Diss. ETH No. 17713
TIK-Schriftenreihe Nr. 97

Entropy-Based Worm Detection for Fast IP Networks

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZURICH

for the degree of
Doctor of Technical Sciences

presented by

ARNO WAGNER

Dipl. Inform.
born January 7th, 1969
citizen of Austria

accepted on the recommendation of
Prof. Bernhard Plattner, examiner
Prof. John McHugh, co-examiner

2008

Printed copies are available for a fee from Verlag Dr. Hut,
http://www.dr.hut-verlag.de, Email: info@dr.hut-verlag.de

Abstract

A significant threat to computers connected to the Internet are Internet worms. A worm is a software program that self-replicates to other computers over a network. Typically this involves a security compromise of the target computer system. An Internet worm replicates using the Internet as a communication medium. In particular, fast Internet worms are capable of compromising large numbers of hosts in a very short time. Observed worms have managed to compromise up to 500'000 hosts in 8 hours (Blaster worm) or 15'000 hosts in 15 minutes (Witty worm). The compromised hosts can then be used for secondary attacks, for example flooding-type Denial of Service attacks. The secondary attacks are typically executed using a malicious payload contained in the worm code, but worms can also be used to build up networks of compromised computers that are controlled remotely. In addition, the propagation phase of a fast Internet worm can cause significant network disturbances.

We present a detection method for fast Internet worms, that is usable on high and very high volume networks, for example Internet backbones. During our investigation, we determined that propagation traffic of fast Internet worms has a specific impact on the entropy of address fields in network traffic data. One specific change is that each worm-infected host contacts many others in a short time, causing source IP address entropy to drop, and target IP address entropy to increase. Such a connection pattern is rare in ordinary Internet traffic and usually only seen in scanning activities. However, non-worm scanning has lower intensity and is generally not a network-wide event. A similar change can be observed in the port numbers of the propagation traffic.

We used three different approaches to better understand fast Internet worms and their impact on network traffic. First we built a simulator that is capable of simulating Internet-wide worm outbreaks realistically in a quan-

titative fashion. The simulator gave us insights into what aspects are most important for the propagation speed of an Internet worm. In a second step, we modelled the impact of worm propagation traffic theoretically. We were able to demonstrate that entropy changes in a characteristic fashion in the presence of worm propagation traffic. In addition, we did extensive measurements on a large set of traffic data to determine the changes in entropy characteristics during worm outbreaks. The observations confirm the theoretical analysis and show that the expected effects can indeed be observed in real network traffic.

The characteristic Entropy changes were used to construct a threshold-based detector for fast Internet worms. Our method is generic and does not require knowledge of the specific vulnerabilities used by the worm to compromise the target computers. The implemented prototype is scalable and very lean with regard to computational effort. Memory consumption can be reduced to a small constant when entropy estimation using sample entropy is replaced with entropy estimation by using a data compression algorithm.

A primary quality measure for a worm detector is the number of false alerts, i.e. false positives, it produces. Typical Internet traffic contains a multitude of anomalies, most of them of no further consequence. If a detector reports all these anomalies indiscriminately, it becomes of little practical use. We evaluated the detector design on half a year of network data from a medium-sized Internet backbone, demonstrating that is has low detection latency and produces a low number of false positives. Validation was performed with entropy estimated by sample entropy and with entropy estimated by compression. The detection quality using compression is comparable. We also compared the two entropy estimation approaches directly (i.e. not in the context of a detector) and found that results are generally similar, although compression shows a higher sensitivity to short-term anomalies.

In addition to the scientific contribution, challenges and solutions for the problem of capturing and handling large amounts of flow-level network data are presented and discussed. These include experiences with design, implementation and operation of a data capturing and processing system and software over a period of several years.

Zusammenfassung

Internet-Würmer stellen eine ernste Bedrohung dar. Ein Wurm ist ein Programm das sich selbstständig über ein Netz auf Rechner verbreitet. Dies erfordert normalerweise einen Bruch der Sicherheit des Zielsystems. Bei einem Internet-Wurm ist das benutze Netzwerk das Internet. Schnelle Internet-Würmer können in sehr kurzer Zeit eine grosse Anzahl von Rechnern infizieren. Beobachtet wurden Zahlen von 500'000 kompromittierten Rechnern in 8 Stunden (Blaster Wurm) und 15'000 Rechnern in 15 Minuten (Witty Wurm). Die infizierten Rechner können für weitere Angriffe genutzt werden, beispielsweise für Denial of Service Angriffe (Vermindern der Dienstgüte eines Dienstes bis hin zum Totalausfall, zum Beispiel durch Fluten des Zielsystems mit Anfragen). Der entsprechende Angriffscode kann bereits im Wurmcode enthalten sein, Würmer können jedoch auch genutzt werden, um Gruppen ferngesteuerter Rechner aufzubauen. Zusätzlich kann die Verbreitungsphase eines schnellen Internet-Wurms erhebliche Beeinträchtigungen des Netzes mit sich bringen.

In dieser Arbeit stellen wir eine Methode zur Erkennung von schnellen Internet-Würmern vor, die für sehr schnelle Netze geeignet ist. Bei unseren Untersuchungen stellten wir fest, dass die Verbreitungsphase eines schnellen Internet-Wurms einen characteristischen Einfluss auf die Entropie der IP-Adressfelder im Netzwerkverkehr hat. Eine spezifische Veränderung ist, dass jeder infizierte Rechner in kurzer Zeit viele andere kontaktiert, und daher die Entropie in den Quelladressen fällt und in den Zieladressen steigt. Dieses Muster ist untypisch und wird normalerweise nur beim Scannen beobachtet. Scannen, das nicht von einem Wurm kommt, hat jedoch niedrigere Intensität und ist generell kein netzwerkweites Ereignis. Eine ähnliche Veränderung der Entropie kann während einem Wurmausbruch bei Portnummern beobachtet werden.

Wir nutzen drei verschiedene Ansätze, um Internet-Würmer und ihren Einfluss auf das Netz besser zu verstehen. Erstens wurde ein quantitativer Simulator entwickelt, der in der Lage ist, Wurmausbrüche im Internet realistisch zu simulieren. Hauptresultate sind Erkenntnisse, welche technischen Aspekte eines Wurms am wichtigsten für seine schnelle Verbreitung sind. In einem zweiten Schritt wurde ein theoretisches Modell für den Einfluss des durch Wurmverbreitung erzeugten Netzverkehrs geschaffen. Es konnte gezeigt werden, dass die entstehenden Entropieveränderungen allgemein einem charakteristischen Muster folgen. Zusätzlich wurden umfangreiche Messungen auf einer grossen Datenmenge vorgenommen, die das theoretische Modell bestätigen und zeigen, dass die erwarteten Veränderungen tatsächlich in echtem Netzverkehr beobachtet werden können.

Auf Basis dieser spezifischen Entropieveränderungen wurde ein Schwellwert-orientierter Wurmdetektor implementiert. Der verwendete Ansatz ist generisch und hängt nicht von der spezifischen Verwundbarkeit, die der Wurm nutzt, ab. Der Prototyp skaliert gut im Verhältnis zur verarbeiteten Datenmenge und benötigt nur wenig Rechenleistung. Der Speicherbedarf kann auf einen sehr kleinen Wert reduziert werden, wenn Entropie durch Kompression geschätzt wird, anstelle einer Auszählung von Stichproben.

Ein primäres Qualitätsmass für einen Wurm-Detektor ist die Anzahl verursachter Fehlalarme. In normalem Internetverkehr ist eine Vielzahl von Anomalien enthalten. Wenn ein Detektor auf diese reagiert ohne sie zu unterscheiden, hat er wenig praktischen Wert. Der vorgestellte Detektor wurde auf einem halben Jahr Netzverkehr eines Internet Backbones mittlerer Grösse evaluiert. Hierbei wurde demonstriert, dass er schnell reagiert und eine niedrige Anzahl von Fehlalarmen verursacht. Auf den selben Daten wurde auch eine Detektorvariante validiert, die Komprimierbarkeit von Datensätzen für die Schätzung ihrer Entropie nutzt. Die Ergebnisse sind von vergleichbarer Qualität. Beide Ansätze zur Entropieschätzung wurden auch direkt (nicht im Kontext eines Detektors) verglichen. Es wurde gezeigt, dass sie generell zu ähnlichen Ergebnissen führen, wobei der kompressionsbasierte Ansatz eine höhere Sensitivität gegenüber kurzzeitigen Anomalien aufweist.

Zusätzlich zum wissenschaftlichen Beitrag werden Probleme und Lösungen zu Fragestellungen der Aufzeichnung und Verarbeitung grosser Mengen von Netzwerkdaten im NetFlow Format diskutiert. Hierbei werden die Erfahrungen mit dem Design, der Implementierung und der mehrjährigen Nutzung von Aufzeichungs- und Verarbeitungssystemen, sowie der erstellten Software beschrieben.

Preface

The creation of a PhD thesis is a lengthy process. At the beginning, there is an idea. The idea for this particular theses was to look at fast Internet worms in more detail, prompted by an email exchange I had with Robert (”Bob”) X. Cringley. In his column “Calm Before the Storm” [31], published on July 30, 2001, he predicted that the Code Red worm would reactivate soon. I sent him a comment that this was not the way worms worked and that he should check his facts. Fortunately, I decided to monitor the packets coming into my own computer at the expected time of reactivation. About half an hour into the reactivation, I sent an apology to Bob. The scan traffic from the re-awoken worm was clearly visible on my side. At this stage, the damage was done and I was committed to understanding the problem of Internet worms better.

The scientific core idea of this thesis, namely to use entropy statistics as basis of a detection mechanism, came to me some time later, after I had established a data capturing and storage infrastructure. During the outbreak of the Nachi.a worm (the Blaster “anti-worm”, which failed to stop Blaster, but managed to cause a lot of additional damage), I noticed that while the raw data volume (on flow level) increased significantly, the compressed data volume increased only moderately. I later found out that others had the idea of looking at entropy statistics independently from me, but nobody seemed to really have followed up on it.

Initially, I underestimated the effort of dealing with the network data that we had the good fortune to obtain access to because of our good historic contacts to SWITCH. I had to build significant infrastructure, both software and hardware, before I could start any scientific work. Fortunately, in time, the entropy idea turned into a working detector and I was in the unique position of doing evaluation measurements on a really large set of real traffic data.

This theses owes its existence to many people. I am grateful to Robert X. Cringely for raising my interest in worms in the first place. I am also grateful to Nicholas Weaver for his essay ”How to 0wn the Internet in Your Spare Time” that drove home the point that fast Internet worms are more than just a nuisance. Professor Plattner made my work possible in the first place by hiring me and by facilitating access to the SWITCH data. Professor McHugh provided valuable feedback on the thesis draft. Thomas Dübendorfer was a valuable collaborator in the DDoSVax project and contributed numerous ideas. Michael Collins, Andrew Kompanek and the rest of the CERT netSA team invited me on several occasions, to FloCon and otherwise, and always provided good discussions and insights.

I could not have built the computing infrastructure for my work without the help of the Services Group at TIK. Hans-Joerg Brundiers and Thomas Steingruber provided significant assistance in obtaining the components for and setting up the computer cluster “Scylla”. On the side of SWITCH, Simon Leinen and Peter Haag provided significant help with regard to the NetFlow data we received from SWITCH. I am also grateful to the SNF (grant 200021-102026/1) and SWITCH for financing the DDoSVax project. I thank Jan Gerke and Placi Flury for being good friends during the process of writing this PhD, and all the other members of the communication systems group for discussions and feedback. I also had the pleasure of supervising a number of students in their thesis work at ETH. Some of these were done under joint supervision with Open Systems, where Stefan Lampart and Roel Vandevall provided excellent support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The emergence of the Internet as a global communication infrastructure brought with it a multitude of new threats to networked computers. Many widely used operating systems and applications are still plagued by frequent vulnerabilities. While professionally administrated Internet hosts can be secured to a reasonable degree today, for many computers the administrator is a non-expert, typically the same person as the computer user. This often results in insecure environments, where security patches are installed late or not at all and many vulnerabilities remain unsecured for a long time. Huge monocultures of hosts that have similar vulnerabilities increase the overall risk.

Consequentially, Internet security is still in its infancy. There are numerous threats that can seriously impact Internet users, hosts connected to the Internet and general Internet network stability. Among the most important problems are spam[1], which can render email accounts hard to use by overloading the user with a flood of unwanted messages. There are phishing attacks, where well-known Internet sites, such as those of banks, are faked and users are redirected in an attempt to steal their passwords and account details for use in theft, identity-theft and other criminal activity.

A significant threat are bot-nets, i.e. coordinated groups of compromised hosts, that can be used for sending spam, conducting Denial of Service (DoS) attacks by flooding sites with traffic and other hostile activities. Bot-nets can

[1] Unsolicited commercial email and other unwanted mass-mailings. There are three accepted spellings. “SPAM” refers to the original Trademark by Hormel Foods, see `spam.com`. “Spam” uses the trademark as a name. Finally, “spam” is frequently used when any confusion with the trademark should be avoided.

have thousands of member hosts. Researchers have found bot-nets as large as 350,000 hosts [32]. However, most attack activity from bot-nets seems to involve only several thousands of hosts. One possible explanation is that bot-net operators currently still face command and control problems that limit bot-net activity [83].

Traditionally, bot-nets are slowly built up, until they are large enough for their intended purpose. An alternative approach is to use a fast Internet worm to quickly compromise a large number of hosts and carry out the intended attack a short time later. An Internet worm is a piece of self-replicating code, that replicates not locally, but remotely on other hosts reachable over the Internet. Typically, this requires a security compromise on the remote hosts, since most networked software refuses to execute unrequested code that was sent to it over the network or otherwise places strong restrictions on it. If, for example, a massive Distributed Denial of Service (DDoS) attack is planned, the attack code can then be part of the worm code and the attack can be triggered at a specific time. This neatly avoids the command and control issues and at the same time makes tracing the originator of the attack harder, at the price of far lower flexibility. Examples of well-known fast Internet worms are the Blaster worm [37], the Witty worm [115] and the Slammer worm [73]. The first known (slow) Internet worm is the Morris worm [96], which had its initial outbreak on November 2, 1988.

Worm propagation can involve user activity. For example, email worms often need active attachment opening by the user, in order to exploit a local vulnerability. Propagation can also be fully automatic, with no user interaction needed, but at the cost of more elaborate exploit code and fewer suitable vulnerabilities. Since user interaction is typically quite slow and can delay infection of a host by hours or longer, fast Internet worms rely on vulnerabilities that can be exploited in a fully automated fashion. Fast Internet worms that have been observed so far have managed to compromise up to several hundred thousand hosts in a matter of a few hours [37]. Significantly faster infection speeds are a real possibility [98].

Complete host compromise is not required in order for a host to be useful as part of an attack. Application compromise is enough, as long as the application in question has network access. An attractive target are P2P file-sharing applications [106], although in the past worms have typically exploited vulnerabilities in basic operating system services, such as the RPC (Remote Procedure Call) port mapper, and in server software, such as web-servers [37, 73, 102, 115].

This thesis tackles the problem of detecting fast Internet worms early in the propagation phase and on high traffic-volume networks, such as Internet backbones. Internet backbone networks typically deliver traffic between a large number of hosts and consequentially offer a more global traffic view, compared to access-networks. This reduces misdetection caused by localised DoS attacks and scanning activity. At the same time, backbone network stability can be at risk by worm propagation traffic as well. An early detection capability is a valuable tool for operators of these networks. The main focus of this thesis is on creating a computationally cheap detector for fast Internet worms, that is reliable (i.e. has a low number of false positives), fast (i.e. detects worms early in their propagation phase), capable of real-time operation and suitable for use on very high traffic volume networks.

1.1 Detection: Selectivity is the Key

Anomaly detectors in general, and in particular those intended to be connected to a network with high traffic volume, have to be selective in what they trigger on. Building a general backbone anomaly detector that works on traffic intervals with a length in the minute-range or longer is easy: It just needs to output the "something detected" state for each data interval. Usually, an anomaly, for example a port-scan, will be present at any given time, resulting in a very low false-positives rate for such a detector design. The rate of false negatives would be zero by construction. For obvious reasons this type of detector is completely useless.

One problem is that unspecific detection alone is of limited value in a defensive network monitoring setting. Alerts have to be used to determine actual risk created by the anomaly. For example, an attack against a type of system that is not installed in a specific network carries very low risk and typically does not require countermeasures. On the other hand, massive scanning activity directed to a specific port, that is used by a service critical to a specific network, deserves high attention and may require immediate action. Presently, determining the risk caused by an anomaly does usually involve a decision by a human being at some stage. Before this stage, the detection result needs to be specific enough that the number of false alerts is small. Otherwise the number of messages can cause information overload problems for the human operator. The human operator needs to have enough attention and concentration left to be able to read and recognise critical events with little delay.

A second problem is that unspecific anomaly detection is of very little scientific value. Again, evaluation of network traffic anomalies depends on the type and intensity of the anomaly. Very low intensity scanning, for example, is strictly speaking an anomaly, but is prevalent enough that it forms part of the normal traffic profile. The only thing that can be done with unspecific detection is anomaly counting. However, since one high intensity anomaly can easily have orders of magnitude more impact than a low intensity anomaly, these counts have very little meaning.

For this reason, it is necessary to build detectors for specific types of anomalies, that do not trigger on other types. In fact, the frequency of a detector to trigger on the wrong type of anomaly, thus generating a false positive, becomes one of the primary detection quality measures. For the same reason, if a detector is capable of detecting several types of anomalies, its detection quality and usefulness can only be judged reasonably after the detection results have been split into the different types, since different anomaly types carry different levels of importance.

Of course, specific anomaly detectors can be combined into clusters of detectors for specific applications. If, for example, a network is vulnerable to flooding-type DoS attacks and to fast worm propagation, then a combined detector can be used for overall alerting, and the human operator can then be provided with the specific information on whether a DoS attack or a worm was detected.

This work aims to provide a detection mechanism for fast scanning Internet worms, but no other anomalies. Fast Internet worms are an interesting subject in their own right, since they can have a massive negative impact, not only within their intended purpose, but also as a side-effect due to their propagation activity. Currently, no other mechanism besides fast Internet worms can compromise hundreds of thousands of Internet hosts in mere hours or even less time, without previous network activity that could give early warning. Fast Internet worms represent a significant threat to the Internet, even if the observed number of incidents is relatively low. Within the set of fast Internet worm models, we will be able to distinguish different TCP/UDP port profiles and scanning intensities. We will also conduct detector calibration on different worm models.

We will use a computationally cheap detection approach, that allows deployment of a larger number of differently calibrated individual sensors, even for very fast networks. The computational effort will be linear in the number of sensors used, with a very small constant. In order to demonstrate efficiency

and a low false positives rate, we will evaluate the detector design on a significant set of unsampled and not anonymised network data from a medium sized Internet backbone, spanning half a year. The data was obtained from SWITCH (Swiss Academic and Research Network, [6]) as part of the DDoSVax project [35]. The SWITCH network carried around 5% of all Swiss Internet traffic in 2003 [75]. The DDoSVax project uses unsampled flow-level representation data of all SWITCH border routers for research purposes, for example for worm detection, detection of DoS attacks and measurements of P2P (Peer to Peer) traffic generated by file sharing applications.

1.2 Anomalies During Worm Outbreaks

Any working worm design has one central characteristic: Each infected host tries to infect several others. This results in a one-to-many connection profile. If the worm cannot determine in advance which connections will be successful, most of the ensuing connection attempts will be unsuccessful and hence introduce an asymmetry into the network traffic: A smaller number of IP addresses (the ones of infected hosts) will be seen more often, while a larger number of IP addresses (the target addresses) will be seen only once or a few times. This change is comparable to the pattern seen when host-scanning is performed from a small number of hosts to a large IP address range.

In addition, the infection traffic has specific characteristics with regard to port numbers, as typically the attacked vulnerability in the target system needs connecting to a specific TCP or UDP port. During a worm propagation phase, the specific port is then seen as a target port in more network connections as usual. Unlike the IP address profile, there are (rare) vulnerabilities that do not need a specific target port, for example because the attack is against a part of the network stack directly. An example is the vulnerability used by the Witty worm [115], were a specific flaw in a firewall product was used. For the source ports, a worm implementor has the choice of using a variable of fixed source port. Both choices have been seen in deployed worm implementations [37, 73, 102, 115].

The described effects may have a significant effect on the entropy of the IP address fields and port number fields found in captured data. This thesis will show that for worms with a high enough scanning intensity (*fast* worms) the entropy changes can be used to build a practical detector, that is both accurate and efficient enough for deployment in networks carrying a high traffic volume. It will be demonstrated that both fast detection and a low

number of false positives can be obtained simultaneously on real network traffic gathered from a medium sized Internet backbone network.

1.3 Problem Statement

The central claim of this thesis is that it is possible to build a detector for fast Internet worms based on entropy characteristics of backbone traffic observed only at flow-level. This detector can detect fast Internet worms early, has a low rate of false positives and low resource needs.

Demonstrating that this central claim holds, requires a number of engineering and scientific contributions to be completed successfully. In detail these are the following:

Design, build and operate a NetFlow data capturing system for the SWITCH network (Engineering)

In order to obtain the basis data for this research, we created a system for capturing, transport and storage of the SWITCH NetFlow data. At the time this work was started, SWITCH used the router-exported data stream only for real-time monitoring tasks, such as generating traffic volume statistics, but no short- or long-term storage was done.

Design and implementation of NetFlow data processing libraries and tools (Engineering)

No NetFlow processing tools suitable for our purposes were available. We needed batch processing capability for large datasets, with a focus on statistical and other analyses. Therefore we created our own tool-set that works primarily on files of NetFlow v5 datagrams. Processing of datasets that exceed days or weeks of captured data is possible in an efficient way with this tool-set. It also serves as a basis for other work on the stored SWITCH data, see Chapter 8 for a selection.

Creation of a worm simulator to better understand worm characteristics (Engineering / Science)

In order to better understand the impact of worm parameters such as, for example, worm code size, scanning speed and infection delay, we created a

simulator that allows us to study these effects. A central problem we solved was how to model the Internet realistically with regard to worm propagation simulation.

Model the entropy-effects of fast Internet worms theoretically (Science)

We created a theoretical model that explains the impact of worm outbreak behaviour on flow data entropy characteristics. This model serves as a basis to explain and understand what effects are to be expected during Internet worm outbreaks and facilitates the search for them.

Identify and quantify the effect that the outbreak of a fast Internet worm has on NetFlow dataset entropy scores (Science)

In order to allow the design of an entropy-based worm detector, we studied the effects of real worms on NetFlow data entropy characteristics. This involved both measurements on real data as well as theoretical observations. This gave us conclusive insights into the actual entropy-related effects of fast worm propagation.

Evaluate the suitability of compression for entropy estimation (Science / Engineering)

Since calculating sample entropy for backbone traffic data is a memory intensive task, with memory needs dependent on the traffic volume, data compression was evaluated as a possible alternative method for estimating entropy characteristics of flow-level traffic data. We found that compression forms a valid alternative to sample entropy. Compression has significantly lower resource needs and a slightly higher rate of false positives.

Design a detector for fast Internet worm outbreaks based on entropy measurements and evaluate its characteristics (Science / Engineering)

We designed, implemented and evaluated a detector for fast Internet worms. The detector was evaluated on real traffic data. A focus of the evaluation was on the detector calibration method, and on the number of false positives generated for different calibrations on a significant portion of the SWITCH traffic data. We found that the detector can deliver both early detection and

a low rate of false positives. This is the case when using sample entropy and when using entropy estimated by compression as basis.

1.4 Thesis Overview

Chapter 2 gives an overview of related work in the areas of anomaly detection, entropy estimation and worm detection, as well as worm simulation. Chapter 3 focuses on worm traffic and its specific characteristics. Target scanning strategies are discussed and our approach to simulating worm propagation in the Internet is presented. In Chapter 4 we discuss the concept of entropy, important properties for our work and which parameters of flow-level traffic data are suitable for an entropy analysis. The chapter finishes with some examples of observed entropy changes during worm outbreaks. The main focus of Chapter 5 is the possibility to estimate entropy. We discuss direct estimation by observed frequencies and estimation by compression. A quality and performance evaluation of the different possibilities is given. Chapter 6 presents an entropy based detector design that is real-time capable. The detector works on time-series, i.e. data is separated into intervals and detection is done for each whole interval. In order to demonstrate that the approach is effective, Chapter 7 presents validation measurements for entropy estimation by compression on flow-level traffic data from the SWITCH backbone network, as well as detection results and evaluation of false positives on half a year of recorded SWITCH traffic data. The scientific part of this thesis concludes in Chapter 8, where the accomplishments are reviewed, the relevant publications written as part of this thesis are presented and possible directions for future work are given.

The engineering contributions are documented in the Appendices. Appendix A documents the libraries and tools created as the software infrastructure for NetFlow data processing, that served as the basis for most measurement work done in this thesis. We describe design objectives, technical solutions, performance figures and briefly compare our approach to other tool-sets. Appendix B briefly describes experiences made with a Linux cluster of 24 PCs that was created as part of this thesis to serve as the processing infrastructure. Appendix C describes the data capturing and storage system and the specific problems encountered together with their solutions. A special focus is fault-tolerant operation, since the data capturing system is intended for multi-year continuous operation. The SWITCH network is also briefly described.

Chapter 2

Related Work

In order to set the playing field, we will now review a selection of the relevant results and publications. Note that in most cases additional literature references can be found in the relevant chapters.

2.1 Fast Internet Worms

One of the first to recognise the immense threat worms are to the Internet is N. C. Weaver who coined the term "Warhol Worm" [114] for very fast worms. This work is extended in [97].

Many past fast Internet worms have been analysed in detail. Examples are the Code Red worm variants in [34, 80, 121], the Blaster worm in [12, 17, 37, 39], the Witty worm in [90, 93, 104, 115] and the Nachi worm in [11, 13, 44]. In [123], Zou et al. analyse different scanning strategies suitable for fast Internet worms with regard to their performance, similarities and differences.

2.2 Worm Simulation

Generally speaking, the Internet is difficult to simulate, a good overview can be found in [42]. One problem is that the Internet traffic mix changes relatively fast. For example P2P (Peer-to-Peer) traffic, often for filesharing applications, forms a major part of todays Internet Traffic [51, 109], but was not a major concern a few years ago. An other problem is that getting an overview

of the current Internet topology is difficult, even if the speed of the individual connections is ignored, because this would require obtaining detailed information from every backbone operator and every ISP on the planet. Still, with a narrow focus the task becomes easier.

2.2.1 Modelling the Internet

Simulating the spread of a worm on the Internet requires parametrisation for the worm and the Internet, including vulnerable host population, infection speed and so on. The choice of these parameters directly impacts the level of realism a simulation has. It should be noted that the relevant Internet characteristics change over time. Each set of parameters is valid only for a specific time period.

The Simple Epidemic Model

A first approach is to use a simple epidemic model, that is parametrised with the population size, the vulnerable population, the number of contacts per time unit and the infection probability. For worm simulation, the infection probability for a vulnerable target is typically one. The Internet is not explicitly modelled and full connectivity between each pair of hosts is assumed. As a consequence, a realistic set of parameters can generally only be derived from observations of worm outbreaks in the real Internet or from more complex simulations or analyses. This limits the usefulness of this type of model for research purposes. It is still well suited for demonstration purposes. A simulator that uses this model is [122]. An epidemiology-based simulation of the Code Red worm can be found in [121]. In [97], a variant of the simple epidemic model is used to simulate worms with two scanning stages, where an initial hit-list scan is followed by a different scanning strategy once the hitlist is exhausted.

The Last Mile Model

A more complex approach models the Internet by using speed and delay of the last mile connections. The worm is modelled by its scan strategy, the number of bytes to be transferred for successful infection and by the delay inherent in a system compromise. These values are then used to parametrise a simple epidemic model. The advantage is that once the characteristics of the worm are known and a certain last mile speed profile of the Internet is

selected, it is possible to simulate the corresponding worm outbreak realistically. This approach allows exploring the outbreak behaviour of different worms under different conditions. The worm characteristics can be derived from measurements in a test-bed or from more theoretical observations. The last mile Internet speed profile can be derived from Internet measurements. We use this type of model in our own Simulator and derive its parametrisation from global P2P filesharing application speed surveys. Chapter 3.7 presents the details and results.

The Scale-Free Model

A scale-free network is one where the distributions of the degree of nodes (i.e. the number of neighbours) follows a power law. Basically there are few nodes with a lot of neighbours and also many nodes that have few neighbours.

There is a lot of work on worm propagation in scale-free networks, for example [21,67,79,113]. The basic problem is that, concerning global scanning strategies, for example random scanning, the Internet is a fully meshed network where every host can reach every other directly by using the IP address of the target. For this reason, the scale-free network model has only very limited applicability to worm propagation, unless the worm uses a topological scanning strategy (see Section 3.4.4).

Topological scanning works well for email worms, Instant-Messaging worms and similar worms, that use a local address-book to select targets for infection. However, fast Internet worms typically use random target selection, sometimes combined with an initial hitlist. It is currently unknown whether topological scanning can achieve propagations speeds comparable to observed fast Internet worms. Primary requirements would be a very fast local search for targets, and the local availability of a lot of target addresses in many cases.

Agent-Based Simulation Models

A third possibility is to actually simulate individual hosts and network connections between them. This can be done in an agent-based framework, where hosts are stationary agents sending each other messages to represent infection attempts. Hosts are initialised with their probability to be infected and a time until they start infecting others as well. The worm is its scanning and exploit profile with regard to data transfer size and needed exchanges over the network. More complex parametrisation is possible, for example a probability

for an infection attempt to crash a host or a possible mix of different operating systems with a similar vulnerability that have different timing characteristics when exploited. This Internet model also allows simulation of faster local connections and, if Internet topology is added, simulation of congestion effects. It is very resource intensive and simulations may have to be restricted to a fragment of the Internet size or a maximum number of vulnerable hosts. It is also difficult to obtain realistic parametrisation information with regard to the required level of detail. This type of Internet model is used for example in [118].

