_list[j
                                  ]+l))-55)*pow(16,(1-k));
670                           }
671                         }
672                         l++;
673                       }
674                       *(new_sig+i) = (char)sig_number;
675                       sig_number=0;
676
677                       *(data+i) = *(new_sig+i);
678                       fprintf(sig_output, "%c",*(detect_matrix[ntohs(ip->ip_len)].sig_list[j]+
                            i));
679                     }
```

```
680                     l=0;
681                     fprintf(sig_output,"'\n");
682                     libnet_do_checksum((u_char*)ip, IPPROTO_TCP, data_length);
683
684                     if(ntohs(udp->uh_check)==sum)
685                     {
686                       fprintf(sig_output,"Virus Identified\n");
687                     }
688                   }
689                 }
690               }
691           }
692         shareptr[memswitchwrite].cTCP++;
693         shareptr[memswitchwrite].saTCP -= ((shareptr[memswitchwrite].saTCP - ntohs(ip->ip_len)
              )/shareptr[memswitchwrite].cTCP);
694         shareptr[memswitchwrite].saTotal -= ((shareptr[memswitchwrite].saTotal - ntohs(ip->
              ip_len))/shareptr[memswitchwrite].cTotalIP);
695         shareptr[memswitchwrite].tcp_sport[ntohs(tcp->th_sport)]++;
696         shareptr[memswitchwrite].tcp_dport[ntohs(tcp->th_dport)]++;
697
698                 /* Check for SMTP */
699                 if(a){
700             if((ntohs(tcp->th_dport)==25))
701             {
702             if(v){printf("Got SMTP Packet\n");}
703                         shareptr[memswitchwrite].smtp_total_bits += ntohs(ip->ip_len);
704                         if(tcp->th_flags & TH_SYN){
705               if(v){printf("Got SMTP Syn\n");}
706                   shareptr[memswitchwrite].smtp_count++;
707                               /*printf("\nSMTP Syn\n");*/
708                                 if(
709                                       (ntohl(ip->ip_dst.s_addr)!=(0x5005B6C1)) &&
710                                       (ntohl(ip->ip_dst.s_addr)!=(0x5167DD0B)) &&
711                                       (ntohl(ip->ip_dst.s_addr)!=(0xD4DAA208))
712                                   ){
713 /*                                  fprintf("PACKET SMTP HEADING WRONG WAY\n");*/
714 /*                                  printf("%u.%u: %x,%x\n",ntohl(ip->ip_dst.s_addr));*/
715 /*                  fprintf(smtp,"%x,%x\n",ntohl(ip->ip_src.s_addr),ntohl(ip->ip_dst.s_addr));
      */
716                 gettimeofday(&smtp_time, &smtp_zone);
717                   fprintf(smtp,"%i.%06li,%d.%d.%d.%d,%d.%d.%d.%d\n",smtp_time.tv_sec,smtp_time
                        .tv_usec,(u_char)(ntohl(ip->ip_src.s_addr)>>24),(u_char)(ntohl(ip->
                        ip_src.s_addr)>>16),(u_char)(ntohl(ip->ip_src.s_addr)>>8),(u_char)(ntohl
                        (ip->ip_src.s_addr)),(u_char)(ntohl(ip->ip_dst.s_addr)>>24),(u_char)(
                        ntohl(ip->ip_dst.s_addr)>>16),(u_char)(ntohl(ip->ip_dst.s_addr)>>8),(
                        u_char)(ntohl(ip->ip_dst.s_addr)));
718                               shareptr[memswitchwrite].Non_NTL_smtp_total_bits += ntohs(ip->
                                    ip_len);
719                   shareptr[memswitchwrite].Non_NTL_smtp_count++;
720                 smtp_packet_count++;
721                 if(smtp_packet_count > SMTP_FILE_SIZE){
722                   smtp_file_count++;
723                   fclose(smtp);
724                   sprintf(smtp_file,"smtp_Log%06d.csv",smtp_file_count);
725                   smtp = fopen(smtp_file,"w");
726                   smtp_packet_count=0;
727                 }
728                 if(v){printf("SMTP Not heading for NTL Servers\n");}
729                               }
730                         }
731
732             }
733                 }
734
735         /* Increment the TTL for the value in the packet */
736         shareptr[memswitchwrite].tcp_ttl[ip->ip_ttl]++;
737
738         /* Increment the counter which corresponds to the new packet size */
739         shareptr[memswitchwrite].tcp_packet_size[ntohs(ip->ip_len)]++;
740
741         /* Check for set Set flags with a logical operand */
742         if(tcp->th_flags & TH_FIN)
743         {
744           shareptr[memswitchwrite].FIN++;
745         }
746         if(tcp->th_flags & TH_SYN)
747         {
748           shareptr[memswitchwrite].SYN++;
749         }
750         if(tcp->th_flags & TH_RST)
```

```
751           {
752             shareptr[memswitchwrite].RST++;
753           }
754           if(tcp->th_flags & TH_PUSH)
755           {
756             shareptr[memswitchwrite].PUSH++;
757           }
758           if(tcp->th_flags & TH_ACK)
