A Network Telescope Information Visualisation Framework

Submitted in partial fulfilment
of the requirements of the degree of

Bachelor of Science (Honours)

of Rhodes University

Samuel Oswald Hunter

Grahamstown, South Africa
November 2010

Abstract

Network telescopes are able to provide a sampled view of the Internet with regard to nefarious traffic. More specifically they are able to provide empirical data based on malicious traffic, unbiasedly targeted towards unused address space. Network telescopes accomplish this by monitoring ranges of unused internet address space in which no legitimate traffic should exist. Analysis of traffic captured by network telescopes has been shown as an effective measure in characterising nefarious traffic caused by worm propagation and distributed denial of service attacks. By choosing the correct metrics for analysis on this traffic one is able to extract information that gives insight and a greater understanding of the current state of illegitimate traffic on the Internet. Collecting the data and extracting the information is however only the first half of the process towards understanding and interpretation of results. Through correct visualisation, large sets of data can be summarised in a compact and easily understandable format. This research focuses on the analysis, manipulation and visualisation of traffic obtained from network telescope monitoring. Analysis was achieved by outlining specific metrics that were used to extract information from captured traffic. This information was then manipulated and prepared for different methods of visualisation. An information visualisation framework named Ember was created with the goal of managing and visualising multiple data sets. The Ember framework was then used alongside existing third party libraries to create a network telescope dashboard. Lastly a case study was conducted with the network telescope dashboard. It was shown that the dashboard was effective at visualising and assisting in the interpretation of results.

ACM Computing Classification System Classification

Thesis classification under the ACM Computing Classification System (1998 version, valid through 2010):
C.2.3 [Network Operations]: Network monitoring, Public networks
D.3.3 [Language Constructs and Features]: Frameworks
K.6.5 [Security and Protection]: Invasive software,
General-Terms: Metrics, Framework, Security

Acknowledgements

Its been a long walk and at times a slow crawl to reach this point, I acknowledge all the support I have received from my friends and family. I would also like to extend a special thanks to my supervisor, Barry Irwin for his guidance and support throughout this year. I would also like to thank my parents for their continued supported and encouragement over the years. I thank my friends for making this a memorable year. Finally, I would like to thank the NRF and Telkom SA, Comverse SA, Stortech, Tellabs, Easttel, Bright Ideas Projects 39 and THRIP through the Telkom Centre of Excellence at Rhodes University for the financial support they provided.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

On a daily basis the many of hosts that are connected to the Internet experience continuous probes and attacks from malicious entities. These entities such as worms, viruses and trojan horses are often also the cause of denial of service attacks. Their automated propagation make up the vast majority of the traffic that is captured by network telescopes. One of the characteristic that a majority of worms share is that during their propagation phase they tend to target internet addresses indiscriminately although sometimes with a local bias. They hold no regard for whether the address has been allocated by service providers or which host device is associated with it. This along with the fact that unused address space is not advertised and no live hosts are associated with it means that the only traffic that should exist there, will be that with potentially malicious intent. Thus removing the problem faced by traditional IDS of having to discriminating between legitimate and il-legitimate traffic.

With the steady advance of technology, internet enabled devices are becoming cheaper and more accessible to the public. This along with decreasing connectivity prices and increases in network bandwidth[44] has meant that more and more devices are connecting to the internet every day. This increases the attack vector for these malicious entities and helps to ensure an increasing population of hosts for future attacks.

The analysis of traffic captured by network telescopes is made easier by the absence of legitimate traffic that would otherwise have obfuscated traffic with malicious intent. This however does not greatly simplify the task of analysis as there is still a great volume and variety of unsolicited traffic to work through.

Any information obtained from this type of traffic analysis is in turn only useful if it can be interpreted and understood correctly. There are many visualisation techniques that can be used to display information and attempt to infer meaning. Some examples of information visualisations that are often used with regards to network traffic analysis include charts, graphs, sparklines, heat signatures and animations. These techniques can be made even more effective if they are grouped into meaningful sets that complement each other. Providing different views into the same data

1.1 Motivation

This thesis focuses on the development of an information visualisation framework which will then be used alongside existing third pary libraries to create a dashboard for monitoring traffic obtained from a network telescope. Information visualisation has been proven as a successful means of summarising large sets of data such as the data obtained from a network telescope. Simple metrics for traffic analysis will be identified and implemented for use by the dashboard.

A strong motivation for the research described in this thesis comes from the need for effective and efficient visualisation of network telescope data. This visualisation need to be done in manner that lends itself to the explanation of results and helps users to identify similarities and anomalies that might warrant further investigation. There is also a need to reduce large quantities of data into smaller re-usable sets of useful information, this is often a time consuming process and poses a serious problem for a real-time tool such as an information dashboard. Lastly there is a need to automate the process of analysing un-solicited traffic which in the past has predominantly been a manual process.

Past research in the field of un-solicited traffic analysis has proven successful in observation, identification and tracking of Distributed Denial of Service(DDoS) attacks[5][27], worm prorogation[21][32] and network scanning[9]. An example that serves as motivation for the use of network telescopes in the detection and tracking of a worm outbreak follows. Less than a month after Microsoft disclosed the MS08-67[40] vulnerability that affected multiple versions of the Windows operating system, the Conficker[39] worm was set loose onto the internet. Figure 1.1 shows traffic observed by CAIDA[1] and indicates unique IP source addresses per day that were scanning network telescope addresses on TCP/445[5]. It should be noted that several outages caused gaps in the data displayed in Figure 1.1.

[1]CAIDA - http://www.caida.org/home/

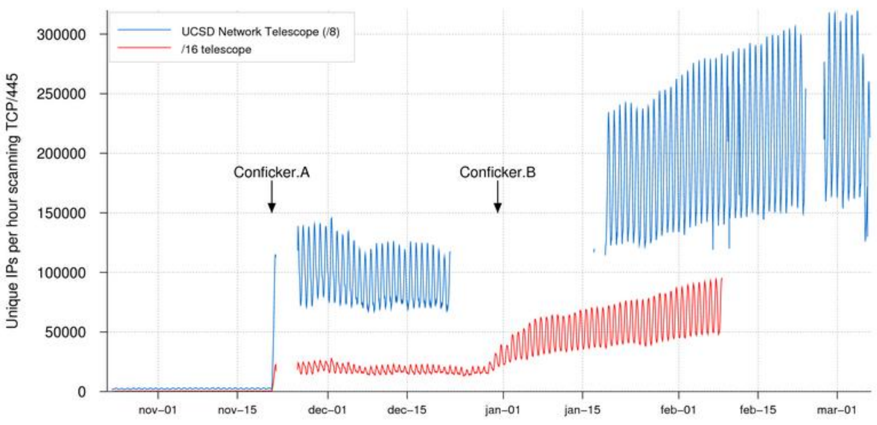


Figure 1.1: Unique source IP addresses scanning on TCP/445(hourly)[5].

According to Emile Aben two network telescopes from CAIDA started recording scanning of TCP/445 on the 23rd of October 2008, at which point around 1000-2000 unique source IP addresses where detected per hour[5]. A sharp increase in scanning was observed at midnight on the 21st of November UTC. From this point onwards the level of scanning remained high as Conficker had officially taken hold. This example illustrated how useful the analysis of un-solicited traffic from network telescopes had been in detecting the date of origin for Conficker.A and Conficker.B variants.

Traffic as a result of DDoS attacks can also be extracted from network telescope traffic, this type of traffic is often referred to as backscatter[27]. Backscatter from denial of service attacks are often received from the target of the attack, an example of this would be SYN/ACK reply packets received after a SYN flood. During the early morning hours of December the 10, 2003 the UCSD Network Telescope started receiving backscatter traffic from the SCO group[17]. This indicated the start of a DDoS attack at which point approximately 34,000 packets per second where being directed towards the SCO's webserver.

On the 11th of December the SCO group's FTP server became the primary target for the attack. During the morning of the 11th, the web and FTP servers combined were receiving over 50,000 packets per second from a SYN flood. Replies from the SCO group's web and FTP servers from the resulting SYN flood were sent to spoofed IP addresses that had been forged by the attackers. The replies from the DDoS attack were as a result detected by the network telescopes whenever the addresses fell into the same IP address range as the network telescopes. The DDoS attack against the SCO group is illustrated in Figure 1.2, the values shown in the graph were calculated using a backscatter analysis technique[27].

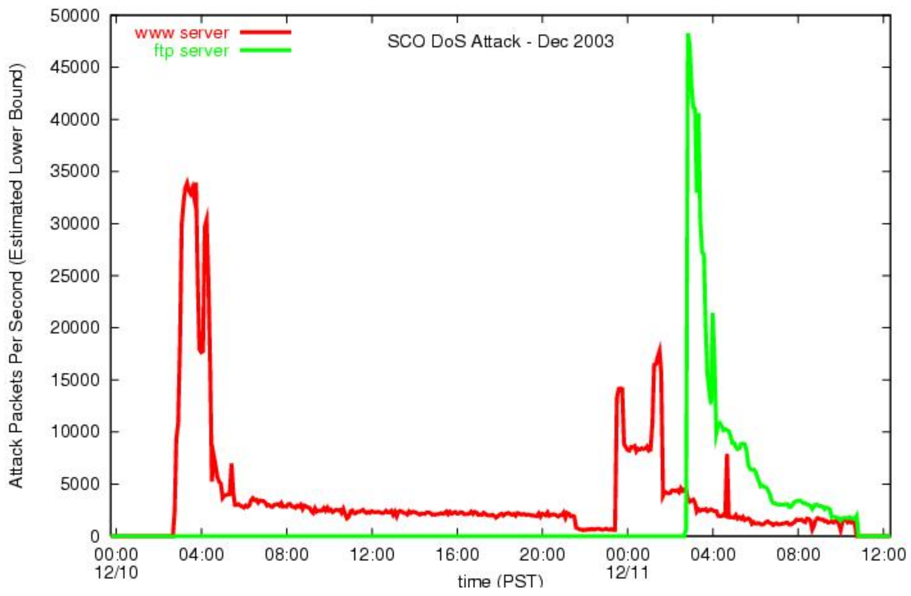


Figure 1.2: Estimated attack packets per second every 4 hours[17].

This real word example of detecting a DDoS attack by analysing network telescope traffic further motivates the importance of network telescopes and information visualisation in modelling malicious activity on the internet.

The research in this thesis is concerned with the development of a visualisation framework and its application in the creation of a network telescope dashboard that can be used to help detect and visualise events similar to the Conficker outbreak and SCO denial of service attack.

1.2 Research Outcomes

The main goals of this thesis are now summarised.

Development of a Visualisation Framework: The Ember framework will be developed in an attempt to simplify the process of creating and maintaining multiple visualisations. These visualisation must also be able to make use of multiple data sets. The framework needs to be able capable of maintaining state over various charts, allow users to navigating multiple charts and provide additional information regarding charts. Most importantly the Ember framework needs to be designed in such a manner that allows it to be easily integrated with other projects and expandable with minimum effort.

Create a Network Telescope Dashboard: A dashboard needs to be created for the Rhodes University network telescope (RUscope). The dashboard must be capable of analysing network telescope traffic and visualising the results. It will need to employ a form of data cache as a high latency is likely to exist when querying a large network telescope traffic database. The RUscope dashboard should also be able to visualise and assist in the discovery of DDoS attacks and the monitoring of Internet based worm activity.

Simplified Traffic Analysis Metrics: For use with the RUscope dashboard, simplified traffic metrics need to be defined. These metrics will need to be accurate enough to return useable results that would be sufficient for a proof-of-concept dashboard.

1.3 Organisation of Thesis

Chapter 2 Discusses related work in the field or network telescopes, un-solicited traffic, data visualisation, metrics and traffic analysis. It provides an overview and background information relating to the research conducted in this thesis.

chapter 3 Provides information on the development of an information visualisation framework called Ember. It explains the various components that make up Ember and details how the Ember framework could be used.

chapter 4 Makes use of the Ember framework in the development of a network telescope that will monitor traffic from the Rhodes University network telescope. The network telescope dashboard was called the RUscope Dashboard.

chapter 5 This chapter details a case study were the RUscope Dashboard was used, details regarding the traffic analysis used in the dashboard is provided here. A discussion after using the RUscope Dashboard to analyse and visualise traffic is also provided.

chapter 6 Lists possible future extensions for the Ember framework and concludes the research that has been conducted.

Chapter 2

Background and Related Work

An information visualisation framework is responsible for the transformation of data into useful information and then visualising to convey information and understanding. The information is obtained by making use of specific metrics and using them to extract information from a dataset. Once the data has been analysed, it needs to be made use of in an efficient manner. By visualisation the information one is able to more easily interpret and understand what it represents.

This research is concerned with the construction of such an information visualisation framework and its application in a proof of concept dashboard. The datasets that will be used by the dashboard will come from traffic captured by a network telescope, as such it is important to understand how a network telescope works and how the environment it operates in contributes to the data it observes. Further concepts that surround the information visualisation framework such as metrics and traffic analysis also need to be explored in detail.

This chapter will address past research in the fields of network telescopes, un-solicited traffic, network traffic analysis, metrics and information visualisation. The remainder of this chapter is structured as follows:

- Section 2.1 starts by introducing the concept of network telescopes as a means to gather data from nefarious and potentially malicious traffic found on the internet. The section will also differentiate between the different types of network telescopes and provide an overview of similar technologies.

- Section 2.2 will explore the underlying architecture and configuration of network telescopes. The resource requirements associated with network telescopes are also addresses in this section.
- Section 2.3 will address problems that face network telescopes and influence their ability to monitor unsolicited traffic. A brief overview of ethical considerations with regard to network telescopes is provided at the end of the section.
- Section 2.4 looks at some of the major network telescopes that are currently in operation as well as introducing their operators and some of their contributions to the field.
- Section 2.5 explains how the probability of observing given packet or event on a network telescope can be calculated according to the size of the network telescopes address range.
- Section 2.6 explores the different techniques analysing network telescope traffic for the detection of DDoS events and worm propagation. This section also introduces the topic of metrics.
- Section 2.7 formally defines metrics and introduces basic statistical methods that can be used to analyse and manipulate data.
- Section 2.8 introduces the topic of information dashboards and visualisation.
- Section 2.9 serves as a chapter summary.

2.1 Network Telescopes

Network telescopes are also known as darknets or blackholes, the "dark" or "black" referring to the address space that is empty and thus not in use by devices. The basic setup for a network telescope is to have a server to which traffic that would normally be destined for un-used address space is forwarded. Different configurations exist for network telescopes, some respond to incoming traffic (Active Telescopes) such as the IMS which makes use of a lightweight responder[7] and others simple capture all traffic forwarded to them (Passive Telescopes). Similar technologies also exist such as honeypots which attempt to lure malicious traffic and intrusion detection systems (IDS's) that monitor live network traffic.

A network telescope ability to capture traffic destined for unused address space results in it receive no legitimate traffic with the exception of traffic caused by miss configured hardware. The traffic captured by the network telescope is thus highly suitable for providing us with information regarding nefarious traffic such as that caused by denial of service attacks[27] and the automated propagation of internet based worms and viruses[21]. As mentioned before in Chapter 1, a network telescope operates on unused address space and not a live network coupled with legitimate hosts sending traffic. This means there is no need to filter legitimate traffic and all traffic received may be considered un-solicited and potentially malicious.

When looking at traffic observed by a network telescope various research papers refer to the term "backscatter"[6][21][27] which refers to residual traffic observed from other hosts that may have been the target of distributed denial of service attacks and are responding to spoofed source addresses. Other traffic such as network scanning from worms and malicious users also amount to backscatter while a very small portion of backscatter is the result of miss-configured hardware[6].

There are two main categories for network telescopes, namely active and passive. These categories refer to the type of traffic that a network telescope is able to capture. However a passively configured network telescopes is still able to capture initial active packets, it is however unable to respond to these packets and thus incapable of capturing any further traffic after the initial message. Active traffic is considered as any type of packet that tries to solicit a response. This includes packets that are used for port scans as well as TCP SYN packets which try to initiate a communication session. Passive traffic on the other hand is sent to the network telescope and requires no further communication, this includes traffic as the result of a DDoS attacks and initial traffic from most UDP based worms. Active and Passive network telescopes will be discussed in more detail in the following two sections.

Honeypots and intrusion detection systems are similar to network telescopes as they to capture potentially malicious traffic. Honeypots are however more pro-active than network telescopes and often advertise themselves as vulnerable in order to attract the attention of malicious entities. Intrusion detection systems may be used to monitor traffic on a live network for known bad signatures of malicious traffic. Much research[8][14][33] has gone into the collaboration and aggregation of these various technologies to produce a more holistic view of nefarious internet topology.

2.1.1 Passive Network Telescopes

The paper "Inferring Internet Denial-of-Service Activity" by Moore et al.[27] shows how it is possible to infer Denial-of-Service (DoS) activity from a passive network telescopes. A passive network telescope is only capable of receiving incoming traffic and has no means of responding to any packets. The lack of response capability means the telescope is unable to complete the 3-way TCP handshake required to receive TCP payloads, this however is not required to infer some DoS attacks as will be shown in section 2.6.2. Passive network telescopes can however collect payload data from UDP and ICMP packets as they do not require active responses[14] and contain a wealth of information in the initial packet.

It has been shown that worm and virus attacks may also be inferred using a passive network telescope, the paper "Observing Internet Worm and Virus attacks with a Small Network Telescope" by Harder et al.[21] examines different methods of analysing worm and virus traffic on a small network telescope. Alternative methods also exist to detect the spread of malware such as using "honeypots" that attempt to attract malicious traffic and are able to respond to and in some instances capture[3]malware[31].

While passively configured network telescopes are unable to record TCP based exploit data or details of miss configured application requests[6], they are still capable of detecting source addresses and packet header information. Passive network telescopes are also able to record earlier worms such as Witty[32] and Slammer[26] which propagated over UDP and were able to be delivered to their target via a single packets payload.

2.1.2 Active Network Telescopes

Actively configured network telescopes are capable of issuing responses to certain requests, by doing so they are able to receive application level data that may lead to a better understanding of an exploit attempt. For example a telescope might be configured with a responder to increase its level of interactivity with traffic. A responder would reply to a TCP SYN request with a TCP SYN-ACK packet and in so doing receive the first packet containing a data payload[6]. This payload could contain enough information to accurately identify a threat such as the blaster worm[37].

The Internet Motion Sensor (IMS)[7] project is an example of a distributed darknet monitoring system that makes use of a lightweight responder to elicit the initial packet of each TCP connection. This allows the IMS to retrieve more information than a passively configured network telescope, by implementing a very simple stateless TCP responder. The IMS also employs a novel payload storage technique, for each packet it receives it creates a MD5 checksum of the payload[7]. The checksum is then compared to a signature database for that day, if the signature does not exist in the database (it's new) the payload is stored and the signature added to the database[14]. If the signature had already been captured that day it, the signature is logged but the payload discarded. This catching technique allows the IMS to reduce its storage requirements but does not hinder its ability to monitor new payloads or payload frequency.

2.1.3 Honeypots and Honeyfarms

Honeypots are very similar in nature to network telescopes, they are however at the other end of the spectrum with regards to capturing nefarious traffic. This is because honeypots emulate a vulnerable service or host in order to attract malicious traffic. They can be used to monitor individual addresses or function over a range of addresses[3].