The Internet as a Test-bed

The most realistic option is to replace simulation with the Internet as the test bed. Typically this is done using generation limited worm instances. The results provide current and realistic viability information about a specific worm design and also serve to test a concrete implementation. This approach has the advantage of no simulation error, but is completely inflexible with regard to exploration of different scenarios. In addition, it causes significant damage, is unethical and typically criminal and therefore is not a valid research option. It is however frequently used by worm authors. Evidence of this are outbreak-like traffic patterns that have been observed hours or days before large outbreaks of fast Internet worms in the past. An example can be found in [37].

2.3 Anomaly Detection for High-Volume Networks

Anomaly detection in Internet Backbone networks is difficult because of the amount of traffic involved.

2.3.1 Black Hole Sensors

A way to deal with these problems is to operate a ”black hole” sensor. This type of sensor does not monitor backbone traffic, but rather traffic to a large unused address space. Black holes detect traffic directed to random addresses or to a fraction of all Internet addresses that includes the black hole. Host scans and systematic searches for specific vulnerabilities in hosts often cause

this type of traffic. Black holes can also detect backscatter traffic, for example caused by responding packets to flooding attacks with spoofed source addresses. A black hole is typically implemented by using a router configuration that redirects all packets addressed to a specific address range to a single machine. This machine counts and classifies the traffic, but does not answer. One example of a large black hole is the CAIDA [26] black hole, also used for worm analysis [2, 90].

An variant of black hole sensors is a white hole installation. It works similar to a black hole, but in addition has a countermeasure capability. For example, malware that does *importance scanning* (i.e. tries to identify the distribution of a host population with a specific vulnerability and then prioritises its attack activity accordingly) can be slowed down using white holes [49].

2.3.2 Connection Counting

It is possible to build sensors that rely on a connection counting approach, where the number of outgoing and incoming connections of a set of hosts is monitored. This approach causes relatively high effort, since state has to be kept and maintained for each monitored host. A second drawback is that both directions of each network connection have to be available, which is not always the case in a backbone scenario, due to asymmetric routing.

An early approach is the Network Security Monitor [52]. In [101] hosts are classified into different classes called *Connector*, *Responder* and *Traffic*, that correspond to hosts that initiate a lot of connections, have a lot of incoming connections and have both, respectively. This classification is then used for anomaly detection and can, for example, be used to identify network worms and email worms that send email directly from the compromised hosts.

2.3.3 Origin-Destination Flows

Origin-Destination (OD) flows are a method in general traffic analysis, and have been used for highway traffic patterns, flows of goods in logistics, migration patterns, and other applications, where traffic originates in a set of points and goes to a set of points. It is based on estimation theory (e.g. [57]). The traffic flow is individually measured for each pair of (Origin, Destination) and the results are represented in matrix form. This matrix can then be used to estimate normal flow activity by different methods, for example least-square estimation.

In [58] authors apply OD flows and Principal Component Analysis (PCA) to the problem of detecting general network anomalies. To achieve this, they use network entry points as origin as well as destination and summarise traffic by calculating sample entropy scores for port and IP address fields. The resulting four sets of sample entropy values are each put into a separate matrix per 5 minute measurement interval. In a second step, a multi-way subspace method is used to estimate a traffic characteristics baseline. Anomalies show up as multi-dimensional deviations from the baseline. The anomalies are then clustered using unsupervised learning. Identifying the nature of the anomalies in a cluster is done manually. Measurements given in [58] found 444 traffic anomalies in three weeks of Abilene data, containing 43 false positives and 64 events that could not be classified by manual analysis. Injected worm traffic could also be detected. Abilene data is sampled at 1 out of 100 packets and the last 11 bits in IP addresses are zeroed.

In [95] the authors show that OD flows are superior to input link aggregation and input router aggregation as traffic data aggregation mechanism for traffic data from the Abilene and GEANT networks. GEANT traffic data is sampled at 1 out of 1000 packets. In [87] the sensitivity of PCA for network traffic anomaly is examined. One identified problem is that the number of false positives in PCA-based detection is very sensitive to dimensionality of the normal subspace and the threshold calibration. This induces the problem of over-fitting, especially when measurements are done on relatively small sets of data, for example traffic data from only a few weeks of observation. A second problem is that large anomalies can contaminate the normal subspace and may not be detected as a result. Measurements in [87] were done using one week of data form the Abilene and GEANT networks. It is unclear how well the measurement results and the OD approach transfer to other networks and what impact sampling and anonymisation have on the findings.

2.4 Worm Detection

A lot of work has been done on worm detection in the recent past. In [89], the authors describe a fast detector for scanning worms in local networks. It uses sequential hypothesis testing to identify infected hosts. SWorD [38] is a fast worm detector based on a counting algorithm and intended to be used on the network edge. It keeps a list of each host that has raised attention by generating scan-like traffic. An approach based on detectors in local networks that uses a Kalman filter to detect exponential behaviour is described in [120]. A

distributed worm detector, that uses sensors on individual hosts is the subject of [27]. Billy Goat [119] is an intrusion detector for corporate networks. It analyses traffic to unused IP addresses in order to identify infection attempts. There is also significant work on automated worm signature generation using honey pots. One example is [81].

2.5 Entropy

The fundamental reference for the concept of entropy in information theory is [91] by C. E. Shannon who defined the concept first. Information theoretic entropy (entropy for short) intuitively describes how much uncertainty is in a data-stream, i.e. how much information (bits) are needed to describe the variations in the data stream that cannot be predicted. Entropy is measured in bit/bit or bit/symbol. In digital computing bit/bit, i.e. a unit-less measure in the range $[0 \ldots 1]$ is often used. If bit/symbol is selected instead, the symbol set has to be described in order for the measure to be meaningful.

Information theoretic entropy was inspired by the concept of entropy from the second law of thermodynamics, first described by Rudolf Julius Emanuel Clausius, a German physicist and mathematician, in 1850. Clausius also defined the concept of thermodynamic entropy in 1865. The second law of thermodynamics intuitively states that energy spreads out and disperses if not specifically hindered to do so [59], i.e. in a closed system after an infinite amount of time all the available energy is evenly distributed in the available space. This final state is regarded as having maximum entropy. The connection to information theoretical entropy is that the evenly distributed energy state may be seen as the space being filled with random “noise”. In information theory, the signal from a source without memory that has an evenly distributed output over its symbol set also has maximum entropy and is basically “noise”.

Information theoretical entropy is closely connected to data compression in the sense that no lossless stream compressor can achieve a compression result smaller than the entropy of the original binary data stream. The entropy here has to be measured in bit of entropy per bit of data.

When compressing a fixed binary object, Kolmogorov complexity [63] is the dual concept to entropy. The Kolmogorov complexity of a binary object basically describes the smallest possible representation in terms of the size of the description of an algorithm that generates the object. Kolmogorov

complexity can not be measured for a specific binary object, as that specific object could be hard-coded into the language the algorithm is written in. On the other hand, the average Kolmogorov complexity of an infinite sequence of binary objects is equal to the entropy of the sequence.

In [60] entropy was used to analyse audit data (e.g. sendmail logs) for anomalies. The authors also analysed tcpdump data from a small access network at MIT. One of the main results is that entropy measures lead to a high false-positives rate when applied to smaller data sets. The authors argue that, for example, system call trace data on a target system is far more likely to detect a buffer overflow exploit being used than entropy data on the network traffic.

2.5.1 Entropy Estimation

Entropy can be directly computed when the probability of each individual symbol in a data stream without inter-symbol dependencies (i.e. generated by a source without memory) are known. The direct way to estimate entropy is to assume independence and then estimate symbol probability by observed frequency. This method is called *sample entropy*.

A second approach is to use data compression methods, in the hope that the compression algorithm will achieve a compression ratio close to the (binary) entropy of the data. This method generally fails if there is hidden structure in the data.

Current research on data compression mostly focuses on lossy compression of multimedia data, exploiting imperfections in the sensors (e.g. human eyes and ears) of the final data consumer. These techniques are not useful for entropy estimation outside of their specific field of application and are therefore outside of the scope of this work.

Work on estimating entropy has been done in other fields. For example the Entropy Estimation workshop at NIPS'03 [76] deals with entropy estimation for use in fields like Bioinformatics and speech processing with a specific focus on sampled data. While compression methods have been studied in these fields, the processed datasets are far smaller than in our work, but usually have significantly more structure so more complex estimation methods are used.

2.5.2 Compression in Data Analysis

In [116] compression signatures of worm code are used to identify worm families. A worm family consists of worms sharing a significant part of the same code basis. The author also uses the method to do anomaly detection in ssh connections and other types of network connections. The work is extended in [117].

In [105] the Authors apply data compression techniques in order to identify data clusters in fields as diverse as music, genetics, virology, language and others. The presented clustering techniques are general, and try to identify similarities between data subsets. The examples given use smaller and relatively strongly structured data sets.

Chapter 3

Worm Traffic

This chapter defines the notion of an Internet worm and discusses propagation mechanisms and strategies. Except for the occasional example, we do not discuss specific worms in this chapter, only concepts and mechanisms. Chapter 7 includes descriptions of several worms that initially broke out in 2004. Chapter 2 discusses relevant publications. We are primarily concerned with fast Internet worms, i.e. worms that reach initial saturation (i.e. infection of most vulnerable and reachable hosts) in a matter of hours.

3.1 Definitions

Note that definitions different from the ones we give here are in use.

Definition 1 *An* **Internet worm** *is a piece of self-replicating code that does its replication over the Internet, i.e. the target is accessed using a layer 3 or layer 4 protocol, typically TCP or UDP. In order to propagate, the host on which the worm code is executed (called* **infecting host** *or* **infected host***) contacts an other host (the* **target host***) over the Internet, replicates its code onto the target and triggers execution (***infection***) of its code on the target. We will sometimes call the running copy of the worm code on the target a* **child** *or* **child instance** *of the worm. All hosts that have the same number of infection steps from the initially infected host(s) are called an* **(infection) generation**.

While in principle worms that propagate using the ICMP protocol or using raw IP in some fashion (i.e. where the protocol field in the IP header is ignored or not used) are possible, we are not aware of any worms that use these means to propagate.

A distinction that is sometimes made is between the notion of a *worm* and a *virus*. The idea is usually that a worm can propagate without user interaction, while a virus cannot. We do not make this distinction. In a sense we allow the user to be part of the execution environment. In this way our definition includes email worms and other application worms that require a user on the remote host to open an email attachment, for example, in order to trigger worm code execution.

Definition 2 *A* **fast Internet worm** *is an Internet worm that infects most of the vulnerable (reachable) host population in less than a day.*

We are aware that this definition is not too precise. Nonetheless we are not aware of a better one.

Definition 3 *The* **initial outbreak** *(or just* **outbreak** *for short) of a fast Internet worm is the time from its first infection over the Internet until it reaches saturation.* **Saturation** *is typically reached when around 90% of the vulnerable host population that is reachable has been infected.*

Again, the term *saturation* is not too well defined. Intuitively it is reached when the target selection strategy of the worm produces mostly unsuccessful selections, since most vulnerable hosts have already been infected.

3.2 General Worm Mechanisms

Every Internet worm has to have a certain minimal functionality in order to be viable:

- A worm has to be able to identify possible infection targets.
- A worm has to be able to transfer its code to a selected target.
- A worm has to be able to induce a vulnerable target to run the transferred worm code.

- A worm should be able to identify already infected targets and refrain from re-infecting them.

Interestingly, the first three requirements are already enough. The fourth merely improves efficiency. Also, if a service on the target system is capable and willing to propagate the worm without having its security compromised, then a worm can do without any kind of system compromise at all.

A compromise of the target system to some degree is customary nonetheless, especially when some other purpose, like espionage, sending of spam or attacks on other systems is intended. A second reason for target system compromise is that many worms use security vulnerabilities to obtain resources on the target system. The advantage is that in this way the basic execution services of the target system become available to the worm and any functionality its designer wants can be easily implemented.

The typical worm uses a propagation mechanism that works like this:

1. Select a potential target
2. Attempt to contact the target
3. Compromise the targets security in some way to obtain the resources to transfer and execute a copy of itself.
4. If more infections are desired, goto step 1
5. Do damage on the local machine or do damage somewhere else using the local machine

The last step is optional and can also be done earlier. However, in order for a worm to propagate as fast as possible, it is a sound design choice to not impair the functionality of an infected host until the worm has completed most or all of its intended propagation activity from that host. In addition, the damage may be done later to delay the discovery of the worm or in order to allow coordinated attacks from several infection generations.

Note that we also regard data collection activities, such as looking for passwords or credit card numbers, as causing “damage”.

3.3 Infection Mechanisms

The primary requirement for propagation is, as stated, that a worm can transfer code to a target and induce the target to execute that code with a permission

level high enough that further propagation from the target is possible. Often this is a multi-stage process: First the worm transfers a specific piece of data, that causes an initial execution of some worm supplied code. In order for this to work, the worm has to use a **remote exploit**, i.e. a vulnerability that can be exploited over the network. The next steps will then be used to execute more complex code and optionally to achieve a **privilege elevation**, i.e. execution with higher privileges. For the latter, the worm needs to use a **local exploit**. i.e. a vulnerability that can be exploited locally and that increases the level of control that the worm has over the target host. We will not discuss the possible types of remote or local exploits here. Instead we refer the reader to the literature as discussed in Chapter 2.

3.4 Target Selection Mechanism

Of primary interest to this thesis is the target identification and selection mechanism a worm uses, since target selection has by far the largest influence on the actual worm traffic seen in the Internet during an outbreak. The reason is that, while the target address may or may not be assigned to a host and if there is a host, this host may or may not be vulnerable, the worm code has to select targets and then try to contact them. These connection attempts, also called **scan traffic** is the most visible sign of a fast Internet worm in its main propagation phase.

3.4.1 Random Scanning

Perhaps the most simple target selection strategy is purely random scanning. For this, the target selection code usually includes a Pseudo Random Number Generator (PRNG) or uses an OS service with this functionality. Infection targets are then selected by generating a 32 bit random number and using that as the target IP address. In a more advanced setting, ranges that do not contain normal hosts, such as multicast-addresses, can be excluded.

Care needs to be taken, that the random target selection is implemented correctly. Interestingly, many worm writers seem to get this wrong [2, 73]. Mistakes include constant PRNG seeding after propagation, use of inferior PRNGs with non-even value distribution and even PRNGs that cannot generate all output values and hence miss many possible targets.

3.4.2 Local Preferential Scanning

Pure random scanning works reasonably well, but one disadvantage is that it does not take advantage of the better network connectivity to hosts in the same LAN or otherwise in close proximity. Local-preferential scanning is very similar to random scanning, but it dedicates a portion of the scan activity to addresses in the same subnet the attacking host is in. Typical implementations have preferences for the /24 subnet and the /16 subnet of the attacking host.

One way to implement this type of strategy is to randomly scan in more often in the local /16 subnet, but to scan the local /24 subnet fully. The latter can be done in a simple, linear fashion, although this may trigger IDS and/or IPS system sensors.

Local-preferential scanning has several advantages. One is that the probability of actually finding hosts with addresses close to the attackers IP address is usually far higher than for randomly selected addresses. After all, the local subnet contains at least one host already, namely the infected host. This means that it is not an unused subnet. The second advantage is that the traffic over the Internet access and backbone networks is reduced. Pure random scanners run the risk of overloading the Internet access connection and thereby hindering their own propagation. A further advantage is that the network latency to hosts in close proximity is lower, leading to faster scanning and infection performance.

3.4.3 Hitlist Scanning

A completely different approach to random scanning is hitlist scanning. To implement this strategy, the worm-designer precomputes a list of vulnerable targets. This list is then included in the worm when it is deployed. The worm then not only propagates its own code, but also parts of the hitlist to be used by the respective child instance. Propagation schemes with some degree of redundancy are possible. For example, each so far unused target address could be propagated to two or several child instances of the worm, so that if a child instance cannot work through its list fragment completely, some other child instance may still be successful. With this type of redundancy the individual copies should be worked through in different orders to maximise propagation speed.

The use of a hitlist scanner for the full vulnerable population for a specific exploit only makes sense if this population is relatively small. Otherwise the

transfer of the hitlist will slow down the worm considerably. A second concern is that the hitlist needs to be obtained in a way that does not arouse suspicion. Otherwise the worm could find a situation were the potential targets have already been warned before its initial propagation.

A typical use of hitlist scanning is to have a relatively small hitlist of very attractive targets, e.g. hosts with high bandwidth or host that are geographically well placed. The worm then does its initial propagation with a hitlist strategy and then changes over to another strategy after one or a few infection generations, e.g. random scanning. For a good discussion of how fast a hitlist-scanner could actually be, see [98].

3.4.4 Topological Scanning

Topological scanning bears some resemblance to hitlist scanning. However, the information about potential targets is not precomputed, but instead extracted from the data available on the local host. Possible sources of IP addresses are ARP caches, contact lists of P2P applications, open Internet connections, browser caches, address books of any kind and other sources. Host names and URLs can also be used since they can be converted to IP addresses by DNS lookup. It should be noted that worms that do DNS lookup will generally be quite slow and likely not qualify as fast Internet worms according to our definition.

One primary example of topological worms are email worms. Although they are not *Internet* worms by our definition, they represent a very important class of application layer worms. Another class of application layer worms are IM (Instant Messaging) worms, that have also been observed in the wild. P2P filesharing could provide an equally viable platform for application layer worms, but so far no P2P worms have been observed as to our knowledge.

3.5 Port Characteristics

Scan traffic of a fast Internet worm has some limitations on how source and target ports can be selected. These are different for TCP and UDP scan traffic. We will now discuss the different possibilities.

3.5.1 TCP: Source Port

In ordinary TCP traffic, the source port for the connection initiating host, i.e. the host that sends out the initial SYN packet, is chosen at random by the network stack from a port range unlikely to be used as server ports. Each concurrent connection gets its own source port, so that answering traffic can be identified by the port it is sent to. It is possible to drop this requirement and match answering packets by remote IP address and port. This is, for example, done in servers that accept multiple connections on a single port, such as web servers.

For a worm, it would be possible to use a static source port and match the answering traffic by remote IP address. However, this causes additional effort and does not have any real benefit. It also prevents the worm from using the normal network stack, since the normal, OS integrated network stack cannot do this type of matching.

3.5.2 TCP: Destination Port

The primary limitation for destination port selection in a worm is the exploit used. If an exploit works only on a specific port, then all attack traffic has to be addressed to that port. In addition, the connection initiating SYN packet in a TCP connection is unable to transport data. Even if a port independent exploit was possible, the initial SYN would have to be sent to a port where the remote system sends an answer. With variable ports, the worm would need to do a port scan in order to find such an open port. This scan would slow the worm down significantly. In addition we are not aware of any TCP exploits that can be used on a larger range of target ports.

For these reasons a worm using one or more TCP based exploits will likely target one or a small number of TCP ports on the target system.

3.5.3 UDP: Source Port

Since UDP is connectionless, UDP based exploits can be and usually are single-packet exploits. This means the attacking host sends a single packet to the target host and is then either contacted back by the successfully executed exploit code or has to do a second polling step. For both options, the UDP source port is immaterial and can be chosen in an arbitrary fashion.

3.5.4 UDP: Destination Port

As in the case of TCP, the target port for an UDP exploit depends on the actual nature of the exploit. If the vulnerability is present in a service running on a specific port, the same rationale as for TCP destination ports applies and the target port will be fixed.

Unlike TCP, UDP permits transfer of data in the first packet sent. This allows exploit code to be sent to random destination ports, since establishing a connection is not needed. In order for this to work, the vulnerability needs to be in a service that processes all UDP payloads, such as a firewall or a proxy. For example, the Witty worm (see Section 7.2.2), exploits a vulnerability in a firewall product and sends single attack packets to random target ports.

3.6 Expected Impact of IPv6

IPv6 offers a 128 bit address space [56, 86]. It is not quite clear how much structure will be contained in addresses actually assigned to hosts in the future. For example only one eighth of the address space is currently assigned to global unicast addresses. Furthermore 64 bits may be used for interface identification, e.g. to hold the MAC address of an Ethernet interface. In case of a 48 bit MAC address, there are significantly less than 48 bits of randomness in these 64 bits, although the structure is not very simple. Still we expect that random scanning will be ineffective with wide deployment of IPv6. One possible way around this problem is topological scanning as discussed before. In fact, topological scanning is already in use by email worms and seems to work reasonably well for them. It remains to be seen how effective such mechanisms are and whether worms can achieve fast propagation speeds under IPv6 without resorting to large hitlists.

3.7 Simulating Worm Traffic

As part of this work we examined the possibility of simulating Internet-wide worm traffic. The results have been published in [110]. We will describe and briefly discuss them now. The simulation code and implementation documentation is available from the author of this thesis upon request.

3.7.1 Why Predict Worm Behaviour?

The benefits of predicting worm behaviour are numerous:

- Better understanding of the behaviour of worms observed in the past
- Estimations of a worm's threat potential
- Estimations of the impact of future worms on the Internet
- Forms a basis for the design of worm detection mechanisms
- Determination of parameters relevant for worm characterisation

Traffic Prediction

Traffic prediction for the worm spreading phase helps to estimate the decrease in performance of an affected network. Slow spreading worms might not even be visible in traffic monitoring tools as they are well hidden in regular traffic variations. However, if specific characteristics of a worm are known, a detection might still be possible.

Speed Prediction

The Sapphire worm infected more than 90% of all vulnerable hosts in the Internet within 10 minutes [73]. Since manual intervention is too slow to deal with this, there is a need for semi- or fully-automatic tools that detect and analyse a spreading worm and activate countermeasures in near real-time.

Threat Evaluation

Given that modern worms have the potential to infect most vulnerable hosts in the Internet within a short time, these worms pose a real threat to the Internet infrastructure. It is important to determine what the possibilities and limitations of this attack tool are in order to concentrate countermeasure efforts towards the most vulnerable places.

3.7.2 Worm Characteristics Relevant for a Simulator

A worm writer basically implements the following process:

1. Identify a vulnerable host
2. Compromise the target host
3. Transfer the worm and activate it

For some vulnerabilities all these steps can be combined into a single network packet, as was done in the case of the Sapphire worm. For others, the steps have to be done separately.

We believe that for the study of worm propagation a very abstract view of these steps is sufficient. Steps 2 and 3 can be modelled as an exchange of a specific amount of data with a specific protocol and optional time delay, i.e. disregarding the concrete nature of the vulnerability used for target compromise. Step 1 is a little more complicated, but can still be modelled disregarding vulnerability details.

TCP vs. UDP

The main choice in the transport protocol is whether it is connection-oriented or not, for simplicity, this is represented by TCP and UDP. For worms that infect a distributed application, like a P2P system, other models might be needed [106]. The protocol used is usually directly determined by the vulnerability that is exploited by the worm.

In the case of UDP, resource consumption in the attacking host is small. A typical scenario is to send out UDP packets to random hosts, while keeping very little state information for each target, or none at all if the attack can be executed by sending a single UDP packet. Disadvantages are that the size of a UDP packet is constrained to around 50 kiB[1] and data packets with a payload larger than 1472 Bytes will be transported using IP-fragmentation.

Use of TCP causes additional effort for connection establishment and error handling. On the plus side there is no data size limit. The most significant disadvantage of TCP is that a connection attempt to a non-existing host fails only after a long timeout and consumes OS resources until it does. There are ways around this, but they require that the worm implements its own modified version of TCP, which makes worm design more difficult and increases worm size.

[1] This is OS dependent. We found that e.g. Solaris has a limit around 50 kiB, Linux a little higher. 64 kiB is the definite protocol limit.

Amount of Data Transferred

The time a worm needs to propagate after a vulnerable target has been identified depends mainly on worm size and available bandwidth. Additional delays may be present, e.g. if a reboot of the attacked host is needed. Data transfers form a specific signature of a worm and can be used for detection purposes. Obviously, a large worm will generally propagate significantly slower and far more visible, so worm writers will often aim to write small worms.

Scanning Strategy

The scanning strategy is the method used to select the next host to be probed. See Section 3.4 for a discussion of the possible options.

Latency vs. Bandwidth Limit

Even though the Code Red I and Sapphire worms both used random scanning, their propagation speed was different by several orders of magnitude. The number of Sapphire infected hosts doubled initially every 8.5 seconds while the Code Red Iv2 worm population had an initial doubling time of about 37 minutes [73]. The reason for this difference lies in the choice of the transport protocol and in the size of the transferred worm code.

The Sapphire worm uses a single UDP packet with a total size of 404 bytes. Since there is no connection establishment with UDP, the spreading speed is mostly independent of latency but strongly dependent on bandwidth. An infected host can send as many infection packets as its network link and protocol stack allow.

Code Red uses TCP, which implies the use of a three way handshake for connection establishment. As a consequence, latency is the main limit on propagation speed. In addition OS constraints limit the number of parallel connection attempts that can be made. Latency limited worms can also become bandwidth limited when their scanning traffic exceeds network resources. For Code Red this happened after about 15 hours.

3.7.3 Simulation and Alternatives

We will now discuss different ways to study the characteristics of a piece of self-propagating code.

Mathematical Models

The most powerful approach is probably the creation of a realistic mathematical model that allows behaviour prediction in a closed form, i.e. with no or very little iteration. The problem with this approach is that such models are not generally available and are usually hard or even impossible to create.

Testbeds

Testbeds allow to actually run self-replicating code in an isolated and limited environment so as to observe its behaviour. The most obvious limit of a testbed is that it cannot be created in a size approaching the size of the Internet. Another serious problem is that a testbed needs to use real self-propagating code, which is difficult to obtain. There are also legal and moral problems with creating and handling such code.

Real World "Experiments"

If a testbed is too limited, why not use the Internet itself? While worm code authors certainly take this freedom, this is not an option for scientific study because of the damage potential. In a very limited sense the use of the whole Internet is possible, namely in observing the behaviour of worms that have been set free by people not hampered by ethical considerations. We have observation equipment in place in a moderately sized backbone network to observe the next Internet outbreaks. See Appendix C for details.

Simulation

In a sense simulation is a mathematical model in which some of the functions used rely heavily on iteration. In order to reduce computational complexity, abstraction and approximation of the inner mechanisms of the object studied is often used. This allows computation of functions that are not well understood in a mathematical sense. The analytical approach of mathematical modelling is replaced with an experimental approach, in which scenarios are simulated and then analysed. Simulation is often a very effective tool to understand complex processes.

A significant drawback of simulation is that due to abstraction the simulation results can differ significantly from real behaviour of the system under study. A way to verify and optimise simulation accuracy is to simulate events

that have been observed in the real system and compare simulation outputs to the measured data.

3.7.4 Simulator Design

The main component of the simulator is a script written in Perl. The simulator can be started from the command line. It was developed under Linux, but should run under most Unix-like operating systems without modification. It first reads the parameter values and then opens two plot windows. The *speed plot* shows the number of infected hosts vs. time and the *traffic plot* shows the total scanning and infection traffic vs. time. Plain text output is also available. The simulator code is available upon request from the author of this thesis.

Simulator Structure

Our aim was to create a modular and flexible simulator that can easily be extended. We chose the scripting language Perl as basis for the implementation, since it is well suited for rapid prototyping and is fast enough for our purposes as our evaluation in 3.7.5 shows. Perl modules are used to structure the code and to facilitate extensions. Plotting is done with gnuplot. A pipe is kept open to each instance of gnuplot and automatically[2] flushed to generate an updated plot when the simulator has finished a number of iteration steps.

Internet Model

The Internet model is at the very core of our simulator. We were looking for a model that is complex enough to represent prevalent characteristics of today's Internet. At the same time it had to be simple enough to enable efficient simulations. Inspired by the Napster and Gnutella P2P client connection measurements in [88], we chose a model that disregards the properties of the hosts and focusses on the speed of the last mile connection. We discuss other possible Internet models suitable for worm simulation in Section 2.2.1.

Our chosen model divides the Internet into n different groups of hosts that belong to sub-networks with similar characteristics. Each host group has two defining parameters: *bandwidth* and *latency*. The bandwidth and latency of a connection between any two groups are chosen as the minimum

[2]This can be done in Perl by using `select(G); $| = 1;`, with `G` being the handle of the pipe.

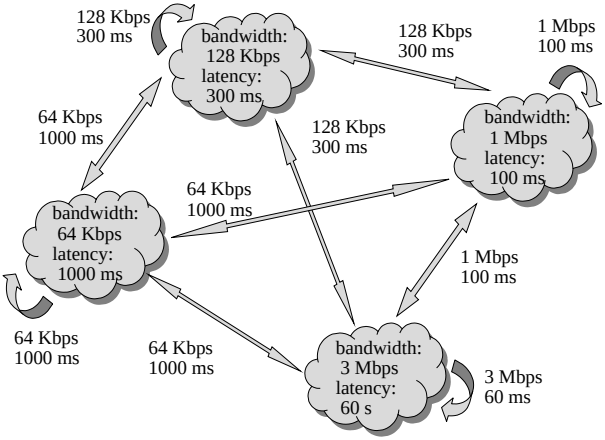

Figure 3.1: *Example of a configuration of our Internet model*

bandwidth and maximum latency of the groups. Figure 3.1 shows a 4-group configuration of the Internet model that is used in our simulator. Details of the host distribution can be found in Table 3.1. We also specified a 10-group configuration, given in Table 3.2. The average bandwidth per host in Table 3.1 is 1157 kbit/s for Napster and 1544 kbit/s for Gnutella, for Table 3.2 it is 1176 kbit/s.

The given percentages are measurements from [88] and assume that the user population of Napster and Gnutella are representative for the whole Internet. The measurements were done in May, 2001. The differences between the Napster and the Gnutella numbers show that this approach is not very accurate. Still these are the best figures we were able to find.

The details of the underlying measurements as well as more information on host characteristics in the Napster and Gnutella P2P filesharing systems can be found in [88]. For continued usefulness of the model and the simulator these numbers will have to be updated from time to time.

Our Internet model turned out to be powerful enough to simulate many cases of worm behaviour. Still for some cases modifications were needed to get realistic results.

The model could easily be extended to support asymmetric connections in order to simulate ADSL or Cable modem connections that, for example, in some European countries have a downstream speed that is two to four

Bandwidth	Napster	Gnutella	Latency
64 kbit/s	32%	10%	1,000 ms
128 kbit/s	5%	14%	300 ms
1 Mbit/s	38%	38%	100 ms
3 Mbit/s	25%	38%	60 ms

Table 3.1: *Internet models with 4 groups*

times faster than the upstream speed. Also, the TCP slow start behaviour is not modelled. However, as most worms are of rather small size, it could be represented by choosing a lower bandwidth than the actually available bandwidth.