759           {
760             shareptr[memswitchwrite].ACK++;
761           }
762           if(tcp->th_flags & TH_URG)
763           {
764             shareptr[memswitchwrite].URG++;
765           }
766
767           break;
768         /* ICMP */
769         case (char)1:
770           shareptr[memswitchwrite].cICMP++;
771           shareptr[memswitchwrite].saICMP -= ((shareptr[memswitchwrite].saICMP - ntohs(ip->
                ip_len))/shareptr[memswitchwrite].cICMP);
772           shareptr[memswitchwrite].saTotal -= ((shareptr[memswitchwrite].saTotal - ntohs(ip->
                ip_len))/shareptr[memswitchwrite].cTotalIP);
773
774           /* Increment the TTL for the value in the packet */
775           shareptr[memswitchwrite].icmp_ttl[ip->ip_ttl]++;
776
777           /* Increment the counter which corresponds to the new packet size */
778           shareptr[memswitchwrite].icmp_packet_size[ntohs(ip->ip_len)]++;
779           break;
780         /* UDP */
781         case (char)17:
782           if(f){
783                   if (detect_matrix[ntohs(ip->ip_len)].number != 0)
784                 {
785                 data_length = ntohs(ip->ip_len) - (4*(int)ip->ip_hl);
786                 data = (char*)(packet + size_ethernet+size_ip + 8);
787
788                 sum = ntohs(udp->uh_check);
789
790                 for(j=0;j<detect_matrix[ntohs(ip->ip_len)].number;j++)
791                 {
792                   if(detect_matrix[ntohs(ip->ip_len)].port_list[j]==ntohs(udp->uh_dport))
793                   {
794                     fprintf(sig_output,"Checking for '");
795
796                     for(i=0;i<(data_length - 8);i++)
797                     {
798                       for(k=0;k<2;k++)
799                       {
800                         if((char)*(detect_matrix[ntohs(ip->ip_len)].sig_list[j]+l)!=(char)0)
801                         {
802
803                           if((int)*(detect_matrix[ntohs(ip->ip_len)].sig_list[j]+l)<59)
804                           {
805                             sig_number += ((int)(*(detect_matrix[ntohs(ip->ip_len)].sig_list[j]+
                                  l))-48)*pow(16,(1-k));
806                           }
807                           else
808                           {
809                             sig_number += ((int)(*(detect_matrix[ntohs(ip->ip_len)].sig_list[j]+
                                  l))-55)*pow(16,(1-k));
810                           }
811                         }
812                         l++;
813                       }
814                       /* *(data+i) = *(detect_matrix[ntohs(ip->ip_len)].sig_list[j]+i); */
815                       *(new_sig+i) = (char)sig_number;
816                       sig_number=0;
817
818
819                       *(data+i) = *(new_sig+i);
820                       fprintf(sig_output, "%c",*(detect_matrix[ntohs(ip->ip_len)].sig_list[j]+i)
                            );
821                     }
822                     l=0;
823                     fprintf(sig_output,"'\n");
824                     libnet_do_checksum((u_char*)ip, IPPROTO_UDP, data_length);
825
```

```
826                     if(ntohs(udp->uh_check)==sum)
827                     {
828                       fprintf(sig_output,"Virus Identified\n");
829                     }
830                   }
831                 }
832               }
833         }
834         shareptr[memswitchwrite].cUDP++;
835         shareptr[memswitchwrite].saUDP -= ((shareptr[memswitchwrite].saUDP - ntohs(ip->ip_len)
                )/shareptr[memswitchwrite].cUDP);
836         shareptr[memswitchwrite].saTotal -= ((shareptr[memswitchwrite].saTotal - ntohs(ip->
                ip_len))/shareptr[memswitchwrite].cTotalIP);
837         shareptr[memswitchwrite].udp_sport[ntohs(udp->uh_sport)]++;
838         shareptr[memswitchwrite].udp_dport[ntohs(udp->uh_dport)]++;
839
840         /* Increment the TTL for the value in the packet */
841         shareptr[memswitchwrite].udp_ttl[ip->ip_ttl]++;
842
843         /* Increment the counter which corresponds to the new packet size */
844         shareptr[memswitchwrite].udp_packet_size[ntohs(ip->ip_len)]++;
845         break;
846       default:
847         shareptr[memswitchwrite].cUnknownIP++;
848         shareptr[memswitchwrite].saTotal -= ((shareptr[memswitchwrite].saTotal - ntohs(ip->
                ip_len))/shareptr[memswitchwrite].cTotalIP);
849         break;
850     }
851   }
852   /* Increment the total packet counter */
853   packet_count++;
854
855 }
856
857 void report(void)
858 {
859   uint32_t count,ip_count=0, ip_countd=0;
860   int difference_sec;
861   long difference_usec;
862   uint16_t ttl_position[5];
863   long ttl_value[5];
864   uint16_t udp_ttl_position[5];
865   long udp_ttl_value[5];