One of the key differences between a honeypot and a network telescope is the resource requirements and the method of capturing potentially malicious traffic. Honeypots interact at greater depths with a threat (such as a worm) and is thus more resource intensive. Honeypots also attract or encourage malicious traffic as opposed to network telescopes that just listen for any traffic directed towards it. This allows honeypots to characterize the vulnerability that's has been exploited and its affect on a machine[8].

Honeypots can be deployed in many different flavors, a single physical host could act as a high interaction honeypot[36] alternatively that same host could emulate 10 virtual honeypots[31] thus creating a virtual honeynet or honeyfarm. Honeypots may also be categorized as either a low or a high interaction honeypot. This refers to the level of interaction allowed with a honeypot, low interaction honeypot might only emulate venerable services, while a high interaction honeypot might be a complete physical machine with a vulnerable operating system. Banks of honeypots are sometimes used to handle traffic that was initially detected by network telescopes, filtered and deemed important enough for further analysis[8].

By analyzing the data captured from a honeypot it is possible to learn more about attack patterns such as how an attacker might elevate their privileges to gain root access on a host. Other information could also be determined like what type of attacks are currently the most prevalent, which ports they are targeting and which services they try to exploit.

Lastly honeypots such as Nepthenses[3] are used to collect worms and could provide insight to their spread and underlying architecture.

2.1.4 Intrusion Detection Systems

Intrusion detection systems provide automated monitoring and analysis of events that attempt to "compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer or network"[25]. Intrusion detection systems employ one of two methods to detect threats, the first makes use of known signatures of "bad" events, it compares traffic patterns to these signatures and logs the event if the traffic matches a known signature[25]. The second method involves traffic patterns and the deviation there of, when anomalies in traffic are detected the second method logs the event. Anomaly detection is a topic still undergoing further research and is employed in only a few intrusion detection systems in a limited form[25]. While intrusion detection systems are effective at identifying known threats, it comes at a price of resources as the traffic comparison against known signatures involves considerable resources especially as the signature database grows.

2.1.5 Hybrid Systems

A passive network telescope is not able to respond to requests, which might be needed to identify certain threats. It should be noted that even the use of an active network telescope might not guarantee that enough information is captured. While a distributed collection of network telescopes would be able to register global events, they lack the ability of assessing how exactly the threat is spreading. On the other side of the spectrum honeypots, antivirus software and intrusion detection systems posses the ability of collecting[3] or at least providing more detailed information on a threat. However due to the relatively small scale of addresses that they operate on, they are incapable of interpreting events on a global scale, such as those required to identify the early growth stages of a worm[8].

For this reason hybrid architectures have been developed that take advantage of the scale provided by network telescopes and the detailed threat interpretation provided by honeypots. A particular system explained by[8] provides a method whereby a collection of distributed network telescopes are used to track initial threats. The network telescope architecture then uses packet filtering techniques too select particular sources of the traffic that is then routed from the network telescopes to honeyfarms. These honeyfarms consist of multiple honeypots that are then able to interpret the threats at a lower level[8]. While this thesis's current research is not concerned with the combination of honeypots and network telescopes it is a field of notable importance and may one day be integrated with the information visualisation framework that has been developed. This integration between information visualisation and multiple data and analysis resources could produce a tool capable of very accurately detecting and modeling new threats.

2.2 Network Telescope Architecture and Configuration

Multiple considerations need to be addressed when configuring a network telescope, for instance the method of forwarding traffic to the telescope that would have been destined for un-used address space needs to be decided upon. Two common formats that are used to collect telescope traffic include pcap[1] and NetFlow[6], these formats record raw packet data that will need to be processed before analysis.

Storage and processing is also an important factor, as traffic that has been captured needs to undergo filtering to extract the desired packet information and then needs space to be stored for later analysis. It would be inefficient to analyse entire raw packets as they contain more information than would be used, instead we apply a filter to the packets and insert they attributes we wish to keep into a database for further analysis. An example of database used to store network telescope traffic is show in Figure 4.4 which can be found in section 4.3.

2.2.1 Packet Forwarding

When dealing with a small network telescope such as one capturing traffic from only a few IP addresses, Bailey et al.[6] suggests configuring the network telescope to send ARP replies to the router for each un-used address. This however is not scalable for monitoring large blocks of un-used address space and may be improved by configuring an upstream router to statically route entire address blocks to the network telescope. This requires an

[1]pcap is a packet capture API implemented in libcap and WinPcap, http://www.tcpdump.org/

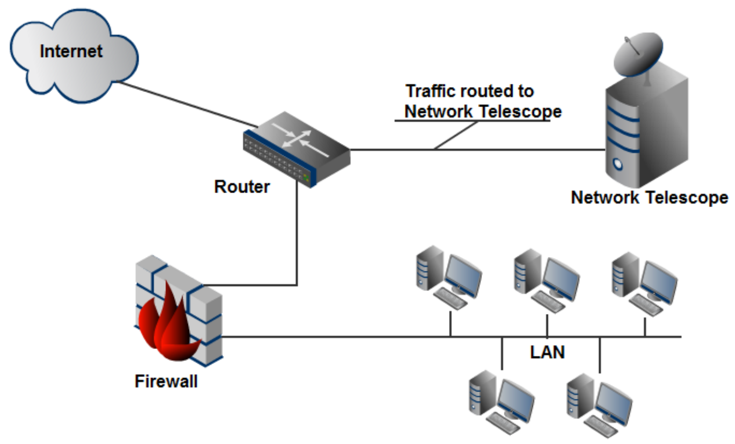


Figure 2.1: A network diagram illustrating a router that has been configured to route traffic to a network telescope.

entire address block to be dedicated to the network telescope and while this is easiest it might not be ideal, for a more flexible solution[6] suggests routing all packets that would have been dropped by the router to the network telescope. With the above examples it is assumed that the un-used IP addresses are in fact globally addressable and reachable. There are however methods that allow monitoring of unused non-routable addresses[13], such as those found inside service providers and large organisations internal networks. Figure 2.1 illustrates a typical configuration for a network telescope.

2.2.2 Storage, Filtering and Bandwidth Requirements

The once a network telescope has captured traffic, that traffic needs to undergo a filtering process by which information pertaining to whichever study is being done is extracted from the packets. This information then needs to be stored into a database for later analysis. For instance, if a study only requires source and destination ports, there is no need to insert packet header and address information the database. Once the packets have been filtered, they need to be stored, storage provisions need to be made and storage capacity needs to be decided on.

The storage requirements of network telescopes is dependent on the amount of traffic that will be received, Bailey et al.[6] has found that on a /24 sensor the average traffic rate can be approximated to 9 packets per second, while a /16 sensor would receive roughly 75 packets per second and a /8 sensor around 5000 packets per second. The average packet size from the Rhodes University Network telescope which contains just over 40million packets is 101 bytes, using the average packet size and estimated packet arrival rate according to sensor size one is able to make a approximation of the size of storage that will be required. Average bandwidth requirements according to Bailey et al.[6] is displayed in the table below.

Table 2.1: Suggested bandwidth requirements according to network telescope sensor size[7]

Sensor Size	Bandwidth Requirements
/24	7 Kbps
/16	60 Kbps
/8	4Mbps

The bandwidth requirements and packet arrival rates shown in Table2.1 were calculated from data collected by the globally deployed Internet Motion Sensor (IMS)[7] which is a distributed darknet monitoring system and will be formally introduced in section 2.4.3.

2.3 Problems Facing Network Telescopes

Due to the inherently distributed architecture of the internet there is always a chance that traffic does not end up where it should be. Routers may fail, DNS services may be miss configured and firewalls might drop legal traffic. These are just a few examples of the difficulties that traffic traversing the internet might encounter. The public nature of the internet also means that any host may attempt to send data to any other accessible host on the internet, this intern could result in the poising of network traffic. It has also been shown by Cook et al.[14] that the traffic collected at individual address blocks monitored by network telescopes may vary greatly from other address blocks that exist on separate IP ranges.

This section will highlight some of these inconsistencies encountered during traffic analysis and why certain considerations should always be kept in mind when conducting research with network telescope traffic. It will start by introducing the topic of network telescope placement and the variance of results obtained by monitoring geographically displaced network telescopes that are part of the IMS project. A brief introduction to internet topology and architecture will follow to explain some of the problems encountered during traffic routing from source to destination. The topic of result poisoning follows which is one of the key reasons that telescope specific data such as actual IP addresses should never be published in research. Lastly the section will highlight ethical considerations pertaining to traffic captured by network telescopes and similar technologies such as honeypots.

2.3.1 Placement of Network Telescopes

Although the Internet threats observed by network telescopes such as DDoS and worm propagation are globally scoped, data from the IMS indicate widely different trends between separate network telescopes[14]. These differences where noted across three dimensions namely, over all protocols and services, a specific protocol and port and lastly signatures of known worms. Another publication[33] shares this view that multiple points of monitoring are required coupled with a collective interpretation to provide a more comprehensive view of nefarious network traffic. Thus observing traffic from only a single point would provide very little, if any information about the background activities[33]. Contrasting this view there are however still important information that can be learnt from a single vantage point,[21] showed how a small /24 telescope was used to identify and distinguish between port scans, host scans and DDoS attacks. While the results found in[21] might not correlate strongly with global findings, their findings were still useful in understanding current threats.

2.3.2 Internet Topology

The Internet is dependent on an ever expanding, interconnecting network of physical devices such as routers and switches, which are ultimately responsible for getting packets from point A to point B. These devices however can be overcome by traffic and the result of which is a loss of packets. Packets delivery from point A to point B can be slowed down by processing delays, queuing delays, transmission delays and propagation delays[23]. These delays can obscure the actual arrival rate of packets that may biased some of the results obtained from network telescope traffic analysis. Packet loss can also be a serious problem, in the event of a large scale Distributed Denial of Service attack routers may queue packets and eventually drop[27] packets if the queue becomes too long. A good description of the vulnerability of traffic on the internet is presented by[23] as follows "unfortunate that the physical laws of reality introduce delay and loss as well as constrain throughput." Miss configured of hardware and software can result in arbitrary but legitimate packets ending up in a network telescope, an example of this could be a NetBIOS configuration that sends small numbers of unsolicited packets to a monitored address range[27].

2.3.3 Result Poisoning

Due to the nature information that can be learnt from the analysis of network traffic, it would be in the best interest of certain nefarious entities to try and obscure the data obtained from network telescopes. According to Shinoda, Y., Ikai, K., Itoh, M.[33] "Knowledge of a monitor's sensor location can severely reduce its functionality as the captured data may have been tampered with and can no longer be trusted. " For this reason findings obtained from network telescope traffic should never include actual address ranges used by the networks telescopes, delays should be applied to results and results aggregated where possible. The power of network telescopes lies in their capability to collect traffic from sources that believe they are sending data to legitimate hosts.

2.3.4 Ethical Considerations

Depending on the configuration of network telescopes and honeypot technology detailed information may be extracted from the traffic they obtain. Coupling the information that could be extracted and the underlying internet architecture that traffic depends on to reach its information there is always a chance that legitimate traffic might be captured. Even knowledge of illegitimate traffic might harm certain entities such as the naming of organisations that have been under attack from DDoS attacks or ISPs that have not put in place corrective measures such as ingress filtering[2] to hinder DDoS attacks. As such service providers and content providers might consider this type of information confidential[27]. Care should always be taken in the publication of findings and the collection of data. This is also brings to light one of the advantages of passively configured network telescopes, by design they are unable to see TCP payloads. This means that passively configured network telescopes do not infringe on privacy with regard to the majority of internet communication.

2.4 Existing Network Telescopes and Operators

As a result of the importance and usefulness of network telescopes, various projects and collaborative efforts have developed to help manage the knowledge base and steer research in the field of network traffic analysis. This section of the chapter will introduce some of

[2]Ingress Filtering - Defeating Denial of Service, http://www.faqs.org/rfcs/rfc2827.html

the big contributions to the field, not by specific authors but rather by collective effort of many. CAIDA is introduced first and followed by Team Cymru and and the Internet Motion Sensor project. Lastly a brief introduction to the Rhodes University Network Telescope is given.

2.4.1 CAIDA

CAIDA, the Cooperative Association for Internet Data Analysis is a "collaborative undertaking among organizations in commercial, government, and research sectors aimed at promoting greater cooperation in the engineering and maintenance of robust, scalable global internet infrastructure.". Amongst their various contributions they also supply data sets, produced by monitoring locations in several large Internet Service Providers (ISPs) and several other network telescopes to produce large datasets of network traffic. CAIDA also produces various important reference material that was used in the thesis such as the "Network Telescopes: Technical Report"[28].

2.4.2 Team Cymru

Team Cymru[3] is a non-profit internet security research firm that is dedicated to making the internet more secure. They provide in-depth information regarding setting up and maintain network telescopes. In addition to network telescope construction and use they provide secondary services that may be used during analysis of captured traffic, such as a malware hash registry[4] that provides a look up service for captured malware.

2.4.3 IMS

The Internet Motion Sensor project that is run by the university of Michigan consists of a heterogeneous set of sensors and data aggregators which can be divided into two main categories[14]. Blackhole sensors (network telescopes) that collect treat data and topology sensors that provide context regarding the data collected by the blackhole sensors. While the blackhole sensors passively collect traffic over ranges of IP address space they include an active component that responds to so TCP SYN requests in an attempt to illicit more data[14].The architecture employed by the IMS is shown in Figure 2.2

[3]Team Cymru - http://www.team-cymru.org/
[4]MHR - http://www.team-cymru.org/Services/MHR/

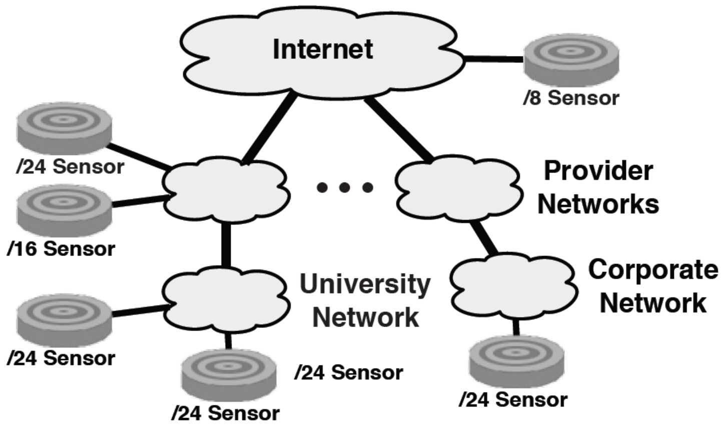


Figure 2.2: Internet Motion Sensor architecture[7].

The IMS's novel method of traffic storage by use of filtering, hashing and categorising has already been explored in section 2.1.2.

2.4.4 Rhodes University Network Telescope

The Rhodes University network telescope (RUscope) is the primary provider of captured network traffic used for this thesis. The network telescope monitors a /24 address block (traditionally known as a class C) and is passively configured so that it only captures traffic targeted towards it and does not respond to any requests. After packets are filtered they are inserted into a Postgres[5] database for further analysis such as that achieved by this project. The network telescope was established on in August 2005 and has captured approximately 65 million packets. The network telescope dashboard created during this research was used to monitor traffic from RUscope and is detailed in chapter 4.

2.5 Event Probability

The amount of traffic a network telescope observes is proportional to the size of the address block it is monitoring. The CAIDA "Network Telescopes: Technical Report"[28] refers to the analogy of network telescopes as astronomical telescopes and explains how "having a larger "size" (fraction of address space or telescope aperture) increases the quantity

[5]PostgreSQL - http://www.postgresql.org/

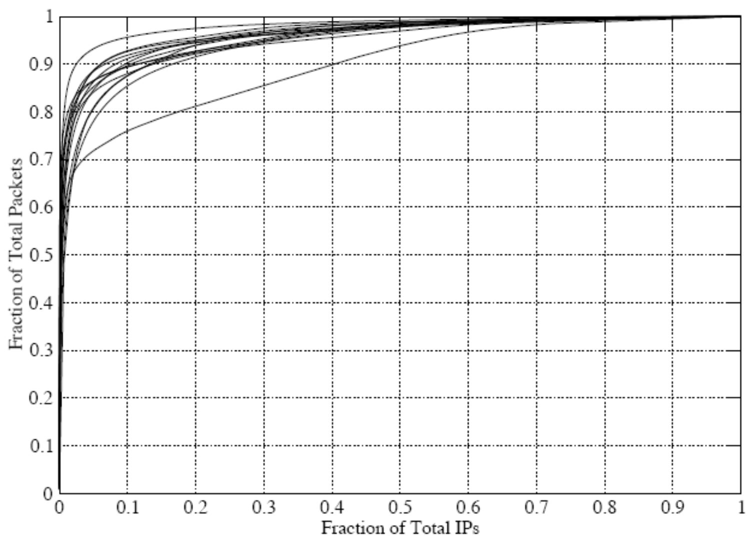


Figure 2.3: The contribution of individual IP to the total number of packets as seen at 14 darknets. Over 90% of the packets are from 10% of the source IP addresses[8].

of basic data available for processing"[28]. The size of the address block or 'aperture' also influences the probability of that network telescope observing a given event. When looking at the Internet Protocol version 4 (IPv4) address space which allows a host a 32 bit address, there are a total of 2^32 possible IP addresses. Address blocks are commonly assigned according to the number of leading bits that uniquely identify that address, a /8 address block would describe a range of 2^24 addresses who all share the first 8 bits of their IP address. A /32 would then describe a single host, the probability of a network telescope monitoring a unique host in IPv4 address space is thus given by $p(x) = 1/2^x$ where p is the probability of monitoring the host and x is the size of the address block. For a telescope monitoring a /8 network, $p8 = 1/2^8 = 1/256$ which means the telescope has a 0.39% chance of observing a packet from a single unique host. A formula for calculating the probability of 1 of m packets being observed can then be calculated by $E(X) = nm/2^32$ where n is the number of unique addresses being monitored[27].

In conjunction with event probabilities it should be noted that there appears to be a disproportionate distribution between traffic observed and source address distribution. Research by Bailey et al.[8] found that nearly 90% off all traffic observed by network telescopes were sent by less than 10% of the total observed source addresses from each of 14 network telescopes. The distribution is illustrated in Figure 2.3.

It has also been shown that the traffic observed by a network telescope does correlate to its locality in IPv4 address space, which results in different network telescopes observing separate and repeated events[6][8]. The concept of locality and traffic was explored in greater detail in section 2.3.1.

2.6 Analysis of Network Telescope Traffic

Traffic observed by network telescopes could be explained as either a miss configuration of host hardware or software, backscatter produced by spoofed source addresses (most likely the event of a Distributed Denial of Service (DDoS) attack), scanning from worms or other types of probing[14]. The raw packet data obtained by network telescopes would be of little use if there were no method to distinguish the specific threats. There are however various techniques[27][21] that are used to infer these types of traffic and they will be the introduced in this section. By understanding the traffic and how it is generated it become easier to identify it.

2.6.1 Distributed Denial of Service Attacks

The purpose of Denial of Service(DoS) attacks is to consume the resource of its target, that being a host or network[27]. The result of which denies legitimate users access to that resource. DoS attacks may be divided into two main categories, that of logic attacks and that of flooding attacks. Logic attacks such as the "*Ping of Death*" exploit existing software vulnerabilities to crash or severely downgrade the service/availability of a remote server[27]. Logic attacks are often executed by sending a few well crafted packets to a vulnerable operating system or application and if the correct combination of packets is sent the target service or host could stop or crash[23]. Flooding attacks on the other hand focus on consuming as much of the targets CPU, memory or network resources. This is achieved by sending large numbers of spurious requests and botnets are often used for this purpose. Flooding attacks may be rather easily achieved by sending large volumes of small packets as quickly as possible as this can overwhelm routers and NICs packet processing capabilities. DDoS attacks create backscatter traffic that could be directed to a network network telescope's address range, this traffic is passive.