The nature of our Internet model is well suited for a quantitative analysis of worm spreading, however it is not suited for traffic prediction for a specific host.

Bandwidth	Napster	Latency
14.4 kbit/s	4%	1000 ms
28.8 kbit/s	1%	1000 ms
33.6 kbit/s	1%	1000 ms
56 kbit/s	23%	1000 ms
64 kbit/s	3%	1000 ms
128 kbit/s	2%	300 ms
256 kbit/s	44%	300 ms
512 kbit/s	14%	100 ms
1.544 Mbit/s	5%	60 ms
44.736 Mbit/s	2%	60 ms

Table 3.2: *Internet model with 10 groups*

Implemented Worm Parameters

Table 3.3 provides an overview of all worm and Internet parameters implemented by our simulator and gives their value range.

Worm parameter	Unit	Lower limit	Upper limit
Hosts in the Internet	hosts	1	2^{32}
Vulnerable hosts	hosts	1	Internet hosts
Start population	hosts	1	vulnerable hosts
Simulation time span	seconds	0	no limit
Transport protocol	TCP or UDP	–	–
TCP resend on timeout	enable/disable	–	–
TCP timeout	milliseconds	0	no limit
Worm size (w/o header)	bytes	0	65535
Parallel scans (TCP) or scans per second (UDP)	–	0	no limit
Additional time to infect	milliseconds	0	no limit
Hitlist	enable/disable	–	–
Hitlist length	hosts	0	Internet hosts
Hitlist vulnerability	-	0%	100%

Table 3.3: *Simulation parameters*

Implemented Scanning Strategies

The simulator implements three different scanning strategies, namely *Random Scanning* with even distribution, *Hitlist Scanning* with a user-defined hitlist and *Local Forced Scanning* that scans local IP addresses with a higher rate than remote addresses.

The effect of hosts being already infected during worm spreading is taken into account by reducing the success probability of an infection attempt:

$$P(\text{infect}) := \frac{|\text{vulnerable hosts}| - |\text{infected hosts}|}{|\text{all hosts}|} \tag{3.1}$$

For each time step the simulator sums up the *infection probabilities*, as defined in (3.1), for each host scanned to determine the number of newly infected hosts. An error is introduced here because two scanning hosts could select the same target in a time step. This error is small as long as the number of vulnerable hosts is significantly lower than the number of total hosts. For simplicity, expression (3.1) is used in the simulator.

Output and Reporting

The simulator produces a text file that describes all parameter values for the simulation, as well as numeric data files suitable for Gnuplot input. In addition, the graphical plots are displayed and updated on the screen while the simulation is in progress.

Traffic: The traffic plot shows the total traffic generated by the scanning and propagation of the simulated worm over time.

Spreading Speed: The spreading speed plot shows the total number of infected hosts over time.

Simulator Limitations

The simulator assumes an even distribution of the vulnerable hosts over the different speed groups. The Code Red worms attacked many installations of the IIS web server with the owners of the hosts not even aware they were running a web server, because IIS had been installed as part of other software packages. Accordingly, the vulnerable hosts were pretty evenly distributed over all speed groups. However, if a worm targets an application that is only installed on hosts that are specifically designated as servers, the vulnerable hosts will tend to be in the faster groups.

Countermeasures by network and host operators are not modelled in the simulator. The effects of such countermeasures will vary heavily depending on human behaviour and technical parameters and hence can hardly be modelled reliably.

3.7.5 Impact of Internet Model

The Internet model serves as an approximation of the real Internet. Since precise overall Internet bandwidth and latency figures are not available, the model also serves as a method to estimate bandwidth and latency based on a limited observation of these characteristics in real distributed Internet applications, in our case P2P filesharing. It turned out that the Internet models needed to be adjusted to some degree to obtain realistic simulation results.

The Sapphire Worm

Sapphire is bandwidth-limited. Its propagation speed is roughly linear with the bandwidth directly available to the already infected hosts. When a high

number of hosts have been infected, there can also be additional limitations because of ISP and backbone bandwidth limits. We assume a vulnerable population of 75,000 hosts.

Figure 3.2 shows a simulation graph obtained with the 10-group Internet model from Table 3.2. The initially infected population was 100 hosts distributed over the different speeds according to group size. The simulation shows a significantly slower propagation than the observed propagation speed of the Sapphire in [73]. A likely explanation is that the Internet became faster since the Napster measurements were taken.

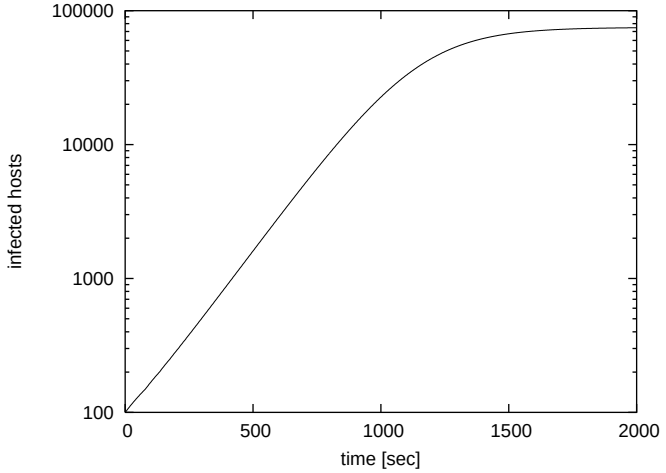

Figure 3.2: *Sapphire worm: Infection speed with original model*

If the 100 initially infected hosts are chosen from the fastest group and, in addition, the fastest group is enlarged to 10% (taking evenly from the other groups) the initial doubling time is about 6 seconds and the scanning rate after 3 minutes is about 50 million per second, giving a very rough approximation for the observed Sapphire worm behaviour. The simulation then reaches an infection level of 90% after about 275 seconds, as can be seen in Figure 3.3. Figure 3.4 shows the infection traffic for the adjusted model. It can be seen that the lack of fast hosts causes the propagation speed to be sub-exponential.

From these experiments we conclude that the initially infected population (obtained via hitlist or pre-infection), while critical for the propagation speed, need not be large. 100 fast vulnerable hosts are probably easy to find. From

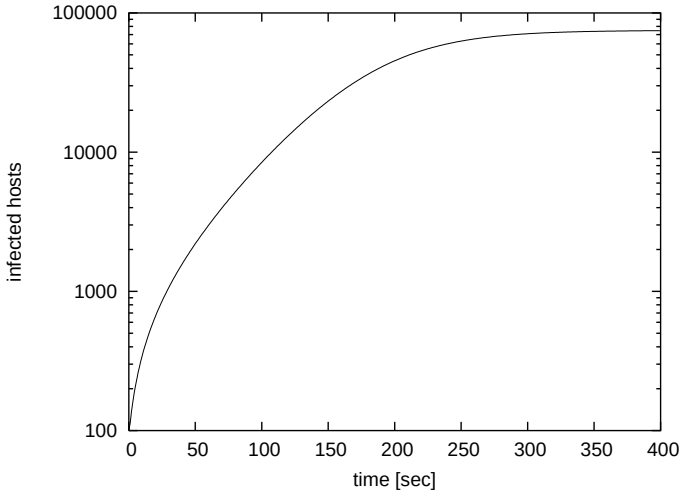

Figure 3.3: *Sapphire worm: Speed with adjusted model*

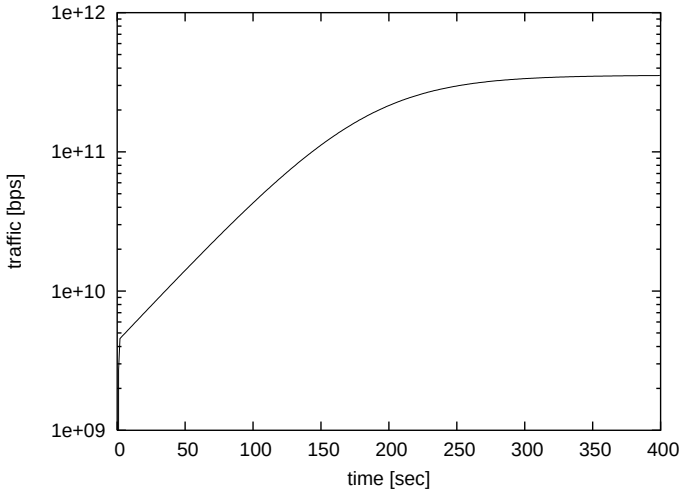

Figure 3.4: *Sapphire worm: Traffic with adjusted model*

there on plain random scanning is quite effective.

To demonstrate the possibilities of the simulator, we give some more examples. The parameters are the same as for the second simulation above. Figure 3.5 shows the Sapphire worm with 15,000 vulnerable hosts. The worm now needs about 1030 seconds for a 90% infection degree. This demonstrates that UDP worms with random scanning can still be used for relatively small vulnerable populations.

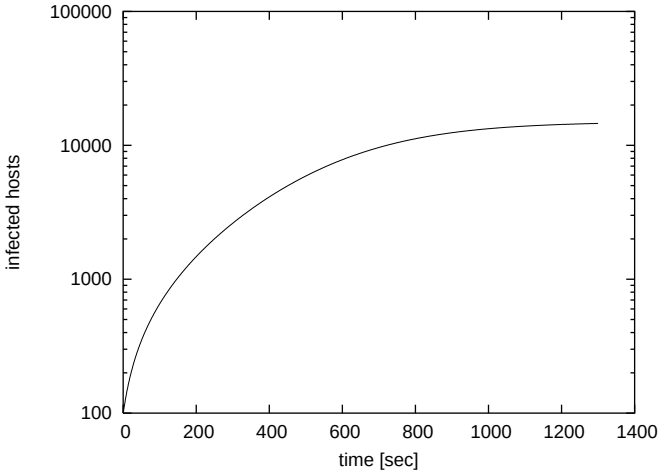

Figure 3.5: *Sapphire worm: 15,000 vulnerable hosts*

Figure 3.6 demonstrates the effect of an infection latency, for example a reboot after infection, here chosen to be 100 seconds. Infection of 90% of the vulnerable hosts now takes about 660 seconds, which shows that even with a significant infection latency the worm is still quite fast.

Code Red

To validate our simulator's results for TCP-based worms, we tried to approximate the behaviour of Code Red Iv2. Therefore we combined data from different analyses in order to choose the most accurate parameters for our simulation. The plot by CAIDA [74] as shown in Figure 3.7 was used as a reference to estimate the simulator's accuracy.

For the simulation we assumed 360,000 vulnerable hosts (3.7). The TCP timeout of CodeRed Iv2 was set to 21 seconds ([121]) and the number of

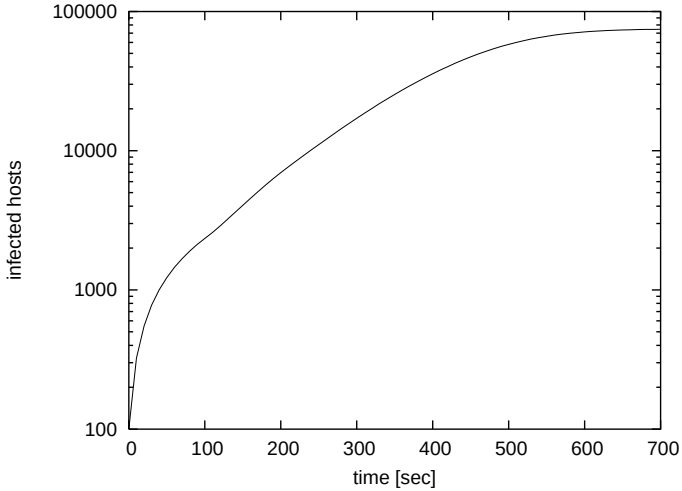

Figure 3.6: *Sapphire worm: 100 sec. infection latency*

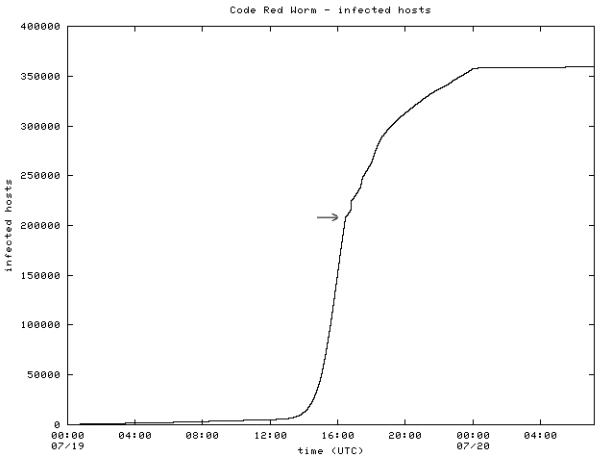

Figure 3.7: *Code Red Iv2: Measurements of infected hosts by CAIDA*

parallel threads sending out scanning packets was set to 100 ([80]). TCP resending was disabled and a time step of 1 sec for the simulation was defined.

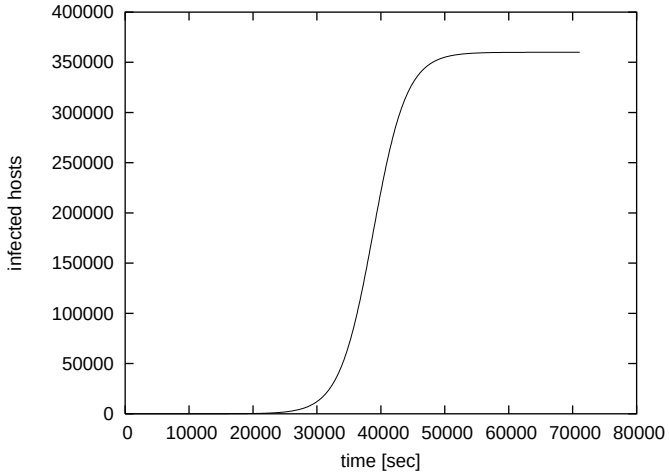

Figure 3.8: *Code Red Iv2: Infection speed simulation*

Our results for the number of infected hosts against time using the 4-group (Napster-based) model are shown in Figure 3.8. They closely match the reference plot. Figure 3.9 superimposes both figures to allow for easier comparison. The log scale plot in Figure 3.10 shows the exponential increase in the number of infected hosts nicely.

The simulation plot does not show the effects of countermeasures put into place by network and host administrators that are present in CAIDA's plot. The arrow in Figures 3.7 and 3.9 marks where countermeasures begin to affect the worm's propagation speed.

Finally in the traffic log as shown in Figure 3.11, it can be observed that at the saturation level of 360,000 infected hosts, a traffic of roughly 0.5 GBit/s is generated. Each host accounts for roughly 1.5 kbit/s as 100 parallel threads on each host send TCP SYN packets within each 21 seconds timeout interval. The fluctuations in the traffic shaping stems partially from a high synchronisation of the hosts due to a fixed time reference for all hosts in our simulator.

A simulation of CodeRed Iv2 with the 10-group model showed only negligible differences to the 4-group case. A decrease of CodeRed's worm size to the size of Slammer showed only a slight decrease in the generated traf-

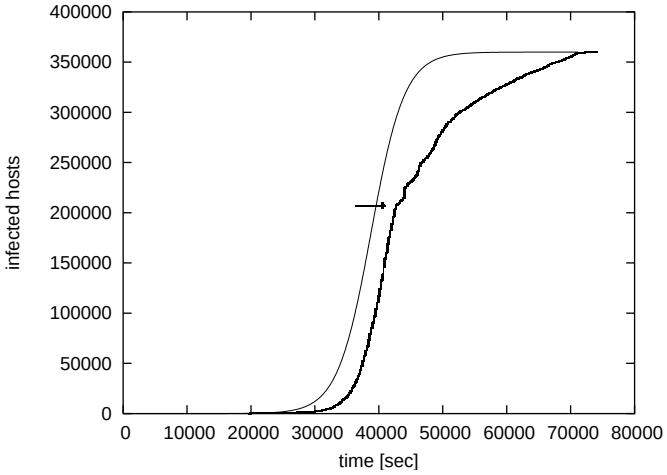

Figure 3.9: *Code Red Iv2: CAIDA vs. simulation, CAIDA graph scaled and shifted right for better visibility.*

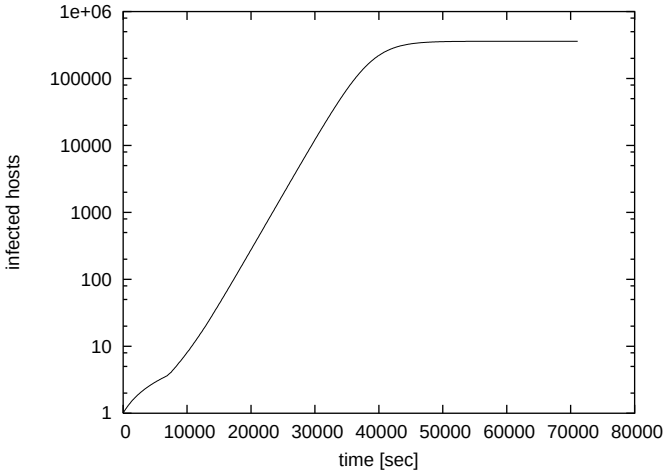

Figure 3.10: *Code Red Iv2: Infection speed simulation, logscale*

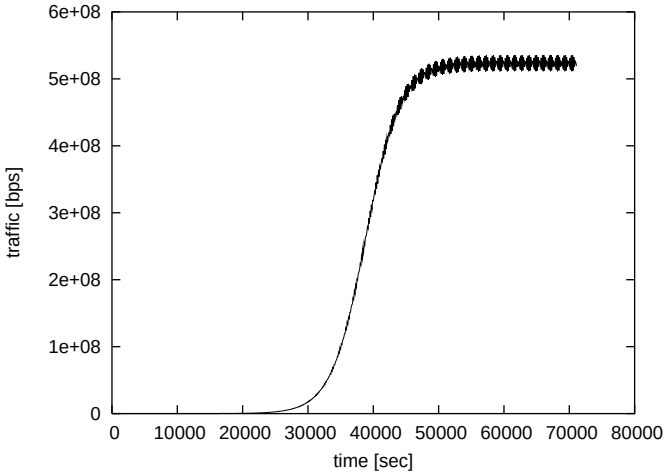

Figure 3.11: *Code Red Iv2: Traffic simulation*

fic. This is not surprising, as the rather large worm code only propagates to vulnerable hosts and hence most of the traffic is caused by scanning other hosts.

Simulator Performance

Simulator performance varies widely with the input parameters. We did most of our experiments on an AthlonXP 2200+ under Linux 2.4., with a time resolution of 50ms for the UDP simulations and 1 second for the TCP simulations.

For UDP simulations we observed a simulation runtime below 20% of simulated time for a 4 group model. With a 10-group model overall simulation time was still lower than simulated time in many cases. A high infection latency time is the one factor that significantly slows things down, since more state has to be kept. Reducing the time resolution drastically speeds things up, this allows for a balance between accuracy and performance. We believe that simulator performance is good enough for many applications.

The Perl-based implementation allows for easy modification if a need to simulate additional effects arises. We feel that this flexibility is more important than the benefits of a faster implementation with a compiled language. The simulator presented here is not suitable for a simulation with a large

number of host groups, that, for example, model individual subnets. In addition, we are not aware of speed and topology statistics that could be used as a basis of such a model. Even if they were available, we think that Internet topology is too unstable for such a detailed model to stay usable without frequent and possibly costly updates.

Chapter 4

Entropy in Worm Traffic

Worm outbreaks in the Internet change the observed traffic mix. The main reason is the worm scan traffic, that is generated in the search for hosts to be infected. Actual infection traffic plays only a minor role, since most connection attempts do not result in successful infection and usually fo not even reach a valid target address. The scan traffic is most pronounced in random scanning worms, where potential targets to be infected are selected at random from the Internet address range. See Chapter 3 for a more in-depth discussion of observable worm traffic.

4.1 Observable Worm Traffic Parameters

For the DDoSVax project, the primary observed data is in Cisco NetFlow [9, 28, 29] format[1]. For the purpose of worm observations this means that mainly flow information, i.e. source and destination IP addresses and ports, as well as flow length in packets and bytes, are the primary available data. In addition, the data is pre-aggregated into flows, i.e. unidirectional packet streams from a source IP address/port pair to a destination IP address/port pair. In a sense a flow forms a "communication" event, possibly having a second flow as a closely related event, if the communication is bidirectional. Failed connection attempts are often unidirectional, and consists of a single flow.

[1] See Appendix C for a discussion of the captured data properties and the capturing system.

As it turns out, the parameters most influenced by worm traffic are IP addresses and ports, where entropy in particular changes in a characteristic fashion. Volume metrics, such as byte counts, packet counts and flow counts are of minor or no use, since they are very sensitive to (D)DoS attacks, flash crowds and other non-worm network events.

4.2 Entropy

Entropy for the purpose of this thesis means information theoretic entropy as defined by Shannon [92]. The name "entropy" is inspired by statistical thermodynamics, where entropy refers to the amount of "disorder" in a thermodynamic system. While thermodynamic entropy is not a subject of this theses, a good summary of it can be found in the Wikipedia article on entropy [40].

4.2.1 Intuition

Information theoretic entropy, or information entropy for short, describes the expected information in a symbol emitted from a symbol source. The source is state-free, i.e. each symbol has a (static) specific probability of being emitted that is independent of the history of emitted symbols. The measure of entropy is bits/symbol and intuitively describe the minimum expected average number of bits needed to describes a symbol the source emits.

On an intuitive level, this is equivalent to the problem of encoding the symbols of an infinite symbol sequence individually so that the least number of bits per symbol are used. The symbols in the sequence need to have global, independent occurrence probabilities for each position of the symbol stream. This also prompts the intuition that entropy is related to data compression and forms a limit of the best case performance per-symbol data compression can reach.

For example, the entropy per symbol corresponds to the average symbol size in bits generated by (static) Huffman coding [54, 55]. However, Huffman coding has to use integral bits, while the true entropy measure per symbol is a real number. It turns out that the maximum error is just short of one bit per symbol, see Section 5.2.1.

Most other compression schemes assume and use inter-symbol dependencies. In real-world situations these generally exist, so that entropy based on

per-symbol probabilities is not necessarily a lower size bound for a compression result. Still, the general intuition that entropy per symbol corresponds in some form to the average bit-size per symbol produced by a data compression algorithm is valid.

4.2.2 Definition

Definition 4 Entropy *according to Shannon is defined in terms of a random event x with possible outcomes* $1,\ldots,n$ *with an associated* **discrete probability distribution** $P = p_1,\ldots,p_n$. *Here,* p_i *is the probability of outcome i. P fulfils* $0 < p_i \leq 1$ *and* $\sum p_i = 1$. *Then the entropy of event x is:*

$$H(x) = -\sum_{i=1}^{n} p(i) log_2(p(i)) \quad \textbf{[bit]}$$

$H(x)$ is the entropy of a single event. When dealing with a series of statistically independent events produced by a process x with distribution P for the individual events, $H(x)$ is the entropy per single event. It is often useful to associate a symbol with each possible outcome and regard the random event as a process that produces one or a sequence of symbols.

Outcomes with zero probability do not contribute to the entropy. However, the above definition is customarily extended to allow them as well where the zero values are treated as limits $\varepsilon \to 0$ with $\varepsilon > 0$. Since $\varepsilon \cdot log_2(\varepsilon) \to 0$ for $\varepsilon \to 0$, $\varepsilon > 0$ and $\varepsilon \cdot log_2(\varepsilon)$ is continuous when extended by the limit at position 0, this is unproblematic.

$log_2(p_i)$ is also called the **surprisal** of the outcome i. This makes $H(x)$ the expected value of the **surprisal** of the outcome of x. Since entropy is not influenced by which specific outcome has which probability, we also write $H(P)$ for a discrete probability distribution P as defined above, instead of $H(x)$.

A further refinement is possible if the symbol stream is comprised of binary encoded symbols of a fixed, known number of bits per symbol, e.g. a stream of IP addresses or TCP port numbers. In that case $H(P)$ can be stated in units of bit/bit, taking the number of bits into account that each symbol is encoded with. This is the form mostly used in this thesis. Clearly a symbol encoded into n bits cannot have more than n bits of entropy, hence the possible value of $H(P)$ is in the range $0 \leq H(x) \leq 1$ bit/bit.

4.2.3 Properties

One of the most important properties of entropy for this thesis is that it increases when the observed data pattern becomes more random, i.e. the observed symbols have a more equal probability of occurring. The inverse is true as well. Formally:

Theorem 1 *Let $P = p_1, \ldots, p_n$ be a discrete probability distribution, with $n \geq 2$. Let w.o.l.g $p_1 \leq p_2$. Let further $P' = (p_1 - k, p_2 + k, \ldots, p_n)$ for a real number $k > 0$ with $0 \leq p_1 - k$ and $p_2 + k \leq 1$. Note that P' is also a probability distribution. Then:*

$$H(P) > H(P')$$

Informally: When the probability distribution P becomes less even, then $H(P)$ decreases.

Proof: It is enough to show the inequality only for the first two terms in the sum for $H()$, since all other terms in the sum remain unchanged and hence do not influence the claimed inequality.

The proof requires two steps. First we show

$$\begin{aligned}
&-(p_1 \cdot log_2(p_1) + p_2 \cdot log_2(p_2)) > \\
&\qquad ((p_1 - k) \cdot log_2(p_1) + (p_2 + k) \cdot log_2(p_2)) \qquad (1)\\
&\Leftrightarrow \\
&p_1 \cdot log_2(p_1) + p_2 \cdot log_2(p_2) < (p_1 - k) \cdot log_2(p_1) + (p_2 + k) \cdot log_2(p_2) \\
&\Leftrightarrow \\
&0 < -k \cdot log_2(p_1) + k \cdot log_2(p_2) \\
&\Leftrightarrow \\
&k \cdot log_2(p_1) < k \cdot log_2(p_2) \\
&\Leftrightarrow \\
&log_2(p_1) < log_2(p_2)
\end{aligned}$$

We have $0 < p_1 < p_2$ and hence the last line is true because the logarithm is strictly monotonicly increasing and k is positive.

For the second step we use Gibb's inequality [46, 47], due to Josiah Willard Gibbs (1839 - 1903). Gibbs inequality states that for two discrete probability distributions $Q = (q_1, \ldots, q_n)$ and $R = (r_1, \ldots, r_n)$

$$H(Q) \geq \sum_{i=1}^{n} q_i \cdot log_2(r_i)$$

with equality exactly when $p_i = q_i$ for all $i \in \{1, \ldots, n\}$.

We still need to show that

$$-((p_1 - k) \cdot log_2(p_1) + (p_2 + k) \cdot log_2(p_2)) > \\ -((p_1 - k) \cdot log_2(p_1 - k) + (p_2 + k) \cdot log_2(p_2 + k)) \quad (2)$$

But this follows directly from Gibb's inequality by choosing $Q = (p_1 - k, p_2 + k, p_3)$ and $R = (p_1, p_2, p_3)$ with $p_3 = 1 - p_1 - p_2$, or omitting p_3, if it were zero.

Taking (1) and (2) together gives the claim. □

Entropy also increases when the number of observed symbols increases. More formally:

Corollary 1 *Let P be as in the last theorem. Let $P' = (p_1 - k, \ldots, p_n, k)$ for a random event x', for an arbitrary real value $k > 0$ with $p_1 - k > 0$. Then*

$$H(P) < H(P')$$

Proof: Note that $H(P)$ does not change if we add a zero probability event to P, i.e. replace it by $P'' = (p_1, \ldots, p_n, 0)$. Now remember that zero value probabilities in the definition of entropy are really limits $\varepsilon \to 0$ with $\varepsilon > 0$, i.e. we can use $P'' = (p_1, \ldots, p_n, \varepsilon)$. This allows us to apply the previous theorem in reverse.

□

4.2.4 Changes During Worm Outbreak

One property of normal Internet traffic we observed in the SWITCH traffic data is that it is mostly symmetrical with regard to a "sends data to" relation. For most packet-streams sent from host **A** to host **B** there is some answering packet-stream from **B** to **A**. This observation holds both for TCP and UDP traffic. On a flow-level this means that for a flow-record describing traffic from **A** to **B**, there usually is a flow-record describing traffic from **B** to **A**. A typical example is a successful TCP connection, where some traffic flows in both directions. This observation does not hold for the amount of data sent. For example, in P2P filesharing, the SWITCH network is a provider, i.e. much more P2P data flows out of the SWITCH network than into it, see [51].

Note that this thesis splits observations of traffic properties into four classes of traffic, namely TCP, UDP, ICMP and other traffic. The main focus of our observations is on TCP and UDP traffic. One effect is that an ICMP "destination unreachable" as an answer to a TCP "SYN" might be missed. However, experiments have shown that a TCP "SYN" to an IP address in the SWITCH network will typically either result in a TCP "SYN ACK" or receive no reply at all.

The traffic symmetry is not perfect. In the SWITCH traffic data we see more active external IP addresses than internal ones. A typical hour during the day in 2004 had traffic from around 800'000 external IP addresses, while only around 200'000 internal IP addresses were found to be traffic sources. This is likely due to IP-range scans, such as manual scans, scans from residual worm populations, scan-like traffic from P2P filesharing clients in their start-up phase and other scan sources. SWITCH has about 2.2 million IP addresses in their address range, and a major part of the random scan activity remains unanswered.

Note that traffic symmetry is not present at all if volume metrics, i.e. packet counts and byte counts, are used. In addition, the effect of a worm outbreak on volume metrics is typically very minor and not suitable for detection purposes.

IP Addresses

During the outbreak phase of a random scanning worm, the flow-mix changes. On one hand, massive random scanning activity can be observed. Most of these connection attempts remain unanswered and the flow-mix becomes decidedly asymmetric. The effect is that the entropy of the target IP addresses in the observed flow data increases noticeably. This is because many more target IP addresses are seen, which all get one or very few flows sent to them. This causes an increase in the entropy of the flow target IP address fields. In addition the target IP probabilities become more even, causing additional entropy increase.

On the other hand, some (few) infected hosts start to strongly contribute to the source IP addresses seen, since they generate a significant amount of the overall flows seen. This leads to a less even probability distribution for the individual source IP addresses, but at the same time the number of different source IP addresses remains mostly the same. This leads to an overall decrease of source IP address entropy in the observed traffic flows.