866   uint16_t tcp_ttl_position[5];
867   long tcp_ttl_value[5];
868   uint16_t icmp_ttl_position[5];
869   long icmp_ttl_value[5];
870   char file[100];
871   char* ip_str;
872   char currentline[201];
873   struct in_addr in;
874   FILE* output;
875   FILE* Dist;
876   int highest_packet_location[5];
877   int highest_packet_count[5];
878   int udp_highest_packet_location[5];
879   int udp_highest_packet_count[5];
880   int tcp_highest_packet_location[5];
881   int tcp_highest_packet_count[5];
882   int icmp_highest_packet_location[5];
883   int icmp_highest_packet_count[5];
884   int i,j;
885   int highest_port_frequency[10];
886   int highest_udp_port_frequency[5];
887   int highest_port_location[10];
888   int highest_udp_port_location[5];
889   int portlist[65535];
890   int portlist_udp[65535];
891   int x;
892   /* Time variables */
893   struct timeval  start,end;
894   struct timezone tpz;
895     struct timespec t;
896   char output_s[50];
897   char output_st[9500];
898   char command[500];
899
900   int highest_id, highest_id_location, highest_idb, highest_id_locationb;
901   float highest_id_ratio;
902
```

```
903   int highest_sum, highest_sum_location, highest_sumb, highest_sum_locationb;
904   float highest_sum_ratio;
905   float normal[256];
906   float current[256];
907   float packet_norm[1500];
908   float packet_curr[1500];
909   float ks_packet, ks_ttl;
910
911   Dist = fopen("TTL_Distribution.csv","r");
912   if(Dist==NULL){
913     printf("Error Opening TTL Distribution File\n");
914     exit(1);
915   }
916
917   fgets(currentline, 200, Dist);
918   x=1;
919   while(!feof(Dist)){
920       normal[x] = atof(strtok(currentline, " \n"));
921       if(v)printf("Added '%f' at %d to TTL Distribution\n", normal[x],x);
922       x++;
923       fgets(currentline, 200, Dist);
924   }
925   fclose(Dist);
926   Dist = fopen("Packet_Distribution.csv","r");
927   if(Dist==NULL){
928     printf("Error Opening Packet Size Distribution File\n");
929     exit(1);
930   }
931   x=0;
932   fgets(currentline, 200, Dist);
933   while(!feof(Dist)){
934       packet_norm[x] = atof(strtok(currentline, " \n"));
935       if(v)printf("Added '%f' at %d to Packet Size Distribution\n", packet_norm[x],x);
936       x++;
937       fgets(currentline, 200, Dist);
938   }
939   fclose(Dist);
940
941
942   sprintf(file, "Log%u", file_count);
943   file_count = file_count * FILE_SIZE;
944   alert_file=fopen("alert.csv","a");
945
946
947   /* Check to see if we have received a stop command */
948   while(shareptr[memswitchread].written!= -1)
949   {
950     if(shareptr[memswitchread].written==1)
951     {
952       /* Set the reading flag, to prevent the parent from writing in this memory */
953       shareptr[memswitchread].read = 0;
954             if(v){
955                 printf("Child Processing Data\n");
956             }
957
958         /* Get the start time of the next capture */
959         gettimeofday(&end, &tpz);
960                     nanosleep(&t,NULL);
961         difference_sec = end.tv_sec - start.tv_sec;
962         difference_usec = (end.tv_usec - start.tv_usec);
963         start.tv_sec = end.tv_sec;
964         start.tv_usec = end.tv_usec;
965
966         /* Reset Variables */
967         ip_count = 0;
968         ip_countd = 0;
969
970         /* Packet Size Counts */
971         for(i=0;i<5;i++)
972         {
973           highest_packet_location[i] = 0;
974           highest_packet_count[i] = 0;
975
976           tcp_highest_packet_location[i] = 0;
977           tcp_highest_packet_count[i] = 0;
978
979           udp_highest_packet_location[i] = 0;
980           udp_highest_packet_count[i] = 0;
981
982           icmp_highest_packet_location[i] = 0;
```

```
983             icmp_highest_packet_count[i] = 0;
984
985             ttl_value[i] = -1;
986             udp_ttl_value[i] = -1;
987             icmp_ttl_value[i] = -1;
988             tcp_ttl_value[i] = -1;
989
990             highest_udp_port_location[i] = 0;
991             highest_udp_port_frequency[i] = 0;
992           }
993
994           for(i=0;i<10;i++){
995                       highest_port_frequency[i] = 0;
996             highest_port_location[i] = 0;
997                   }
998
999           /* IP Identification */
1000          highest_id = 0;
1001          highest_id_location = 0;
1002          highest_idb = 0;
1003          highest_id_locationb = 0;
1004          highest_id_ratio = 0;
1005
1006          /* IP Checksum */
1007          highest_sum = 0;
1008          highest_sum_location = 0;
1009          highest_sumb = 0;
1010          highest_sum_locationb = 0;
1011          highest_sum_ratio = 0;
1012
1013          for(i=0;i<65535;i++)
1014          {
1015            portlist[i] = shareptr[memswitchread].tcp_sport[i] + shareptr[memswitchread].