One of the best known DoS attacks is that of the SYN flood, the SYN flood is a type of flooding attack that aims at immobilising a server by initiating as many TCP connections with the target as possible. For every SYN request packet the victim receives, it has to process that packet by going through a list of connections, if no match is found resources need to be allocated to the new connection. According to Moore et al.[27] "*even a small SYN flood can overwhelm a remote host.*" This shows how a single host may cause significant damage to a target, however often nefarious agents on the internet with make use of multiple hosts to perform attacks. Attackers compromise multiple hosts and leverage their combined bandwidth and processing power to mount more powerful attacks. These compromised hosts are referred to as "*zombie*" hosts and together make up is known as a botnet. An example of a SYN flood is illustrated in more detail in Figure 5.2 found in chapter 5.

In an attempt to conceal their location and create multiple connections in the case of a SYN flood, attackers forge or "spoof" the source addresses of each packet they send[27]. Due to the "*spoofed*" nature of source addresses, they are also the cause of backscatter detected by a network telescope. During a DDoS attack, the victim attempts to send SYN-ACK or RST[29] replies to the "*spoofed*" source addresses and if these addresses happen to fall into the same address range of a network telescope they are observed. Targeted hosts are not the only cause of backscatter from DDoS attacks, occasionally network devices between the "*spoofed*" address and the target send their own ICPM messages[35] to the spoofed address. Table 2.2 illustrates common packet requests and their responses.

Table 2.2: Packets requests and responses[27]

Packet Sent	Response from victim
TCP SYN (to open port)	TCP SYN/ACK
TCP SYN (to closed port)	TCP RST (ACK)
TCP ACK	TCP RST (ACK)
TCP RST	No response
TCP NULL	TCP RST (ACK)
ICMP Echo request	ICMP Echo reply
ICMP TS request	ICMP TS reply
UDP pkt (to open port)	protocal dependent
UDP pkt (to closed port)	ICMP Port Uncreachable

2.6.2 Inferring DDoS Attacks

Before attempting to infer DoS traffic one needs to make the following assumptions. Addresses spoofed by attackers need to be uniformly distributed across the entire IP address space, that is the attacker needs to spoof the IP addresses at random. This assumption is often effected by ISP's that employ ingress filtering[18]. Ingress filtering monitors source addresses and drops packets with source addresses outside its client address range. This can cause neither that all packets may arrive at their target nor that the IP addresses are uniformly distributed. It's possible to check if a set of observed addresses are uniform to a network telescope range by calculating the Anderson-Darling(A2) test[16]. However as discussed earlier in section 2.3 and 2.5, distributed network telescope results vary from different address ranges. Although Moore et al.[27] still affirms that if the distribution of source addresses is not random, then it would be impossible to calculate an un-biased attack rate from the arrival rate.

The next assumption is that of reliable delivery, all attack traffic is assumed to have been delivered to a victim and all backscatter to the network telescope without issue[27], included in this assumption is that all packets elicit a response. These statistics are needed to correctly interpret the rate of packet arrival and estimate the size of an attack. Problems with this assumption include the volatile topology of the internet as discussed in section 2.3.2. Again ingress filtering may hinder the packet arrival as could intrusion detection software and firewalls that rate limit the packet arrival. These issues could result in the under-estimation of results.

According to Moore et al.[27] the last assumption needed is that all unsolicited packets received by the network telescope represent backscatter. Any host on the internet is free to send packets to any other host, this includes host addresses in a network telescope's range. Another issue that faces the last assumption is that of nefarious agents and the poisoning of telescope results as introduced by section 2.3. These assumptions where used by Moore et al.[27] to infer DoS attack traffic from network telescope data. They go on to say that their approach "*provides at worst a conservative estimation of current denial-of-service activity*", as would be expected from the uncertainty presented by traffic delivery on the internet.

2.6.3 DDoS Attack Classification and Metrics

Due to the vast number of approaches one could take to analyse network telescope traffic, Moore et al.[27] has decided to focus on two classifications of traffic, flow-based and event-based. By categorising denial of service attacks within one of these two categories, Moore et al.[27] is able to determine the severity of attacks on short time scales with event-based. While making use of flow based classification it is possible to learn the following metrics; how many, how long and what kind[27].

A flow was defined by Moore et al.[27] as "*a series of consecutive packets sharing the same target IP address and IP protocol.*" A flow is then considered to exist from the 1st packet received until the last packet received within say a 5 minute gap from the second last packet to be received, this timeout variable can affect end results. To ignore insignificant backscatter any flow with less than a 100 packets or minimum duration of 60 seconds may be ignored. Flows also need to contain packets that have been sent to multiple addresses in the network telescopes range[27]. The following data was used by Moore et al.[27] for flow based analysis:

- TCP flags, to determine what flows consist of.
- ICMP payload, from ICMP packets with TTL expired.
- Address uniformity, whether they pass the A2 test and are uniformly distributed.
- Port Settings, whether port range is fixed, for A2 uniformity test.
- DNS information, DNS address of the victims source address.
- Routing information, prefix mask and origin according to a local BGP table.

Event-based classification makes use of the victims IP address and notes details over a fixed time window to examine time-domain qualities[27]. These qualities include the distribution of attack rates or simultaneous attacks. The event-based analysis was achieved by dividing the captured traffic into one minute intervals and noting the each attack event during that one minute[27]. Attack events consisted of a victim sending at least ten backscatter packets to the network telescope in that minute.

These two methods, flow and event categories are just two examples of classification used during the analysis of network telescope traffic. Many other metrics may be extracted with different means. Arrival times discrepancy is introduced by Harder et al.[21], they show that inter arrival time of backscatter and normal traffic differ due to "*distinctive peaks around 10 and 120 milliseconds are missing*"[21].

2.6.4 Worm Propogation

Worms operate by spreading over a network to different hosts by exploiting vulnerabilities in the host operating system or in application level software[43], exploiting there vulnerabilities allows the worm to execute code and become self replicating. Worms scan ranges of network addresses for vulnerable hosts that they can exploit and in so doing are able to propagate and spread by themselves.

At any time there is a steady flow of background traffic on the internet caused by worms and viruses this traffic flow also includes port scans and backscatter from DDoS attacks[21]. This traffic will be visible to any routable, un-firewalled host connected to the internet. Furthermore it has been shown that many worm propagation (target selection) strategies are biased towards local addresses[43]. By targeting local addresses the worm is able spread at a faster rate, due to smaller network distances, common administrative practices and also exploitation of an already breached firewall. Examples of such worms include Code Red II[10], Nimda[11] and Blaster[12].

The existence of this never ceasing background traffic caused by worms does bring concern, that this traffic can pose a serious threat to the usability of the internet and do in fact cause instability during their epidemic stages. With increasing bandwidth speeds[44] and more devices being connected to the internet the platform for worms to spread is increasing. The main purpose of network telescopes when it comes to the observation of worms is defined by their configuration. Passive network telescopes only record packet traffic from worms and do not respond to any requests, their port scans can be detected and a good visual example of detected port scans are illustrated by van Riel, J.P., Irwin, B.[42] with the InetVis research. Actively configured network telescopes on the other hand are able to respond to some worm requests to obtain additional information and as such are able to identify TCP and UDP based worms.

Some concepts and terminology surrounding computer based worms include: worm virulence refers to the extent to which a worm generates traffic and which paths (routers) become most congested by it[7]. Worm demographics refer to the number of hosts infected the geographic and topological placement of hosts. The worm demographics also refer to infected host attributes such as operating systems, available bandwidth and application level software that was exploited (if any). Figure 2.4 shows a graph indicating the three phases of the Blaster worm life cycle.

2.7 Security Metrics

Security Metrics can be described as metrics derived by comparison of two or more measurements that had been taken over a time to a predetermined baseline[30]. Metrics are generated from analysis, as opposed to measurements which are generated by counting.

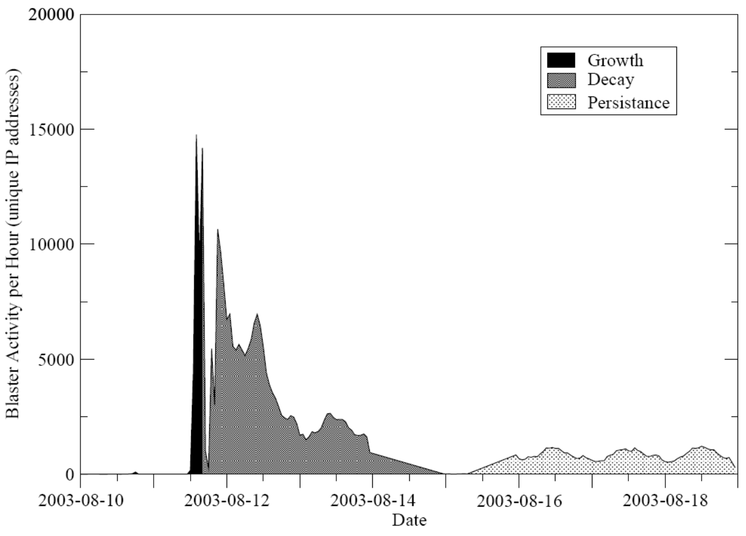


Figure 2.4: A snapshot of Blaster worm showing the three phases of the worm lifecycle[7].

Jaquith extends this definition by abstracting Security Metrics to *"an emerging field of study"* that is *"Drawing from examples in fields like manufacturing, logistics and finance"* he finished his definition by stating that Security Metrics *"put numbers around activities that safeguard information resources"*.

Metrics quantify particular characteristic of data to facilitate insight in the chosen subject area. Good metrics should be consistently measureable, repeatable, specific, cheap to gather, expressed as a cardinal number or percentage and have at least one unit of measurement[22][30]. Security metrics thus determine what information one is interested in learning and providing empirical data and understanding on. The outputs of which should provide useful information and interpretation that can be used in conjunction with risk analysis and threat mitigation. Network telescopes are used to gather data and if there is any consideration for obtaining sensible information from that data useful and practical security metrics need to be determined that will be used to analyse the data.

Raw data needs to be processed to produce information, information then needs to be analyzed to yield insight. Insight into the data is what is accomplished by using correct security metrics visualization techniques. A multitude of analytical methods exist that can be used to transform data into something that explains itself and this section will attempt to explain some of them as listed below.

- Average (Mean)
- Median

- Standard Deviation
- Grouping and aggregation
- Time series analysis

2.7.1 Average(Mean)

Calculating the average value from a dataset is a standard aggregation technique[22] used to give an overview of a data set. The average is calculated by adding all the values from a set of elements and then dividing that sum by the total number of elements. While using averages is a good method of aggregating results it does however cause problems for some data sets. Outliers are obscured and the richness of underlying data is lost. For this reason[22] states that "means are a poor choice for aggregating highly variegated data sets." he continues by adding elaborating the means have a tendency to "obscure hidden insights and often steamroll over spikes and valleys that an analyst might consider interesting." Arithmetic means are however acceptable to use with certain data sets when the scope of data represented is narrow[22].

2.7.2 Median

The median is different from the mean as it denotes a value and proportion. The median represents the number that separates the top 50% of elements from the bottom elements and is calculated by sorting a dataset in descending order and then choosing the element in the middle of the dataset. The median is often used instead of the mean, as it can give better insight to data that has strong outliers and has said to offer significant advantages[22] over means in respect to measuring performance.

2.7.3 Standard Deviation

Standard deviation reflects the dispersion of a dataset from its mean. A high standard deviation could indicate irregular or unpredictable data, while a low standard deviation would indicate a high degree of data clustering around the mean. The standard deviation is calculated by first calculating the mean for the data set. Then square the difference of each element from the mean, add together all the squared differences and divide the result by the number of elements and this produces what is known as the variance, the square root of which gives the standard deviation.

2.7.4 Grouping and Aggregation

Grouping and aggregation of data is one of the methods used to transform a large quantity of raw data into useful information[22]. Grouping refers to the organization of similar and complementing data from the same scope of analysis onto a combined unit. While aggregating refers to the calculation of summary statistics for each group of data, examples of which could include the sum, mean, standard deviation, minimum and maximum values. Grouping and aggregation can break down large data sets into meaningful "chunks" that are easily understood and might even bring to light underlying relationships between data.

2.7.5 Time Series Analysis

A very important analysis technique that is used in conjunction with network telescope traffic is that of Time Series Analysis as it is used during the inference of denial of service attacks[27], worm propagation[21] and load calculations[28]. Time series analysis is explained by[22] as attempting to understand the evolution of a dataset over time and will contain a series of observations for a particular attribute that have been taken at regular intervals.

2.7.6 Automation of Metric Calculations

By automating the generation of security metrics, it is possible to improve the aspects of repeatability and increased measurement frequency with minimal if any ongoing effort. This is achieved by automating the process of data gathering, computation and presentation. As part of this thesis is concerned with the development of a network telescope dashboard that will be capable of performing automated periodic analysis, it is important to explore the automated process of metric calculations.

Some of the benefits from automating metric calculations include added accuracy, repeatability, increased measurement frequency, reliability, transparency and audit ability[22]. Characteristics and considerations of an automated metric system includes the following.

A real time focus, this could be based in days, weeks, months[22]. If the information is represented over to fine an interval too much time will pass before it could be made use of, too large and it abstracts details. Automating the metric calculations also allow for the aggregation of aggregated results, that is observed events could be further aggregated.

2.8 Information Dashboards and Visualisation

Stephen Few defines a dashboard in *Information Dashboard Design: The Effective Visual Communication*[19] as follows:

A Dashboard is a visual display of the most important information needed to achieve one or more objectives; consolidated and arranged on a single screen so the information can be monitored at a glance.

As an integral part of this thesis is concerned with the production of an information dashboard for a network telescope it is important to outline a clear and concise definition thereof. The network telescope information dashboard will have to provide a concise and high level visual representation of relevant information derived from the analysis of the network telescope traffic.

According to Few a dashboard should be displayed only on a single screen[19], this severely limits the amount of information that could be displayed. As such great consideration should be taken to select relevant and complementing information. It has been shown in section 2.1 and 2.6 that the amount of information obtained from a network telescope could be great and as such could easily overwhelm an observer or even obfuscate important information. Thus Few explains how a dashboard should scale the available information in an attempt to provide an aggregated overview which allows relevant information to be conveyed and indicate what would require further drilling down.

The concept of an information dashboard is not new and has been used extensively in the fields of finance and management. Any field where mission critical choices depend on large combinations of constantly updated information will benefit from the use of dashboards.

The volume of data obtained from network telescopes can easily overwhelm observers and allow information to become lost within the data, to avoid this different mechanisms are used to interpret and display data. Some data can easily and very effectively be represented by purely numerical means, other information however is much better interpreted when presented in a visual manner.

The following guidelines were provided by Andrew Jaquith[22] who believes they should always be considered when generating visual representations such as graphs and charts. Graphics need to be kept simple, colours might imply extra meaning or spark biases and as such they should be avoided. Furthermore Jaquith states that empirical data should always be indicated clearly and graphs should always be observed in context of their data. It should be noted that the negative perception of colour and its ability to distract, as illustrated by Jaquith is contrasted by Few, who explains that colour can more easily help distinguish attributes such as those used in a pie graph with its legend. Both of their view points hold merit and care should be taken to choose neutral colours unless the aim is to biased or more strongly outline a certain piece of information.

By using the correct and relevant visualisation techniques during the development of a dashboard it is possible to create a concise summary of information that is easily understood and conveyed to the observer with little effort on their part.

2.9 Summary

Network telescopes have proven to be a useful tool in the capture of nefarious network traffic. The data they collect provides an indication of large and small scale network events such as worm lifecycles and denial of service attacks. By analysing raw traffic obtained from network telescopes and using reputable metrics, it is possible to extract useful and accurate information. Using that information and transforming it into a visual representation to provide insight is an invaluable asset. The purpose of a dashboard is to convey meaning and understanding at a glance. By conveying network telescope data onto a dashboard one is able to visualise potentially malicious activity and more easily identify trends. The next chapter will detail the design process of an information visualisation framework that can be used to visualise information obtained from a network telescope.

Chapter 3

Information Visualisation Framework Design

The design of the information visualisation framework involved detailed planning and requirements analysis. The framework was named Ember as it shares certain characteristics of coal embers left after a fire has burnt out. A metaphor where the flames represent data and the residual heat left in the embers lingers long after the flame is gone represents the information that is displayed and conveyed to the user.

The first steps in planning was to identify the main objectives of the framework, secondary and complementary objectives also had to be identified so that a clear design plan could be constructed that would allow for future extensions and ensure a modular design. The systems development life cycle involved rapid prototyping which helped to identify the advantages and constraints of multiple languages and libraries that would be used during development.

The rest of this chapter is outlined as follows:

- Section 3.1 provides an overview of the entire Ember framework and explains how the different components that are addressed in the chapters following the overview are used.
- Section 3.2 details the container components which are used to navigate and manage multiple charts.
- Section 3.3 explains how graph scripts are used in the Ember framework to construct charts.

- Section 3.4 looks at the Dashboard Generator class which is responsible for generating the HTML and JavaScript a browser requires to render Ember frameworks components.
- Section 3.5 addresses the two libraries that were used in the Ember framework.
- Section 3.6 serves as a chapter summary.

3.1 Framework Overview

The framework core is made up of two very important classes, the XML Parser and the Dashboard Generator Class. These classes are responsible for the generation of a dashboard as defined by an XML configuration file. This allows the Ember framework to be used to create any type of dashboard that will be able to represent information of any kind. As will be detailed in the next Chapter 4, the Ember framework was used to construct a network telescope dashboard, which would be able to monitor the status of a specific network telescopes and display analysed information. The choice of PHP allowed for rapid development and the added benefit of platform independent deployment.

The main goals of a dashboard as was detailed in Chapter 2.8 was that of conveying information and understanding to a user, not overwhelm a user and not biases information in a negative or obtrusive manner. These attributes of a dashboard had to be kept in mind when deciding the objectives of the Ember framework and the network telescope dashboard that would be created with it.

The Ember framework would be responsible for the generation a dashboard, it must thus be able to generate any kind of dashboard. This objective places allot of focus on the modularity and the generic design principles of the dashboard components. These components should be able to be used with any kind of information that needs to be displayed in whichever way deemed fit by the user, within reasonable constraints of course. Development and testing was done with Mozilla Firefox other browsers are not explicitly supported and as a result the rendering of components might not work as intended. Object oriented design principles and the model-view-controller principle were used during the design of the Ember framework. The structure of a dashboard using the Ember framework is illustrated in Figure 3.1.