These two effects have different direction, since the scan-flows from worm infected hosts are mostly unanswered. Otherwise an increase or decrease in entropy of the IP fields might be visible, but no difference between source and destination IP addresses would be present.

Note that these effects are very weak and usually remain below the noise threshold for entropy in packet headers. The aggregation of all packets belonging to a connection is really needed to see the entropy changes during worm outbreaks.

Ports

With regard to ports, the effects are conceptually similar, but can take different forms. It depends on what port characteristic the dominant scan-traffic component has. Both source and destination ports in scan-traffic can be fixed or random. For source ports this is mainly an implementation choice. If the OS network stack is used by the worm, source ports will be random from an area in the higher port numbers. For destination ports the used exploit limits what characteristics can be used. Typically the destination port is fixed, but not necessarily so. See Chapter 3 for a more detailed discussion.

By the nature of entropy, random port values in scan-traffic increase the observed entropy, while fixed values decrease it. In the absence of any major attack, SWITCH flow-level network traffic has roughly one-half random port numbers (selected by the network stack on the connection-initiating side as source port) and one-half fixed ports (the well-known port for the service on the other side). Unanswered port-scans increase the randomness in the ports seen, P2P traffic with fixed ports on both sides can decrease the amount of randomness in the port values. Overall the normal port number entropy has a value that still leads to a noticeable entropy increase or decrease (depending on the worm characteristics) during the outbreak of a fast Internet worm.

4.2.5 Observation Examples

We will now illustrate the entropy effects with two examples, both taken from the SWITCH network data. For easier comparison we use a vertical scaling of bit/bit in the plots, i.e. bit of entropy per bit of data input, instead of bit/symbol, i.e. bit per IP address or port number. With a bit/bit scaling the possible entropy range is $0 \ldots 1$.

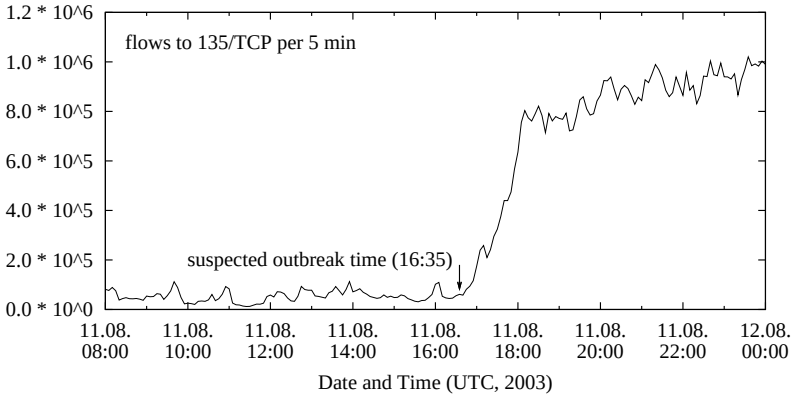

Figure 4.1: *Blaster worm: Flow count*

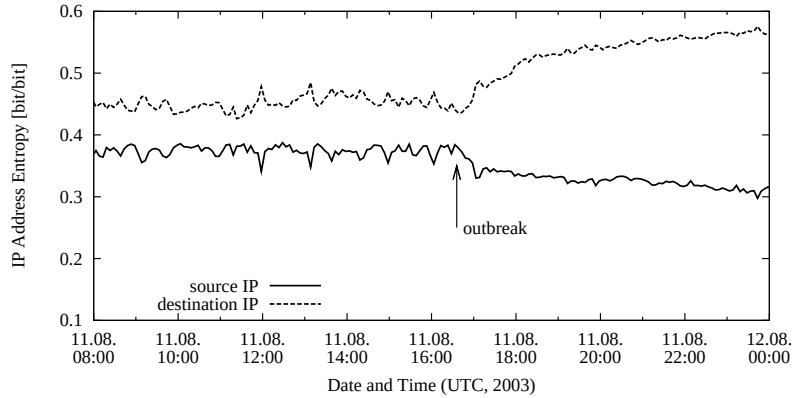

Figure 4.2: *Blaster worm: IP address entropy (TCP traffic)*

The first example is the Blaster worm [12, 17, 61]. First observed on August 11th, 2003, Blaster uses a TCP random scanning strategy with fixed destination and variable source port to identify potential infection targets. Blaster is estimated to have infected from 200'000 to 500'000 hosts worldwide in the initial outbreak. A flow count for the Blaster worm outbreak as seen in the SWITCH network can be found in Figure 4.1. This count is from the routers `swiCE1` and `swiCE2`, see Appendix C. Figure 4.2 shows the changes in the IP address field entropy on flow level during the initial outbreak. As expected, the source IP address entropy falls, due to a smaller number of hosts (those infected) starting to generate a large fraction of the observed flows.

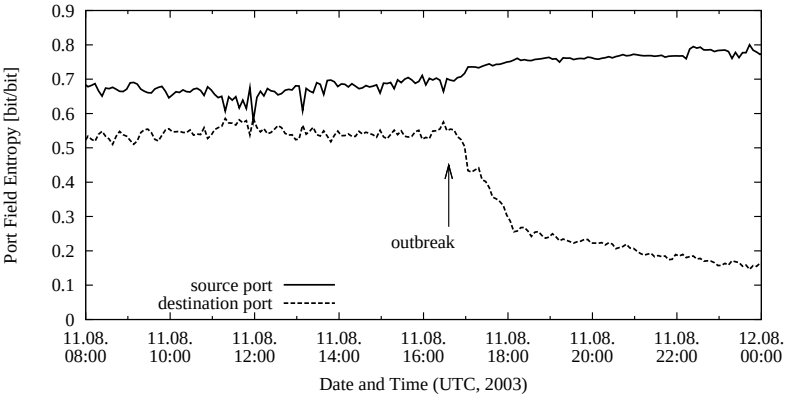

Figure 4.3: *Blaster worm: Port field entropy (TCP traffic)*

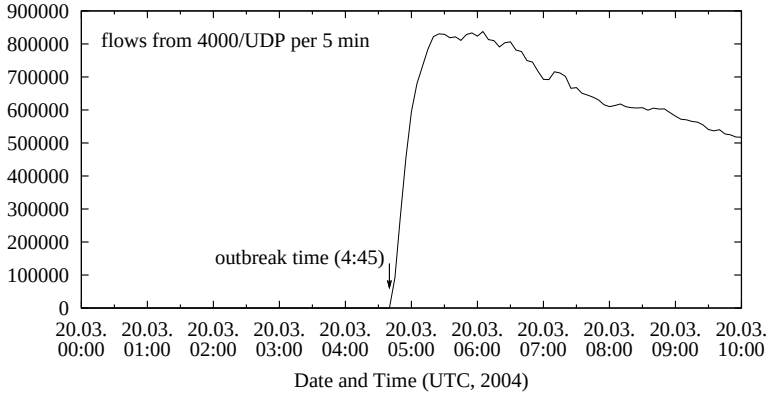

Figure 4.4: *Witty worm: Flow count*

The destination IP address field entropy increases, because more IP addresses are observed (due to random scanning) and the probability distribution of the individual IP addresses becomes more equal, since a large fraction of the observed flows goes to randomly selected targets.

The port field entropy changes are shown in Figure 4.3. Since Blaster uses an exploit on port 135/TCP, this target port number is constant. As a consequence, the target port entropy in the observed traffic falls sharply after the outbreak, since a single value starts to become much more frequent. Since Blaster uses the OS network stack, the source port is chosen randomly from a range in the higher port numbers. As a result, the entropy of the source port fields in the observed flow data increases.

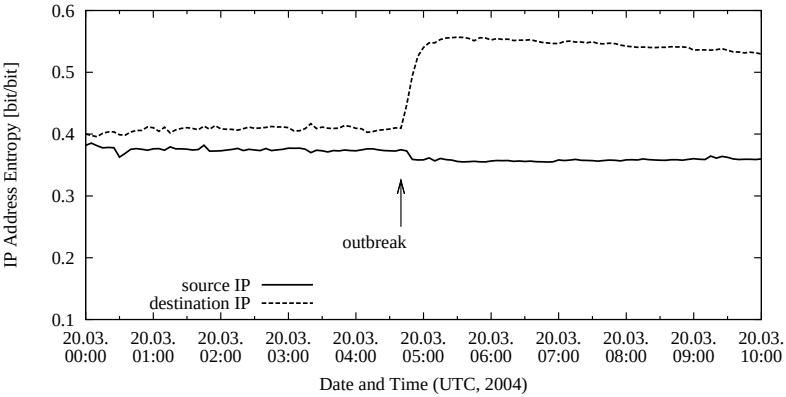

Figure 4.5: *Witty worm: IP address entropy (UDP traffic)*

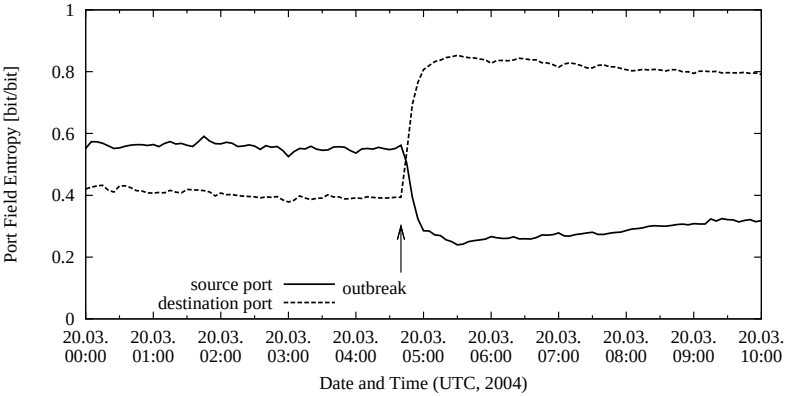

Figure 4.6: *Witty worm: Port field Entropy (UDP traffic)*

The Witty worm [90, 104], first observed on March 20th, 2004, is a good second example, because it has some unexpected characteristics. Witty attacks a specific firewall product. It uses UDP random scans with *fixed* source port and *variable* destination port. Witty infected only about 15'000 hosts. A plot for the count of Witty generated network flows in the SWITCH network can be found in Figure 4.4. This count is also from the routers `swiCE1` and `swiCE2`, see Appendix C. The IP address entropy changes for Witty, shown in Figure 4.5 are similar to Blaster, if different in magnitude. The changes in the port address fields of the observed flows, however, are exactly the opposite of the observations for Blaster, since Witty fixes the source port and varies the target port.

Chapter 5

Entropy Estimation

Assume we have a symbol stream emitted from a (real) source. If the individual symbol probabilities $p(i)$ are known, and they do not depend on the other symbols in the symbol stream, then entropy can be directly and accurately calculated using the definition from Chapter 4. Usually the symbol probabilities are only known beforehand when the symbol stream is synthesised or produced by a well understood process. For our purposes neither is the case. Hence we need methods to estimate the entropy in a symbol stream.

5.1 Direct Entropy Estimation

The most direct way to estimate $H(x)$ for a given, finite symbol sequence x, over the symbol set $1,\ldots,n$, is to estimate each individual symbol probability $p(i)$ by its actual number of occurrences $f(i)$ in the sequence, divided by the length of the symbol stream $l(x)$. This gives the following estimation for the per-symbol entropy:

$$H(x) \approx -\sum_{i=1}^{n} \frac{f(i)}{l(x)} log_2 \frac{f(i)}{l(x)} \quad \text{[bit]}$$

The main source of errors on this type of estimation is that the symbols probability for different positions are often not independent in practice, and this approximation method will yield a value that is too large. Obtaining a more accurate measurement for this case would entail combining individual

symbols into larger ones to capture the interdependency. This only works if there is a maximum distance over which symbols depend on each other. If there is no such maximum distance, other measures than entropy are needed for accurate measurement of information.

The computational effort for this type of estimation is determined by the effort of estimating the individual probabilities. There is no way around counting how often each symbol is contained in the sequence, but this is also enough. Hence we have the following algorithmic complexity:

- Time: $O(l(x))$
  The basic assumption here is that counting each stream position takes constant effort. This is realistic, if arrays or hash-tables are used to store the individual probabilities. The constants are determined by the element-access effort for the table type used.

- Space: $O(n)$
  For each observed value we need one space unit to store its frequency.

In practice, the limiting factor is the table used to store the symbol counts. For example a hash-table used to store frequencies of IP addresses has a per-element memory need of 4 Bytes for the frequency and 4 bytes for the key (IP address). Table-external collision resolution (e.g. chaining colliding table elements in a linked list) typically adds another 4 bytes per element for a pointer and 4 bytes in the base table if a load-factor of around 1.0 is assumed. The worst case storage need for such a hash-table (reached e.g. when spoofed, random addresses are observed) is $8 \cdot 2^{32}\text{Bytes} = 32GB$ for table-internal collision resolution and $16 \cdot 2^{32}\text{Bytes} = 64GB$ for table-external collision resolution. This number may be prohibitively large. The relevant figures for the SWITCH data are discussed in Section 5.3.

5.2 Estimation by Compression

A second, more indirect approach to entropy estimation uses the connection between entropy and channel capacity. The intuition is that there is no way to push more bits of entropy per time unit through a specific channel than its channel capacity, while this limit can (theoretically) be reached. Compressing the input data stream or diluting it with redundant symbols does not change this limit. If a (theoretical) perfect compressor, that removes all redundancy,

is used on the original, binary encoded, symbol stream, we must get a bit-stream that goes through the channel with the highest possible entropy per bit sent. Hence this compressed binary stream must have exactly one bit of entropy per bit used in the encoding.

This idea can be used to estimate entropy in a finite data-stream: First compress perfectly and obtain the exact entropy as the resulting number of bits. Then divide this number by the original number of bits in order to obtain the original entropy in [bit/bit].

The main source of error in this approach is that real compressors are not perfect. Since we are mainly interested in relative comparison of entropy in normal data and during a worm outbreak, this type of error is not necessarily a problem. One definite advantage of compression over direct entropy estimation, as discussed in Section 5.1, is that most modern compressors assume that symbols are interdependent and will take this into account during compression.

We evaluated different compression methods, all of them lossless, in order to find one suitable for our purposes. We now briefly describe each compression method.

5.2.1 Huffman Coding

Huffman coding [54, 55] assigns a code word of variable size to each binary symbol to be encoded. It does so based on observed symbol probabilities. The code words are in a prefix format, which allows decoding in the absence of length fields. The prefix property does not impact code word length. The average number of bits needed per symbol in a Huffman encoded data stream x is

$$\text{Huff}(x) = -\sum_{i=1}^{n} p(i)\lceil log_2(p(i))\rceil \quad \text{[bit/symbol]}$$

This number is very similar to entropy, except that the number of bits for each specific symbol is rounded up to the next integral value. The only additional error introduced compared to direct entropy estimation is the rounding. i.e. the maximum additional error is smaller than $1 bit/symbol$. If the probabilities are all fractions of the form $1/2^i$ with i a natural number, then $\text{Huff}(x)$ and $H(x)$ are identical. On the other hand the computational effort for Huffman coding, even if only done to the extent needed to obtain the output size, is comparable to direct entropy estimation, since Huffman coding first deter-

mines symbol frequencies and then builds an encoding tree based upon these frequencies. For this reason Huffman coding has not been evaluated further.

5.2.2 GZIP

The GNU zip compressor [50, 84, 85] is a well-known, well established and standardised compression program and compression library in the UNIX world. GNU zip uses the LZ77 [62, 65] algorithm and binary Huffman coding. This combination is usually referred to as the DEFLATE algorithm. LZ77 works by keeping a ring-buffer with the most recently seen data. It then tries to find the current symbol sequence in that buffer and replaces it by a reference-length pair if found. The ensuing stream of original symbols and reference-length pairs is then further compressed with Huffman coding.

GNU zip is a stream compressor, i.e. each byte in the compressed data stream can depend on any or all bytes in the uncompressed data stream up to that point. The compression performance of gzip is average in all regards. It compresses reasonably fast and achieves reasonable compression sizes. A comparison on the raw data relevant for this thesis can be found in Section 5.3.

5.2.3 BZIP2

The bzip2 compressor [24, 25] was first publicly released in July 1996 as version 0.15 by Julian Seward. Today bzip2 has reached version 1.03 and has been production-stable for many years. Currently bzip2 is used, e.g., to compress Linux kernel source packages before distribution.

Different from gzip, bzip2 is a block-compressor. It takes between 100kB and 900kB (depending on parametrisation) of input data and compresses it into a block of output data. The output data block is not dependent on any previous data compressed. If a bit-error occurs in the compressed data, only one block of input data is lost.

The first compression step done in bzip2 is the Burrows-Wheeler transform [22, 23] which transforms repeated symbol sequences into sequences of identical letters. It then performs a move-to-front transform [8, 16] in order to condition the data for a final step of Huffman coding. Compression performance of bzip2 is very good, but it is slow and uses a relatively large amount of memory. See Section 5.3 for a performance comparison with the other compressors.

5.2.4 LZO

The LZO [7, 66] compressor family was created by Markus F. X. J. Oberhumer. LZO stands for *Lempel-Ziv-Oberhumer*. It is another variant of the Lempel-Ziv algorithm. The GNU command line tool that encapsulates the LZO library is called *lzop*.

LZO is primarily optimised for very fast decompression and very low memory consumption. Compression speed can be varied between very fast, with a low compression factor, to relatively slow and comparable to gzip in compression factors. We are primarily interested in the LZO compressor in its fastest variant. Again, see Section 5.3 for a performance comparison with the other compressors.

5.2.5 Compression Comparison Example

An example that shows the compression performance of the three different compressors side by side can be found in Figure 5.1. The plot shows the compression ratio changes during the Witty worm outbreak for the destination IP fields on flow level for the gzip, bzip2 and LZO compressors, respectively. The y-axis gives the relative compressed size, i.e. a value of 1.0 means no compression at all, 0.5 means the compressed data took half as much space than the raw data, etc..

It can be seen that the three different plots are shifted vertically against each other, in a way that is consistent with the expected compression performances of the three compressors. However, the shapes are very similar, as is the change during the outbreak. This supports the expectation, that while the compression ratio is very dependent on the compressor used, changes in compression ratio are less dependent and, even more importantly, the reaction to strong changes in the input data caused by a worm, is only weakly dependent on the compressor used.

5.3 Performance and Scalability

In order for an approximation method to be viable, it has to have some significant advantages over other measurement methods. For entropy estimation by compression, these advantages are both in speed and memory needs.

Table 5.1 gives the maximum memory needs for the different compressors. For the compressors, these numbers are absolutes and do not depend

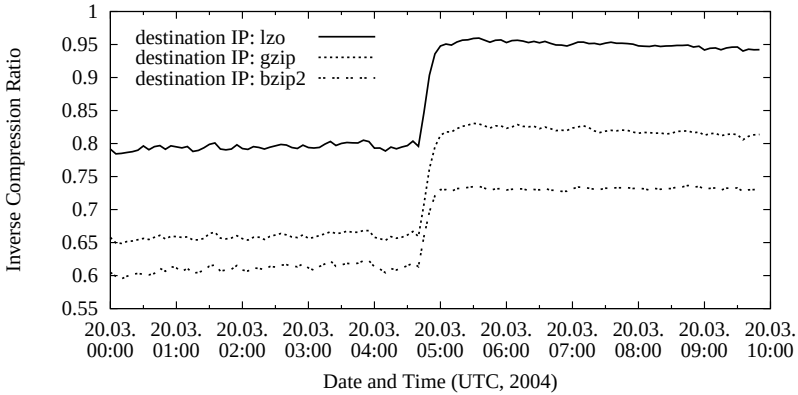

Figure 5.1: *Witty worm: Compressor comparison*

on the input data processed. These needs are per compressor instance. While bzip2 uses a moderate amount of memory, the memory needs of gzip and especially lzop are very small.

For comparison we give the memory consumption to be expected for entropy estimation by direct frequency estimation (i.e. counts for the different values) in Table 5.1 as well. The number given assumes a non-outbreak situation with around 60 million flows per hour and measurements in 5 minute intervals. We also assume the use of a hashing method that uses chained collision resolution (colliding elements are placed into a linked list) and a load factor around 1.0. This leads to 16 bytes of memory per observed source and destination IP address. The `hashed_table` data structure, described in more detail below and in Appendix A, has these properties. Ports can be counted with far smaller memory effort, an array with 256kB size is sufficient for both source and destination port.

Using internal collision resolution and direct element storage, it would be possible to reduce the number of bytes needed per element down to a minimum of 8 bytes. This would however reduce the table speed drastically, because the number of collisions would increase dramatically and secondary collisions (collisions happening as a result of collision resolution) would become a concern. In practice, a table with internal collision resolution would only be filled to a load factor significantly smaller than 1.0. This in turn increases the memory needed and the size advantage shrinks or vanishes.

Note that the worst case memory consumption for direct entropy estimation by frequency counts is directly dependent on the number of flows seen in each measurement interval. The worst-case assumption is that all IP addresses seen are different. On one side, this means that the worst case is usually not reached during normal operation. On the other side, the number of flows can increase dramatically during an attack. For example, a simple SYN-flooding attack with randomly spoofed source IP addresses places a huge load on the table used to estimate source IP address entropy. As a consequence, a large amount of memory needs to be kept available for the counting algorithms usage in order for it to be of any use during attacks.

Table 5.2 shows the relative CPU times needed in order to compress all four fields of interest for a typical hour of data, containing 60 million flows. The measurement is for 5 minute intervals without overlap. It can be seen that even the slowest compressor (bzip2) performs a lot faster than necessary for real-time operation on the SWITCH data. The fastest compressor (LZO in its lzo1x-1 variant) is fast enough so that in a real-time sensor the main bottleneck will be transferring the data into memory.

For comparison we also state the time needed for direct entropy estimation by estimation of the value frequencies. The measurements were again done using the `hashed_table` data structure described in more detail in Appendix A. It can be seen that direct frequency counting is comparable in CPU needs to the slowest compressor we evaluated.

Method	Memory needed per data stream
bzip2	7600 kB (fixed maximum)
gzip	256 kB (fixed maximum)
lzo1x-1	64 kB (fixed maximum)
direct frequency counts	60 MB (at 60 mill. flows/hour)

Table 5.1: *Entropy estimation memory needs (worst case)*

Since CPU load is comparably low for all approaches, memory becomes the primary concern, at least on the SWITCH data In addition to the significantly lower memory needs of the compression approach, all compressors have fixed bounds on maximum memory needed. Direct entropy estimation by counting value frequencies has an upper memory bound dependent on the number of different values seen. This presents a potential weakness in the algorithm, which could lead to failure due to memory exhaustion in attack situations.

Method (Library)	CPU time / hour (at 60'000'000 flows/hour)
bzip2 (libbz2-1.0)	169 s
gzip (zlib1g 1.2.1.1-3)	52 s
lzo1x-1 (liblzo1 1.08-1)	7 s
direct frequency count (5 minute intervals)	176s

Table 5.2: *Average CPU time (Linux, Athlon XP 2800+)*

5.4 Validation

Entropy estimation by compression is a heuristic approach. Its primary advantages are high speed and very low memory usage. Due to its heuristic nature, its validity has to be demonstrated for each specific application, compression algorithm and data set. In the last section, we demonstrated its viability as an entropy estimator for the purpose of detecting fast Internet worms. We will now drop this restriction and examine the correspondence between sample entropy and entropy estimated by LZO compression.

5.4.1 Basis Data

We explore the suitability of compression for entropy estimation, using NetFlow data from the SWITCH network (see Appendix C). The validation data spans the whole year of 2004, but is limited to the flows exported by the routers `swiCE1` and `swiCE2`, which represents about half of the exported flow-level data. The reason for this limitation is to avoid artefacts that result from an overload in the remaining router, `swiIX1`, during the observation period.

5.4.2 Estimation by Compression

We only consider the fastest compression method LZO here. It is likely that the other two compression methods will have comparable performance. LZO is the fastest compressor with the least resource usage, and showing that it performs well is sufficient, since this removes the incentives to use the other compressors.

Recall that the entropy value estimated by compression is given as the original size in bits of a (finite) symbol stream x divided by its compressed

size in bits. We will call this estimation with the LZO compressor $H_{lzo}(x)$ here. Formally $H_{lzo}(x)$ is given by

$$H_{lzo}(x) = \frac{size(x)}{size(lzo(x))} \text{ [bit/bit]}$$

5.4.3 Entropy Measurement

We compare entropy estimated by compression with entropy estimated by value frequency, i.e. against sample entropy. Recall that sample entropy uses the number of occurrences $f(i)$ of symbol i in data stream x, divided by the overall number of symbols $l(x)$ in x to obtain an estimation $\tilde{p}(i)$ for the probability $p(i)$ of symbol i occurring, i.e.

$$\tilde{p}(i) = \frac{f(i)}{l(x)}$$

This allows us calculate an estimation $\tilde{H}(x)$ of the entropy $H(x)$ by

$$\tilde{H}(x) = -\sum_{i=1}^{n} \tilde{p}(i) log_2(\tilde{p}(i)) \text{ [bit/symbol]}$$

In order to normalise this measure to bit/bit, we need to divide by the symbol size s as well. The symbol size is 32 bit for IP addresses and 16 bit for port numbers. We get

$$\tilde{H}(x) = -\frac{1}{s}\sum_{i=1}^{n} \tilde{p}(i) log_2(\tilde{p}(i)) \text{ [bit/bitl]}$$

We will show that $H_{lzo}(x)$ and $\tilde{H}(x)$ are strongly correlated. The first method used consists of scatterplots with $H_{lzo}(x)$ on the horizontal axis and $\tilde{H}(x)$ on the vertical axis. The second method uses the standard deviation of $H_{lzo}(x) - \tilde{H}(x)$. Both comparisons are done for the full 2004 data. TCP and UDP flows are treated separately.

5.4.4 Linear Regression

Linear regression is a well-known statistical method that can be used to determine a linear relationship between the first and second component of a set

of two-dimensional coordinates. Its first result is a line given as

$$y = a + bx$$

were a and b are constants. The line describes the estimated linear relation between the coordinates. The second result is a correlation coefficient $r \leq 1$, that describes the accuracy of the linear relationship. Values close to 1 stand for a good correlation, i.e. a close to linear relationship. The exact definition can be found in most introductory texts on statistics, e.g. [53].

In order to determine the quality of the approximation of entropy values by LZO compression, we have calculated the correlation coefficients in one-week intervals (basis measurement interval is 5 minutes) for the SWITCH network traffic of the year 2004.

Findings - TCP

The plots for the correlation coefficient r for TCP traffic can be found in Figures 5.2, 5.3, 5.4 and 5.5. All, except the destination IP plot, indicate a good and mostly linear relationship between sample entropy values and compressibility-derived entropy approximation using the lzop compressor.

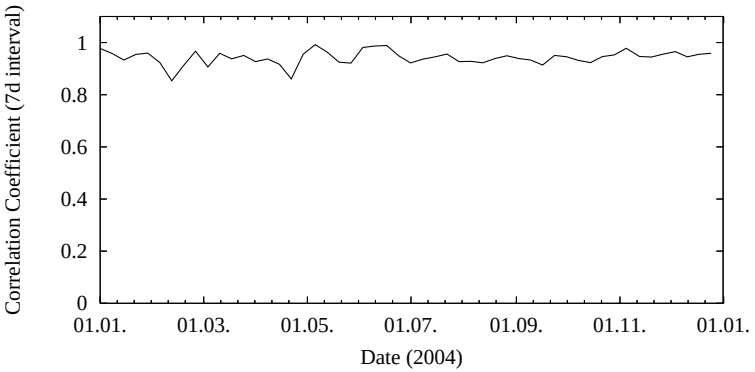

Figure 5.2: *TCP - Correlation coefficient, source IP*

The destination IP plot in Figure 5.3 shows reasonable correlation values, except for a number of sharp, negative spikes. A primary suspect for these spikes is scan activity against a subnet. If the scans are conducted in a way so that the repetitions strongly show up during the 5 minute sample entropy

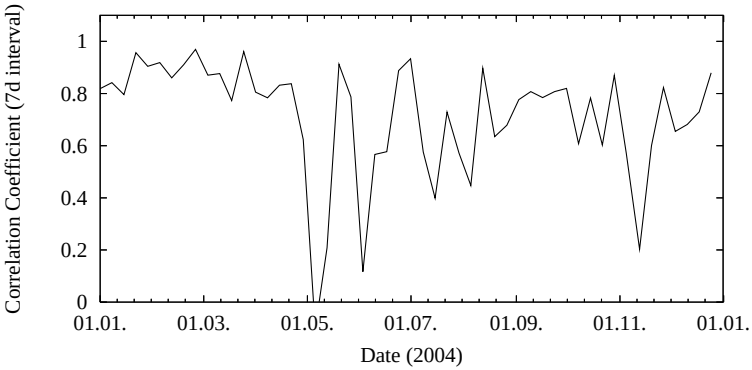

Figure 5.3: *TCP - Correlation coefficient, destination IP*

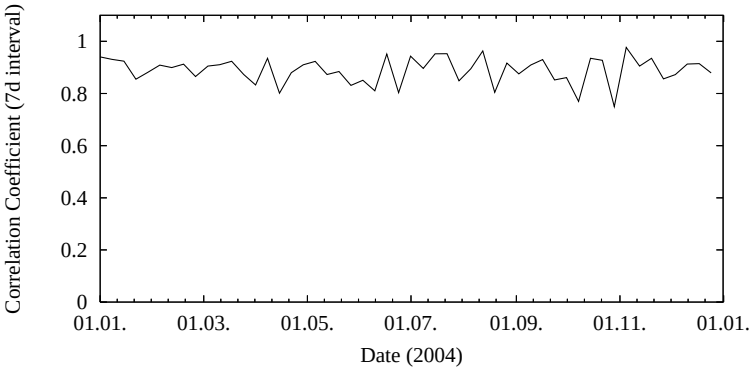

Figure 5.4: *TCP - Correlation coefficient, source port*

measurement intervals, yet are not near enough together in the destination IP address stream found in the NetFlow data, then the lzop compressor will not detect the repetition. At the same, time the sample entropy calculation may be influenced because of its larger measurement interval. As a consequence, sample entropy can be significantly lower during specific scan activity, than LZO estimated entropy.