                  tcp_dport[i];
1016            portlist_udp[i] = shareptr[memswitchread].udp_sport[i] + shareptr[memswitchread].
                  udp_dport[i];
1017          }
1018
1019          /* Calculate the time */
1020          if(difference_usec < 0)
1021          {
1022            difference_usec = difference_usec + 1000000;
1023            difference_sec--;
1024          }
1025
1026          strcpy(output_st,"");
1027          sprintf(output_s,"%d",site);
1028          strcat(output_st, output_s);
1029          sprintf(output_s,"%lu,", end.tv_sec);
1030          strcat(output_st, output_s);
1031 /*         fprintf(smtp,"%lu,", end.tv_sec); */
1032          if(v){printf("%i.%06li\n",difference_sec,difference_usec);}
1033          sprintf(output_s,"%i.%06li,",difference_sec,difference_usec);
1034          strcat(output_st, output_s);
1035
1036
1037
1038          /* Count the number of sub-nets captured and reset */
1039          for(count=0; count<IP_RANGE; count++)
1040          {
1041            ip_count += shareptr[memswitchread].ip_src_array[count];
1042            shareptr[memswitchread].ip_src_array[count] = 0;
1043            ip_countd += shareptr[memswitchread].ip_dst_array[count];
1044            shareptr[memswitchread].ip_dst_array[count] = 0;
1045          }
1046
1047          for(i=0;i<256;i++){
1048            current[i]=shareptr[memswitchread].ttl[i];
1049          }
1050          /* perform KS on the TTL distribution */
1051          ks_ttl = ks(normal, current,256,256);
1052          if(v){printf("KS for TTL was %f\n", ks_ttl);}
1053
1054          /* perform KS on the Packet size distribution */
1055          for(i=0;i<1500;i++){
1056            packet_curr[i]=shareptr[memswitchread].ip_packet_size[i];
1057          }
1058          ks_packet = ks(packet_norm, packet_curr,1500,1500);
1059          if(v){printf("KS for Packet Sizes was %f\n", ks_packet);}
1060
```

```
1061          /* Calculate the value and position of the largest TTL */
1062          for(count=0; count<256; count++)
1063          {
1064            for(i=0;i<5;i++)
1065            {
1066              if(shareptr[memswitchread].ttl[count] > ttl_value[i])
1067              {
1068                for(j=0;j<4-i;j++)
1069                {
1070                  ttl_value[4-j] = ttl_value[4-j-1];
1071                  ttl_position[4-j] = ttl_position[4-j-1];
1072                }
1073
1074                ttl_value[i] = shareptr[memswitchread].ttl[count];
1075                ttl_position[i] = count;
1076                i=5;
1077              }
1078            }
1079
1080            for(i=0;i<5;i++)
1081            {
1082              if(shareptr[memswitchread].udp_ttl[count] > udp_ttl_value[i])
1083              {
1084                for(j=0;j<4-i;j++)
1085                {
1086                  udp_ttl_value[4-j] = udp_ttl_value[4-j-1];
1087                  udp_ttl_position[4-j] = udp_ttl_position[4-j-1];
1088                }
1089
1090                udp_ttl_value[i] = shareptr[memswitchread].udp_ttl[count];
1091                udp_ttl_position[i] = count;
1092                i=5;
1093              }
1094            }
1095
1096
1097            for(i=0;i<5;i++)
1098            {
1099              if(shareptr[memswitchread].tcp_ttl[count] > tcp_ttl_value[i])
1100              {
1101                for(j=0;j<4-i;j++)
1102                {
1103                  tcp_ttl_value[4-j] = tcp_ttl_value[4-j-1];
1104                  tcp_ttl_position[4-j] = tcp_ttl_position[4-j-1];
1105                }
1106
1107                tcp_ttl_value[i] = shareptr[memswitchread].tcp_ttl[count];
1108                tcp_ttl_position[i] = count;
1109              i=5;
1110              }
1111            }
1112
1113            for(i=0;i<5;i++)
1114            {
1115              if(shareptr[memswitchread].icmp_ttl[count] > icmp_ttl_value[i])
1116              {
1117                for(j=0;j<4-i;j++)
1118                {
1119                  icmp_ttl_value[4-j] = icmp_ttl_value[4-j-1];
1120                  icmp_ttl_position[4-j] = icmp_ttl_position[4-j-1];
1121                }
1122
1123                icmp_ttl_value[i] = shareptr[memswitchread].icmp_ttl[count];
1124                icmp_ttl_position[i] = count;
1125                i=5;
1126              }
1127            }
1128
1129            shareptr[memswitchread].ttl[count] = 0;
1130            shareptr[memswitchread].udp_ttl[count] = 0;
1131            shareptr[memswitchread].tcp_ttl[count] = 0;
1132            shareptr[memswitchread].icmp_ttl[count] = 0;
1133          }
1134
1135
1136
1137          /* Calculate the most common IP sizes */