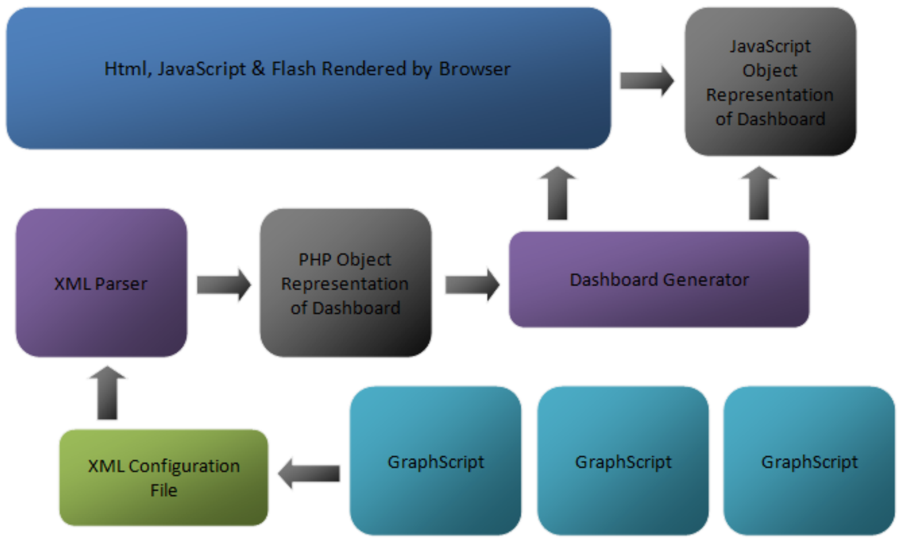

Html, JavaScript & Flash Rendered by Browser
JavaScript Object Representation of Dashboard
XML Parser
PHP Object Representation of Dashboard
Dashboard Generator
XML Configuration File
GraphScript
GraphScript
GraphScript

Figure 3.1: Ember framework structure when integrated into a dashboard.

Before any further discussion is entered it should be noted that the word graph and chart are used interchangeable throughout this work and both have the same meaning in that they are form of graphic that embodies information and their purpose is to convey understanding and correct interpretation of that information to a user.

3.2 Containers

Containers are used to display charts in the Ember framework, each container can hold multiple graphs and offers various means of interacting with their respective graphs. Interaction such as zooming in on a specific graph or obtaining a textual description of a specific graph is thus controlled by the containers. Containers also allow for the navigation and selection of different graphs, this can either be done manually or automated by the use of a toggle cycle function. Figure 3.2 illustrates a container object and its various components.

Each container has its own container ID so that it can be referenced from within JavaScript, containers also use an array of chart ID's that are contained in the JavaScript Object Representation of the dashboards. These ID's allow the container to differentiate between charts and target specific charts. The rest of this section will briefly describe the functionalities of a container object in the Ember framework.

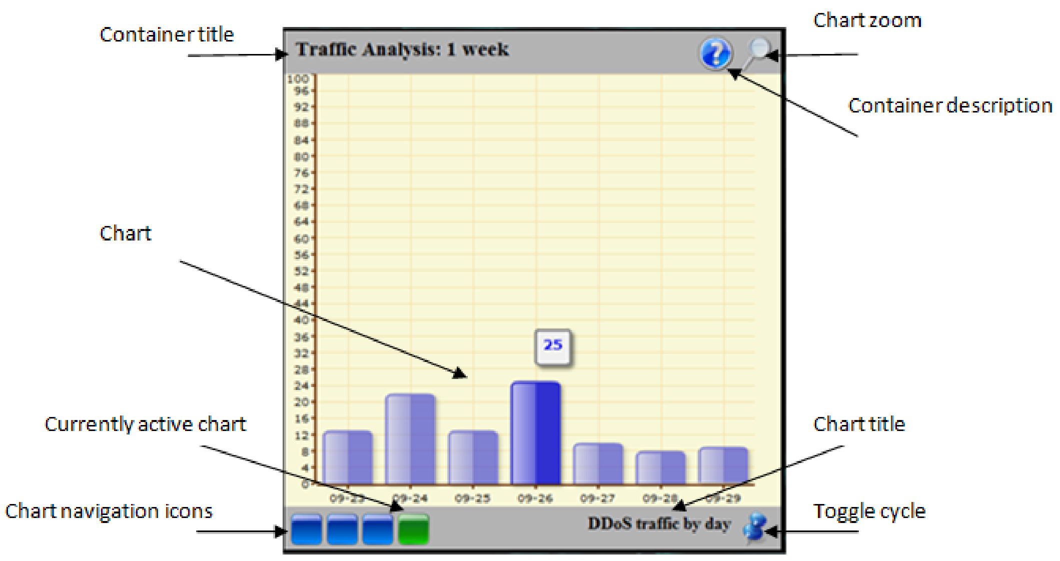


Figure 3.2: An example of a container that contains four charts and is currently displaying a bar chart.

3.2.1 Zoom

The zoom function located on the top right hand corner of each container allows the currently selected graph to be full screened. The zoom icon resembles a magnifying glass for easy identification. When a chart has been zoomed in, that chart will no longer cycle. If the automated cycle was turned on, until such time as the zoomed state has stopped the displayed chart will not change. The main use of the zoom function is to focus on a specific graph, this could be used during a presentation if attention needed to be drawn to a single chart.

3.2.2 Description

The description function located also on the top right hand corner of each container displays a small pop-up window which contains a general description of all the charts managed by that container. This description is defined in the backing graph script for each chart and is retrieved from that graph script by using a HTML GET request. The description normally motivates the use for that specific chart and data combination, describes any assumptions on the data and provides additional information to help interpret a chart correctly.

3.2.3 Navigation

The containers contains a navigation icon for each chart it can display, the icons are blue squares and are positioned in the bottom left of the container. At any given time one of the navigation icons will be green, this indicates the currently active and displayed graph in the container. Selecting a different navigation icon will replace the current graph being displayed in the container. The replacement is done via JavaScript code that modifies values stored in the Document Object Model. This is done by replacing the `graphID` variable and backing graph script location on the Flash object in the browsers memory, the order in which these attributes are modified is important as the wrong order would result in anomalies when the chart is displayed.

3.2.4 Toggle Cycle

The toggle cycle functionality in the container allows one to automate the processes of navigating through the charts held in a specific container. The Toggle cycle button is located in the bottom right hand corner of the respective container and is illustrated by a blue thumb pin icon. Once the toggle cycle icon is clicked, it changes its colour to green, just as the navigation icons had. Once the icon is green it means that that container is currently in its cycle state and as such it will automatically cycle through all the graphs contained in it, displaying each one for a pre-defined time. Initial benchmarking has shown 8-10 seconds is sufficient to see and interpret the results contained on the chart.

This ability allows the dashboard to stand alone without interaction and still be able to show a large amount of data without overloading an observer with all the information at once. The cycle function is handled in JavaScript with the `window.setInterval()` function, an anonymous function is created with an existing function to handle the logic of cycling though a container and is passed to the `setInterval()` along with a time in milliseconds. The JavaScript source code to start/stop cycling can be found in Appendix A.1, along with updating the state of that container and the modification to the CSS class to update the thumb pin that is displayed.

3.3 Graph Scripts

The graph scripts in the Ember framework form the basic building blocks of the dashboard that will be generated, they are self contained classes and for each graph that is to be displayed in the dashboard a backing graph script has to exist. The graph scripts are written in PHP and require a unique name for each graph, their purpose is to define all the characteristics of that graph. The graph scripts also output a correctly formatted text string that the Open Flash Chart 2[1] library uses to construct a flash object. OFC2 will be discussed in greater detail in section 3.5.1, for now it suffices to say that the library only requires a text string to draw a given graph from.

Each graph script is made up of a class with private variables, a constructor and multiple accessor methods, sample code for a graph script is illustrated in Appendix A.2

The `open-flash-chart.php` script is required to construct the actual chart object that will later be called with a the `toPrettyString()` method which returns a well formatted text string which is used by the Flash object to display the chart. Each graph scripts class variables hold information pertaining to the graph and can be obtained via two methods. The first method is used by calling the graph scrips accessor functions, this is the same method employed by the Dashboard Generator class; which is detailed in section 3.4. The second method of attaining these attributes is through a HTML GET request and is illustrated in Appendix A.2.

The datasets used by a chart can be directly coded into the graph script or fetched from a database. PHP allows for simple database access through the use of a PDO[2] connection. Once data has been obtained for a graph script it is used to define the chart through functions provided by OFC-2 which is detailed in Appendix A.3.

Each OFC-2 chart object has multiple cosmetic settings such as background colour, tooltip labels, alpha fills and default styles which allow for deep customisation of the look and feel for each chart. In addition to the aforementioned attributes each chart object is also given a SWFscript declaration which needs to be added to the HTML page where the Flash object will be displayed, this is detailed in Appendix A.4.

The dashboard generator class creates objects for each of these graph scripts that it uses to construct the HTML and JavaScript that is used by a browser to render a dashboard. The dashboard generator will be discussed in section 3.4. Graph scripts can be used to display one of two special chart types, namely static or animated charts. These chart types are explained in the following section.

[1]Open Flash Chart 2 - http://teethgrinder.co.uk/open-flash-chart-2/
[2]PHP Data Objects - http://php.net/manual/en/book.pdo.php

3.3.1 Static Charts

Static charts are charts that illustrate a single dataset and are displayed as a static chart, ie it does not change over time. Static graphs are incapable of showing data over a dynamic timeline and thus are more suited towards illustrating data that is not greatly influenced over time. While static charts are useful they are thus very limited in the information they can convey to a user and because of this it was decided to include the ability for a chart to change or evolve over time, this required a more complex graph script and was as such decided to separate the two types of charts. A template for static charts is provided within the Ember framework to simplify the process of creating new charts.

3.3.2 Animated Charts

Animated charts are able to show data over a time interval, changing the shape of charts dynamically and are used to display multiple datasets in a single chart. An example of this could be a horizontal bar graph that changes its values every couple of seconds to illustrate data sets over a pre-determined time frame. By being able to dynamically illustrate changes in data, it greatly enhances the ability of a graph to convey the evolution of data over time, adding a new dimension to the information. This also allows one more easily identify sudden change or linear advance in the information being displayed and thus allows for easy identification trends and anomalies. A template for animated charts is provided within the Ember framework to simplify the process of creating new charts.

3.4 Dashboard Generator

The dashboard generator class is the core of the Ember framework, it is responsible for dynamically generating the actual HTML, JavaScript and embedded Flash required for rendering a particular dashboard in the browser. It does this by making use of an XML constructor class which creates a PHP dashboard object representation from details specified in a configuration file. This includes the number of containers, number of graphs in each container, the backing graph script location and a dashboard title. The source code for the XML constructor is provided in Appendix A.5. Figure 3.3 provides an overview and sequence of operation between the XML constructor, dashboard generator and graph script classes. The rest of this section will look in detail at the XML configuration file

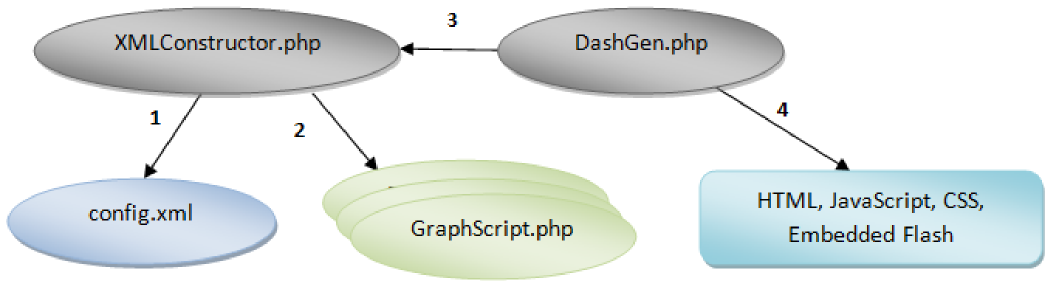


Figure 3.3: Overview of the dashboard generator, XML constructor and graph script classes.

and the XML Parser class responsible for constructing the dashboard object from the configuration file.

Sequence of operation, detailed in Figure 3.3

1. The `XMLConstructor.php` parses the `config.xml` file, which contains the names and path locations of all the `GraphScript.php` files.
2. The `XMLConstructor.php` then calls and initialises the `GraphScript.php` files, using the objects to create a PHP object representation of the dashboard.
3. The `DashGen.php` script then takes the object representation of the dashboard and starts building up the HTML, JavaScript and CSS required to display and handle the required logic for a dashboard.
4. When called from a browser the `DashGen.php` script outputs the web page it had generated.

3.4.1 XML Constructor

The Ember framework makes use of an XML Constructor class which is responsible for parsing the configuration file and constructing a partial representation of the dashboard which is then passed on to the dashboard generator class for further use. The object representation of the dashboard that is created by the XML constructor is illustrated in Figure 3.4.

The Dashboard object contains enough information so that the dashboard generator class can complete the construction of the dashboard object. The XML parser that the XML

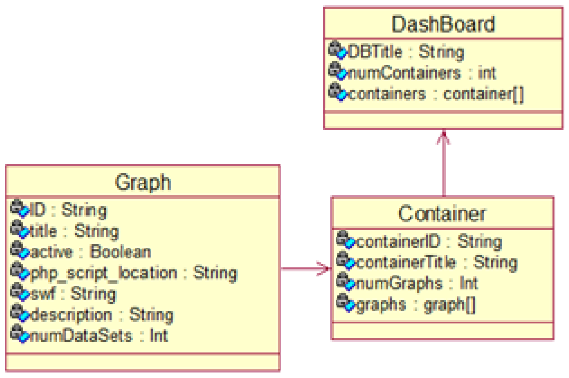
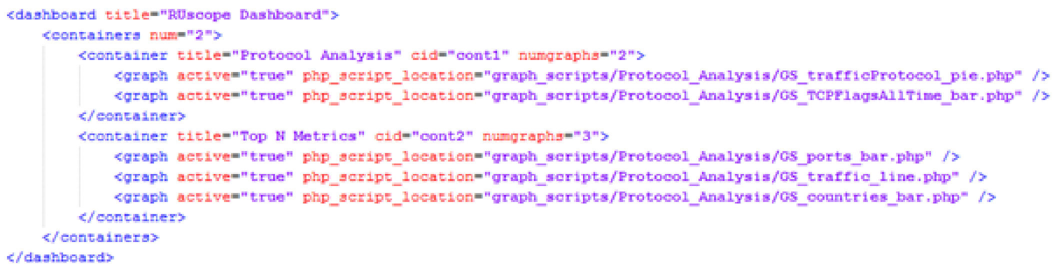


Figure 3.4: The object representation of a dashboard.



```
<dashboard title="RUscope Dashboard">
    <containers num="2">
        <container title="Protocol Analysis" cid="cont1" numgraphs="2">
            <graph active="true" php_script_location="graph_scripts/Protocol_Analysis/GS_trafficProtocol_pie.php" />
            <graph active="true" php_script_location="graph_scripts/Protocol_Analysis/GS_TCPFlagsAllTime_bar.php" />
        </container>
        <container title="Top N Metrics" cid="cont2" numgraphs="3">
            <graph active="true" php_script_location="graph_scripts/Protocol_Analysis/GS_ports_bar.php" />
            <graph active="true" php_script_location="graph_scripts/Protocol_Analysis/GS_traffic_line.php" />
            <graph active="true" php_script_location="graph_scripts/Protocol_Analysis/GS_countries_bar.php" />
        </container>
    </containers>
</dashboard>
```

Figure 3.5: Example of the XML configuration file used to define part of a dashboard.

constructor class uses is a simple XMLReader[3] object found within PHP, it reads through the elements and attributes in the configuration file and inserts the information in the object outlined in Figure 3.4.

3.4.2 Configuration file

The Ember framework makes use of XML to specify the attributes of a dashboard, it does this by keeping an XML configuration file which defines the construction of the dashboard. An example of an XML configuration file used by the Ember framework is shown in Figure 3.5.

The root element in the file is the *dashboard* which has a title attribute and contains a containers element. The xml configuration file also illustrates the hierarchy between the different elements of a dashboard; the containers and the graphs they contain. Multiple

[3]XMLReader - http://php.net/manual/en/book.xmlreader.php

container elements are then found, each of which stipulates the number of graphs contained in that container and then graph elements which keep track of the location for the backing graph script. The graphs active attribute is used to indicate whether or not a given graph is currently active in a container and thus if that container should be able to display that graph or not.

By abstracting part of a dashboard attributes to an xml configuration file, it makes it possible to easily modify the configuration file and then simply call the dashboard generator class to parse it again and thus reconstruct a given dashboard. This helps the Ember framework to be used in a versatile manner and insures modularity. The backing graphs scripts are queried for additional information once their pathname has been extracted from the configuration file; this information is then used to all the additional properties required for the dashboard object representation.

3.5 Libraries

Two different libraries have been used in the construction of the Ember framework, the most notable of which was the OFC-2 library which was created by John Glazebrook . The use of these libraries was largely motivated by the amount of time required to develop a framework by the extensibility provided by them. These libraries were also used to reduce development time. This allowed us to not only leverage high quality software but also avoid re-inventing the wheel and enabling more time to be spent on the core characteristics of the Ember framework. The rest of this section is dedicated towards the two libraries used in the Ember framework, namely OFC-2 and Shadow Box. It will explain how these libraries fit into the framework and give some details towards their capabilities.

3.5.1 Open Flash Chart 2

Open Flash Chart 2 is a open source library used to create various types of flash charts. It supports languages such as Java, PHP, Perl and Python that may be used to construct the data files used by the library to create the flash charts. OFC-2 was used to create all the charts found in containers.

OFC2 supports the construction of the following charts:

- Line charts

- Bar charts
- Horizontal charts
- Stacked bar charts
- Candle charts
- Area charts
- Pie charts
- Scatter charts
- Radar charts

3.5.2 Shadowbox

Shadowbox is a web-based media viewer application, its written in JavaScript and CSS and can be customised to a user's needs. Shadow box was used to implement the zoom function found in the Ember framework containers. Due to the way Open Flash Charts produce the flash charts embedded in the containers, shadow box was unable to work properly with the magnification of flash objects.

A work around solution is however suggested, but due to partially implementation code in the OFC-2 library the work around could not be implemented at this time. It is however listed for future implementation.

By clicking on the magnifying glass icon the currently active graph in the container should be magnified to double its original size and centred on the screen. Due to limitations imposed on flash objects by ShadowBox this does not work, however if a chart is output to as an image file it could be passed to ShadowBox in a similar manner, in which case the magnification would work correctly. OFC-2 supports the output of charts as images but during testing it was discovered that this is not the case, hopefully the next version of OFC-2 would include this.

3.6 Summary

This chapter provided a detailed overview of the Ember framework and its components. The concept of a container was introduces and it was shown how a container could be used to manage multiple charts. It was shown how the Ember framework makes use of a configuration file to define containers and charts. The concept of graph scripts were introduced and it was shown how they were used by the Dashboard Generator class. The Dashboard Generator class is responsible for dynamically creating the HTML and JavaScript used by a browser to render the Ember frameworks components. It was shown how an information visualisation framework such as Ember can be used to display a large amount of data in a small space by utilising the chart cycling function. The next chapter will detail the design of a network telescope dashboard that makes use of the Ember framework.

Chapter 4

Network Telescope Dashboard Design

The primary design goal for the network telescope dashboard was to provide a concise and high level visual representation of relevant information that had been derived from the analysis of the network telescope traffic. This was accomplished by making use of the Ember framework and other components which had not been included in the Ember framework. Motivation for the additional components found in the network telescope dashboard was a result of the Ember frameworks limitations imposed by the constraints that were required to allow Ember to be used as a generic information visualisation framework. The additional components were added to complement the dashboard and allowed for specialised data visualisation and vitally important geopolitical data visualisation. The dashboard was created to monitor traffic collected by the Rhodes University network telescope (RUscope) and was thus named the RUscope Dashboard.