Figures 5.6 and 5.7 show scatterplots of the data around the first and last negative spikes in Figure 5.3. Both plots show that the values are closely grouped together in the higher entropy area, consistent with stronger scan activity. The plots also show that the actual approximation is still reasonable,

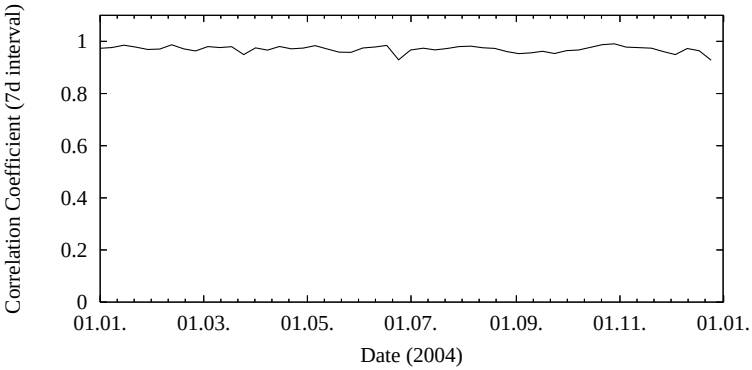

Figure 5.5: *TCP - Correlation coefficient, destination port*

since the error is strongly localised. The results from linear regression analysis are not very significant for a situation were the values are clustered in a small area, since it is then very sensitive to small errors.

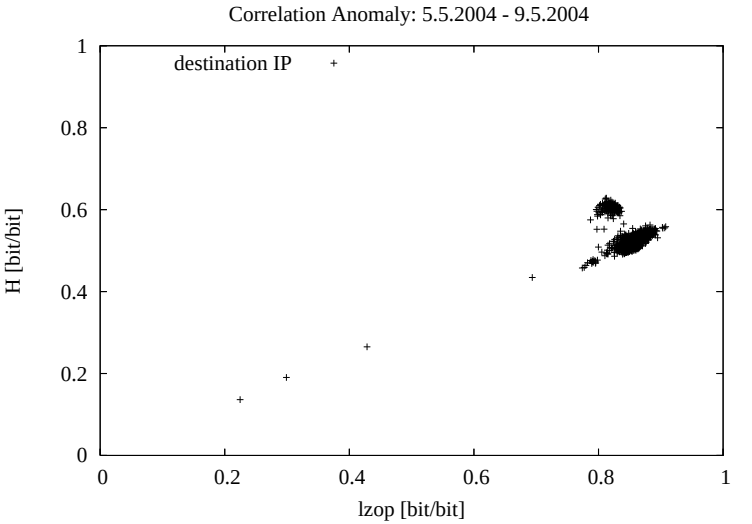

Figure 5.6: *TCP - Correlation anomaly*

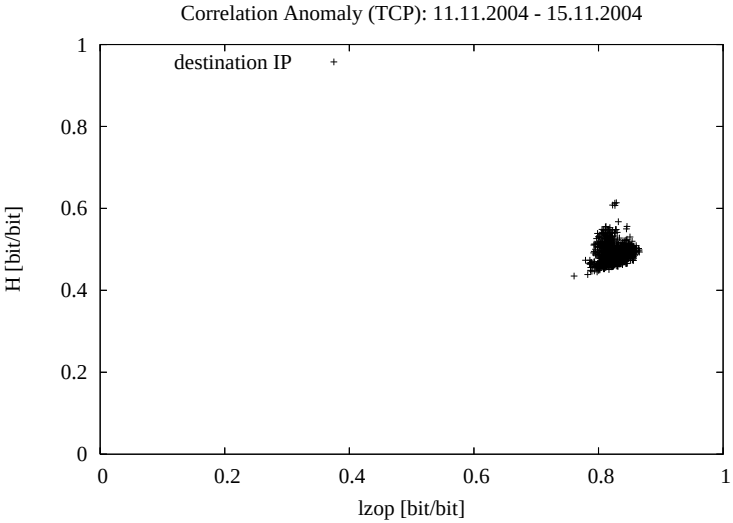

Figure 5.7: *TCP - Correlation anomaly*

Findings - UDP

The results of the linear regression analysis for UDP traffic are similar to the TCP results. Figures 5.8, 5.9, 5.10 and 5.11 show good correlation for all LZO approximations, except the destination IP values. The reasons for the suboptimal destination IP address results are the same as for the TCP case, namely scanning activity with partially randomised target addresses, where a larger data window than the 64kB used by the lzop compressor is needed to recognise the limited range of the scan targets. Figure 5.12 gives a scatterplot of the time around one of the stronger anomalies. The observed clustering is similar to the observation for the TCP case.

5.4.5 Standard Deviation

To complement linear regression analysis, we examine the standard deviation of $H_{lzo}(x) - \tilde{H}(x)$. This expression characterises the distance between the two values in each pair. While linear regression analysis determines a non-local correlation by trying to fit the value-pairs with a line, the standard deviation of $H_{lzo}(x) - \tilde{H}(x)$ gives a measure of the variation in distance between the

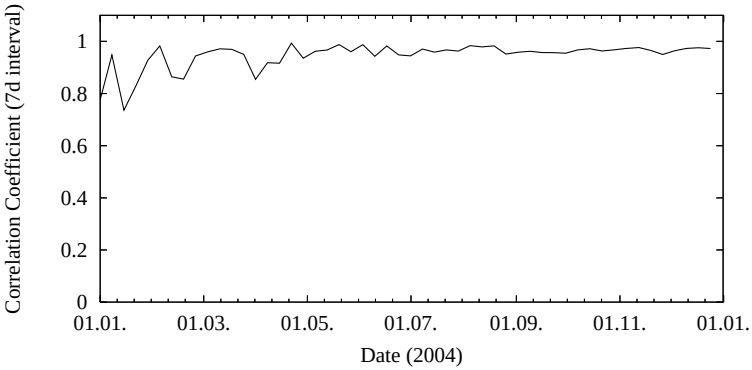

Figure 5.8: *UDP - Correlation coefficient, cource IP*

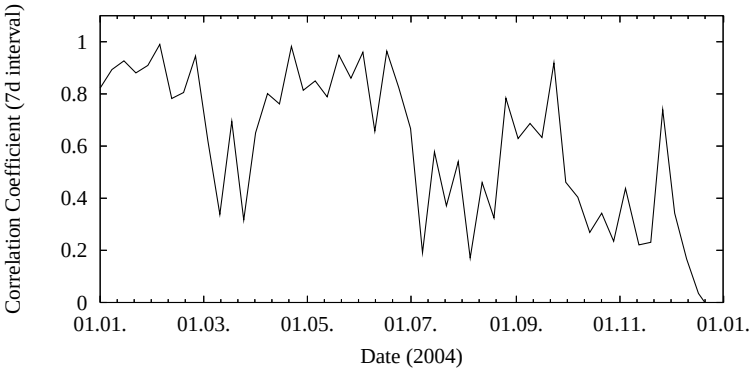

Figure 5.9: *UDP - Correlation coefficient, destination IP*

two values, which is still meaningful when the value pairs are all clustered together in a small area.

Since we do not know the distribution of $H_{lzo}(x) - \tilde{H}(x)$, we cannot compute its standard deviation directly. Instead, we will estimate the standard deviation σ by s defined as

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

were $\overline{x}$ is the arithmetic mean of $H_{lzo}(x) - \tilde{H}(x)$ over the measurement

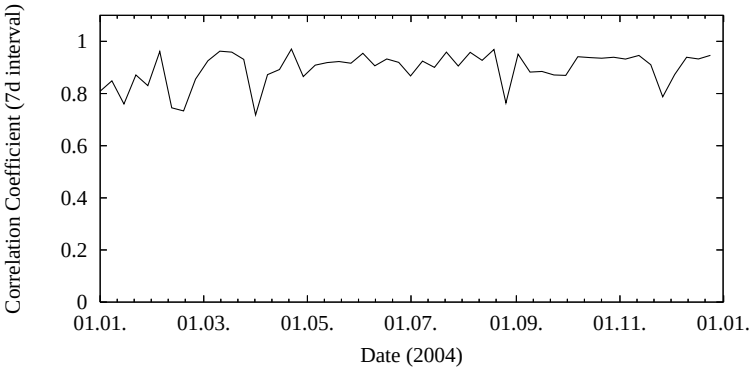

Figure 5.10: *UDP - Correlation coefficient, source port*

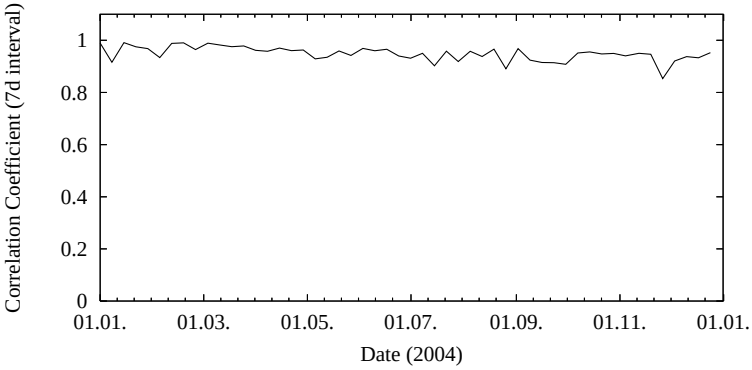

Figure 5.11: *UDP - Correlation coefficient, destination port*

interval (e.g. a day). The x_i are the values for $H_{lzo}(x) - \tilde{H}(x)$ for the individual measurements on the basis date. We use a random error assumption.

In our case the measurement intervals are 5 minutes long without overlap, as before. Finally, N is the number of basis intervals we calculate the standard deviation for, i.e. $N = 288$ for a day.

Findings

The plot in Figure 5.13 shows s calculated individually for each day in 2004 for TCP. The plot for UDP is in Figure 5.14.

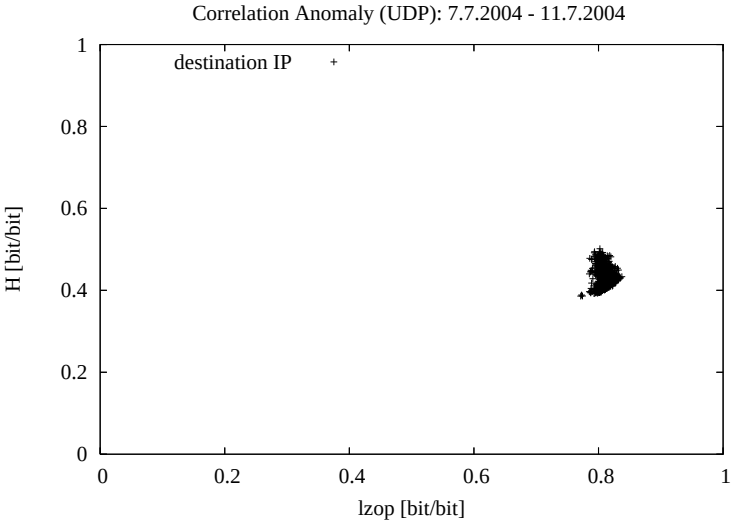

Figure 5.12: *UDP - Correlation anomaly*

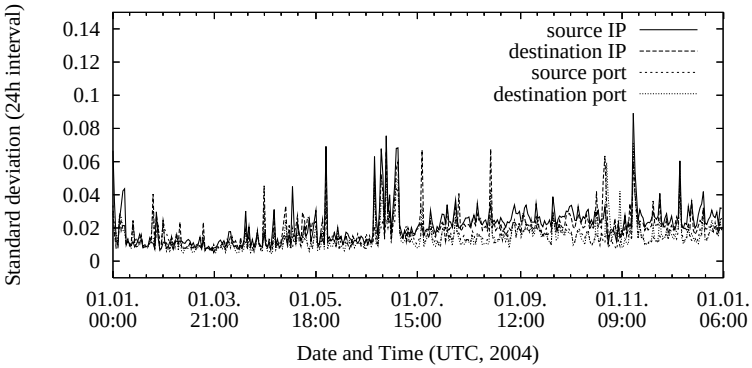

Figure 5.13: *TCP: Estimated standard deviation of $H_{lzo}(x) - \tilde{H}(x)$ per day*

It can be seen that the standard deviation for all IP and port fields is consistently very low over the whole year for both TCP and UDP traffic. We omit detail plots, since they do not show more than the yearly plots. We observe that attacks of any kind do not lead to large differences between entropy estimated by value frequency and entropy estimated by LZO compressibility.

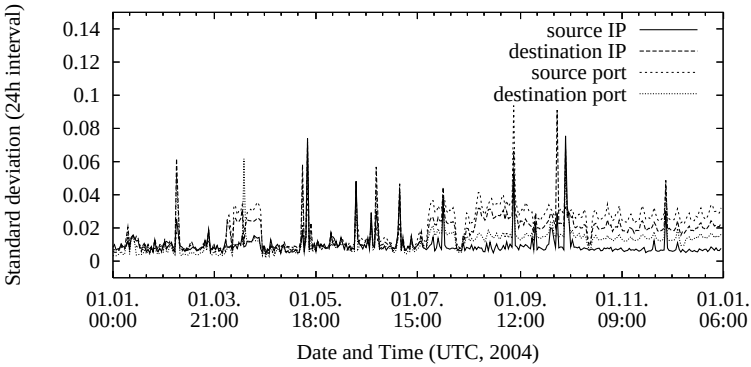

Figure 5.14: *UDP: Estimated standard deviation of $H_{lzo}(x) - \tilde{H}(x)$ per day*

This demonstrates that LZO compressibility is a good approximation for relative entropy changes in the observed data. However, it should be noted that while the match is quite good for each individual day, there are long-term drift effects that do not show up in these plots.

Scatterplots

In order to demonstrate some of the observed effects further, we provide a number of scatterplots for longer time intervals. The measurement interval length is 5 minutes, as before. All plots have $\tilde{H}(x)$ on the vertical axis and $H_{lzo}(x)$ on the horizontal one. We give one scatterplot for the whole year 2004 data and use plots for individual quarters of 2004 to illustrate specific effects.

Ideally the scatterplots would show a 1:1 relationship between $\tilde{H}(x)$ and $H_{lzo}(x)$, i.e. the measurements would be clustered closely around the z = y line. The actual results show a linear relationship below this line in some cases, i.e. compression reacts stronger to entropy changes than direct estimation by value frequencies. Other cases show linear clustering on a line parallel to the x = y line, i.e. $H_{lzo}(x)$ is larger than $\tilde{H}(x)$. We will now discuss the individual results.

TCP: Source IP

The scatterplot for the source IP addresses can be found in Figure 5.15. As can be seen $\tilde{H}(x)$ and $H_{lzo}(x)$ are strongly correlated for source IP addresses.

There is some shift over the year. Figure 5.16 shows only the second quarter of the same year. Clearly the values are less scattered. The reason is that less long-term parameter shift occurs due to the shorter measurement interval.

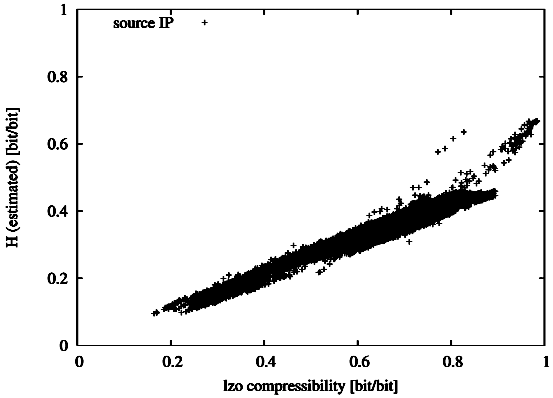

Figure 5.15: *TCP: $\tilde{H}(x)$ vs. $H_{lzo}(x)$, source IP, 2004*

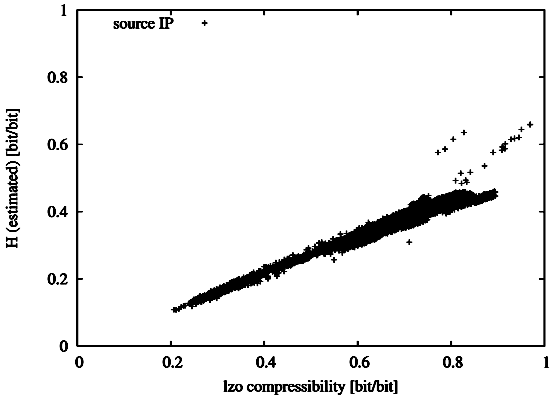

Figure 5.16: *TCP: $\tilde{H}(x)$ vs. $H_{lzo}(x)$, source IP, 2nd quarter 2004*

TCP: Source Port

The scatterplot for the source port numbers can be found in Figure 5.17. The measurements show that $H_{lzo}(x)$ values go up to 1, while $\tilde{H}(x)$ values remain below about 0.85 with the exception of a small number of measurements. The TCP source port field is clearly a high-entropy field. One other effect that can be observed is that the left side of the plot seems to have two "tails", one at about 0.2 above the other. This can also be seen very clearly in the scatterplot for the third quarter in Figure 5.18. The effect is less pronounced in the other quarters (not shown) but still there. Since the number of dots in both tails is very small, we expect they are due to some specific scanning activity that may have lasted only hours but is recurring during the whole year. One tail would be $H_{lzo}(x)$ without the scanning activity, the other $H_{lzo}(x)$ with the scanning activity. Note that within the respective tail, the ratio $\tilde{H}(x)$ vs. $H_{lzo}(x)$ is quite linear.

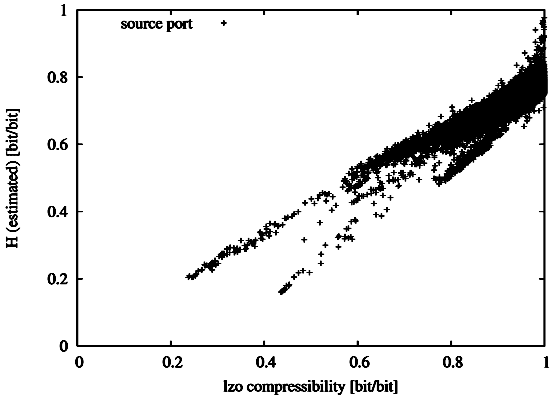

Figure 5.17: *TCP: $\tilde{H}(x)$ vs. $H_{lzo}(x)$, source port, 2004*

UDP: Source IP

The scatterplot for the source IP addresses can be found in Figure 5.19. It shows a clear, almost straight line, with good concentration. The individual quarters show no remarkable differences, so we omit them.

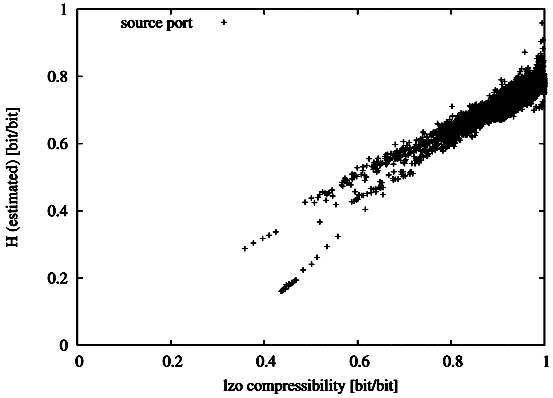

Figure 5.18: *TCP: $\tilde{H}(x)$ vs. $H_{lzo}(x)$, source port, 3rd quarter 2004*

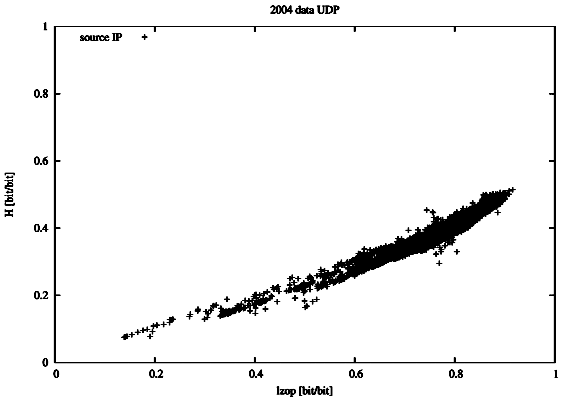

Figure 5.19: *UDP: $\tilde{H}(x)$ vs. $H_{lzo}(x)$, source IP, 2004*

UDP: Destination IP

The scatterplot for the destination IP addresses can be found in Figure 5.20. As in the TCP case, it shows a "bulge" around coordinates (0.8, 0.4). Remark-

ably this bulge is the only thing left in the fourth quarter (see Figure 5.21), where the number of measurement intervals with good LZO compressibility is low. This is due to a rise in UDP host scanning activity towards the end of the year 2004.

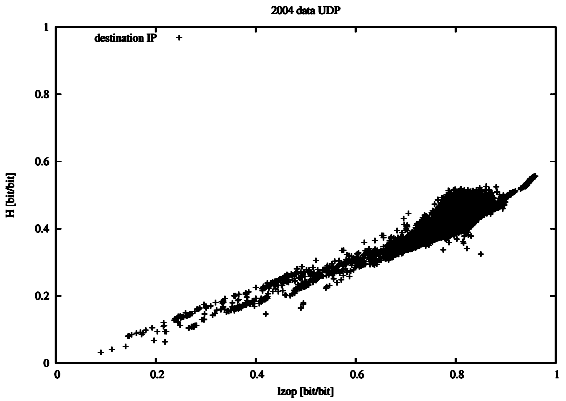

Figure 5.20: *UDP: $\tilde{H}(x)$ vs. $H_{lzo}(x)$, destination IP, 2004*

5.5 Discussion

As the analysis by linear regression shows, entropy estimation by compression has a low relative error when applied to traffic source IP, source port and destination port fields. This finding holds for TCP and UDP traffic. For the destination IP field, however, there are significant errors, that do not follow the correlation trend at all. Scatterplots of measurement times with low correlation in the destination IP fields (Figures 5.6, 5.7, 5.12) show the reason for the errors: The destination IP entropy measurements have very low variability during low correlation times, as can be seen by the close grouping in the scatterplots. While both, sample entropy and entropy estimated by compression, reflect the low variability, noise has a far stronger effect on the correlation coefficient in this configuration, than for periods of higher entropy variability. Since we are primarily looking for larger entropy changes, this type of error does not represent a strong concern for our work, but it may be problematic in

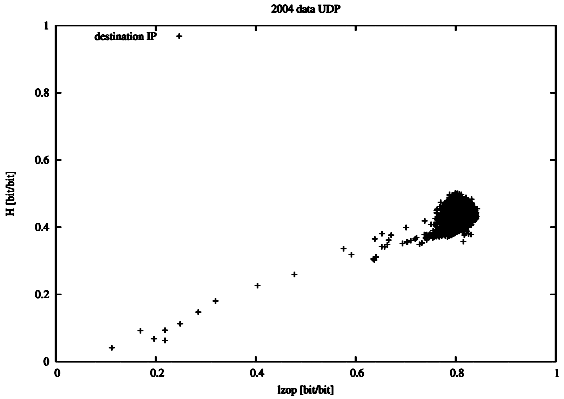

Figure 5.21: *UDP: $\tilde{H}(x)$ vs. $H_{lzo}(x)$, destination IP, 4th quarter 2004*

other applications. In Chapter 7 we will show that estimating entropy by compression leads to results that are comparable to estimating entropy by sample entropy, when applied to our worm detector. The main difference is a slightly higher number of false positives, when entropy is estimated by compression.

Chapter 6

Entropy Based Worm Detection

This chapter will present the actual detection mechanism. In a second step, we will explain how a detector can be calibrated for a specific network. Actual detection results, validating the design, are given in Section 7 of Chapter 7.

The primary design goal for a worm detector, is of course, to detect worms. Detection should be as early as possible, to allow more time for analysis and countermeasures. A second, just as important goal, is a low rate of false positives, i.e. network events not caused by worms, that trigger the detector. As it turns out, a worm detector based on entropy has to be calibrated for a certain outbreak strength and the specific port profile of the worm. The source port can be variable or fixed, and mainly depends on the worm implementation details. The destination port can be variable or fixed, depending on the vulnerabilities(s) used to compromise target systems.

When calibrating our worm detector for a specific worm profile, parameters for the individual detection thresholds have to be estimated. We use a procedure where worm traffic data with the desired trigger strength is inserted into baseline traffic from the target network. In a second step the rate of false positives is evaluated on a longer baseline traffic set without worm traffic in it. Calibration can also be done with observation data from real worms, possibly with modifications to the worm traffic. We use partial removal of the worm traffic in order to evaluate detector sensitivity for a detector that has had its sensitivity increased by dividing all thresholds by a factor larger than one.

6.1 Detector Design

6.1.1 Approach

As discussed in detail in Chapter 4, the outbreak of a fast Internet worm changes entropy characteristics of flow-level Internet traffic data. Typical fields that exhibit changes are source and destination IP address fields and port numbers. Since these fields are already sufficient to build a working detector, we limit the design to them. Refinements that use additional metrics, such as flow-counts per time, are discussed in Section 6.4.

For the actual detection, flow-level traffic data is first grouped into time intervals. Typically, these are discrete intervals of fixed length, e.g. 5 Minutes. The value sequence of the data field under consideration is then extracted and its entropy is estimated (see Chapter 5). The resulting time-series of entropy values is used to determine a baseline by taking the average of l interval measurements directly before the current interval I_n. This average is then compared with the value for I_n. If the difference exceeds a specific threshold and has the right sign (see Section 4.2.4), then for this data field I_n has a positive detection value. For a final decision, the other fields are evaluated in the same fashion. If all data fields of I_n have a positive detection value, then we have detected an outbreak event (or a false positive) in measurement interval I_n.

Note that we have to build a separate detector for each different worm behaviour profile. A worm profile is the set of signs of its characteristic entropy changes during outbreak, relatively to the baseline entropy values. It hence consists of four sign values, one for each IP address and one for each port number. If, e.g., a worm has scan traffic with a fixed source port address, it cannot be detected by a detector built for a worm with a variable source address port, since in the first case the source address port field entropy decreases during the worm outbreak, while in the latter case, it increases.

A first refinement, that serves to reduce false positives, is to not only require detection in I_n, but in k consecutive detection intervals $I_{n-k+1}, \ldots, I_n$. In this case the baseline average is computed over the intervals $I_{n-k-l}, \ldots, I_{n-k}$ to avoid influence on the baseline by the already changed characteristics of the current detection intervals $I_{n-k}, \ldots, I_n$. With this refinement, we have an overall positive detection at time n, if we have a positive detection for all k consecutive intervals $I_{n-k+1}, \ldots, I_n$. The advantage is a reduced rate of false positives. The disadvantage is that detection latency is increased by $k-1$ times the interval length. Note that k may not be arbitrary long. It has to be

short enough that the worm scan activity is present to a sufficient degree in all k intervals, otherwise fast worms that stop scanning after a short time will not be detected. For a worm that finishes its spreading phase in k measurements intervals and then stops scanning, a k' detector with $k' > k$ may not detect the worm outbreak at all. Typical values for k are $k \in \{1, 2, 3\}$.

6.1.2 Design

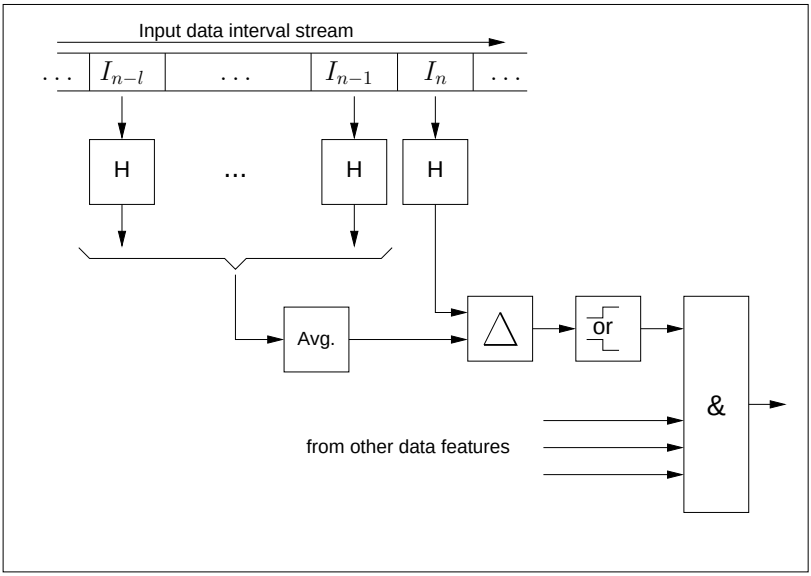

Figure 6.1: *Single interval detector*

The dataflow for a single interval detector can be found in Figure 6.1. On the top, left side, is the input data stream separated into intervals and reduced to the traffic feature under observation. For our detector, the observed traffic features are source IP address, destination IP address, source port number and destination port number. The entropy of each interval is determined (by an estimation procedure, see Chapter 5) and averaged over the baseline time range, giving the feature baseline value at time n. The baseline value is then compared to the value of the respective traffic feature in the current interval I_n and the difference (with sign) is fed into a threshold element. This element

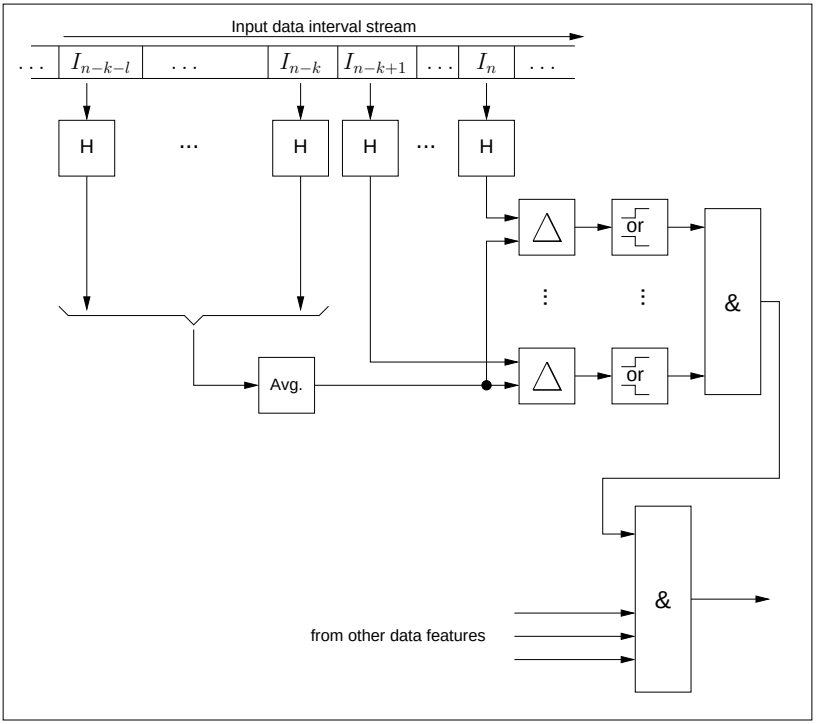

Figure 6.2: *Multiple interval detector*

outputs a value of true, if the difference between current value and baseline (with correct sign) exceeds the threshold set for the respective traffic feature.