1138          for(count=0; count<1538; count++)
1139          {
1140            for(i=0;i<5;i++)
```

```
1141            {
1142              if(shareptr[memswitchread].ip_packet_size[count] > highest_packet_count[i])
1143              {
1144                for(j=0;j<4-i;j++)
1145                {
1146                  highest_packet_count[4-j] = highest_packet_count[4-j-1];
1147                  highest_packet_location[4-j] = highest_packet_location[4-j-1];
1148                }
1149
1150                highest_packet_count[i] = shareptr[memswitchread].ip_packet_size[count];
1151                highest_packet_location[i] = count;
1152                i=5;
1153              }
1154            }
1155
1156            for(i=0;i<5;i++)
1157            {
1158              if(shareptr[memswitchread].udp_packet_size[count] > udp_highest_packet_count[i])
1159              {
1160                for(j=0;j<4-i;j++)
1161                {
1162                  udp_highest_packet_count[4-j] = udp_highest_packet_count[4-j-1];
1163                  udp_highest_packet_location[4-j] = udp_highest_packet_location[4-j-1];
1164                }
1165
1166                udp_highest_packet_count[i] = shareptr[memswitchread].udp_packet_size[count];
1167                udp_highest_packet_location[i] = count;
1168                i=5;
1169              }
1170            }
1171
1172
1173            for(i=0;i<5;i++)
1174            {
1175              if(shareptr[memswitchread].tcp_packet_size[count] > tcp_highest_packet_count[i])
1176              {
1177                for(j=0;j<4-i;j++)
1178                {
1179                  tcp_highest_packet_count[4-j] = tcp_highest_packet_count[4-j-1];
1180                  tcp_highest_packet_location[4-j] = tcp_highest_packet_location[4-j-1];
1181                }
1182
1183                tcp_highest_packet_count[i] = shareptr[memswitchread].tcp_packet_size[count];
1184                tcp_highest_packet_location[i] = count;
1185                i=5;
1186              }
1187            }
1188
1189            for(i=0;i<5;i++)
1190            {
1191              if(shareptr[memswitchread].icmp_packet_size[count] > icmp_highest_packet_count[i])
1192              {
1193                for(j=0;j<4-i;j++)
1194                {
1195                  icmp_highest_packet_count[4-j] = icmp_highest_packet_count[4-j-1];
1196                  icmp_highest_packet_location[4-j] = icmp_highest_packet_location[4-j-1];
1197                }
1198
1199                icmp_highest_packet_count[i] = shareptr[memswitchread].icmp_packet_size[count];
1200                icmp_highest_packet_location[i] = count;
1201                i=5;
1202              }
1203            }
1204            shareptr[memswitchread].ip_packet_size[count] = 0;
1205            shareptr[memswitchread].udp_packet_size[count] = 0;
1206            shareptr[memswitchread].tcp_packet_size[count] = 0;
1207            shareptr[memswitchread].icmp_packet_size[count] = 0;
1208          }
1209                  /* Calculate the most common UDP ports */
1210          for(count=0; count<65535; count++)
1211          {
1212
1213            for(i=0;i<5;i++)
1214            {
1215              if(portlist_udp[count] > highest_udp_port_frequency[i])
1216              {
1217                for(j=0;j<4-i;j++)
1218                {
1219                  highest_udp_port_frequency[4-j] = highest_udp_port_frequency[4-j-1];
1220                  highest_udp_port_location[4-j] = highest_udp_port_location[4-j-1];
```

```
1221                }
1222
1223                highest_udp_port_frequency[i] = portlist_udp[count];
1224                highest_udp_port_location[i] = count;
1225                i=5;
1226              }
1227            }
1228
1229            shareptr[memswitchread].udp_sport[count] = 0;
1230            shareptr[memswitchread].udp_dport[count] = 0;
1231            portlist_udp[count]=0;
1232                  }
1233          /* Calculate the most common tcp Ports */
1234          for(count=0; count<65535; count++)
1235          {
1236
1237            for(i=0;i<10;i++)
1238            {
1239              if(portlist[count] > highest_port_frequency[i])
1240              {
1241                for(j=0;j<9-i;j++)
1242                {
1243                  highest_port_frequency[9-j] = highest_port_frequency[9-j-1];
1244                  highest_port_location[9-j] = highest_port_location[9-j-1];
1245                }
1246
1247                highest_port_frequency[i] = portlist[count];
1248                highest_port_location[i] = count;
1249                i=10;
1250              }
1251            }
1252