Figure 4.1 illustrates the component layout that was chosen for the RUscope dashboard, the layout is only presented now as a means of providing a general overview of what the dashboard would look like and illustrate how the various components that will be addressed in this chapter fit together.

This chapter will start by introducing the data manager script which was responsible for obtaining, manipulating and saving all the information that was visualised by various components found on the dashboard. After the data manager has been introduced the chapter will detail the two separate databases that where used for the RUscope dashboard. It will then look at how the Ember framework was incorporated into the dashboard and

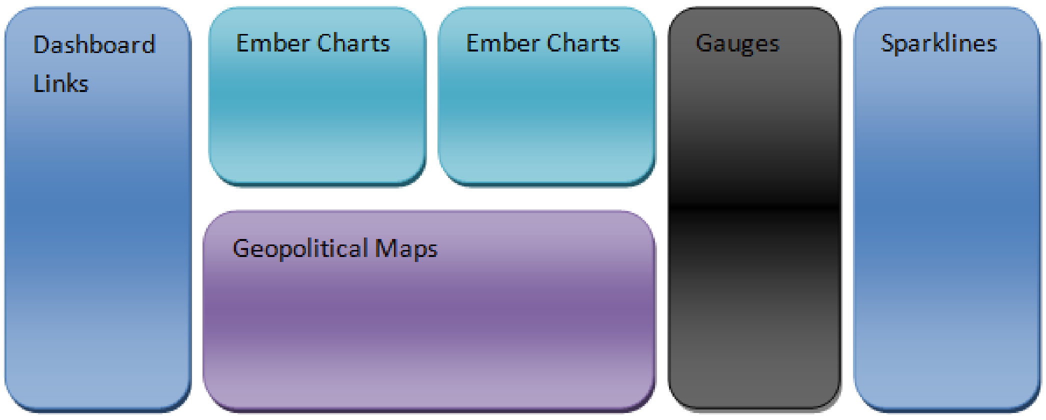


Figure 4.1: Dashboard component layout.

illustrate the various charts that were chosen for the dashboard. The various components used to extend the dashboard will then be detailed. These components include gauges, sparklines and a geopolitical flash map animation. Once the additional components have been outlined the chapter will briefly introduce the additional libraries that were used to create the components. The chapter is then concluded with a summary.

4.1 Data Manager

One of the most import components added to the RUscope dashboard is the data manager, a python script responsible for all the data management of the dashboard. The data manager script includes a set of functions that query the Rhodes University network telescope, manipulate data that is returned and populates the dashboard's local SQLite database. The motivation for writing the DataManager script in python is because python is a platform independent language and can run on any architecture as long as a python interpreter has been installed.

The data inserted into the local SQL ite database is used by the OFC-2 charts, sparklines, gauges and Ammap Flash map. The use of the local SQLite database is required as queries to the network telescope database can take from a few seconds to several minutes to execute. Since a dashboard is a tool that needs to be used in real-time it cannot afford to wait for long periods of time to retrieve datasets. The local SQLite database thus acts as a data cache for information extracted from the actual network telescope database. The data manager script is also responsible for generating an XML data-file used by Ammap to create the traffic density animation displayed on the dashboard.

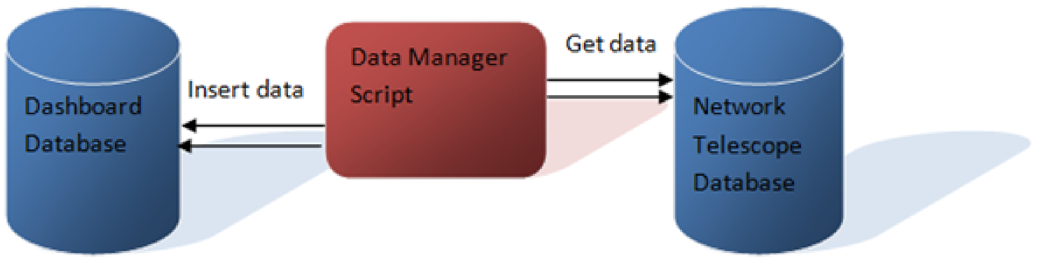


Figure 4.2: Data manager interaction with network telescope database and local SQLite database.

As the dashboard was responsible for displaying the most recent information relating data inserted into the telescope database, the data manager script needs to repopulate the local dashboard database whenever new data is inserted. This is achieved by setting the data manager script up as a batch process which runs every 6 hours, when it starts to execute it first queries the network telescope database for the data of the last packet to have been inserted into the database. If this date is newer than the last latest date found in the sqlite database, the data manager script will execute all of its queries and repopulate the local SQLite database with the new datasets. Figure 4.2 illustrates the data manager script and its relationship with the two databases.

4.2 Local Database

It was shown section 2.2.2 and 2.6 that a network telescope was capable of collecting vast amounts of traffic overtime, this of course was dependant on the sensor size of the network telescope. Since RUscope captured traffic from a /24 address block, the average amount of packet data captured according to Table 2.1 would be 7Kbps. This amount may seem trivial but it quickly adds up to a sizable collection of packets. The dashboard needed to perform multiple and often complex queries on the network telescope database to gather the data required to display the charts and other visualisations it contained. This created a problem; a dashboard is a tool that needs to be used in real-time. It was required to always have information which could be displayed with little or preferably no delay. This made the way in which data was manipulated, stored and retrieved for the operation of the dashboard slightly more complex than it would have been for a dashboard operating on small datasets that could have been queried when needed.

The problem of waiting for a long period of time while the queries were executed was solved by making use of a local dashboard database. The local database was populated with only the information that was necessary to display the various visualisations in the dashboard. This allowed for fast queries from the dashboard to its own local database. The local database schema is illustrated in Figure 4.3.

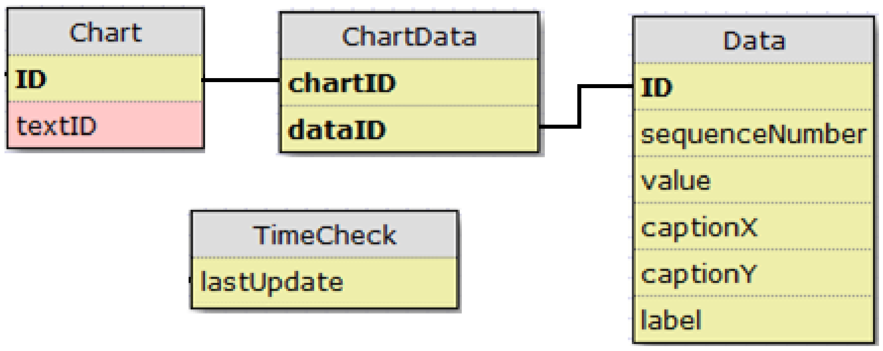


Figure 4.3: Database schema for the local dashboard database.

SQLite[1] was chosen for the local dashboard database because it doesn't require the configuration of a database server, is platform independent and all the data is stored in a single file. There are some tradeoffs when using SQLite, such as a page-based disk structure that is read from disk to cache as its needed. This results in performance problems when working with large datasets, however the amount of information that was being kept in the local SQLite database will not be enough to cause any problems.

4.3 Network Telescope Database

The Rhodes University network telescope (RUscope) records traffic in the form of pcap files which are then parsed by a script that extracts defined values and inserts them into a database. Currently the RUscope database contains just over 65 million packets. This database is then queried by the data manager script and used to populate the local dashboard database. The database structure is outlined in Figure 4.4.

[1]SQLite - http://www.sqlite.org/

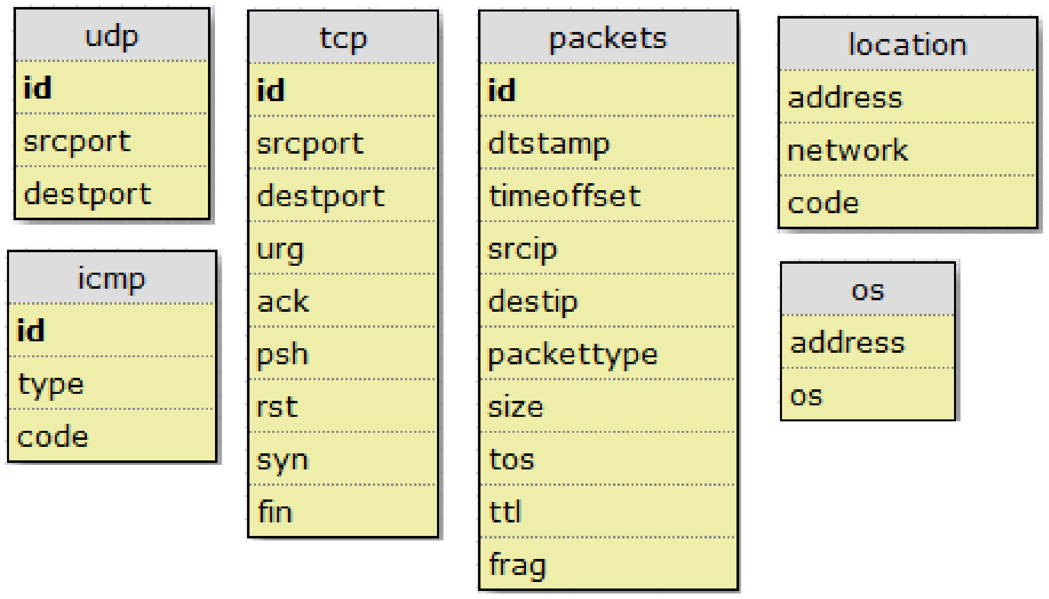


Figure 4.4: Database schema for the RUscope database.

4.4 Using Ember

The Ember framework was used to generate the charts that made up the core of the dashboard, these charts are found in top centre of the dashboards display. Only two containers were used for the dashboard, each contained four of the same charts but operated on different datasets according to their container. The charts obtained their datasets through the use of a PHP PDO database connection the he local dashboard database. The left container's dataset contained data from the last four weeks, the left container's dataset contained data from the last seven days. It should be noted that the latest date used to obtained data from might not be the current date as the network telescope database is only updated every few weeks.

The main goal of the charts that have been used in the dashboard is to display the evolution of results in such a manner as to encourage observers to find similarities or anomalies in the datasets. As mentioned section 3.2.4 the charts inside each container may be set to cycle or remain on one specific chart.

The following Figures show the different charts that where used in the RUscope dashboard, a detailed explanation is given in chapter 5 on how the traffic was analysed.

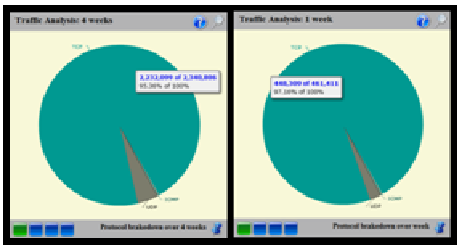


Figure 4.5: Protocol Breakdown, pie chart showing the proportion of TCP, UDP and ICMP packets captured over two intervals.

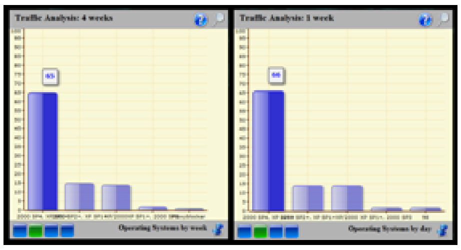


Figure 4.6: Operating System(OS) Breakdown, bar chart showing the top 5 OS's by packet count over two intervals.

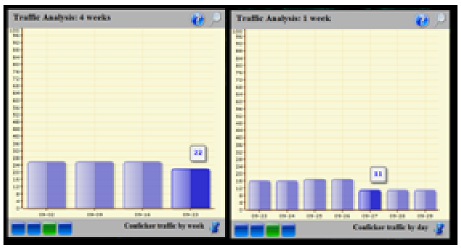


Figure 4.7: Conficker traffic, bar chart showing the normalised packet count of traffic caused by Conficker over two intervals.

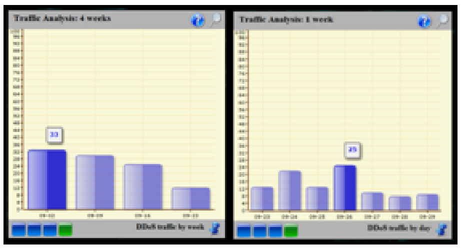


Figure 4.8: DDoS traffic, bar chart showing the normalised packet count for DDoS traffic over two intervals.

4.5 Additional Components

The additional components added to the dashboard were chosen to complement and enrich the information displayed by the Ember framework charts. The components also made it possible to show information from changing datasets in an animated fashion. The rest of this section will give a brief outline of each of these components and elaborate on how they have been used.

4.5.1 Gauges

Gauges are instruments of measurement often found on mechanical devices that need to convey some sort of measurement to a user. A common places where gauges are found are motor vehicles, which motivates how useful they are at conveying information to a user at a glance. As one of the main goals of a dashboard is to convey information at a glance the use of gauges on the RUscope dashboard are highly appropriate. The gauges implemented in the dashboard are from the Open Flash Gauges 2 library. The gauges are represented as Flash objects JavaScript is used to update their data variables. All of the gauges that were used are animated and updated over time a pre-defined time interval (3-5 seconds). Half of the gauges used also display information pertaining to a specific day in a 30 day cycle. The gauges used in the dashboard are detailed below:

Resource Meter The resource meter gauge was used to show the normalised traffic per day over a period of 30 days as a percentage of total traffic over that time period. The values were then increased by a factor of 20 to illustrate them more clearly on the resource meter. The resource meter can be found in Figure 4.9.

Duel I/O Gauges The dual I/O gauge was used to illustrate the percentage of active versus passive traffic observed during the currently displayed day in the 30 day cycle. "IN" represented Active and "OUT" represented Passive traffic. The dual I/O gauge can be found in Figure 4.9.

Digital Readout Two digital readouts were used, the first to show the total number of packets recorded over the last 30 day period and the second to indicate the total number of packets that had been captured in one of those days (this number was updated to reflect a different day in the 30 day cycle every three seconds. digital readout can be found in Figure 4.9.

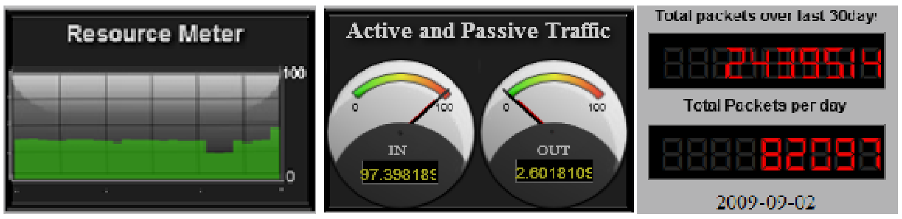


Figure 4.9: Three gauges components; the resource meter, dual gauges and digital readout.

Percentage Bars Three percentage bars where used to show the breakdown of traffic by protocol for one of the days during the 30 day cycle, these percentages were normalised over the total packet count for that day. The three percentage bards are shown in Figure 4.10.

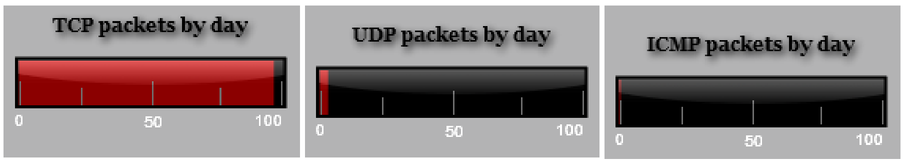


Figure 4.10: Three percentage gauges; TCP, UDP and ICMP.

4.5.2 Sparklines

Spark lines were created by Edward Tufte[41] and are best explained as small, word sized info graphics that are often used in line with a body of text. They add context and historic value to a form of measurement. For this very reason they are often included in text as the add value and context to what would normally just have been a numeric representation of a value. They also allow for easy comparison of data and help show trends and correlations between multiple data sets. Spark lines also have indicators such as min or max value detailed by a coloured dot and normal ranges by lightly colouring a range over the sparkline. The spark lines used in dashboard were positioned in a column on the far right of the dashboard display.

The sparklines where divided into two categories, one for TCP and another for UDP. Under the TCP category five sparklines were used and for UDP only two. The sparklines that were used on the dashboard made use of datasets over an interval of the last 7 days, the values they displayed were also normalised based on that specific metric being displayed in each individual sparkline. Table 4.1 shows a table summarising the types of sparklines used on the Dashboard.

Table 4.1: Sparklines used for dashboard

TCP (over 7 days)	UDP (over 7 days)
Total TCP traffic	Total UDP traffic
Backscatter (TCP SYN/ACK + RST)	SQL Slammer (UDP to port 1434)
Portscans (TCP SYN to any port)	
Conficker (TCP SYN to port 445)	
Blaster (TCP SYN to port 135)	

4.5.3 Geopolitical Mapping

The Ammap[2] library was used to create geopolitical maps with data overlays as they provided an effective means at conveying information through a source that we know and understand; such as a world map. A heat map signature illustrating the number of packets received from each country was created for each of the last 7 days of recorded traffic. A high density of packets was indicated with dark blue and moving towards light grey with lower density traffic. Countries were coloured white if there were no packets recorded for them during that day. Each Ammap interactive Flash map has settings file that is used to enable certain characteristics of the Flash animation. The settings file of the flash map had been configured to allow a user to navigate the map, pause the animation and manually move a slider at the bottom of the map which control the animation's interval.

For each day that was displayed in the animation, the date was provided and a total count of packets for that day also displayed. This allowed a user to put packet counts from specific countries into context. To view the total number of packets received from a given country during a frame in the animation, a user simply had to hover their mouse over the country, Figure 4.11 illustrates one frame. An animation for the dashboard is shown in Chapter 5.

4.6 Libraries

The two additional libraries that were used during the development of the dashboard allowed for a much richer dashboard display and enabled the dashboard to more easily

[2]Ammap - http://www.ammap.com/

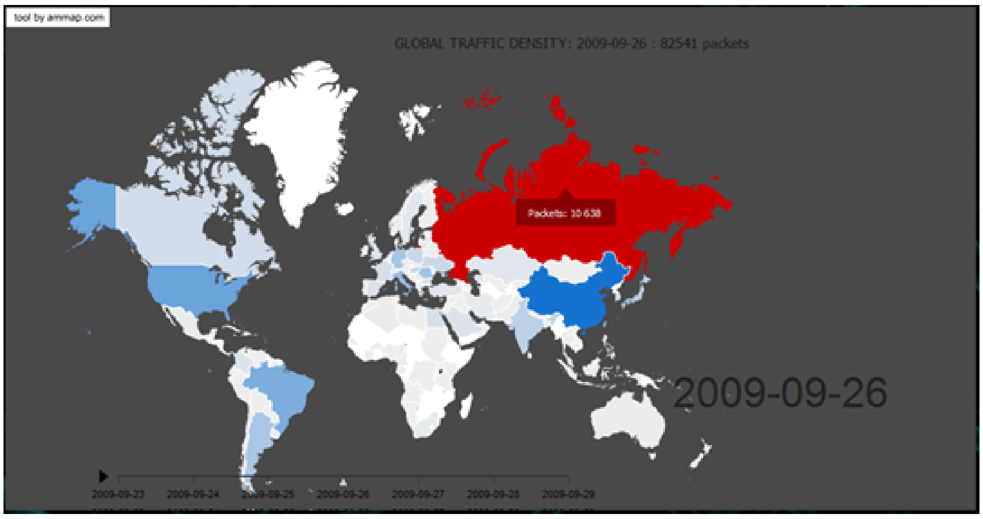


Figure 4.11: Russia is highlighted in one day, of a 7 days geopolitical traffic density animation.

display geographic based information. The Ammap library was used to create detailed flash maps to illustrate data based on geopolitical locality in a simple and easy to understand manner. The JQuery sparklines[3] library that was used to display the various sparklines found on the dashboard allowed for easy data comparison and trend observation by a user.