The detection result for the traffic feature is then combined with the detection results for the other traffic features by an AND operation. If all are positive, a worm scan detection event is detected in interval I_n. Figure 6.3 illustrates the approach using a plot of the per-interval entropy values.

The detector can be extended to a k interval detector, as shown in Figure 6.2. Here, an event for this traffic feature is detected if the detection threshold is exceeded in all k detection intervals $I_{n-k+1}, \ldots, I_n$. A k interval detector for $k > 1$ has a detection latency that is higher by $k-1$ times the interval length. Its primary advantage is that it is less likely to produce false positives, since any event that triggers it has to be present in all k detection intervals.

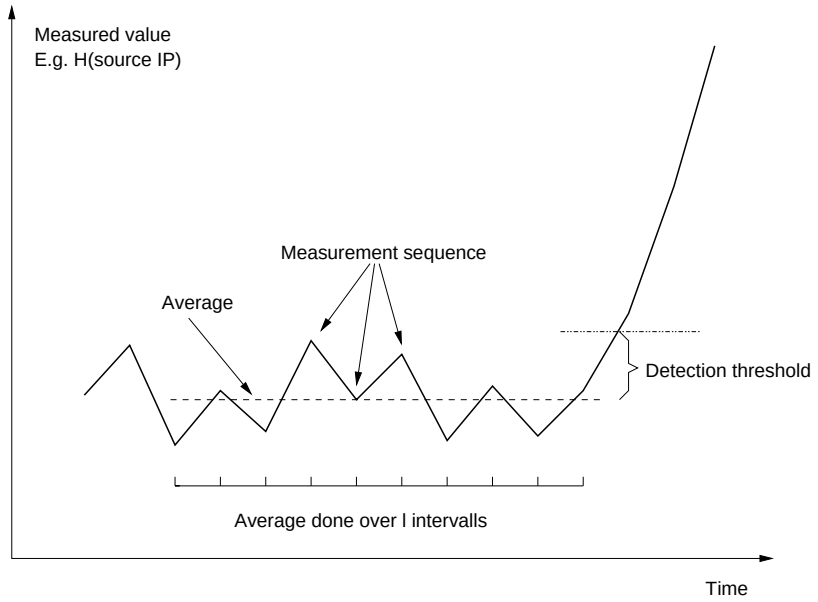

Figure 6.3: *Single interval detector on data plot*

6.2 Calibration

Calibration of the detector is necessary for the following reasons:

- Worm scan activity and hence traffic entropy characteristic changes can be arbitrarily low. Therefore it is not possible to define a definite maximum detector sensitivity. The sensitivity has to be selected so that both the sensitivity level and the rate of false positives is appropriate for the target network and intended purpose.
- Baseline traffic will be different from network to network. The likelihood of false positives of a specific detector parametrisation has to be evaluated for each network with its specific traffic mix.

Also, keep in mind that even if a specific worm or worm model is used for calibration, the detector is not specific to it, but rather at least sensitive enough to detect worms that generate scanning traffic of the same type with the same or higher intensity. It is also quite possible to calibrate the detector

on one worm model and then parametrise it to be more sensitive by a specific factor. If one detector is to be calibrated for several worms with the same entropy profile but different intensities, it is possible to calibrate it on each worm individually and then choose the most sensitive value found for each parameter.

Of course choosing threshold values without adjusting them to a specific worm model is also possible. In this case the sensitivity will have to be evaluated on one or several worm models.

Assuming that a measurement interval length has been selected, the parameters to be determined and tuned in order for the detector to work well are the following:

1. The thresholds in the individual threshold discriminators working on the different input fields
2. The sign for each thresholds
3. The number of intervals to be used for baseline generation

The baseline period length parameter is the easiest in our experience. A value of one hour seems to work well. If, however, at the end of the calibration process, the rate of false positives is not acceptable, longer and shorter baseline periods can be examined as alternatives. Especially if a relatively slow worm needs to be detected, a significantly longer baseline period may be needed.

In order to determine the threshold values and signs, worm models are needed. For TCP the typical situation is that the source port will be variable and hence source port entropy will increase during a worm outbreak. Fixed source ports are possible though, if the worm designer does not use the OS network stack, leading to a decrease of source port field entropy during the outbreak. Destination port field characteristics depend on the vulnerability exploited. Typically the destination port will be fixed for TCP, since data transfer (needed to transport exploit code) with TCP requires a successful handshake. This in turn requires an open TCP port, which will have a specific port number. The IP address fields have a specific behaviour, that would be very hard or infeasible to change, see Chapter 4.

For UDP worms, the situation is more complicated. Any combination of port field entropy is possible. Still, the IP address fields entropy has one specific behaviour, as discussed in Chapter 4.

Once a worm model is determined, a mix of base traffic and worm outbreak traffic is needed. An ideal solution is of course real traffic that was recorded during an actual worm outbreak. Worm models and a possible approach to worm traffic synthesis are discussed in Section 3.7 of this thesis. Of course baseline traffic can be synthetic as well, but we expect that obtaining realistic baseline traffic by simulation may be actually the same or more effort than doing traffic measurements.

With this traffic, the thresholds are adjusted so that each one forms a sharp detector. This can be done, e.g., by determining the exact minimal absolute threshold for timely detection and then relaxing the value by, e.g., 5%. (See Section 7, for a discussion of the effect of relaxing the thresholds.) After this, a significant amount of base-traffic without worm outbreaks in it is needed. On this the rate of false positives is determined by feeding it to the detector. If the rate is unacceptable, the parameters can be refined or additional measures can be implemented to make the detector more discriminating. See Section 6.4 for a discussion of several possibilities.

The calibration process needs the following steps:

1. Get real traffic observations for your network
2. Get worm observations for your network or define synthetic worm models. These supply the signs for the individual threshold values.
3. If your worm traffic is defined only by a model, generate worm traffic and insert the individual flows into a section of baseline traffic at random positions.
4. Determine tight individual thresholds for timely detection.
5. Divide the thresholds by a factor in order to increase the sensitivity. The sensitivity level of a parameter setting is given by the division factor. The tight threshold settings have a sensitivity level of 1.0.
6. Determine the number of detection intervals, if a multi-interval detector is to be used.
7. Determine the rate of false positives on a large section of baseline traffic without worm propagation events.
8. If the rate of false positives is unacceptable, reconsider the worm model, as well as, steps (5) and (6) and try variations of the baseline length. Further possibilities can be found in Section 6.4.

If several worm models are used, the calibration process should be used for each one. It is possible to either build a single detector that uses the minimal absolute threshold of individual detectors, or to build all individual detectors and combine their individual outputs.

6.2.1 Calibration Example

As an example, we will use the Blaster worm, see also Section 7.2.1 and Section 7.4. Blaster uses TCP for its initial contact to the target, and therefore has to use its real source IP address in the traffic. Hence the source address field in all scan traffic from each individual infected host will be fixed. This means, that the worm scan traffic decreases the source IP address field entropy and the threshold sign is negative. Every scanning worm has to use a variable target IP address, otherwise the attack traffic would not reach the target. The effect is that the target IP address field entropy is increased by the scan traffic, and hence the detection threshold sign is positive. Blaster uses a vulnerability in port 135/TCP on the target system, i.e. the target port number in its scan traffic is fixed. As a consequence, the target port number entropy is lower, compared to normal traffic, leading to a negative sign for the target port number detection threshold. Source port numbers in Blaster scan traffic are variable and increase source port number entropy, resulting in a positive threshold sign for the source port number field. Table 6.1 shows the resulting worm profile. With this profile and measurement data from the SWITCH network, we obtain the set of detection thresholds shown in Table 6.2. This concludes initial calibration of the detector for a tight fit to the worm.

The detector sensitivity can then be increased by dividing the values in Table 6.2 by a value larger than one or decreased by dividing them by a value between zero and one. As a next step, the now parametrised detector should be run on a set of baseline data to determine the number of false positives it generates. If the result is unacceptable, variations of the parametrisation can be tried or the detection interval number k can be increased.

source IP	destination IP	source port	destination port
-	+	+	-

Table 6.1: *Blaster worm profile. "+" means entropy exceeds baseline during outbreak, "-" means entropy decreases below baseline during outbreak.*

Sensitivity	source IP	destination IP	source port	destination port
tight (1.0)	-0.06	+0.05	+0.062	-0.16

Table 6.2: *Detection thresholds for Blaster*

6.2.2 Risks of Synthetic Data

While generating synthetic worm traffic is not a major problem (see Chapter 3), synthesising realistic baseline traffic is a hard problem. The specific difficulties for the problem of worm detector calibration lie in the fact that the baseline is needed in order to evaluate the rate of false positives for a specific detector parametrisation. Since false positives are typically caused by anomalies, such as conventional scanning, (D)DoS attacks and the like, false positives are caused by anomalies themselves. Synthesising traffic with a realistic anomaly profile is infeasible without determining this anomaly profile on the target network first. This in turn needs real traffic observations. We expect that generating realistic synthesised baseline traffic is actually significantly harder than recording real traffic in the first place. In addition, it very likely needs knowledge of the specific parametrisation of the detector that is to be calibrated using it, which leads to a circular dependency.

An alternative to using synthesised baseline traffic can be to use baseline traffic from a different network and to adapt it to the specific network conditions of the target network. For global anomalies it should be feasible to obtain a realistic baseline in this way. For local anomalies, that are specific to the target network, this process is however likely to remain unsatisfactory.

6.2.3 Reducing False Positives

One approach to reducing false positives is to use counter-indicators. For example, if entropy metrics indicate massive scanning activity, but the number of traffic targets in the local network does not increase, this indicates an attacker that already knows which hosts are reachable in the local network. In this case the source of the anomaly may still be a worm, however not one with random scanning, but rather one with a precomputed hitlist (see Section 3.4.3). Additional parameters of this nature can be used to generate counter-indicators with mostly the same detection mechanism as presented in this

chapter. In the presence of a counter indicator, the original positive detection is either suppressed or reported in a modified form that indicates it may be a false positive.

A second application is to generate counter-indicators for known anomalies. The idea is that the presence of a positive counter-indicator value invalidates a positive detection result. For example, if there are (D)DoS attacks against the local network from a specific source IP range or with specific traffic characteristics, this can be detected separately and used to suppress worm detection that triggers on the (D)DoS traffic.

Of course there is the risk of attackers using cover traffic, e.g. by conducting repeated (D)DoS attacks against a network, and then executing low-intensity attacks in parallel. However, since this example is not a global anomaly, but a targeted attack against a specific network, it is outside of the focus of this work and better addressed by standard IDS and IPS mechanisms.

6.3 Scalability

6.3.1 Larger Networks

For larger networks, the detector generally benefits from lower noise, and hence from lower numbers of false positives. As a downside, estimating the entropy of the individual traffic flow fields becomes computationally more expensive. If entropy is estimated by compression, as discussed in Chapter 5, the computational effort increases linearly with the size of the input, while memory consumption remains constant. If sample entropy is used, the computational effort also increases linearly with the input size, while the memory need increases linearly with the number of different values seen in a measurement interval.

We believe that with estimation by compression, our detection approach remains feasible and relatively cheap to implement, as long as the problem of importing the raw flow data into computer memory is possible. If the flow data can no longer be imported in real-time, then a flow based approach fails or is degraded to a post-mortem analysis function. Still, for this situation it is possible to work with a randomly selected subset of the exported data or to do entropy estimation for different subsets of the flow data on different machines and average the results.

6.3.2 Smaller Networks

The problem with smaller networks is that the observed traffic data has a larger noise component, since small events, such as a download to an individual computer, can have a significant impact on the overall traffic characteristics. We expect that our detection approach will produce an unacceptable number of false positives for small networks. Since it is not targeted at this type of scenario, we feel that this is not a drawback.

6.4 Refinements

It is possible to improve on the basic detector design presented in this chapter. Improvements may, for example, be desirable in order to obtain more specific detection results or to reduce detection latency.

6.4.1 More Specific Detection Results

A primary approach is to run several instances of the detector with different parametrisation, e.g. more sensitive and less sensitive ones. More sensitive ones can already react to lower-intensity scanning and can be used to generate a pre-warning. Another possible variation is to run several multi-interval detectors with different detection interval numbers. With this it becomes possible to distinguish between short, high-insensitivity scanning activity and worm-scan activity. The worm-scan activity will typically persist over several measurement intervals, while high-intensity scanning for other purposes often stops again after a few minutes.

6.4.2 Reducing Detection Latency

The primary means of reducing detection latency with single-interval detectors is to use a sliding observation window, i.e. a detection interval that is moved by a fraction of the interval length for each detection step. On average this can reduce detection latency by up to half the interval length.

When using multi-interval detectors, it is possible to use more sensitive parametrisation in the earlier detection intervals. This can have significant benefits for slower spreading worms, but increases the rate of false positives.

Chapter 7

Detector Validation

The purpose of this chapter is twofold: First, we will show that the detector design from Chapter 6 performs well on real-world network data. Second, we will demonstrate that estimating entropy by compression instead of by sample entropy is a valid alternative, by providing detection results both for a detector using sample entropy and for one using compression.

An additional validation possibility is the use of simulated traffic. We will discuss this possibility briefly in Section 7.7

7.1 Validation Basis Data

All measurements are done on real data from the SWITCH network, see Appendix C. The validation data spans the first six months of 2004. In order to reduce the data to a manageable size, we only use the data exported by the `swiCE1` and `swiCE2` routers. This represents about half the number of flows seen on the SWITCH network border routers. A second reason for the reduction is that the flow export engine of the third border router, `swiIX1`, was partially overloaded at the time the validation data was recorded, which introduced additional noise and flow-loss, especially at high-load times, such as worm outbreaks and DoS attacks. Depending on the detector parametrisation, this can cause both a higher and a lower number of false positives and would generally have led to unrealistic measurements.

In the following, we evaluate detector characteristics on the Blaster and Witty worm outbreak. For the Blaster worm detection measurements, we use

an additional 3.5 day long data interval from 2003 that contains the initial Blaster outbreak. All measurements are done first with tight thresholds and then with increased sensitivity by dividing the thresholds all by the same factor.

7.2 Worms Used for Validation

7.2.1 The Blaster Worm

We are considering the first observed variant, Blaster.A. For brevity we will refer to it as "Blaster". Blaster had its initial outbreak on August 11, 2003, approximately between 16:30 UTC and 17:00 UTC. The initially infected population was in excess of 150.000 hosts, according to the Internet Storm Center [10]. Saturation (i.e. 90% of the reachable and vulnerable population is infected) was reached after approximately 8 hours. Blaster targeted a vulnerability in the Microsoft RPC mechanism, accessible via Port 135/TCP. A detailed analysis of the Blaster worm and its infection mechanism can be found in [37].

7.2.2 The Witty Worm

We use the Witty worm [90, 104], first observed on March 20th, 2004, as a second example. Witty attacks a specific firewall product. It sends attack datagrams to random addresses with a *fixed* source port of 4000/UDP and *variable* destination port. The firewall product is then compromised while inspecting the datagram. While the attack payload is less than 700 bytes in size, it is padded to a total, random size between 796 and 1307 bytes for each datagram sent. No additional communication is needed to complete the infection.

Witty infected only about 15'000 hosts, but was clearly visible in the UDP entropy statistics. While there is far less UDP traffic than TCP traffic in the Internet, UDP traffic is aggregated to a lower degree in the SWITCH routers and there are almost as many UDP flows exported as TCP flows. The Witty worm analysis by CAIDA [90, 93] places the initial Witty outbreak at 8:45:18pm PST on March 19 (i.e. 4:45:18 UTC on March 20), 2004, which is the most precise outbreak time we were able to find in the literature.

One notable characteristic of Witty is that it was very competently designed [115]. In fact, analysis of the worm code did not reveal any imple-

mentation weaknesses, contrary to many other worms. The other notable characteristic of the Witty worm is that it managed to reach saturation within about 15 minutes, despite the low number of vulnerable hosts. Before the Witty outbreak, it was not known whether a comparably short time to saturation was feasible for such a small number of vulnerable hosts and whilst using a random scanning strategy.

7.3 Quality Measures

We use several different metrics for measuring the actual result quality of a worm detector. These are detection delay, false negatives and false positives. Detection delay is the period of time after worm propagation traffic becomes visible until the worm is detected. False negatives describe which outbreaks are not detected. False positives describe which, and how many, other non-worm anomalies are misclassified as worm outbreaks.

Detection Delay

With our detector, the detection delay depends on the measurement interval length. We have to gather data for a specific time and then process it to determine whether a worm outbreak event was visible during this measurement data interval. For our detector, this can only be done at the end of each measurement interval. In addition, the anomaly has to habe enough impact during a measurement interval in order to be detected in it. If, for example, a worm outbreak shows up weakly at the end of a measurement interval and not at all before the interval, then it will likely only be detected during the next measurement interval. If multi-interval detection is used, the overall observation time becomes the effective measurement interval. As optimisation, the basis interval can be shifted in an overlapping fashion to reduce detection latency.

False Negatives

A false negative is a serious problem. It basically means that an event, which should have been detected, was not detected at all. For our work, this is difficult to define. Since we do not consider slow worms, failure to detect a slow worm is not a false negative. In order to generate a false negative with our detector design, a worm would need to increase scanning activity very slowly, so that it becomes part of the baseline. However, this would automatically

make the worm a slow worm. Still, if the detector is parametrised with very high detection thresholds, it is quite possible to miss outbreaks of fast worms. As in Chapter 6, we call the highest thresholds, that still lead to reliable detection of a worm, a set of *tight* thresholds. If they are exceeded in a detector parametrisation, the worm will not be detected at all or only later. False negatives can also be generated by choosing a very short baseline period, then the effects of the worm outbreak can become part of the baseline before the detection thresholds are exceeded.

It is quite feasible and very efficient to run several differently parametrised detectors in parallel, and then observe their different detection results. Rather than seeing the differences as a problem, we suggest that they actually represent a pre-classification of the observed worm outbreak event into different intensity classes.

False Positives

False positives are not directly a problem. What makes them problematic is that they can overwhelm the second stage detection system, i.e. the system tasked with analysing alerts, as well as, selection and implementation of countermeasures. Today, the second stage is typically implemented by manual analysis. This approach does not scale well, and getting additional experts to perform the analysis is expensive and difficult. Therefore a high number of false positives can result in a DoS-like attack on the overall detection mechanism. In fact, in current IDS systems, false positives are one of the primary and often unsolved challenges.

In our scenario, detection quality increases when more detection intervals are used (as long as the detection interval length is kept constant). It is possible to continue detection with additional intervals after initial alerting and to de-alert the user. This makes some false positives transient. Also, depending on the specific network protection requirements, alerts from our worm detector can be combined with other measurements, e.g. network stability indicators. This provides the possibility to restrict fast (and less reliable) alerting to situations where network stability is potentially impacted. If network stability is not impacted, detection can then be done with longer delay (by using a multi-interval detector) and lower probability of false positives.

7.4 Validation Results for the Blaster Worm

Figures 7.1 and 7.2 show the entropy profiles during the Blaster worm outbreak, for sample entropy (top) and entropy estimated by LZO compression (bottom). The detector signal is shown for tight thresholds (see Section 6.2).

	source IP	destination IP	source port	destination port
Sample entropy	-0.06	+0.05	+0.062	-0.16
Compression	-0.08	+0.033	+0.06	-0.17

Table 7.1: *Blaster: Tight detection thresholds*

We calibrated one detector using sample entropy and one using compression on the real Blaster outbreak using the procedure described in Section 6.2. The resulting tight detection thresholds are given in Table 7.1. The detectors were then run on the DDoSVax TCP data for the first half of 2004. The basis measurement interval was 5 minutes in all cases. The baseline was determined over one hour, i.e. 12 measurement intervals. Measurements were taken only on flows entering the SWITCH network, since the host population within the SWITCH network is small enough to cause significant noise, and local events have a significant impact on the characteristics of flows leaving the SWITCH network.

To determine the dependency of false positives on the number of detection intervals in a multi-interval detector, measurements were taken with one, two and three detection intervals. Note that the number of detection intervals corresponds to k in Section 6.1 and in Figure 6.2. We use the simple multi-interval approach, where all detection intervals use the same thresholds. To determine the impact of more sensitive detection thresholds on false positives, the tight detection thresholds were divided by a factor and the measurements were then repeated. For Blaster, the factors 1.4 and 2.0 were used.

Since entropy is a combined additive and logarithmic measure, dividing a threshold by a factor of q can reduce the number of flows needed to trigger the detector by a factor significantly different from q. The actual effect depends on the concrete traffic mix. Table 7.3 lists the effects of increasing sensitivity for the Blaster worm as observed in the SWITCH data. The effect was determined experimentally by randomly removing a specific number of Blaster flows until detection failed.

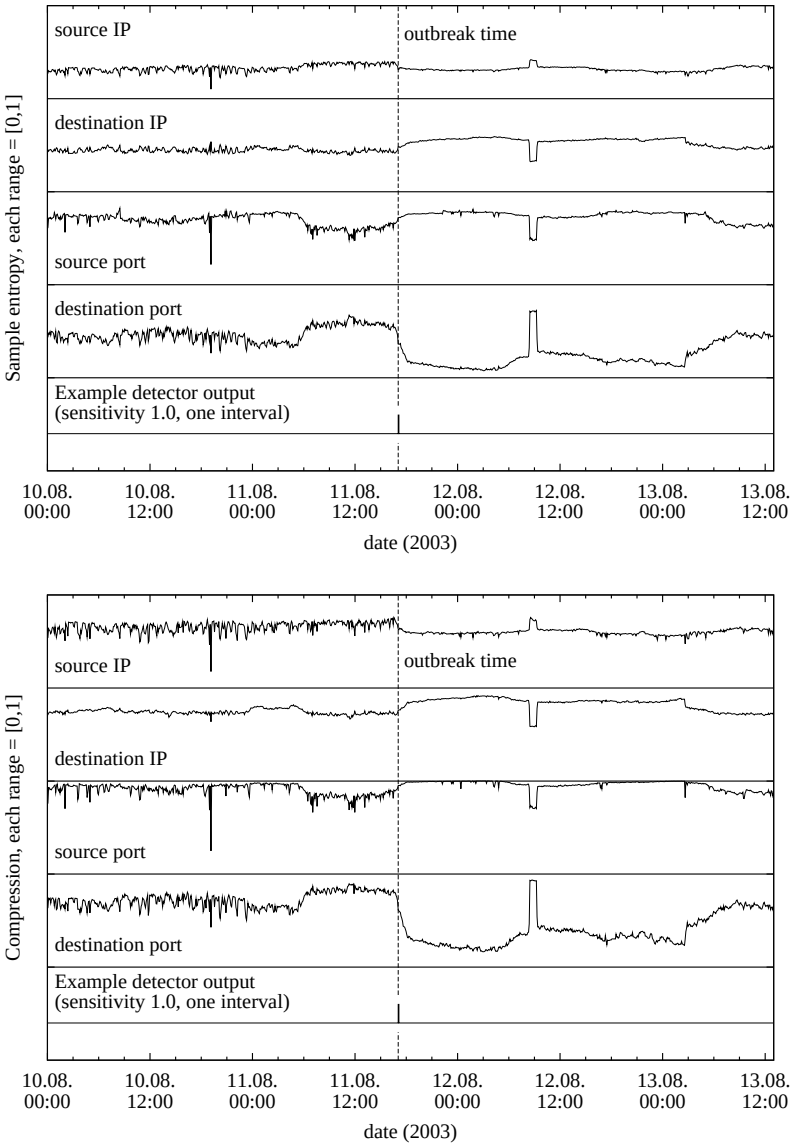

Figure 7.1: *Blaster worm: Sample entropy (top) and compression (bottom)*

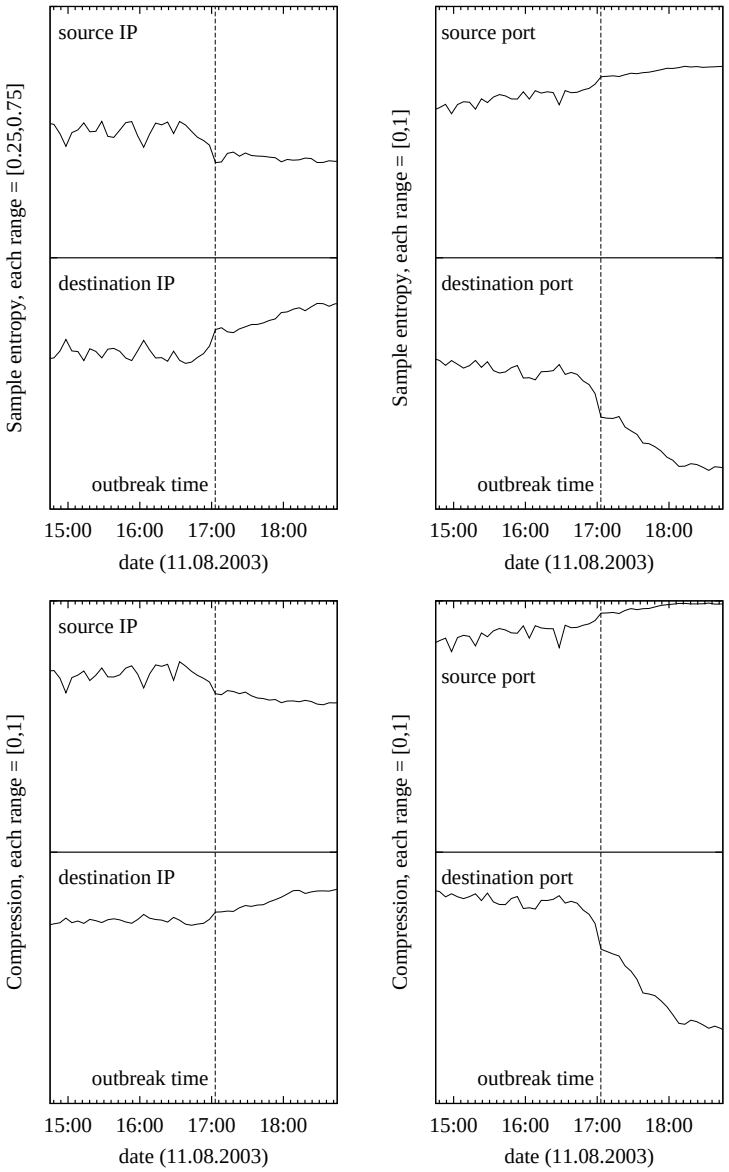

Figure 7.2: *Magnification of Figure 7.1 around outbreak time*

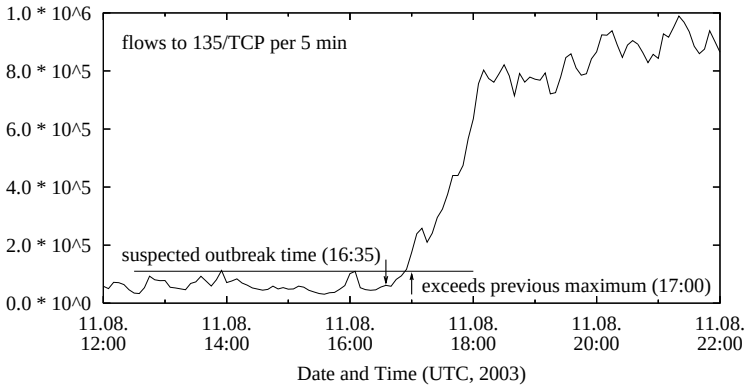

Figure 7.3: *Blaster worm: Flows to port 135 TCP*

Sensitivity	Sample entropy			Compression		
	Intervals			Intervals		
	1	2	3	1	2	3
tight	3 (0)	0 (0)	0 (0)	6 (0)	0 (0)	0 (0)
1.4	23 (1)	13 (1)	3 (0)	24 (1)	10 (1)	4 (1)
2.0	55 (14)	32 (3)	9 (2)	50 (8)	26 (2)	9 (1)

Table 7.2: *Blaster: False positives vs. threshold tightness*

Sensitivity	Blaster flow count	
	Sample entropy	Compression
1.4	0.739	0.740
2.0	0.596	0.600

Table 7.3: *Blaster: Sensitivity vs. reduction in number of Blaster flows needed to trigger the detector*

Our detector found the Blaster outbreak first in the measurement interval starting at 17:03 UTC on August 11th, 2003. More sensitive threshold settings did not result in earlier detection. While the exact Blaster outbreak time is unknown, the earliest time the Blaster outbreak may have been visible in the SWITCH traffic was around 16:35, when a slight increase in the

connection attempts to port 135/TCP could be noticed [37]. The first time the number of flows to port 135/TCP exceeded the peak number of flows to port 135/TCP in the 5 hours before the outbreak, was at 17:00 UTC, as can be seen in the traffic plot in Figure 7.3. The first successful infection of a host in the SWITCH network was at 17:42 UTC [37].

False Positives

The numbers of false positives during the six month reference data interval are listed in Table 7.2. Closer inspection of the false positives revealed that a large number was due to burst scan activity on several days in the second half of June 2004. The scan activity caused false positives with both entropy estimation methods used. The numbers in brackets in Table 7.2 represent the number of false positives without this specific scan activity.

The burst scans in June 2004 were all done at 5:25 UTC, 7:15 UTC, and with lesser intensity at 8:40 UTC. They lasted for up to an hour with bursts between 5 and 10 minutes long. The number of overall network flows per 5 minute interval entering the SWITCH network during the scans, was up to 4.5 times higher than observed directly before and after the anomaly. Closer inspection revealed that during these intervals a large number of flows was sent from a relatively small number of hosts to a large fraction of the SWITCH IP range, suggesting a targeted, combined host and port scan on the SWITCH host population.