1253            shareptr[memswitchread].tcp_sport[count] = 0;
1254            shareptr[memswitchread].tcp_dport[count] = 0;
1255            portlist[count]=0;
1256          }
1257
1258
1259          /* Calculate Id numbers */
1260          for(count=0; count<65535; count++)
1261          {
1262
1263            if(shareptr[memswitchread].identification[count] > highest_id)
1264            {
1265              highest_id = shareptr[memswitchread].identification[count];
1266              highest_id_location = count;
1267            }
1268
1269            if(shareptr[memswitchread].checksum[count] > highest_sum)
1270            {
1271              highest_sum = shareptr[memswitchread].checksum[count];
1272              highest_sum_location = count;
1273            }
1274          }
1275
1276          for(count=0; count<65535; count++)
1277          {
1278            if(shareptr[memswitchread].identification[count] > highest_idb && count !=
                   highest_id_location)
1279            {
1280              highest_idb = shareptr[memswitchread].identification[count];
1281              highest_id_locationb = count;
1282            }
1283            shareptr[memswitchread].identification[count] = 0;
1284
1285            if(shareptr[memswitchread].checksum[count] > highest_sumb && count !=
                   highest_sum_location)
1286            {
1287              highest_sumb = shareptr[memswitchread].checksum[count];
1288              highest_sum_locationb = count;
1289            }
1290            shareptr[memswitchread].checksum[count] = 0;
1291          }
1292
1293
1294          /*if(a){*/
1295                  /* Print out SMTP Traffic for bot-net analysis */
1296              /*if(shareptr[memswitchwrite].smtp_count > SMTP_TRIGGER)
1297              {
1298                for(count=0; count<shareptr[memswitchwrite].smtp_count;count++)
```

```
1299                {
1300                    in.s_addr = shareptr[memswitchread].smtp_machine_ip[count];
1301                    ip_str = inet_ntoa(in); */
1302                    /* printf("IP was %i.%6li,%s\n", difference_sec, difference_usec, ip_str); */
1303 /*                   fprintf(smtp,"%i,%s\n",end.tv_sec,ip_str);
1304
1305                }
1306            }
1307                }*/
1308
1309
1310        /* Calculate the ratio between id numbers */
1311        highest_id_ratio = ((float)highest_id/(float)highest_idb)*100.0;
1312
1313        highest_sum_ratio = ((float)highest_sum/(float)highest_sumb)*100.0;
1314
1315        /* Output stuff to file */
1316        sprintf(output_s,"%u,",ip_count);
1317        strcat(output_st, output_s);
1318        sprintf(output_s,"%u,",ip_countd);
1319        strcat(output_st, output_s);
1320        sprintf(output_s,"%d,", highest_id_location);
1321        strcat(output_st, output_s);
1322        sprintf(output_s,"%f,", highest_id_ratio);
1323        strcat(output_st, output_s);
1324        sprintf(output_s,"%d,", highest_sum_location);
1325        strcat(output_st, output_s);
1326        sprintf(output_s,"%f,", highest_sum_ratio);
1327        strcat(output_st, output_s);
1328
1329        for(i=0;i<5;i++){sprintf(output_s,"%lu,",ttl_value[i]);strcat(output_st, output_s);}
1330        for(i=0;i<5;i++){sprintf(output_s,"%hu,",ttl_position[i]);strcat(output_st, output_s)
                ;}
1331        for(i=0;i<5;i++){sprintf(output_s,"%lu,",udp_ttl_value[i]);strcat(output_st, output_s)
                ;}
1332        for(i=0;i<5;i++){sprintf(output_s,"%hu,",udp_ttl_position[i]);strcat(output_st,
                output_s);}
1333        for(i=0;i<5;i++){sprintf(output_s,"%lu,",tcp_ttl_value[i]);strcat(output_st, output_s)
                ;}
1334        for(i=0;i<5;i++){sprintf(output_s,"%hu,",tcp_ttl_position[i]);strcat(output_st,
                output_s);}
1335        for(i=0;i<5;i++){sprintf(output_s,"%lu,",icmp_ttl_value[i]);strcat(output_st, output_s
                );}
1336        for(i=0;i<5;i++){sprintf(output_s,"%hu,",icmp_ttl_position[i]);strcat(output_st,
                output_s);}
1337        for(i=0;i<5;i++){sprintf(output_s,"%d,",highest_packet_location[i]);strcat(output_st,
                output_s);}
1338        for(i=0;i<5;i++){sprintf(output_s,"%d,",highest_packet_count[i]);strcat(output_st,
                output_s);}
1339        for(i=0;i<5;i++){sprintf(output_s,"%d,",udp_highest_packet_location[i]);strcat(
                output_st, output_s);}
1340        for(i=0;i<5;i++){sprintf(output_s,"%d,",udp_highest_packet_count[i]);strcat(output_st,
                 output_s);}
1341        for(i=0;i<5;i++){sprintf(output_s,"%d,",tcp_highest_packet_location[i]);strcat(