4.6.1 Open Flash Gauges

Open Flash Gauges is a flash library that allowed for the construction of the gauges gauges illustrated in section 4.5.1. The gauges object created by the library worked by receiving data from Javascript and using that data to update a particular gauge object. The dashboard generator class that was introduced in section 3.4 was modified so that is could output the JavaScript required by the Open Flash Gauges 2 library.

4.6.2 Ammap

Ammap is an interactive flash map library primarily used to create geographic and geopolitical flash maps and animations. It was used in the dashboard to create a heat map signature animation of a world map to show traffic density by origin country. Maps created by Ammap require the following 3 files:

[3]JQuery Sparklines - http://omnipotent.net/jquery.sparkline/

- `ammap.swf`
- `data-file.xml`
- `settings-file.xml`

The `ammap.swf` is a Flash ShockWave file which contains the vector graphics and Action Script required by Flash to handle the graphics and logic embedded in the map. The `data-file.xml` contains the data displayed in the Flash map, it also specifies the location (path) of the `settings-file.xml` relative the path specified in the HTML where the Ammap Flash object is embeded. The DataManager script included a function that would generate an Ammap data-file. The `settings-file.xml` allows a user to set various attributes that specify how the map should be rendered.

During prototyping it was discovered that the `data-file.xml` gets cached and thus to load a new `data-file.xml` the browsers cache needs to be cleared and the page refreshed before the new data would be reflected by the Ammap Flash map.

4.7 Summary

This chapter detailed the design and implementation of a network telescope dashboard for the Rhodes University Network telescope. The Ember framework was used in the dashboard along with other libraries. It was shown how the dashboard generator class can be extended to incorporate external components such as gauges, sparklines and flash maps. The problem of latency from complex queries was solved by making use of a local dashboard database that acted as a data cache for the larger network telescope database. The network telescope dashboard that was developed in this chapter is used in a case study for the next chapter. The next chapter will explore the various traffic analysis techniques that where used for the RUscope dashboard and discussion will take place which details the observations after using the dashboard.

Chapter 5

RUscope Dashboard Case Study

The goal of the RUscope Dashboard was to allow for easy and continuous monitoring of activity on the network telescope. The Dashboard also had to assisting in the identification of events and trends which might warrant further investigation. While advanced techniques of traffic analysis exist and have proven to be very effective[6][21][27][29], for the implementation of traffic analysis in the RUscope Dashboard a simplified approach to analysis was chosen. The RUscope Dashboard is show in Figure 5.1.

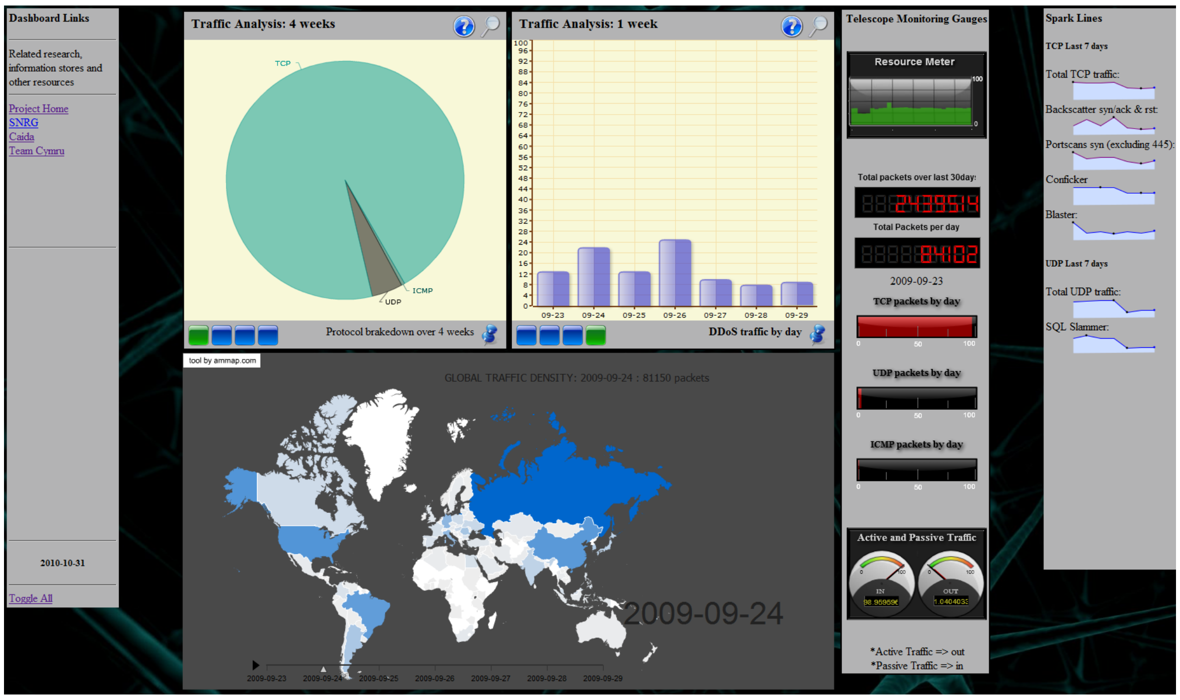


Figure 5.1: RUscope Dashboard.

This chapter will show how the RUscope Dashboard, which was constructed with the help the Ember framework and additional libraries was used to monitor traffic on the Rhodes University network telescope. It will explain the reasoning behind the techniques used for traffic analysis and discusses results and findings after using the RUscope dashboard.

5.1 Deployment

The RUscope dashboard was deployed on a host running Microsoft Windows 7. A dump file from the Rhodes University Network Telescope was used to populate a database on a Postgres database server. Wamp[1] was used the web server on which the RUscope Dashboard was deployed. Finally the data manager script from chapter 4 was set to execute every 6 hours using the Microsoft Windows Task Scheduler.

5.2 Metrics and Traffic Analysis

The traffic collected by a network telescope provides a large dataset that needs to be analysed, to accomplish this one needs to summarise the data into manageable sets that contribute to the information you are after. This can be done by identifying the characteristic of data that contribute to your desired results. Attributes such as packet size, arrival rate, port related traffic and packet compilation can all attribute to the identification of specific events and illustrate metrics. Through the use of effective visualisation one is able to easily identify significant variations in these metrics which could be indicated by a substantial deviation in the values of traffic attributes and the likely observation of a malicious event. The rest of this section will examine the analysis techniques used to identify events in network telescope traffic which was then modelled into various visualisations to be displayed on the RUscope dashboard.

5.2.1 Distributed Denial of Service(DDoS) Detection

A simple approach was taken for the detection and modelling of DDoS attack traffic on the network telescope. Only SYN Floods would be considered as DDoS traffic, as they have been shown to be the most prominent type of DDoS attacks[27][29]. Any TCP SYN/ACK

[1]WampServer - http://www.wampserver.com/en/

or TCP RST packets received on the network telescope would be count towards DDoS traffic. This is also considered as backscatter[27] and for use in the I/O gauge it was considered as passive traffic.

A typical SYN Flood starts when a botnet's command and control initiates an attack on a target[15], zombie hosts (or bots) in the botnet periodically check in with the command and control for attack vectors or updates. Once they receive the attack vector they start sending TCP SYN requests to the target from spoofed IP addresses. The purpose of these SYN requests is to create a large amount of half open TCP connections and effectively tying up all available connections to that host. After receiving each SYN request the victim response with a SYN/ACK reply, since the SYN packets came from spoofed IP addresses they are not likely to be received by bots and are in turn routed out onto the internet. A network telescope is able to receive these SYN/ACK replies when the spoofed IP addresses fall into the same range as what it is monitoring on. Figure 5.2 illustrates a typical SYN Flood.

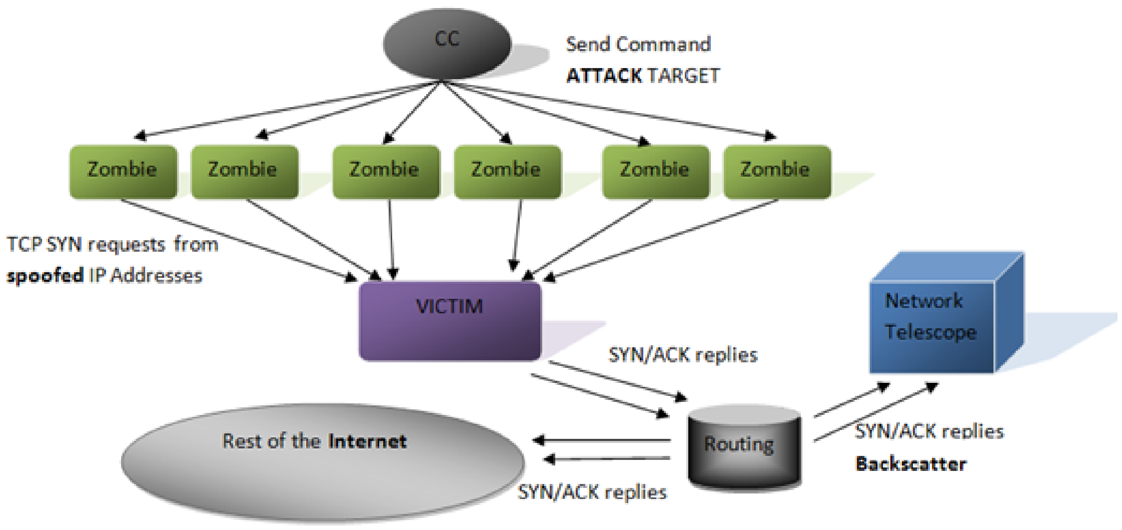


Figure 5.2: Example of a Distributed Denial of Service attack in the form of a SYN flood.

5.2.2 Worm Detection

Self propagating worms are responsible for the majority of traffic observed on the Rhodes University network telescope, this is identified by analysing the telescope traffic for known characteristics of these worms. As we cannot model each and every different worm found on the internet it was decided to choose two worms which have enjoyed prosperous growth over the last few years. Conficker was chosen for TCP and the SQLslammer worm was chosen for UDP.

Port 445 is used by Windows systems for Common Internet File System(CIFS)[2] over TCP which is also known as Microsoft Directory Service. Many windows services make use of CIFS and as a result there are many exploits available for it[29]. One of the most well known worms to exploit vulnerabilities on this service is Conficker[39] which exploits the Microsoft Windows vulnerability MS08-67. Conficker along with Blaster[1] and Nimda[1] was monitored by the RUscope dashboard by looking for packets that sent TCP SYN requests to port 445 on the network telescope with no other TCP flags set.

The Blaster worm was discovered in August 11, 2003[37]. The worm exploited a Microsoft Windows DCOM RPC Interface Buffer Overrun Vulnerability by connecting to TCP port 135 and then by sending a sufficiently large amount of data to overrun the buffer. Blaster was also responsible for denial of service attacks in the form of SYN[37]. To extract Blaster traffic from the network telescope traffic, all TCP packets with a destination port of 135 and a SYN flags set were taken.

SQL Slammer[38] was first discovered on the 24th of January, 2004. The worm targeted systems running Microsoft SQL Server 2000 and Microsoft Desktop Engine (MSDE) 2000. It spread by exploiting a service over UDP port 1434 which is used as the SQL Server Resolution Service Port[38]. Traffic targeting this port over UDP was extracted from the network telescope database and displayed in the dashboard as SQL slammer traffic.

5.2.3 Port Scan Detection

Port scanning is a method often employed by malicious entities to find vulnerable hosts on the internet. The technique for conducting a port scan is done by sending a packet to a host and waiting for a response. Table 2.2 in section 2.6.1 illustrates common responses for different types of packets sent. The response received can be used to determine if the status of a port and in some cases can even be used to infer the host operating system.

To extract port scan traffic for the RUscope Dashboard, only TCP traffic with the SYN flag set were used, SYN packets to port 445 that had previously been used to identify Conficker traffic was excluded from port scan traffic.

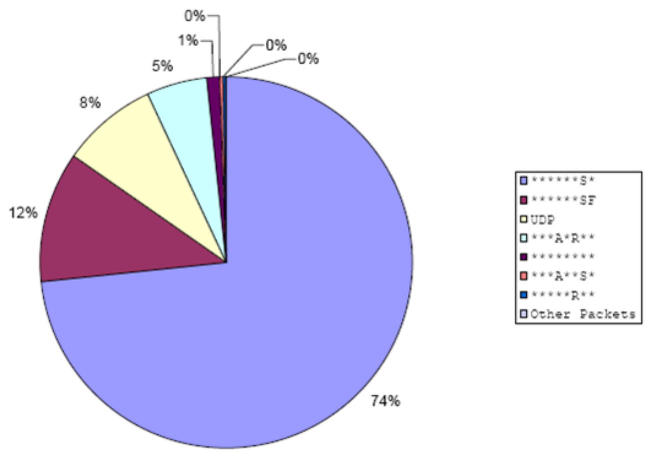


Figure 5.3: Distribution of packet types among port scans[24].

5.2.4 Active and Passive Traffic Detection

Network traffic can be classified and grouped under various categories; this is done an attempt to more easily identify correlations and anomalies in the traffic. Now that we have shown in more detail what traffic we choose to extract for the Dashboard we can group the metrics further as either active or passive. It was mentioned in chapter 1 and 2 that we had classified active traffic as any type packets that expect or solicit a reply of some kind. While on the other hand passive traffic expected no kind of response and was simple sent and forgotten. This however means that UDP is very nearly impossible to classify as either active or passive as it is a stateless protocol. There are however assumptions one can make in attempt to classify it, such as UDP payloads that match known malicious entities or source, destination and packet size inferences. These assumptions have however been simplified down in favour of applicable method suitable for a proof of concept implementation. All traffic that was not considered as active traffic, was considered as passive. This assumption allowed us to forego extracting different kinds of active and passive traffic and instead just count the amount of active traffic, which most of the visualisations already made use of. Passive traffic observation was thus a result of subtracting active traffic counts from the total traffic observed for some given interval.

5.3 Discussion

As the RUscope dashboard was constructed in such a manner as to easily compare historic and more recent traffic. The two Ember container objects each contain four charts, each of these 4 charts in both containers make up a pair. These pairs of charts operate on the same metric but make use different data sets. The container on the left shows data from the last four weeks while the container on the right only shows data from the last 7 days that had been inserted into the network telescope database. This allows for relatively easy correlation and anomaly detection. The gauges container represents data over an interval of the last 30 days and gives a breakdown of per day statistics. The sparklines container and geopolitical traffic animation only operates on the last 7 days of traffic. Note that during the use of the RUscope dashboard for this case study the network telescope database that was used only contained packets captured up until the 29/09/2009. During the rest of this chapter any reference to current day refers to the latest date found in the network telescope database, that is to say the 29th of September 2009.

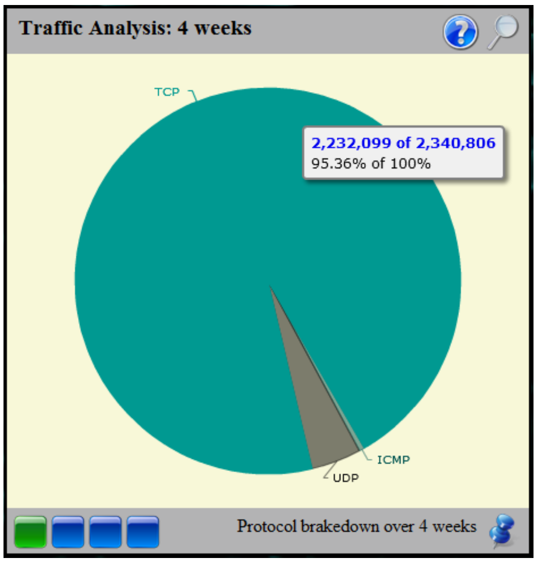
5.3.1 Ember Containers

The first chart found in each of the Ember containers were used to show a brakedown of UDP, TCP and ICMP protocols over the given time interval as illustrated in Figure 5.4. A slight increase in TCP and decrease in UDP traffic between the last 4 weeks and the current week is seen in Figure 5.4(a) and Figure 5.4(b). These values are also illustrated in Table 5.1 and Table 5.2. When looking at the Figures in Figure 5.4 one could see a clear increase in TCP traffic and a decrease for UDP. When the results were compared it showed an increase of 1.8% in the proportion of TCP traffic and a decrease of 1.78% in UDP, with ICMP changing only by a meagre 0.02%. This is also a good example of how useful information visualisation is, with small percentages as observed above it would have been difficult to note a difference if looking at table showing the packet count, this difference is however relatively clear when comparing the two pie charts.

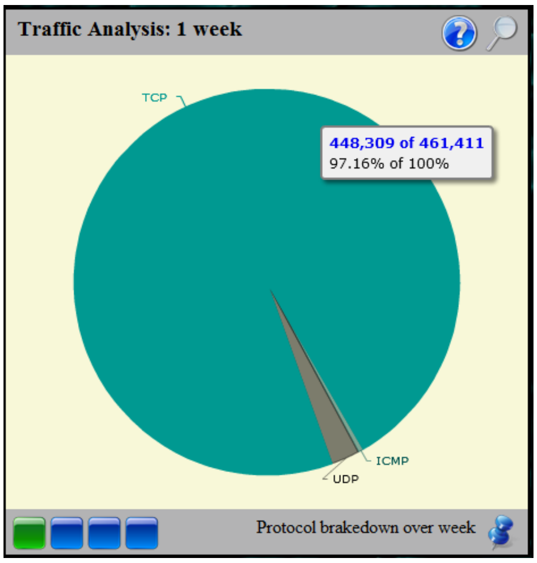
Table 5.1: Values for protocol distribution chart in Figure 6.4(a)

TCP	95.36%
UDP	4.2%
ICMP	0.44%

Figure 5.5(a) and Figure 5.5(b) illustrates a set of bar charts which show the top 5



(a) Protocol distribution over last 4 weeks.



(b) Protocol distribution over last 7 days.

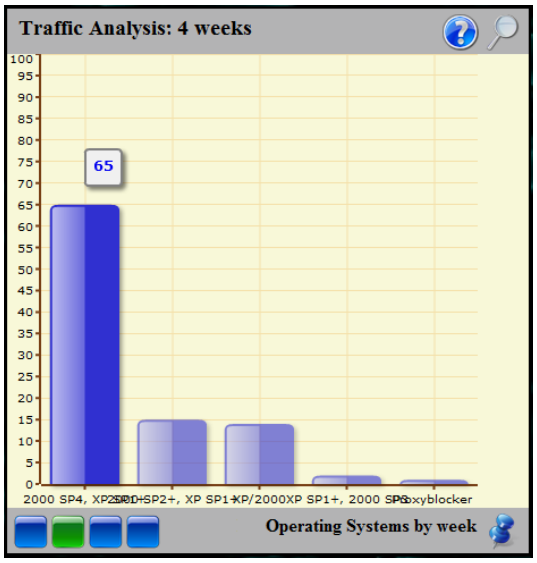
Figure 5.4: Two pie charts showing protocol distribution.