7.5 Validation Results for the Witty Worm

In order to determine detection characteristics for an UDP based worm, we used the procedure in Section 6.2 to determine tight thresholds for the real Witty worm outbreak in the DDoSVax dataset. Again, we determined a set of parameters for a detector that uses sample entropy and for one that uses compression. The reference data set was the same as for the Blaster worm, but restricted to UDP data. As with Blaster, only flows entering the SWITCH network were used. Since the Witty worm outbreak occured within the first half of 2004, we initially used a lock-out time of 6 hours before and after the known outbreak time and detections within this time were not counted as false positives. As it turned out that there were no false positives within the lockout time period, no effort has been made to optimise it. Figures 7.4 and 7.5 show the UDP traffic entropy profiles during the Witty worm outbreak for

sample entropy (top) and entropy estimated by LZO compression (bottom). The detector output plot is for the tight detection thresholds given in Table 7.4.

	source IP	destination IP	source port	destination port
Sample entropy	-0.016	+0.061	-0.082	+0.21
Compression	-0.037	+0.091	-0.085	+0.20

Table 7.4: *Witty: Tight detection thresholds*

Sensitivity	Sample entropy			Compression		
	Intervals			Intervals		
	1	2	3	1	2	3
tight	0	0	0	0	0	0
1.4	0	0	0	0	0	0
2.0	0	0	0	0	0	0
3.0	0	0	0	0	0	0
4.0	0	0	0	1	1	1
6.0	1	1	1	3	3	2
8.0	7	1	1	4	3	2

Table 7.5: *Witty: False positives vs. threshold tightness*

The earliest time that we were able to detect the Witty worm by changes in the UDP traffic entropy profile was during the measurement interval starting at 4:45:00 UTC on March 20, 2004. This is consistent with observations by CAIDA [90, 93] that places the initial outbreak at 4:45:18 on the same day. Similar to the Blaster worm, increasing sensitivity did not lead to earlier detection and our detection time of 4:50 (end of the measurement interval) represents the best possible detection latency our detector can achieve with the given measurement interval length.

For the Witty worm, Table 7.5 gives false positive counts and the impact of increasing sensitivity (left side) and using multi-interval detection (top row). It can be seen that an increase of sensitivity by a factor of at least 6.0 (for sample entropy) and at least 4.0 (for LZO compression estimated entropy) is

Sensitivity	Witty flow count	
	Sample entropy	Compression
1.4	0.739	0.738
2.0	0.598	0.638
3.0	0.483	0.515
4.0	0.423	0.459
6.0	0.360	0.400
8.0	0.332	0.367

Table 7.6: *Witty: Sensitivity vs. reduction in number of Blaster flows needed to trigger the detector*

needed to produce any false positives. The improvement in the number of false positives when using multi-interval detection is relatively small for the range of intervals we used.

7.5.1 Validation Results for a Modified Witty Worm

Because of the special port characteristics of the Witty worm, namely a fixed source port and a variable destination port, we did an additional detection run on the reference interval with the port characteristics reversed, i.e. a variable source port and a fixed destination port. This simulates a variant of the Witty worm that uses a variable source port and a fixed destination port, i.e. the typical port characteristic of a worm that attacks a service running on a specific port. The detection thresholds were the same as for the unmodified Witty worm.

The detection results are listed in Table 7.7. It can be seen that for sample entropy a sensitivity increase of 4.0 is needed to produce any false positives. The results for compression are similar, but a bit worse, with the first false positives occurring at a sensitivity of 3.0. Using multi-interval detection is very effective in reducing false positives for both entropy estimation methods.

7.6 Discussion

We have demonstrated the validity of our detector design on two well-understood fast Internet worms using a significant amount of real traffic data

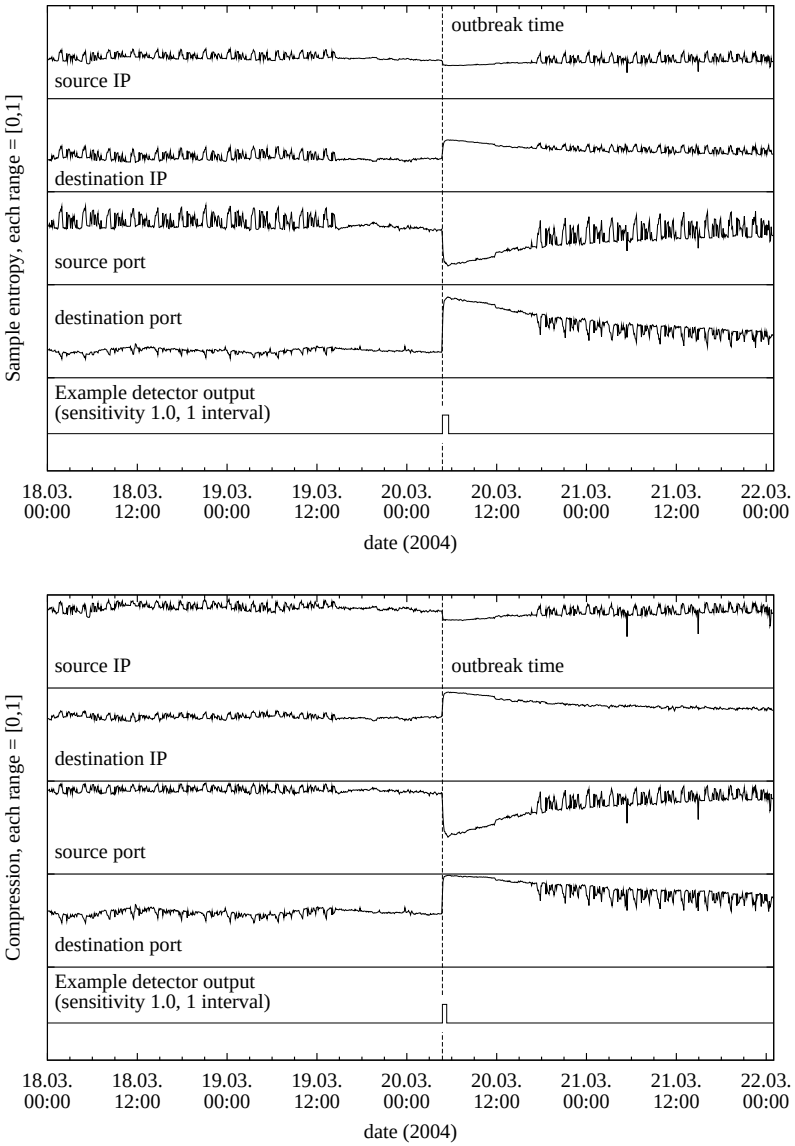

Figure 7.4: *Witty worm: Sample entropy (top) and compression (bottom)*

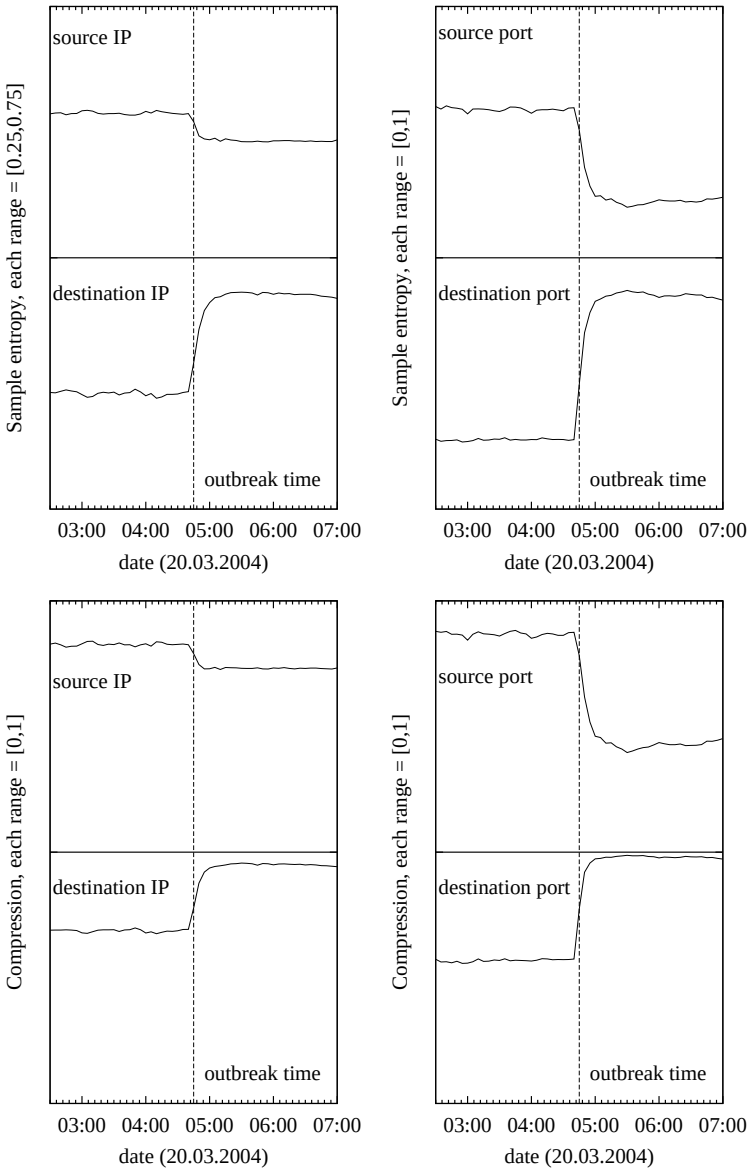

Figure 7.5: *Magnification of Figure 7.4 around outbreak time*

Sensitivity	Sample entropy			Compression		
	Intervals			Intervals		
	1	2	3	1	2	3
tight	0	0	0	0	0	0
1.4	0	0	0	0	0	0
2.0	0	0	0	0	0	0
3.0	0	0	0	25	2	0
4.0	1	0	0	114	14	1

Table 7.7: *Modified Witty: False positives vs. threshold tightness*

from the SWITCH network. False positives were evaluated on half a year of SWITCH network traffic data, giving realistic measurements that also include rarer network events. The numbers of false positives found were low and can be reduced further with multi-interval detection. This demonstrates that the detector design is capable of working well with real network data. We also measured the effects of increasing detector sensitivity by dividing all thresholds by the same factor and demonstrated that this approach to sensitivity adjustment is valid.

In addition, we determined the number of false positives for a detector parametrisation for a hypothetical Witty worm variant with inverse port characteristics. The results demonstrate that the good detection results for the Witty worm are not due to its particular port profile.

Detection Quality

For both worms, detection latency is low. For the Blaster worm, we have detection in the first interval where the number of flows to port 135/TCP (the Blaster scan target port) exceeds the number of flows to this port during the time period shortly before. This result can be seen as optimal for a traffic-mix based approach such as ours. In the case of the Witty worm we have detection in the first measurement interval that contains the time of the actual Witty worm outbreak. This is optimal.

With regard to false positives, we see no or very low numbers of false positives for tight thresholds. When sensitivity is increased, the false positives for the Blaster worm increase relatively fast for single interval detection. Multi-interval detection behaves significantly better under increased sensitivity. For

the original and modified Witty worm, the observed numbers are excellent, even with increased sensitivity. This difference is due to the stronger changes Witty caused in the traffic entropy profile, because of its more aggressive scanning strategy and hence higher spreading speed. Not surprisingly, slower worms may take longer to detect reliably.

The detector for the modified Witty worm had moderately worse characteristics than the one for the original Witty worm, namely a higher number of false positives. The likely explanation is that the port profile of scan traffic for the modified Witty worm closely resembles common port scans. These can then trigger the modified Witty detector if its sensitivity is high enough.

Sample Entropy vs. Entropy Estimated by Compression

For the Blaster measurements, the detection results obtained with sample entropy and with entropy estimated by LZO compression are very similar. One difference is that the LZO based detector at sensitivity level 1.0 is a bit more sensitive to one specific, repeated high-intensity scanning activity at the end of the validation data interval. We have stated the observed numbers of false positives with and without this specific scanning activity.

For the Witty worm, the number of false positives are moderately worse for LZO estimated entropy than for sample entropy. In addition, decreasing detection thresholds has a weaker effect with LZO compression, i.e. for a specific sensitivity, LZO compression based detection actually needs more Witty flows than sample entropy based detection. However, in both cases the results are very good and a significant increase in sensitivity is needed to produce any false positives at all for the half year of reference traffic data. The same effects can be observed for the modified Witty worm. Here again, the sample entropy based detector performs better.

The results obtained demonstrate that LZO compression forms a valid alternative to sample entropy when estimating flow-stream entropy profile changes. Since LZO compression offers a moderate speed improvement and a smaller memory footprint (see Section 5.3) in comparison to sample entropy, using LZO compression can have significant benefits in practical implementations.

Increasing Sensitivity

Increasing detector sensitivity increases the likelihood of false positives. We utilised division of the detection thresholds by constant factors in our measurements, i.e. the same decrease in the individual detection threshold values. An alternative strategy is to calibrate the detector on a worm outbreak event of smaller magnitude. This could potentially increase detection quality and can be used for applications where a simple constant factor sensitivity increase is not good enough. The better result quality comes at the price of higher effort, since the measured or simulated worm traffic data has to be changed.

7.7 Simulation as a Validation Tool

Using synthesised data or simulation to validate an anomaly detector is problematic. An example of a dataset that may have contributed to the development of IDS systems that have trouble dealing with real traffic data because of its, among other things, failure to provide a realistic baseline, is the MIT's Lincoln Lab data from 1998 [64, 70].

If simulation is used to validate a worm detector design, two aspects are critical:

- The worm traffic must be simulated realistically
- The baseline traffic must be simulated realistically

Simulating worm traffic is not too difficult, if a realistic worm model and a realistic Internet model are available. We present one possible approach in Chapter 3.

Simulating realistic baseline traffic for the evaluation of false positives is harder. One possible approach [18] that is currently being investigated, is to identify traffic features of real traffic and then to re-synthesise this traffic with similar characteristics. This however still needs real traffic with the desired characteristics, which is a significant drawback, since the real traffic data could be used directly for validation with less effort and more accurate results. At the same time, it is currently unknown if the re-synthesised traffic will have realistic characteristics with regard to false positives in a worm detector. The main motivation for this research activity is therefore not to allow simulation but to provide anonymisation of a network flow dataset, in order to deal with privacy restrictions.

Whether simulation of a realistic baseline is feasible without using a set of real traffic observations as a basis, is currently unknown. If it were feasible, the simulation results would still need to be compared to results obtained with real traffic to demonstrate that they are realistic. Otherwise there is a real possibility that simulations will provide results the experimenter expected or desired, rather than meaningful ones. This again means real traffic data has to be available and the simulation is, again, not really needed and the second best choice. For these reasons, we are convinced that at this time there is no adequate substitute for testing backbone worm detectors on real baseline traffic.

Chapter 8

Conclusion

We have identified traffic entropy changes present during the propagation phase of fast Internet worms. The changes were analysed theoretically and measured on real traffic from a high-volume network. The insights gained have been used to build a real-time capable detector for fast Internet worms that has low detection latency, a low rate of false positives and low resource needs. The detector scales well and is suitable for real-time online use on very fast networks. Validation of the detector design has been done using an extensive set of real Internet backbone flow-level traffic data.

8.1 Review of Contributions

Design, build and operate a NetFlow data capturing system for the SWITCH network (Engineering)

Since no NetFlow data capturing, transport and storage software suitable for the purpose of of this work was available, we designed and implemented a suitable system. The capturing system handles all tasks involving capturing of the NetFlow UDP datagrams streams to file, compression and transport to long-term storage. Significant effort is spent on fault tolerant operation and automatic recovery in case of errors and system outages. A problem with packet loss in the UDP datagram stream, due to very fast data export bursts and too small socket buffers in intermediate hosts replicating the datagram stream, was analysed and solved as well.

The capturing system has been operational without major outages since the beginning of 2003 and is still perfectly adequate to the task. In October 2007 the total amount of data captured and stored was about 30TB compressed (an estimated 90TB uncompressed with the `bzip2` compression used), which corresponds to roughly 1'800 billion individual NetFlow records, captured over a time of approximately 41'000 hours. The capturing system is described in Appendix C and in [107, 111].

Design and implement NetFlow data processing libraries and tools (Engineering)

At the beginning of this work, the available toolsets and libraries for handling NetFlow data were not suitable for processing large amounts of data or had a primary focus on forensics. Forensics is generally not concerned with the overall picture, but focuses on a small number of specific flows instead. Analysis with forensic tools is typically done interactively by a human operator.

The DDoSVax project and this thesis are specifically not targeted at forensic work, since that would violate agreements with the traffic data source used (SWITCH) and could violate Swiss privacy laws. As a consequence we created a NetFlow data toolset and associated libraries, suitable for batch processing of very large datasets with a focus on more abstract traffic properties. It forms the basis of all NetFlow processing work done in this thesis and has proven to be perfectly adequate to the task. It is suitable for large scale analysis as, for example, the validation work done in Chapter 7 shows, where half a year of traffic data was processed at a time.

The toolset is described in detail in Appendix A. It has also been used in other research activities, for example [20]. A public release as Free Software is in preparation.

One component of the toolset, namely the transparent compressed file reader library `cfiletools`, has been added to RIPE [1] RIS raw data toolset `libbgbdump`, adding support for `bzip2` compressed BGP data files.

Create a worm simulator to better understand worm characteristics (Engineering / Science)

In order to understand what limits propagation speeds of fast Internet worms, we saw a need for a simulator that could realistically simulate an Internet-wide worm outbreak. No such simulator was available, and hence we created

one as part of this work. It uses an Internet model that centres on the speed of the "last mile" Internet access connection statistics of home users and proved to be able to realistically simulate known worm outbreaks, up to the time when network saturation sets in. When saturation effects start to appear, realistic simulation becomes very difficult, since Internet network behaviour at saturation is very hard to predict and dependent on the specific characteristics of the traffic causing the saturation. In measurement data we also observed effects of manual intervention during worm outbreaks, such as installation of network filters, that a simulator cannot predict. Since we were primarily interested in the early behaviour during worm propagation, due to our interest in early detection of fast Internet worms, the simulator was perfectly adequate for our purposes.

The simulator is described in Chapter 3 and was the subject of a publication [110].

Model the entropy-effects of fast Internet worms theoretically (Science)

We were able to model entropy effects during worm outbreaks in a qualitative fashion. During a worm outbreak a lot of target IP addresses are contacted. Ordinarily, many of these would not have associated traffic, since no hosts are attached to them. At the same time the hosts doing these connection attempts are few during the early phase of a worm outbreak. We were able to show mathematically that this behaviour causes a specific impact on flow-level entropy scores for IP addresses, specifically a decrease in source IP address entropy and an increase in destination IP address entropy. Quantitative predictions require realistic modelling of the baseline traffic, which is the subject of ongoing research, see e.g. [19].

The theoretical foundation for the impact of worm propagation traffic on Internet flow-level traffic data is described in Chapter 4.

Identify and quantify the effect the outbreak of a fast Internet worm has on NetFlow dataset entropy scores (Science)

We made measurements on real traffic data from the SWITCH network, which connects most Swiss universities and some research centres. The advantage of this data is that it has a diverse traffic mix, as it also contains a lot of Internet traffic created by students. Observations of fast Internet worm outbreaks in this traffic data pool did confirm the presence of the entropy changes predicted by theory and allowed quantitative observations of these changes.

The measurements also served to identify baseline traffic characteristics and ultimately lead to the fast Internet worm detector design, which is at the core of this work.

Chapter 4 described the details of observations made on captured traffic data. The observed effects have also been described in [111].

Evaluate the suitability of compression for entropy estimation (Science / Engineering)

In order to explore the suitability of compression techniques as an alternative method of estimating entropy and entropy changes, we performed extensive measurements comparing entropy estimated by sample entropy and entropy estimated by compressibility. The results show that compression is a suitable method for estimating entropy changes in flow-level Internet traffic data. While accuracy is worse than for sample entropy, the results are viable for use in a detector for fast Internet worms. Memory consumption in particular is reduced dramatically, typically to a low and constant amount, while computational effort stays very low.

Chapter 5 describes the use of data compression for entropy estimation of flow-level traffic data. The evaluation in Chapter 7 shows that this approach increases inaccuracy only moderately. The idea of using compression as entropy estimator for the purpose of detecting fast Internet worms was published in [111].

The design of a detector for fast Internet worm outbreaks based on entropy measurements and evaluation of its characteristics (Science / Engineering)

In order to allow detection of fast Internet worms in high-speed networks, we created a detector that is capable of solving the detection task with high accuracy, low latency and low resource needs. The detector is suitable for real-time online-line use. The idea is to monitor localised changes in the entropy profile of IP addresses and port numbers for specific change patterns. The detector can be calibrated for different levels of sensitivity and worms with different traffic port profiles. The number of false positives can be lowered even further, if a slightly higher detection latency is accepted, by using a multi-interval detector that tests for the presence of worm-scan activity in several consecutive measurement intervals.

In addition the detector design allows easy combination of differently calibrated detectors into clusters. One advantage is that using detectors with different sensitivity levels can first provide very early detection with low reliability, i.e. a high rate of false positives. The detection reliability level can then be increased by slower, less sensitive detectors. This provides a prealerting capability. Clusters of detectors scale very well, since the main effort lies in the entropy estimation, which has to be done only once per traffic data stream.

The detector and its calibration procedure are described in Chapter 6. Its validation on real traffic data is given in Chapter 7.

8.1.1 Summary

We have solved both the scientific and engineering challenges this thesis work presented. A working, practical and efficient detector for fast Internet worms was designed, implemented and validated on real traffic data from a moderate-sized Internet Backbone network, using two well-understood fast Internet worm outbreaks as benchmarks. The detector is suitable for real-time online use. Measurements of CPU and memory consumption show that the design scales to significantly larger networks, without the need to reduce data volume by sampling or other means. To reduce memory consumption further, we have demonstrated that replacing sample entropy measurements with compressibility scores obtained with a very fast data compression algorithm can be used, with only a small impact on detection quality. This allows implementation of our detector with a very small memory footprint, as traffic data can be discarded immediately after compressing each individual traffic record.

At the same time, the core idea of the detector design has been examined and explained theoretically. Propagation traffic from a fast Internet worm has a strong one-to-many property, that is not present in baseline traffic. In addition, IP addresses that have no hosts attached to them, receive traffic from connection attempts by the worm. A theoretical argument shows that if the baseline traffic does not change and this type of traffic is added to it, the source IP address entropy decreases, while target IP address entropy increases. This is consistent with observations made during real worm outbreaks. For port numbers there is a similar effect, but it is weaker and can be reversed or can have similar impact on source and destination port fields. The reason is that, with regard to IP addresses, a few infected hosts allways need to contact many

potential target IP addresses. For ports, however, each side of the connection can either use a fixed or a variable port, depending on the vulnerability used and the preferences of the worm designer.

In conclusion, we have solved the problem of detecting fast Internet worms in high volume networks, using only flow-level data. Our solution does not require knowledge of the specific vulnerabilities(s) used by the worm to infect target systems or its specific traffic characteristics. This makes a fully generic detector possible. As an additional benefit, the solution has low resource needs and very good detection quality with regard to false positives and detection latency.

8.2 Limitations

Every scientific work has its limitations, and this one is no exception. We will discuss the ones we know about below.

- The sensor we developed focusses on one specific approach, namely detection on entropy-abstracted flow-level data. Other sensor types are possible and have been investigated. However, for a scientifically sound analysis of a sensor type, it has to be evaluated on its own, before combining it with other sensors.

  We submit that exploring the possible synergetic effects of combining different, individually understood, detection techniques for fast Internet worms represents a major scientific undertaking in its own right. One specific danger is over-fitting, i.e. customising parametrisation tightly enough to a specific benchmark data set so that the generality of the result becomes very low. Typically, more complex sensors are more prone to over-fitting, since they can match the specific characteristics of a benchmark data set better.

  For this reason, an evaluation of each individual sensor type, in as transparent a fashion as possible, is essential. We believe that we have reached this goal for the specific sensor type we presented in this work.

- Our detection approach represents new and fundamental research. The impact of long-term changes in network traffic characteristics is currently unknown. Periodic re-calibration of the sensors may be needed in order to maintain sensitivity and a low rate of false positives. It

would be beneficial to determine long-term changes in baseline behaviour and their impact on entropy-abstracted traffic parameters.

- Traffic data from small networks is more noisy, which leads to more false positives and limits applicability of our approach. Determining the actual differences between traffic data from small volume networks and large volume networks could serve to adapt our detector design to smaller networks as well.

8.3 Relevance of Our Results

Most fast Internet worms fall into the time period from 2003 to 2005. Activity has decreased significantly since then and no fast Internet worms had their initial outbreak in the years 2006 and 2007 so far. One reason is a shift to application-generated overlay networks. A number of IM (Instant Messaging) worms were observed in 2005 and 2006. A second reason is that fast Internet worms have lost their initial appeal to the people that write malcode simply to see whether it can be done (and possibly to brag, as the arrest of the Sasser worm author in May 2004 [5] shows). The technology of fast Internet worms is now reasonably well understood. A new worm will just be more of the same and not a reason to receive admiration from peers[1].

A second reason is that fast Internet worms are the equivalent of a WMD (Weapon of Mass Destruction). No precision strikes are possible and collateral damage is huge. This makes fast Internet worms unsuitable for criminal purposes, since law enforcement will be highly motivated to track the originators down. Any way to effect criminal gain. i.e. money, offers a very good possibility of getting caught, since tracing money flow is a tried and true criminal investigation strategy. Consequentially, criminals using compromised Internet hosts have shifted to bot-networks in the last few years, that are built up slowly and ideally in such a fashion that the owners of the compromised hosts do not notice the compromise at all.

A third reason may be that easily exploited vulnerabilities in operating systems have decreased, possibly as a result of the worm epidemics seen over the last few years. This does not mean that operating systems are no longer vulnerable. It just means that the competence level needed for finding a vulnerability and exploiting it successfully is higher now. Analyses

[1] We cannot imagine that writing worms or other malware works well as a strategy "to impress girls". Nonetheless, even more bizarre approaches to that question have been tried in the past.

of past worms show that worm authors are typically not very proficient programmers, with the notable exception of the Witty worm author. There are also still enough vulnerabilities left to make fast Internet worms an ongoing possibility, for example see [4, 43]. As the Witty worm demonstrated, even a small vulnerable population (about 15'000 hosts for Witty) can be enough for successful deployment of a fast Internet worm.

We believe that these developments have made the deployment of "proof of concept" worms unattractive. Not only is a greater effort needed, the recognition gained is small. However, individuals and groups craving global attention may in the future use fast Internet worms in an attempt to do global damage. Currently these groups do not seem to have the technological knowledge to create fast Internet worms. It is also possible that the Internet is not yet widely enough perceived as critical infrastructure to make it a worthwhile target of terrorist-like activity. Nevertheless, the ground work has been laid and the knowledge on how to create fast Internet worms is out there. At the same time, defences are basically non-existent. We submit that detection technology is a crucial part of any defensive strategy. Even if effective defences prove impossible, successful and early detection of the attack increases the effectiveness of any type of damage-limitation since it allows fast analysis of what actually happened. The sensor presented in this work can play a significant role in early detection of attacks based on fast Internet worms. It has the potential to be widely deployed, since it is lightweight and does not need costly infrastructure.

8.4 Directions for Future Work

This work can be extended in several directions. We propose that future efforts are spent on the following tasks:

- Extend the detector to detection of other large-scale anomalies besides outbreaks of fast worms. Possible anomalies to be examined for detection by entropy changes are massive scanning activity, flooding-type attacks that use a large number of flows, because source IP addresses are spoofed and randomised, and attacks against or scans for specific services.

- Specifically for general anomaly detection, it may be beneficial to combine our detector with other traffic metrics and detection schemes. Syn-

ergetic effects could be significant. As our detector is very resource efficient, adding it to installations of other detectors and monitors should be feasible without hardware and software upgrades in many cases.

- The toolset and libraries created are only suitable for NetFlow version 5. It would be beneficial to extend them to include handling capability for NetFlow version 9 [30] and eventually IPFIX [82].

Appendix A

The DDoSVax NetFlow Toolset

The DDoSVax NetFlow Toolset was designed and implemented as part of this thesis. In order to allow effective work with NetFlow data, a basic toolset and library was needed. Design and implementation started at the beginning of the project. Optimisation and extension went on during the project lifetime. At the end of the work on this thesis, the created general NetFlow tools were stable, reliable and efficient and had been used not only for this thesis, but also as a basis of some of the work in the Ph.D. thesis of Thomas Dübendorfer [36] and numerous student theses in the context of the DDoSVax project.

The toolset operates on NetFlow Version 5, the type of NetFlow records collected in the DDoSVax project. All libraries and tools are written in ANSI-C with some GNU extensions. Currently the DDoSVax toolset is only available upon request in a "research version". An open source release under GPL/BSD dual licensing is in preparation.

A.1 Design Approach

The DDoSVax project aims both at near real-time processing of NetFlow data and at offline processing of large data-sets for scientific analysis. The primary data processing mode is *stream processing*, where a stream of NetFlow records is processed record-by-record, usually read from a file or a set of files.

Tools may also read a sequence of NetFlow export packets from the `STDIN` stream, from a named pipe or from a TCP or UDP socket. The toolset is split into a library part, that encapsulates the NetFlow v5 specific operations, and a set of command line tools. The included command line tools allow basic forms of data analysis and transformation and serve as example code that demonstrates the use of the library. For new algorithms, the user is expected to create new code, building upon the library.

A.2 Architecture

A.2.1 Tool I/O and Interconnect

Most tools in our toolset read data from file, usually containing NetFlow export packets in the original export sequence. These files can be raw or compressed. In the second case, the tool detects the compressor (by file extension) and performs transparent decompression.

The primary means of stringing tools together into a longer processing chains are the data stream interfaces offered by Unix-like operating systems, i.e. network sockets (TCP, UDP), FIFOs (named pipes, a.k.a. Unix domain sockets) and the `STDIN` and `STDOUT` streams. These interfaces can be used for input and/or output, depending on the individual tool's functionality. Final results can also be written to file. The toolset contains a tool (`netflow_replay`) that reads NetFlow packets from file and sends them via the stream interfaces mentioned above. This tool can be used to simulate real-time online processing with stored data.