                output_st, output_s);}
1342        for(i=0;i<5;i++){sprintf(output_s,"%d,",tcp_highest_packet_count[i]);strcat(output_st,
                 output_s);}
1343        for(i=0;i<5;i++){sprintf(output_s,"%d,",icmp_highest_packet_location[i]);strcat(
                output_st, output_s);}
1344        for(i=0;i<5;i++){sprintf(output_s,"%d,",icmp_highest_packet_count[i]);strcat(output_st
                , output_s);}
1345        for(i=0;i<10;i++){sprintf(output_s,"%d,",highest_port_location[i]);strcat(output_st,
                output_s);}
1346        for(i=0;i<10;i++){sprintf(output_s,"%d,",highest_port_frequency[i]);strcat(output_st,
                output_s);}
1347        for(i=0;i<5;i++){sprintf(output_s,"%d,",highest_udp_port_location[i]);strcat(output_st
                , output_s);}
1348        for(i=0;i<5;i++){sprintf(output_s,"%d,",highest_udp_port_frequency[i]);strcat(
                output_st, output_s);}
1349
1350
1351        sprintf(output_s,"%li,",shareptr[memswitchread].cTotalIP);
1352        strcat(output_st, output_s);
1353        sprintf(output_s,"%li,",shareptr[memswitchread].saTotal);
1354        strcat(output_st, output_s);
1355        sprintf(output_s,"%li,",shareptr[memswitchread].cTCP);
1356        strcat(output_st, output_s);
1357        sprintf(output_s,"%li,",shareptr[memswitchread].saTCP);
1358        strcat(output_st, output_s);
1359        sprintf(output_s,"%li,",shareptr[memswitchread].cUDP);
```

```
1360          strcat(output_st, output_s);
1361          sprintf(output_s,"%li,",shareptr[memswitchread].saUDP);
1362          strcat(output_st, output_s);
1363          sprintf(output_s,"%li,",shareptr[memswitchread].cICMP);
1364          strcat(output_st, output_s);
1365          sprintf(output_s,"%li,",shareptr[memswitchread].saICMP);
1366          strcat(output_st, output_s);
1367          sprintf(output_s,"%li,",shareptr[memswitchread].cUnknownIP);
1368          strcat(output_st, output_s);
1369          sprintf(output_s,"%li,",shareptr[memswitchread].FIN);
1370          strcat(output_st, output_s);
1371          sprintf(output_s,"%li,",shareptr[memswitchread].SYN);
1372          strcat(output_st, output_s);
1373          sprintf(output_s,"%li,",shareptr[memswitchread].RST);
1374          strcat(output_st, output_s);
1375          sprintf(output_s,"%li,",shareptr[memswitchread].PUSH);
1376          strcat(output_st, output_s);
1377          sprintf(output_s,"%li,",shareptr[memswitchread].ACK);
1378          strcat(output_st, output_s);
1379          sprintf(output_s,"%li,",shareptr[memswitchread].URG);
1380          strcat(output_st, output_s);
1381                  sprintf(output_s,"%li,",shareptr[memswitchread].smtp_count);
1382          strcat(output_st, output_s);
1383                  sprintf(output_s,"%li,",shareptr[memswitchread].smtp_total_bits);
1384          strcat(output_st, output_s);
1385                  sprintf(output_s,"%li,",shareptr[memswitchread].Non_NTL_smtp_count);
1386          strcat(output_st, output_s);
1387                  sprintf(output_s,"%li,",shareptr[memswitchread].Non_NTL_smtp_total_bits);
1388          strcat(output_st, output_s);
1389          sprintf(output_s,"%f,%f",ks_ttl,ks_packet);
1390          strcat(output_st, output_s);
1391
1392          if(n)
1393                  {
1394                       if(alert())
1395              {
1396                printf("Alerting Module Triggered\n");
1397              }
1398                  }
1399
1400          printf("Log%d\n", file_count);
1401          printf("End Report\n\n\n");
1402          /* Counters for the various protocols */
1403          packet_count = 0;
1404          shareptr[memswitchread].cTCP = 0;
1405          shareptr[memswitchread].cUDP = 0;
1406          shareptr[memswitchread].cTotalIP = 0;
1407          shareptr[memswitchread].cICMP = 0;
1408          shareptr[memswitchread].cUnknownIP = 0;
1409          /* Average packet sizes for the various protocols */
1410          shareptr[memswitchread].saTotal = 0;
1411          shareptr[memswitchread].saTCP = 0;
1412          shareptr[memswitchread].saUDP = 0;
1413          shareptr[memswitchread].saICMP = 0;
1414          shareptr[memswitchread].FIN =0;
1415          shareptr[memswitchread].SYN =0;
1416          shareptr[memswitchread].RST =0;
1417          shareptr[memswitchread].PUSH=0;
1418          shareptr[memswitchread].ACK=0;
1419          shareptr[memswitchread].URG=0;
1420          shareptr[memswitchread].smtp_count=0;
1421                  shareptr[memswitchread].smtp_total_bits=0;
1422          shareptr[memswitchread].skype_count=0;
1423                  shareptr[memswitchread].skype_total_bits=0;