Table 5.2: Values for protocol distribution charts in Figure 6.4(b)

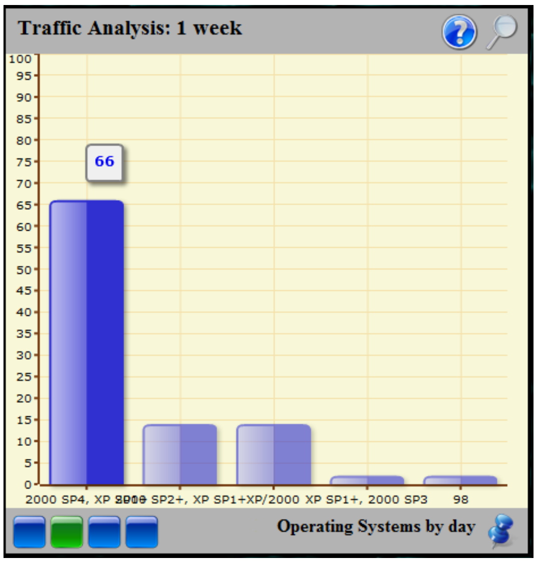
TCP	97.16%
UDP	2.42%
ICMP	0.42%

Operating Systems(OS) that had sent traffic to the network telescope. P0fv2[2] is a passive OS fingerprinting tool that was used to analyse this traffic. The percentages in these bar charts has been arranged in descending order over the interval specified for each of their containers. Some of the longer OS names overlap and make it difficult to read, there is unfortunately nothing that can be done to stop this. It was however possible to highlight the text and copy it to a clipboard to view if it was obstructed. Both Figure 5.5(a) and 5.5(b) show that a large proportion of traffic that had been recorded was from Windows 2000 and Windows XP SP1+ hosts. According to the Conficker Working Group[20] Windows 2000, Windows XP, Windows Vista, Windows Server 2003, Windows Server 2008, and Windows 7 Beta are all vulnerable to Conficker. An additional source[34] mentions how Microsoft Windows XP, SP1, SP2 and SP3 are vulnerable if the security patches for Conficker has not yet been applied. As the largest majority of TCP traffic received is a result of Conficker it can be assumed that most of the hosts that are still infected with Conficker are running on 2000 SP4, XP SP1+ and that their proportion had not had any significant change over the last 4 weeks.

[2]P0f v2 - http://lcamtuf.coredump.cx/p0f.shtml

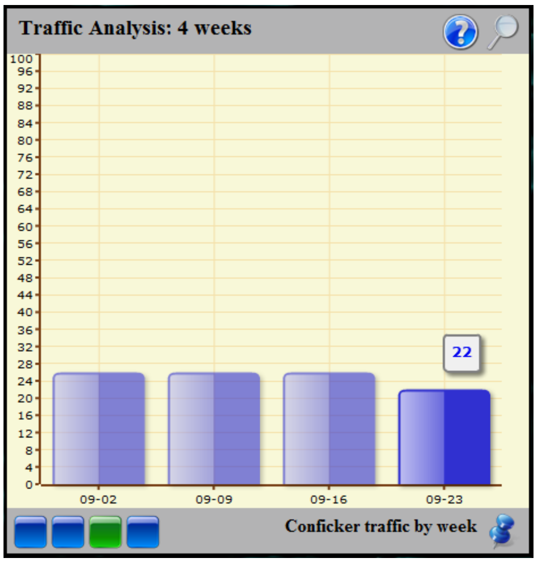


(a) Operating System distribution over last 4 weeks.

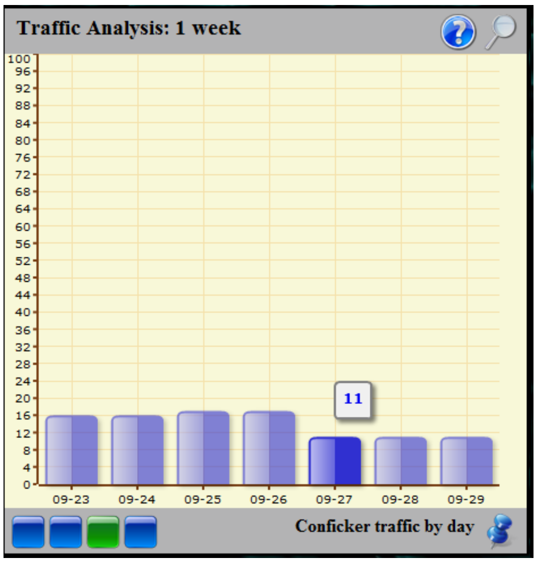


(b) Operating System distribution over last 7 days.

Figure 5.5: Two bar charts showing Operating System distribution.



(a) Conficker packet traffic distribution over last 4 weeks.



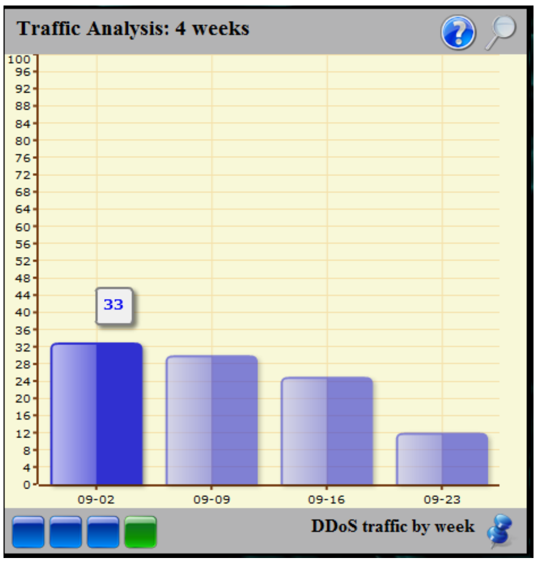
(b) Conficker packet traffic distribution over last 7 days.

Figure 5.6: Two bar charts showing the packet traffic distribution from Conficker.

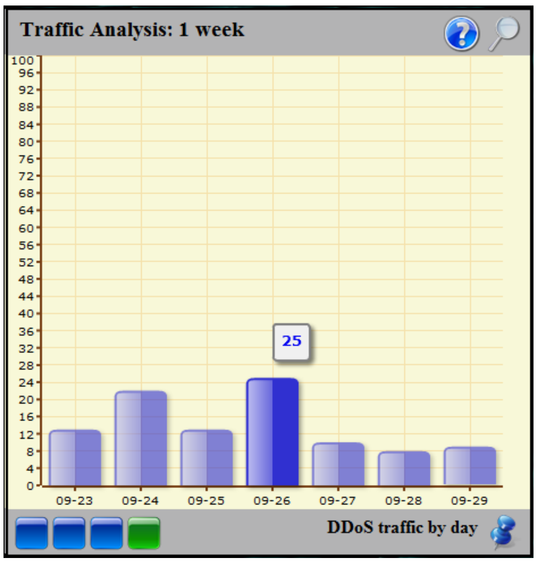
Furthermore the rate at which Conficker traffic was been detected on the network telescope over the last 4 weeks has stayed relatively consistent as can be seen in Figure 5.6(a). A drop in traffic observed during the last week, 09/23 in Figure 5.6(a) shows a slight decrease of 4% in the proportion of Conficker traffic for that week. This decrease can actually be seen better figure 5.6(b) during the interval between 09/27 and 09/29. Across all the visualisations a drop of traffic is observed during this period. The average packet count (of all traffic, including Conficker) over those 3 days was 54353, which translated into an average decrease of 32% for that period.

It should be noted that around this time in 2009 there had been multiple power failures and network outages at the network telescope's location and this might have influenced the drop in traffic. Another possibility as mentioned in section 2.3.2 is the loss of traffic through routing anomalies and congestion. If it was congestion it could possibly have been caused during the growth phase of a new worm, however we did not notice any significant increase in traffic before the decline which makes it unlikely.

Essentially the bar charts in Figure 5.6(a) and Figure 5.6(b) indicated that the traffic the network telescope had been receiving from Conficker was remaining constant, which could mean that conficker had reached its persistent state and that it's next life cycle will bring on a decline in its numbers.



(a) DDoS packet traffic distribution over last 4 weeks.



(b) DDoS packet traffic distribution over last 7 days.

Figure 5.7: Two bar charts showing the packet traffic distribution from DDoS attacks.

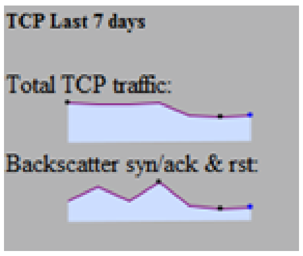
The set of charts in Figure 5.7 showed the proportion of DDoS attacks detected by the network telescope, these are once again broken down into two data sets, with Figure 5.7(a) over the last four weeks and Figure 5.7(b) over the last 7 days. The dataset representing the last four weeks in Figure 5.7(a) shows a steady decrease in DDoS traffic leading up to the present week. Figure 5.7(b) which shows DDoS traffic from the last 7 days indicates a slightly larger but still fluctuating amount of DDoS traffic over the first four days (09/23 - 09/26) and then drops slightly over the remaining three days. Using a calendar we determined that the 09/26/2009 was a Friday at GMT-5. This could indicate that during a working week DDoS attacks are more prevalent than over weekends. The assumption holds if you consider that the monetary loss a business during working hours, if the business is offline would be greater than the monetary loss during non-operating hours. Further analysis would however be required as the traffic represented in these charts are over a global time period and do not represent traffic from a single location over a specific time period.

5.3.2 Sparklines

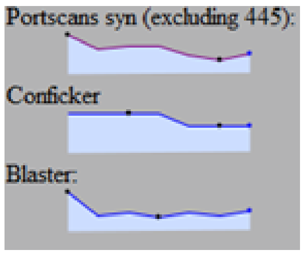
The sparklines found on the Dashboard only represented traffic over a the last 7 days. They were primarily used as a method for simple comparison on traffic composition over multiple metrics and were dived into two categories namely TCP and UDP traffic over the last 7 days. The values in all of these sparklines are normalised to show the proportion based on only that specific metric being displayed in each individual sparkline. Each sparkline uses indicators in the form of blue and black dots, one of the black dots indicate the maximum and the other the minimum values over the given time interval, the blue dot indicates the latest day.

Figures 5.8(a) and (b) show the TCP sparklines, the first sparkline in Figure 5.8(a) indicates the total proportion of TCP traffic over the last 7 days allowing it to be used as a baseline for comparison with the remaining TCP sparklines. As it has been noted in the previous charts there is a definite drop in traffic during the last 3 days. The second sparkline in Figure 5.8(a) shows the proportion of backscatter traffic (SYN/ACK and RST) over the last 7 days while the 1st sparkline in Figure 5.8(b) indicates the proportion of port scanning activity. The 2nd sparkline in 5.8(b) shows the proportion of Conficker traffic over the 7 day period, when comparing the total TCP traffic and Conficker traffic sparklines it is evident that they are very similar and one cant thus again assume that Conficker is responsible for the majority of TCP traffic captured by the Rhodes University network telescope. The last of the TCP sparklines shown in Figure 5.8(b) represents traffic from the Blaster worm over the last 7 days. The sparkline starts with a strong spike in Blaster traffic, which then subsides and stays relatively low over the remaining 6 days as compared to the first.

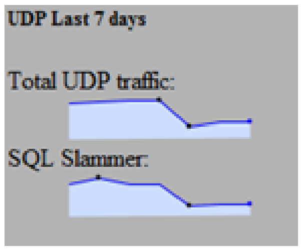
The second category which is used for UDP traffic only contains two sparklines and is



(a) TCP, total traffic and backscatter.



(b) TCP, port scans, Conficker and Blaster.



(c) UDP, total traffic and SQL Slammer worm.

Figure 5.8: Sparklines from the RUscope Dashboard

shown in Figure 5.8(c). The first sparkline as before with the TCP sparklines indicates the total proportion of UDP traffic over a period of 7 days. The second sparkline shows the traffic of the UDP based SQL Slammer worm. The sparkline graph for Slammer is also very similar to that of the total UDP traffic and it can be rightfully assumed that Slammer is responsible for the majority of the UDP based traffic.

It has been shown that from these sets of sparklines that both UDP and TCP traffic are dominated by a single worm; Conficker for TCP and SQL Slammer for UDP. If it was possible to stop or slow down the spread of these two worms, we would severely stunt the quantity of malicious traffic on the internet. There are techniques that may be used to slow down the spread of TCP based worms such as tar pitting which is employed by LaBrea[4], it does however have limitations and its effectiveness is not always guaranteed against more advanced malicious entities.

These sparklines as mentioned before only show normalised traffic flow of each individual traffic, which is to say that the maximum value found on each sparkline is used as the maximum value for the y-axis. The minimum y-axis value however has been set to 0, as opposed to the default for JQuery sparklines which uses the minimum value in the data set as the minimum value for the y-axis. This means that each sparkline is used to show only the flow of traffic for the respective metrics and not the proportion of traffic out of the total for either TCP or UDP.

5.3.3 Gauges

The various gauges used in the Dashboard helped to visualise traffic over a period of 30 days and also provide some insight on day to day patterns of that traffic. The Resource

Meter gauge in Figure 5.9 shows a tightly stacked together bar graph which was used to illustrate traffic flow over the last 30 days. The bar chart was animated and displayed a new bar from the right of the Resource Meter and once the 30th day had made its way to the left of the Resource meter it would be redrawn on the right again, constantly cycling through the traffic. The traffic for each day was normalised over the entire month. As a result of the normalisation the y-axis which is labeled from 0 to 100% each day only attributed 2-3% of the traffic. This did not make the visualisation as effective in showing the flow of traffic. It was decided to multiple the normalised values for each day with a contestant to effectively "zoom" into the graph. This worked well the only concern left was the "100" that labeled the maximum y-axis value, but until we acquire the Flash source files for the gauges we are unable to do anything about their actual appearance.

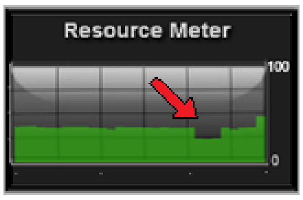


Figure 5.9: Gauges resource meter showing the distribution of traffic over the last 30 days, a drop of 25% in the average traffic is shown.

The Resource meter also shows the drop in traffic over the 3 day period that leads up to the current day. The next interesting thing to note about the resource meter is that it detected a sudden spike in traffic that was reasonably high above the average for the month.

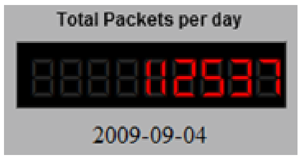
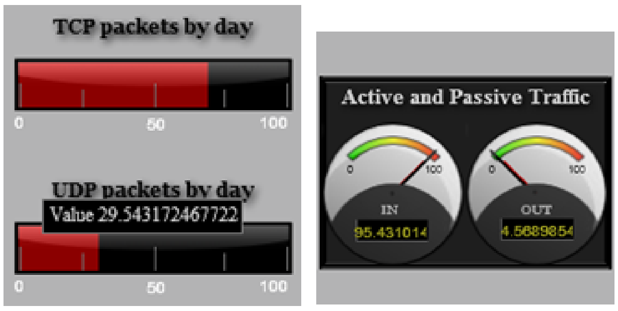


Figure 5.10: Gauges digital readout indicating the number of packets captured by RUscope on the 4th of September 2009.

Further investigation showed that on this day the Digital Readout in Figure 5.10 showed the packet count jump up from approximately 80 000 packets per day to 112537. On that same day the TCP and UDP percentage bars in Figure 5.11(a), which normally show around 96% TCP and 4% UDP for each the day, suddenly showed 70.5% TCP and 29.5% UDP which is a relatively significant change of 25%. As this day was outside of the latest

7 days, the Dashboard held no additional information surrounding it and any further investigation into the sudden spike of UDP traffic would have to be completed manually by querying the network telescope database.



(a) Two percentage bars showing the percentage of TCP and UDP traffic.

(b) Dual gauges showing the percentage of Active(IN) and Passive(OUT) traffic.

Figure 5.11: Percentage bars and Dual gauges showing traffic for the 4th of September 2009.

Lastly the I/O Gauges component shown in Figure 5.11(b) which was used to indicate the levels of Active and Passive traffic stay consistent for most of the month with an average around 4% for active traffic and a standard deviation of 2%. Active traffic is denoted by the gauge that reads "IN" and passive is denoted by the gauge that reads "OUT", we did not acquire the source code for the gauges and were unable to edit the static text they displayed.

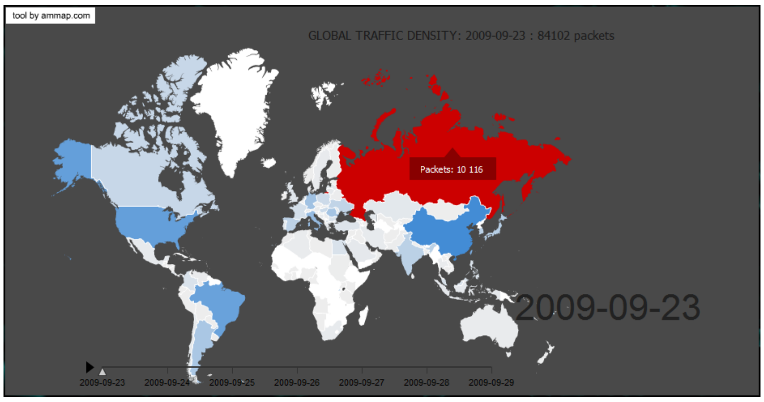
5.3.4 Geopolitical Map Animation

The geopolitical traffic density animation can be seen in figure 5.12. The animation works over 7 frames, each frame showing global traffic density for that day and in so doing animating the flow of traffic from a geopolitical perspective over the last 7 days. Darker colours symbolise more traffic while lighter colours indicate less traffic, no traffic was observed from countries that were coloured white. Figure 5.12 only shows the animation at each second day, the decline in traffic over from the 27/09 and 29/09 period is still very evident. Strangely though traffic from Russia did not slow down at all during the seven days and they managed to send an average of 10533 packets each day, with a minimum of 10116 and a maximum of 11093 packets. Traffic received from Africa was minimal, fluctuating over the 7 day period but not showing anything significant. The

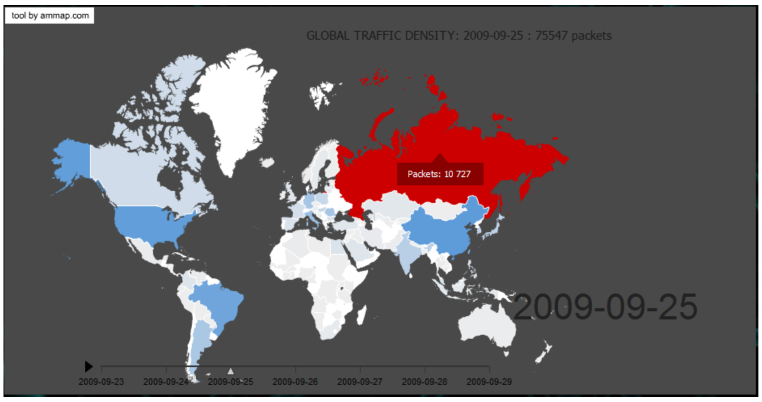
North America, South America and China where responsible for the majority of traffic after Russia, they however also showed a big decline in traffic after 26/09.