1424          shareptr[memswitchread].Non_NTL_smtp_count=0;
1425                  shareptr[memswitchread].Non_NTL_smtp_total_bits=0;
1426
1427 /*         for(i=0;i<1000;i++)
1428          {
1429            shareptr[memswitchread].smtp_machine_ip[i] = 0;
1430          }*/
1431
1432
1433        /* Allow the parent to read this memory */
1434        file_count++;
1435        /* printf("%s", output_st); */
1436        sprintf(command, "./alert_client 62.253.167.161 7113 \"%s\"", output_st);
1437        system(command);
1438
1439        shareptr[memswitchread].read = 1;
```

```
1440        /* Operate on the other block of memory */
1441        memswitchread = !memswitchread;
1442      }
1443    }
1444    /* We are only ever here if the child has not recieved the exit command */
1445    printf("\nChild Exiting Under Parent Request Conditions\n\n");
1446    if(shmdt(shareptr)<0)
1447    {
1448      perror("Child Detach Failed\n");
1449    }
1450
1451    printf("Child Exit\n");
1452
1453    printf("Free Memory\n");
1454    free(shareptrfree);
1455    printf("Memory Free\n");
1456    fclose(smtp);
1457      if(f){
1458          fclose(sig_input);
1459        fclose(sig_output);
1460      }
1461    exit(0);
1462
1463  }
1464
1465  void userexit(int i)
1466  {
1467    int rtrn=0;
1468
1469    if(pid !=0)
1470    {
1471      printf("\nExiting Under User Request Conditions\n\n");
1472      printf("\nExit Command Recieved\nClosing Down Capture\n\n");
1473    }
1474      if(pid==0){
1475          if(a)
1476          {
1477           fclose(alert_file);
1478           fclose(smtp);
1479          }
1480      }
1481    /* Ensure the child recieves the exit command */
1482    shareptr[1].written = -1;
1483    shareptr[0].written = -1;
1484
1485
1486    if(shmdt(shareptr)<0)
1487    {
1488      perror("Detach Failed\n");
1489    }
1490
1491
1492    printf("Releasing Memory\n");
1493
1494
1495    /* Release the Shared memory segment */
1496    if ((rtrn = shmctl(shmid, IPC_RMID, shmid_ds)) == -1)
1497    {
1498      perror("shmctl: shmctl failed" );
1499          printf("%d\n",pid);
1500    }
1501    else
1502    {
1503      printf("Shared Memory Released\n");
1504    }
1505
1506      if(pid!=0){
1507          sleep(5);
1508      }
1509      /* Release the memory */
1510    if(pid != 0)
1511      {
1512      free(shareptrfree);
1513      printf("Memory Freed\n");
1514          if(f){
1515              fclose(sig_input);
1516          fclose(sig_output);
1517          }
1518    }
1519
```

```
1520    exit(0);
1521  }
1522
1523  int alert(void)
1524  {
1525    struct timeval  tp;
1526    struct timezone tpz;
1527    char alert[600];
1528    char command[800];
1529    /*  Placeholder for alerting conditions */
1530    if(((float)shareptr[memswitchread].SYN / (float)(shareptr[memswitchread].FIN +
1531    (float)shareptr[memswitchread].RST))>alerting_vars[0] || ((float)shareptr[memswitchread].SYN
          /
1532    ((float)shareptr[memswitchread].FIN + (float)shareptr[memswitchread].RST)) < alerting_vars
          [1])
1533    {
1534
1535      gettimeofday(&tp, &tpz);
1536          fprintf(alert_file, "Alarm triggered Syn/Fin+Rst of: %f at %u.%06u\n",(float)(((float)
                shareptr[memswitchread].SYN / ((float)shareptr[memswitchread].FIN + (float)
                shareptr[memswitchread].RST, tp.tv_sec, tp.tv_usec))));
1537          sprintf(alert, "Alarm: Sin to Fin ratio of: %f\n",(float)((float)(shareptr[
                memswitchread].SYN / ((float)shareptr[memswitchread].FIN + (float)shareptr[
                memswitchread].RST + 1))));
1538      sprintf(command, "./alert_client arial 7111 \"%s\"", alert);
1539      system(command);
1540      return 1;
1541    }
1542    return 0;
1543  }
1544
1545
1546  float ks(float *normal, float *current, int l_normal, int l_current)
1547  {
1548    int x;
1549    float ks_max=0;
1550    int s_current=0;
1551
1552    /* Get the total value of the array so we can normalise */
1553    for(x=0;x< l_current;x++){
1554      s_current += current[x];
1555    }
1556
1557    /* Make the distribution cumulative */
1558    current[0] = (current[0]/s_current);
1559
1560    for(x=1;x< l_current;x++){
1561      current[x] = current[x-1] + (current[x]/s_current);
1562    }
1563
1564    for(x=0;x<l_current;x++){
1565      if(v){printf("At %d, values were %f and %f\n",x,current[x],normal[x]);}
1566  //  printf("At %d, values were %f and %f\n",x,current[x],normal[x]);
1567      if(fabs(current[x]-normal[x])>ks_max){
1568        ks_max = fabs(current[x]-normal[x]);
1569      }
1570    }
1571
1572    return ks_max;
1573  }
```