5.4 Summary

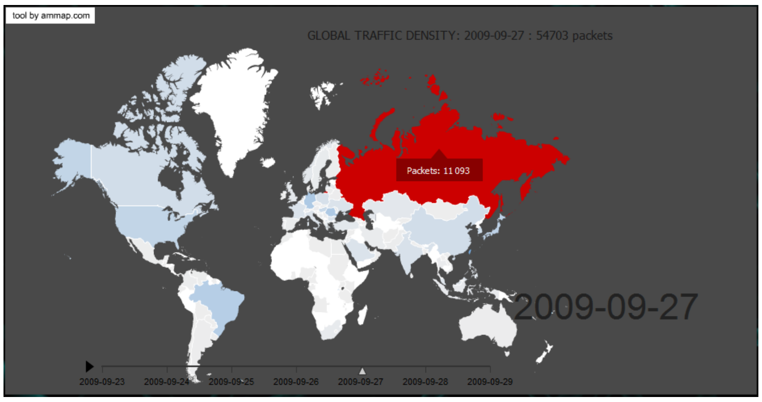
Deployment of the dashboard was relatively easy as cross platform languages such as PHP, Python and SQLite were used during the development of the Ember framework and RUscope dashboard. The chapter then outlined the various techniques used for traffic analysis in the RUscope dashboard. The traffic analysis techniques were simple and relied heavily on the prior knowledge of traffic characteristics. Methods for DDoS, worm and port scan identification were outlined and active and passive traffic was revisited and formally defined for use in the RUscope Dashboard. A discussion followed wherein observations from using the RUscope dashboard were highlighted. Several observations were detailed and their meaning inferred. The next chapter will provide possible future extensions of the Ember framework as well as exploring possible applications for the Ember framework and the design decisions used during the creation of the RUscope Dashboard.



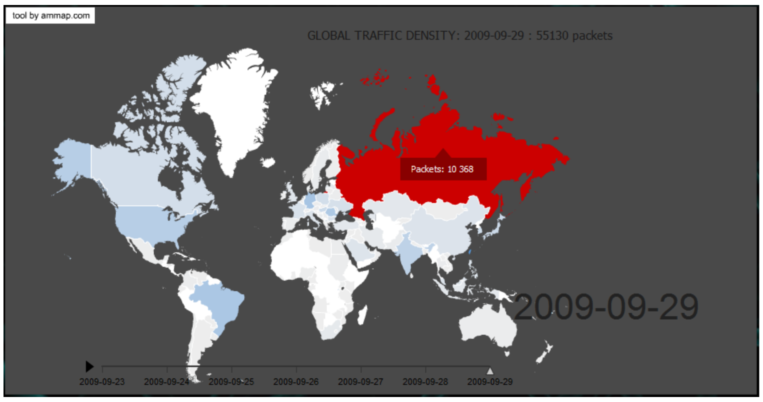
(a) Traffic density for the 23rd of September 2009, high amount of traffic observed from Russia and China.



(b) Traffic density for the 25th of September 2009, high amount of traffic observed from Russia and China.



(c) Traffic density for the 27th of September 2009, drop in global traffic excluding Russia.



(d) Traffic density for the 29th of September 2009, sustained low traffic globally excluding Russia.

Chapter 6

Conclusion

6.1 Future Extensions

The ember framework was designed to be modular and allow for easy implementation of future extensions. This was done by separating the construction of a dashboard from the datasets it would require. The number of container objects and graphs in each container are defined in an external configuration file. The configuration file also contained the location of graphs scripts that were used to construct charts. This separation makes it easy to add additional charts. It was also illustrated in chapter 4 that the dashboard generator class could easily be extended to accommodate additional libraries. The remainder of this section will highlights two possible extensions for the ember framework and then detail possible real-world application of the Ember framework and the RUscope Dashboard design principles.

6.1.1 GUI for Dashboard Construction

While the construction of the dashboard is done dynamically from a configuration file and supporting graph scripts, coding is still required for each graph script and the configuration file. The first possible extension to the Ember framework would be the creation of a complementary tool that would allow for a graphical construction of the dashboard and the graphs contained in each container. The proposed tool would be a simple Microsoft C# application that allows a user to add charts to a container by specifying the chart type and importing a data set. The application would then automatically generate the

PHP graph script file. This tool should also be able to create and modify configuration file used to define a dashboard. In so doing a user could specify the number of containers and which graphs should be contained in each container through a graphical user interface. This tool would further simplify the construction of dashboards and allow for novice users to create dashboards without any programming knowledge.

6.1.2 Data Aggregation and Visualisation Framework

The second possible extension would involve modifying the data manager script to allow for the aggregation of data between multiple sources. This is made easy be the use of the local dashboard database as the modification to aggregate the data would only have to transform the data for insertion into the local database. The local database already has a schema that can be used to store almost any kind of information. Aggregating data from sources such as honeypots and intrusion detection systems would also complement the information displayed from a network telescope. The Ember framework is extensible and able to make use of multiple libraries and data sources without a great deal of modification.

6.2 Summary

The main goal of this thesis was to develop an extensible information visualisation framework that could be used for the construction of tools that require large datasets to be summarised, managed and effectively visualised. This was achieved by the construction of the Ember framework detailed in chapter 3. It was then shown that the Ember framework could be used to construct a network telescope dashboard, through the development of RUscope in Chapter 4. This was done by extending the data manager script to also generate the code required to manage gauges, sparklines and a geopolitical map animation. RUscope was then used in a case study in Chapter 5 and it was shown that it effectively conveyed information, helped observe similarities and anomies in traffic and provided an automated means of generating metrics.

The main achievements of this thesis are now summarised.

Developed a Visualisation Framework: The Ember framework was developed to simplify the process of creating multiple visualisations using multiple data sets. It is capable of maintaining state over various charts, navigating through charts and providing additional information regarding charts contained in dashboards created with it.

Created a Network Telescope Dashboard: The Rhodes University network telescope (RUscope) dashboard was created. The dashboard is capabable of analysing network telescope traffic and visualising the results. It employs a cache database to solve the problem of latency when querying a large telescope traffic database. The RUscope dashboard is also able to visualise and assist in the discovery of DDoS attacks and monitoring worm activity.

Simplified Traffic Analysis Metrics: For use in the RUscope dashboard, multiple metrics where identified. The metrics for identifying worms included Conficker[39],SQL Slammer[38] and Blaster[12]. Metrics to identify traffic from a SYN flood and port scans where also identified.

Network security is of increasing concern and by utilizing network telescopes we are able to get a glimpse of the nefarious side of network activity. When the information obtained from the traffic analysis of network telescope data is appropriately visualised it becomes a great tool for learning and understanding. This allows us to better understand not only the frequency of malicious activity, but also discover patterns, similarities and anomalies found in potentially malicious traffic. By using what we have learnt we are able to better prepare our network to face the ever increasing onslaught of malicious entities circulating the Internet.

Bibliography

[1] Branch router QoS design. http://www.cisco.com/en/US/docs/solutions/Enterprise/WAN_and_MAN/QoS_SRND/BranchQoS.html (last visited October 2010).

[2] Internet file system. http://www.snia.org/tech activities/CIFS/CIFS-TR-1p00%20FINAL.pdf (last visited October 2010).

[3] Nepenthes - detecting malware. http://nepenthes.carnivore.it (last visited June 2010).

[4] Labrea: "sticky" honeypot and ids, 2010. http://labrea.sourceforge.net/labrea-info.html (last visited May 2010).

[5] Aben, E. Conficker/conflicker.downadup as seen from the ucsd network telescope, February 2009. http://www.caida.org/research/security/sco-dos/ (last visited August 2010).

[6] Bailey, M., Cooke, E., Jahanian, F., Myrick, A., and Sinha, S. Practical darknet measurement. In *40th Annual Conference on Information Science and Systems* (March 2006), pp. 1496–1501.

[7] Bailey, M., Cooke, E., Jahanian, F., Nazario, J., and Watson, D. The internet motion sensor: A distributed blackhole monitoring system. In *In Proceedings of Network and Distributed System Security Symposium (NDSS 05* (2005), pp. 167–179.

[8] Bailey, M., Cooke, E., Jahanian, F., Provos, N., Rosaen, K., and Watson, D. Data reduction for the scalable automated analysis of distributed darknet traffic. In *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement* (Berkeley, CA, USA, 2005), USENIX Association, pp. 21–21.

[9] Barnett, R. J., and Irwin, B. Towards a taxonomy of network scanning techniques. In *SAICSIT '08: Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries* (New York, NY, USA, 2008), ACM, pp. 1–7.

[10] CERT. Cert advisory ca-2001-19 "code red" worm exploiting buffer overflow in iis indexing service dll. http://www.cert.org/advisories/CA-2001-19.html (last visited September 2010).

[11] CERT. Advisory ca-2001-26 nimda worm. Advisory, September 2001. http://www.cert.org/advisories/CA-2001-26.html.

[12] CERT. Cert advisory ca-2003-20 w32/blaster worm. Advisory, August 2003. http://www.cert.org/advisories/CA-2003-20.html.

[13] Cooke, E., Bailey, M., Jahanian, F., and Mortier, R. The dark oracle: perspective-aware unused and unreachable address discovery. In *NSDI'06: Proceedings of the 3rd conference on Networked Systems Design & Implementation* (Berkeley, CA, USA, 2006), USENIX Association, pp. 8–8.

[14] Cooke, E., Bailey, M., Mao, Z. M., Watson, D., Jahanian, F., and McPherson, D. Toward understanding distributed blackhole placement. In *WORM '04: Proceedings of the 2004 ACM workshop on Rapid malcode* (New York, NY, USA, 2004), ACM, pp. 54–64.

[15] Cooke, E., Jahanian, F., and McPherson, D. The zombie roundup: understanding, detecting, and disrupting botnets. In *SRUTI'05: Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop* (Berkeley, CA, USA, 2005), USENIX Association, pp. 6–6.

[16] D'Agostino, R. B., and Stephens, M. A., Eds. *Goodness-of-fit techniques.* Marcel Dekker, Inc., New York, NY, USA, 1986.

[17] David Moore, C. S. Sco offline from denial-of-service attack, November 2003. http://www.caida.org/research/security/sco-dos/ (last visited October 2010).

[18] Ferguson, P., and Senie, D. Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing, 2000.

[19] Few, S. *Information Dashboard Design: The Effective Visual Communication of Data.* O'Reilly Media, Inc., 2006.

[20] Group, C. W. Home page. http://www.confickerworkinggroup.org/wiki/ (last visited October 2010).

[21] Harder, U., Johnson, M. W., Bradley, J. T., and Knottenbelt, W. J. Observing internet worm and virus attacks with a small network telescope. *Electron. Notes Theor. Comput. Sci. 151*, 3 (2006), 47–59.

[22] Jaquith, A. *Security Metrics: Replacing Fear, Uncertainty, and Doubt.* Addison-Wesley Professional, 2007.

[23] Kurose, J. F., and Ross, K. W. *Computer Networking: A Top-Down Approach.* Addison-Wesley Publishing Company, USA, 2009.

[24] Lee, C. B., Roedel, C., and Silenok, E. Detection and characterization of port scan attacks. vol. 2004, 2003. San Diego, CA.

[25] Mell, R. B. P. ntrusion detection systems. NIST special publication, November 2001. http://permanent.access.gpo.gov/lps72073/sp800-31.pdf (last visited June 2010).

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

[27] Moore, D., Shannon, C., Brown, D. J., Voelker, G. M., and Savage, S. Inferring internet denial-of-service activity. *ACM Trans. Comput. Syst. 24*, 2 (2006), 115–139.

[28] Moore, D., Shannon, C., Voelker, G. M., and Savage, S. Network telescopes: Technical report. http://www.cs.unc.edu/j̃effay/courses/nidsS05/measurement/moore-telescopes04.pdf.

[29] Pang, R., Yegneswaran, V., Barford, P., Paxson, V., and Peterson, L. Characteristics of internet background radiation. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2004), ACM, pp. 27–40.

[30] Payne, S. C. A guide to security metrics:. http://www.sans.org/reading_room/whitepapers/auditing/guide-security-metrics_55 (last visited June 2010), 19 June 2006.

[31] Provos, N., and Holz, T. *Virtual honeypots: from botnet tracking to intrusion detection.* Addison-Wesley Professional, 2007.

[32] Shannon, C., and Moore, D. The spread of the witty worm. *IEEE Security and Privacy 2*, 4 (2004), 46–50.

[33] Shinoda, Y., Ikai, K., and Itoh, M. Vulnerabilities of passive internet threat monitors. In *SSYM'05: Proceedings of the 14th conference on USENIX Security Symposium* (Berkeley, CA, USA, 2005), USENIX Association, pp. 14–14.

[34] Softpedia. Vulnerable windows machines sitting ducks for the conficker worm.

[35] Song, D. X., Song, D., Perrig, A., and Perrig, A. Advanced and authenticated marking schemes for ip traceback. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies* (2001), pp. 878–886.

[36] Spitzner, L. Honeypot farms, August 2003. http://www.symantec.com/connect/articles/honeypot-farms (last visited June 2010).

[37] Symantec. W32.blaster.worm. http://www.symantec.com/security_response/writeup.jsp?doci
081113-0229-99&tabid=2 (last visited September 2010).

[38] Symantec. W32.sqlexp.worm, February 2007. http://www.symantec.com/security_response/writeup.jsp?docid=2003-012502-3306-99 (last visited October 2010).

[39] Symantec. W32.downadup, July 2010. http://www.symantec.com/security_response/writeup.j
112203-2408-99 (last visited October 2010).

[40] TechNet, M. Microsoft security bulletin ms08-067 critical, October 2008. http://www.microsoft.com/technet/security/bulletin/ms08-067.mspx (last visited September 2010).

[41] Tufte, E. *Beautiful Evidence.* Graphics Press, 2004.

[42] van Riel, J.-P., and Irwin, B. Inetvis, a visual tool for network telescope traffic analysis. In *AFRIGRAPH '06: Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa* (New York, NY, USA, 2006), ACM, pp. 85–89.

[43] Weaver, N., Paxson, V., Staniford, S., and Cunningham, R. A taxonomy of computer worms. In *WORM '03: Proceedings of the 2003 ACM workshop on Rapid malcode* (New York, NY, USA, 2003), ACM, pp. 11–18.

[44] Zou, C. C., Gong, W., and Towsley, D. Worm propagation modeling and analysis under dynamic quarantine defense. In *WORM '03: Proceedings of the 2003 ACM workshop on Rapid malcode* (New York, NY, USA, 2003), ACM, pp. 51–60.

Appendix A

Source Code

The source code appendix contains snippets of code from the Ember framework. The code blocks have however been edited and most function bodies have been removed. The full source code will be made available on request to Rhodes University.

A.1 Toggle Function

```
function startCycle(contID)
{
    tmpPin ='pinID_'+contID;
    tmpCycleName = 'cycle'+contID;
    if (dashboard.containerList[getContainerIndex(contID)].toggleState)
    {
        dashboard.containerList[getContainerIndex(contID)].toggleState = false;
        document.getElementById(tmpPin).className = "pinimg";
        window.clearInterval(window[tmpCycleName]);
    }
    else
    {
        dashboard.containerList[getContainerIndex(contID)].toggleState = true;
        document.getElementById(tmpPin).className = "pindown";
        window[tmpCycleName] = window.setInterval
                                (
                                function() {cycle(contID);}, 8000
                                );
    }
}
```

A.2 Graph Script

```
// This is sample code,
// function bodies have been removed
class GS_protocol_1
{
    private $title = "Protocol brakedown over 4 weeks";
    private $chartID = "GS_protocol_1";
    private $description = "
                <h3>Protocol Breakdown</h3>
                <p>
                    This pie chart ilustrates the
                    traffic captured by the network
                    telescope according
                    to protocol type. 3 Protocols
                    have been selected, nameley
                    TCP, UDP and ICMP.
                </p>
                ";
    private $SWFscript;
    private $chart;
    private $numDataSets = 1;

    function GS_protocol_1()
    function getTitle()
    function getDescription()
    function getSWF()
    function getID()
    function getChart()
    function getNumDS()
}

$thisChart = new GS_protocol_1();

$PARAMS = $_GET;
foreach($PARAMS as $key=>$value)
{
    if ($key == 'req')
    {
        if ($value == 'chart1')
            echo $thisChart->getChart();
        elseif ($value == 'title')
            echo $thisChart->getTitle();
        elseif ($value == 'swf')
            echo $thisChart->getSWF();
        elseif ($value == 'id')
            echo $thisChart->getID();
        elseif ($value == 'desc')
            echo $thisChart->getDescription();
    }
}
```

A.3 OFC-2 Chart Construction

```
// This is sample code,
// Shows how the OFC-2
// Chart object is made
$query =    "SELECT Data.value, Data.label
            FROM (Chart JOIN Data ON Chart.ID = Data.ID)
            WHERE Chart.textID = 'GS_protocol_4week'
        ";

$values = null;
$labels = null;
$arr = null;
```

```
if ($result = $db->query($query))
{
    $table = $result->fetchAll();
    for($i = 0; $i<sizeof($table); $i++)
    {
        $values[] = $table[$i][0];
        $labels[] = $table[$i][1];
    }

}
$db = null;

$pie = new pie();
$pie->set_start_angle( 60 );
$pie->set_animate( true );
$pie->set_tooltip( '#val# of #total#<br>#percent# of 100%' );
$pie->set_colours( array( '#5551','#0001', '#9991')); //
$pie->set_values( array(new pie_value(floor($values[0]),$labels[0]),
                 new pie_value(floor($values[1]),$labels[1]),
                 new pie_value(floor($values[2]),$labels[2])));

$this->chart = new open_flash_chart();
$this->chart->add_element($pie);

function getChart()
{
    return $this->chart->toPrettyString();
}
```

A.4 Embedded SWF

```
<script type="text/javascript">
    swfobject.embedSWF
    (
      "libraries/openflash/open-flash-chart.swf", "GS_protocol_1",
      "460", "400", "9.0.0", "expressInstall.swf",
      {"data-file":"graph_scripts/right/GS_protocol_2.php?req=chart1"}
    );
</script>
```

A.5 XML Constructor Class

```
// This is sample code,
// function bodies have been removed
class Graph
{
    public $active; // indicates if graph is enabled in the container
    public $php_script_location;// location of script that produces the data
// required by open flash chart 2 to draw the graph
// also contains aditional information that is
// kept in the DashBoard ($db) object
    public $title; // Graph title that will be displayed
    public $ID; // Script file name, must be unique, ID EXCLUDES .php extension
    public $description;// Description of graph
    public $swf; // SWF script required for flash
    public $numDataSets;// Number of datasets, more than 1 for animated graphs
}

class Container
{
    public $containerTitle;
    public $containerID;
    public $numGraphs;
    public $graphs = array();
}

class DashBoard
{
    public $DBTitle;
    public $numContainers;
    public $containers = array();
}

class XML_Constructor
{
    private $db;
    private $reader;

    function XML_Constructor()
    {
        $this->db = new DashBoard();
    $this->reader = new XMLReader();
    $this->reader->open("test.xml");
        //** Ommited Code ***   //
        // Parses XML file      //
    }

// Function strips out path and file extension,
// getGraphID("foler/folder/GraphScript.php") => "GraphScript"
    function getGraphID($pathStr)
    function getDB() // return DashBoard Object
}
```