A.2.2 NetFlow Version 5 Export

We now give a brief discussion of the NetFlow v5 binary format, defined by Cisco [28, 29]. Note that Cisco keeps moving the information on its website and that sometimes in the past it has not been available at all. For this reason, we replicate the relevant information here.

Packet Format

NetFlow v5 data is exported as a stream of UDP packets. The packets contain a header, shown in Table A.1, and a number of flow records directly following the header. The format of the flow records is shown in Table A.2. All fields

are exported in network byte order. Positions are specified as byte-offset from the beginning.

The field names are taken from the information available on the Cisco website in 2003. Note that the names follow different conventions, for example capitalised (`SysUptime`), with underscores (`unix_secs`), direct word combination (`dstaddr`) and possibly other, not too clearly identifiable conventions. This mixture gives a strong hint to a not very well controlled historical growth of this export format. In addition to the field definitions by Cisco, we will give our concrete experiences with each data field as manifested in the SWITCH network data.

Position	Name	Description
0-1	version	Fixed to 0x00 0x05 (i.e. Version 5)
2-3	count	Number of flow records in this packet
4-7	SysUptime	Current milliseconds since boot
8-11	unix_secs	Current time (full seconds)
12-15	unix_nsecs	Current time (residual nanoseconds)
16-19	flow_sequence	Total flows seen
20	engine_type	Type of flow-switching engine
21	engine_id	Slot number of the flow-switching engine
22-23	reserved	Unused bytes, should be zero

Table A.1: *NetFlow Version 5 Header Format*

`count:` The number of flows in one flow packet can take any value between 1 and 30. The hardware engines in the routers typically use a fixed number of records per export packet, for example 27 or 29. The software engines are more variable (software engines handle the special routing cases) and can use any legal number of records in an export packet.

`SysUptime:` In theory, this is the export device system uptime in milliseconds when the flow packet was exported. In practice, this value seems to be the time when the header was created in router memory. We have observed packets where this value was set to a time shortly before some of the flow-end timestamps in the flow records of the same packet. This value is an unsigned 32 bit integer and rolls over after roughly 50 days.

`unix_secs, unix_nsecs:` This is the time-since-the-epoch timestamp that corresponds to the `SysUptime` field. While it has space for nanosecond

Position	Name	Description
0-3	addr	Source IP address
4-7	dstaddr	Destination IP address
8-11	nexthop	IP address of the next hop router
12-13	input	SNMP index of input interface
14-15	output	SNMP index of output interface
16-19	dPkts	Packets in this flow
20-23	dOctets	Number of Layer 3 bytes in this flow
24-27	First	SysUptime at start of flow (milliseconds)
28-31	Last	SysUptime at end of flow (milliseconds)
32-33	port	TCP/UDP source port number or equivalent
34-35	dstport	TCP/UDP destination port number or equivalent
36	pad1	Unused
37	tcp_flags	Cumulative OR of TCP flags
38	prot	IP protocol number
39	tos	IP ToS
40-41	_as	Origin AS, source or peer
42-43	dst_as	Destination AS, source or peer
44	_mask	Source address prefix mask bits
45	dst_mask	Destination address prefix mask bits
46-47	pad2	Unused

Table A.2: *NetFlow Version 5 Record Format*

precision, it seems to have millisecond resolution or close to it. This value can be used together with `SysUptime` to calculate the actual start and end time of exported flows. The stated time may be slightly before the packet was exported from the router, just as the value of the `SysUptime` field.

`flow_sequence:` A per-engine counter of the number of flow records exported so far. Note that there is typically more than one flow record in an export packet. This field can be used to determine whether flows were lost in transfer to the collector. In the SWITCH network, sometimes export packets do not arrive in the correct order at the collector.

`engine_type, engine_id:` The `engine_id` identifies the flow engine a packet was exported from. The `engine_type` is supposed to identify the type (software, hardware) of flow engine as well, but we were unable to find a concise definition of its meaning.

`sPkts, sOctets:` Take care that these numbers may not describe the complete flow length. A flow may be cut because an export condition such as idle-timeout, maximum-duration or low router memory may have been reached. The rest of the flow will then be exported later. Cut flows are not specially marked and heuristics have to be used to identify them. UDP and ICMP packets are sometimes aggregated into multiple packet flows, but not allways.

`First, Last:` These values are the flow start and end timestamps, given in the same fashion as the `SysUptime` header field. As these values are 32 bit unsigned, the `Last` value may have rolled over while the `First` value has not, hence giving the appearance of flow-end before flow-start. Both values may also have the same value, typically for single packet flows. In the SWITCH network, the accuracy of the start timestamps is typically high enough to identify which side of a bidirectional connection was the connection initiator.

`port, dstport:` These values give connection source and destination port for TCP and UDP connections. For ICMP the `dstport` and/or the `port` field are supposed to contain the ICMP type and code. We have observed this in some instances, but almost all ICMP flows in the SWITCH data have a of type 0 code 0 (Echo Reply), without corresponding type 8 code 0 (Echo Request) ICMP flows. We believe that only the software engines set these values correctly for ICMP.

`tcp_flags:` This fields is zero in the SWITCH data. It is a big point of dissatisfaction with security researchers and practitioners that many Cisco routers do not export TCP flags. Aparently, this behaviour started in 2002 and is due to performance considerations. The flags would be useful to identify connection initiators and other things, for example flows containing only an RST packet in one direction.

`tos:` As far as we know, the `tos` field is always zero in the data exported from the routers used by SWITCH.

`_mask, dst_mask:` These are the lengths of the respective masks in bits.

Time Handling

NetFlow v5 time reporting has some intricacies. The export time of a flow packet is given as unsigned 32 bit value in milliseconds from the time the

exporting device was started. Flow start and end timestamps are given in the same way. The export time is also given as time-since-the-epoch with nanosecond precision and millisecond resolution. The millisecond fields roll over after roughly 50 days system uptime. Export timestamps may actually be *after* or *before* the individual flow record start and end timestamps. These factors make time calculation using the 32 bit milisecond format error prone. The NetFlow library uses 64 bit signed millisecond values to represent all timestamps internally and provides reliable routines to convert the NetFlow v5 time representation to this format.

The option to reduce timestamp resolution in our NetFlow tools, as, for example, done in earlier versions of the SILK tools [45, 94], was rejected, because millisecond resolution is just about precise enough to determine which side initiated a bidirectional network connection.

Export and Aggregation Modalities

From our observations, the SWITCH network routers export a flow record when one of several conditions is satisfied. If a flow record has been exported and more network packets belonging to the same connection are seen later, they are exported in a new flow record. Note that NetFlow packet export speed is not uniform over time. The routers seem to segment their memory and do flow export individually for the segments in a round-robin fashion. We know of the following export conditions:

- A TCP connection is idle. Around 30 seconds idle time is required to trigger flow export.
- A TCP connection was closed. This condition uses only one direction of a connection and is satisfied if a FIN or RST flag is seen.
- The connection duration exceeded a threshold. At the beginning of the DDoSVax project, this threshold was 15 minutes for the hardware engines and 30 minutes for the software engines. Late in 2003, it was changed to a consistent 15 minutes on all engines. Note that this value is variable in practice. Long flow fragment lengths between 14 minutes and 16 minutes need to be expected.
- A UDP or ICMP packet is typically exported as a single flow record. Aggregation can happen for fast packet sequences.

- Low memory in the router can also be a factor and can cause early export of flows that have not yet met any of the other export conditions.

A.3 Library Components and Tools

Derived from the stream processing model, the basic mode of operation is to process one NetFlow record at a time. The NetFlow library does not deal with sets of records, although there are containers that can be used to store and manipulate such sets. The most important library components are documented below. When writing the code, we took care to use a clean structure and comprehensive in-code documentation in order to facilitate usage by others.

The DDoSVax NetFlow Toolset is under dual-licensing, it can be used both under the GPL Version 2 [48] and the modified BSD license [72]. The original copyright holder is the author of this thesis.

Library Components

Each library is represented by a set of `.c` and `.h` files. The former contain the actual code, the latter the definitions and documentation. The names given below are the prefixes of the filenames of a given library component.

`netflow_v5`

This library component contains the basic flow handling functions and the data structures for flow representation. NetFlow export packets can be read and stored in `struct netflow_v5_header` and `struct netflow_v5_record` data structures. It is also possible to re-synthesise binary export packets from NetFlow headers and records. An important function is `netflow_v5_timing` that, given a flow header and record, calculates start-, end- and export-time in time-since-the-epoch format with millisecond precision. In addition, this library provides functions to convert flow data to different printable representations.

`ip_match`

This library component provides a data structure that allows the definition of sets of IP address ranges and testing IP addresses for membership. Address sets can be read from a textual representation. The SWITCH IP ranges

(AS559) are predefined. The provided functions are intended only for address sets that consist of a relatively small number of linear address ranges, such as a set of subnet addresses. Address set manipulation after creation is not supported.

`ip_table`

In contrast to `ip_match`, `ip_table` is a data structure optimised to represent large, unstructured sets of IP addresses. It is has the functionality of a bit-vector with 2^{32} positions and is implemented as a two-level tree with a worst case memory need of 576MB plus memory management overhead (an additional 160MB with libc6 2.3.2.ds1-2). The table supports insertion, deletion and element count efficiently. Following ideas from object oriented software construction, the table is multi-instance capable.

`hashed_table`

This is a general purpose hash table. It is multi-instance capable and supports true deletion, i.e. the memory allocated by the table shrinks when enough elements are deleted. Keys can be any binary object and are copied on insertion. Care must be taken that some compilers pad structures with unpredictable data, making them unusable as keys. Structures intended to be used as keys should be converted to binary strings first. Elements can be stored directly if they fit into the space of a `void *` or can be referenced otherwise. The table does automatic re-hashing when the number of elements grows or shrinks and allocates or frees memory accordingly. The default hash-function is from the SGI C++ STL Library [100] and efficient for general data [100]. The user can also supply a different hash function.

`collection`

This library component supplies two containers for linearly numbered sequences with array-like access but dynamic size. Elements are `void *` as in the `hashed_table`. One of the implementation, `arrayed_collection` is based on native arrays. The other, `linked_collection` uses linked buckets. Both containers are multi-instance capable and have exactly the same interface and functionality, except for creation. The individual operations can have different time complexities for both implementations. Sorting with a user-supplied comparison function is supported in $O(n \log n)$ time.

`prio_queue`

This implements a general-purpose priority queue based on a heap representation. Priorities must not be changed for elements of the queue. Objects are stored as `void *`. Priority is implemented by a comparison function the user has to supply upon data structure creation. The heap is stored in an array that is dynamically resized when needed. The `prio_queue` is multi-instance capable.

`mt19937`

The Mersenne Twister MT19937 [68, 69] is a high quality Pseudo Random Number Generator (PRNG). It is included in the toolset, since PRNGs in C libraries are frequently of low quality. Using this portable generator avoids problems and removes the need to evaluate PRNGs provided on a platform. This implementation provides pseudo-random values in the form of 32 bit integers, 31 bit positive integers, reals (IEEE 754 "Single Precision") in the range $[0, 1)$ with full 23 bit resolution and doubles (IEEE 754 "Double Precision") in the range $[0, 1)$ with full 52 bit resolution.

`cfile_tools`

The "Compressed File Tools" offer functions for transparently reading compressed and uncompressed files. File access is analog to the binary stream input function `fread` from the ANSI C standard. Reading lines is supported as well. The file compression type is deduced from the filename given. At the moment, uncompressed, gzip and bzip2 formats are supported.

Errors from the compressors and errors from the underlying filesystem access are unified into one mechanism and accessible via the functions `cfr_error` to query stream error status and `cfr_strerror`, to obtain a printable description of the error status.

Command Line Tools

Several command line tools are contained in the DDoSVax NetFlow toolset. They serve as a basis for simpler analysis work and as example code for the library. To this end, the code is carefully documented, even giving explanations about non-obvious NetFlow data properties. The `netflow_to_text` tool, in particular, turned out to be very useful to facilitate explaining the NetFlow

data structures to students. There are also several “iterator templates”, i.e. code templates that read NetFlow data and present each record at a specific place in the code in a set of variables. These serve to greatly speed up the tool creation process. All tools can be called with command line option “`-h`” to display a summary of the available options. In addition, several more specific tools were developed for research applications. As these are only “research quality”, they do not qualify for inclusion in the general toolset.

`netflow_to_text`

This tool reads one input file consisting of NetFlow v5 packets and displays them in various ways. It can also read from `STDIN`. One of the display formats is the complete header and record information on multiple lines. Several other formats print each record on a single line. These formats are especially useful for processing and filtering via scripts written in Perl or Python, and with shell commands like `grep`, `sort` and `wc`.

`netflow_iterator_template2`

This template is basically `netflow_to_text` with the output section removed. All NetFlow headers and records, together with timestamps in directly usable form, available at one place in a loop. The loop is carefully documented and includes sample output statements, demonstrating the data properties.

Note: Originally there was a `netflow_iterator_template`, but it was not very structured and eventually removed and replaced by this template.

`netflow_iterator_template3`

This template is similar to `netflow_iterator_template2`, but can read and process a series of input files.

`netflow_replay`

This tool can be used to replay a NetFlow data file, for example to simulate real-time processing or to split data processing over several hosts. It reads data from `STDIN` or file and sends a stream of NetFlow packets alternatively to a UDP socket, a TCP socket or a FIFO. Replay speed can be unconstrained, with a specific inter-packet delay or a delay that is derived from the original

export timestamps in the netflow packets multiplied by a factor. Figure A.1 gives an example usage scenario.

`netflow_replay_mult`

This is a wrapper-script that uses `netflow_replay` to replay a set of files in lexicographic order of the filenames. All options besides the filenames are passed to `netflow_replay`.

`netflow_mix`

The `netflow_mix` tool can be used to mix two packet streams according to the NetFlow export timestamps. The data is read from two FIFOs given on the command line and written to `STDOUT`. The output can be sent onwards by using `netflow_replay`, redirected to file or processed by a tool reading from `STDIN`. Figure A.1 shows an example scenario.

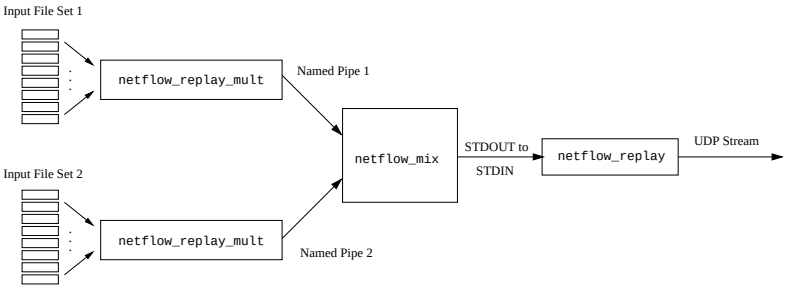

Figure A.1: *Example usage of* `netflow_mix`

`netflow_sample`

In order to allow exploration of the impact of sampling, this tool reads NetFlow data from file and writes a sample to a target file with a ratio given by the user. The target size will not be exactly as expected, since each individual record will have the same probability of being dropped. In this way artefacts in the input data, like nonuniform data export by the routers, do not influence the sampling scheme. The usable sampling rate is in the range of one in $1.0 \ldots 1'000'000.0$ records. The mt19937 library is used as a source of randomness, initialised from `\dev\urandom`.

`netflow_port_stat`

This is an example application that generates traffic statistics for a specific TCP or UDP port.

`netflow_prot_stat`

This is an example application that counts flows, packets and bytes for TCP, UDP, ICMP and other traffic.

`netflow_hosts_stat`

This is an example application that generates traffic statistics for each host seen in the set of input NetFlow data files.

`netflow_ip_count`

This is an example application that counts the number of different IP addresses seen in the set of input files.

`netflow_head`

This application cuts down a NetFlow data file to a set of export found at its beginning.

`netflow_split`

This tool allows splitting NetFlow files that consist of the exported data from several routing engines into separate files. It is customised to the SWITCH data. There are two instances: `netflow_split_19991` for data sent to port 19991 on the capturing system and `netflow_split_19993` for data sent to port 19993. Since the NetFlow export packets do not contain originator information, the sender IP addresses of the collected packets, stored by the collector in files named the same as the packet files but with extension `.stat`, are used.

A.4 Notes on Performance

Most of the NetFlow data files used in the DDoSVax project are compressed with bzip2 [24] (see Appendix C for a discussion of compressor alternatives). As a consequence, decompression forms a significant portion of the overall processing effort. On average, the DDoSVax project captures about 650MB of bzip2 compressed data per hour (as of December 2005, roughly equivalent to 2GB of uncompressed data). On a SCYLLA cluster node (see Appendix B), with the data presen on the local disk, this results in the processing time examples given in Table A.4. Data properties are not constant and disk performance is dependent on file placement. Therefore, these figures should be regarded only as rough approximations.

Activity	CPU time	elapsed time
read compressed data from disk	-	15 s
read raw data from disk	-	44 s
decompression	162 s	-
data parsing	36 s	-
`netflow_sample`, 1 in 1'000'000 compressed input, output to file	174 s	174 s
`netflow_sample`, 1 in 1'000'000 raw input, output to file	8.6 s	44 s
`netflow_to_text` full output compressed input, output to /dev/null	313 s	313 s
`netflow_to_text` full output raw input, output to /dev/null	242 s	243 s

Table A.3: *Processing 650MB bzip2 compressed data on SCYLLA node*

A.5 Lessons Learned

- **Time handling** for NetFlow v5 data is non-trivial and has hidden pitfalls. Encapsulation into a library that calculates a single, clear time value for every time field in the flow header and record is important to facilitate time handling, especially for novice (student) NetFlow users. The internal time format in the DDoSVax NetFlow toolset, namely time-since-the-epoch format with millisecond resolution stored in 64

bit signed (long long) values, has proven to be adequate. As timestamps in the raw data have only millisecond resolution, a higher resolution for the internal format would not have been an advantage. A lower resolution would have been problematic, because millisecond resolution is just about enough to determine originator and responder of a two-sided connection.

- The **examples** and the **code documentation** turned out to be well suited to allow students using the DDoSVax NetFlow data to quickly understand the nature of the data and the usage of the toolset. Especially the `netflow_to_text` tool was very valuable, as it allows low-effort access to the raw data in a human-readable format.
- **Processing speed** of the toolset is adequate, given the availability of a computer cluster. With more limited resources (e.g. a single workstation) data-reduction as a preprocessing step would have been unavoidable and many of the results of the DDoSVax project could likely not have been obtained.
- **Library versatility** is adequate for research. The tools created are not intended to for a complete set, but are basic tools and examples for library usage. This approach worked well in several student theses and can be regarded as a success.
- The primary performance **bottleneck** is CPU and not I/O, contrary to the experience of other projects using (uncompressed) NetFlow data. Overall this turned out to be beneficial, since it allows distribution of computations on several computers when the data comes from a single file server. The relatively high base CPU load from decompression also had benefits, as it allows for a stronger focus on algorithm functionality and limits the temptation to directly write highly optimised code. The problems of too early optimisation, such as far longer implementation time, less code flexibility and a higher rate of code errors, could be avoided in most cases.

A.6 Comparison to Other Toolsets

The SILK tools [45, 94] are another toolset for flow data analysis. They are available under an open source license and are being developed and maintained by Michael Collins and others. While the focus of the DDoSVax

toolset is on non-interactive analysis and analysis of large amounts of data for research and monitoring purposes, the SILK tools focus on interactive work by a security analyst.

The DDoSVax tools expect the user to write code to use the libraries directly in order to do more complex analysis steps. This allows scripted, large-scale analyses. With the SILK tools, the user starts by selecting a time frame of records to work on. The set of selected records is then narrowed down by filtering with command line tools that implement individual processing steps. At the end, the analyst ideally has found a set of flow records that correspond to a single incident.

While the DDoSVax tools have the philosophy of doing as much as possible in memory, the SILK tools use the disk as intermediate storage. For example, determining the IP addresses in a specific flow-set, the end result with the SILK tools will be a file listing all these addresses. With the DDoSVax tools, it will be a specialised hash-table in memory that lists these IP addresses and can be directly used for further steps. Both approaches have advantages and disadvantages.

While the DDoSVax toolset and the SILK tools are designed to work on similar data, they are complimentary in their aims. The SILK tools are aimed at the security analyst that interactively determines the nature and extent of a particular intrusion. For that reason, the SILK tools aim to provide a complete set of command line tools optimised for this usage. The DDoSVax toolset is aimed at batch processing of large numbers of flows for research purposes. It provides libraries that facilitate creation of research tools and includes basic analysis tools and example applications. It does not try to anticipate what the user wants to do in order not to limit the directions research can go in.

Appendix B

Data Processing Infrastructure

B.1 Motivation

The two primary reasons for building a dedicated NetFlow data processing infrastructure were performance issues and security concerns. NetFlow data processing creates significant disk I/O and often the disks represent the main limiting factor for the overall processing speed. This was at odds with the existing computing infrastructure that primarily relied on centralised storage on file servers and was designed with computationally-intensive tasks in mind.

The second concern arose because the data collected is protected by Swiss privacy laws and special care in handling it needs to be observed. The easiest way to implement a good level of protection was by creation of a dedicated system that is accessible only to users that are authorised to work on the data in question.

The cluster system described here was in operation from late 2003 to mid 2007. It was named the “Scylla Cluster”, after the Scylla creature from Greek mythology. Scylla is a sea monster, with six long necks equipped with grisly heads.

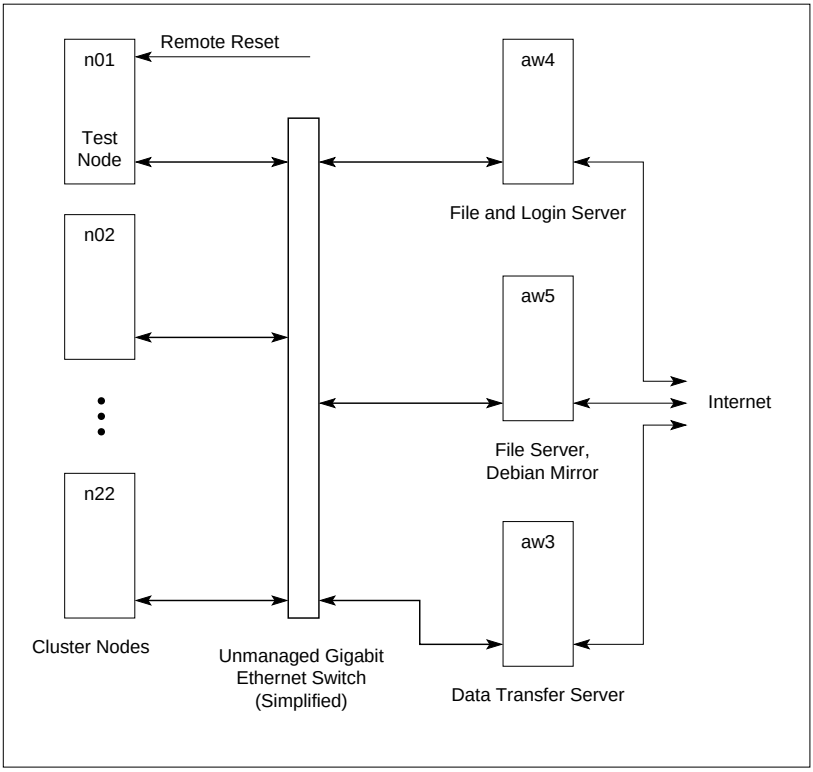

Figure B.1: *"Scylla" cluster structure*

B.2 Structure

The primary cluster structure is given in Figure B.1. There are two servers, `aw4` and `aw5`, that take the role of file-servers and have Internet connectivity. The first one, `aw4`, also doubles as a log-in server. In the standard configuration, users can log in to the cluster nodes from `aw4` without giving a password. Both connect to a private Gigabit Ethernet network, that forms the only network connection of the 22 cluster node PCs `n01`-`n22`. The nodes all use identical set-ups and similar hardware. The first one, `n01`, has a remote reset capability and serves as a test-node. The actual internal network is implemented using one 24 port and one 5 port commodity Gigabit Ethernet switch.

The data transfer system, aw3 (see Appendix C for its function), is connected to the internal network of the cluster as well. This allows fast access to the last 4-6 weeks of captured data, available on aw3.

B.3 Software and Configuration

All systems in the cluster run Debian Linux. The nodes can be installed automatically using FAI (Fully Automatic Installation, [41]). Installation of a new system on all 22 nodes is possible in less than 10 minutes.

Originally the cluster was also running OpenMosix [77], which features automatic load-levelling and process migration between systems. However, due to problems with hardware support and lack of development, OpenMosix was removed later.

All data partitions in nodes and servers are exported on the internal network via NFS and mounted on-demand by the Linux kernel level automounter. The advantage is that if a system becomes unavailable and then available again, there is no need to manually mount exports from other systems and non-available exports will not cause the system to hang.

B.4 Hardware

Year	Servers (total)	Each Node
2003	800GB	120GB
2007	4.7TB	200GB - 250GB

Table B.1: *Initial and final available cluster disk space*

The cluster nodes have one Athlon XP 2800+ CPU, 1 GB RAM and, initially, one 120GB IDE disk. The file server aw4 has two Athlon MP 2800+ CPUs, 2 GB RAM, a PCI-X bus, and, initially 600GB of available RAID5 space. The other server, aw5, has the same hardware as the cluster nodes, but provides an initial available disk space of 200GB in a RAID1 array. All RAID is Linux software RAID.

Over time, the storage space on the nodes and the file servers evolved. The file servers needed more space to store interesting data intervals and analysis

results. For example, the data captured around a worm outbreak event would be made available on a file server, in order to avoid the time-consuming transfer from the tape library. The nodes were extended by additional disk-space as well, as projects started to need more space for the actual data under analysis. Table B.1 shows the initial and final disk space available on the file servers and cluster nodes.

B.5 Security Concept

The primary security measure is to restrict open ports on the servers to the SSH port (Secure Shell, port 22/tcp) and block all others using the Linux kernel firewall. Log-in is done only using SSH certificates to prevent password guessing attacks. Within the cluster, only ordinary security measures, such as using a reasonably current system installation, are used. We expect that, within the cluster, an enterprising user can elevate his or her privileges with moderate effort. Since all users on the cluster are working with sensitive data anyways, we do not consider this to be a significant threat. On the user side, especially student users doing a thesis, are warned that the data they are working on is subject to Swiss privacy laws and that any misuse can lead to personal legal consequences for them. In addition, they are allowed only to log in to the cluster from one machine with a current and maintained Linux installation with firewall settings that block all access from the outside. Students that need more connectivity or want to run an alternative OS are provided with a second computer.

During the operation time of the Scylla cluster, there was no security breach that we are aware of. The only potential incident was an insecure SSH implementation in Debian, which needed to be replaced by an older, non-vulnerable SSH implementation for a limited time. The follow-up system, that replaces the Scylla cluster, uses the same security concept.

B.6 Experiences and Lessons Learned

We encountered several noteworthy operational issues during the lifetime of the computer cluster. These were mainly hardware issues. In addition, some software concerns did arise in connection with the hardware problems.

- The selected network cards were running relatively hot. A measure-

ment with an IR thermometer revealed a surface temperature of around 65C. Estimating the expected lifetime of semiconductors is difficult without exact manufacturer information. One possible generic approach is to use a lifetime of 30 years at 25C and to derate by a factor of 2 for every 10C more. This would give 2-3 years expected lifetime for the cards. Since the cards were the fastest reasonably priced cards available at the design time of the cluster and since the manufacturer (Netgear) gave 5 years warranty, we decided to use the cards anyway.

Not unexpectedly, we lost the first card after about 2 years and it was promptly replaced under warranty. After 3 years we had lost a lot more cards and Netgear never replaced them. To resolve the issue, we had to replace all networking cards with a different brand.

A similar issue arose with the 24-port Gigabit Ethernet switch, also manufactured by Netgear. It failed after 2 years with defective ports. The replacement failed again 1.5 years later, and it turned out to be nearly impossible to get a warranty replacement. When it arrived after several months, it was defective. To resolve the, issue we had to replace the switch with a different brand.

These problems lead to packet loss and link loss issues in the computer cluster. One main effect was that remote volumes mounted via NFS would sometimes become unresponsive and hosts needed to be manually rebooted. This issue was resolved by using NFS over TCP, instead of the default UDP. In addition, we replaced static NFS mounts with automounted ones.

- We encountered failure of two PSUs during the cluster's lifetime. Considering that 24 machines were running 24/7 for about 3.5 years, this is an acceptable rate.

- We were hit by the bad capacitor problem [3, 14], now sometimes known as the "Capacitor plague" in one server mainboard. The manufacturer sent us a mainboard with the same bad capacitors as replacement and we had to fix the board ourselves with quality components. It performed fine afterwards for several more years.

- We did not experience any disk-crashes, besides a small number in one shipment where the packaging had been destroyed by the Swiss postal service. All problems were noted before data loss occurred due to regular S.M.A.R.T. [99] monitoring.

- We had one RAM module (out of 44) with a “weak bit”, i.e. a bit that occasionally lost its state. It took significant effort to find the affected module, also because of the automatic process migration of Open Mosix. Applications would typically not crash, but report wrong results with the frequency of one error per several days of computing time. This problem contributed to the decision to abandon OpenMosix and to operate the cluster without process migration between its nodes.

The main lesson learned is that handling large volumes of data requires extra care. Data corruption can be introduced in RAM, in busses, in unreliable CPUs and in other places. It is necessary to expect occasional corruption and to be able to deal with it. Since our data on tape is stored compressed and the compressor has its own checksum mechanism, recognising data corruption introduced in long-term storage is relatively easy but computationally intensive. When moving data, it is highly advisable to use additional checksums, for example created via the `md5sum` tool, to compare the data at the source and the destination, before deleting the data at the source. The checksums should then be kept at the destination to allow easy checking of the files, in case errors turn up later. This measure not only improves data integrity, it also helpts to isolate the source of any corruption and to facilitate diagnosis and repair, if the corruption event is repeatable.

Overall the Scylla cluster performed well and proved adequate to its task. On the performance side, reading compressed NetFlow data from a file server worked well for up to 6-8 nodes reading in parallel, depending on the actual processing done. When reading uncompressed data, one node could saturate a file server, which is not unexpected. Distributing larger data sets to the local disks in the nodes was possible with low effort, using data-distribution scripts that can run overnight. Reading data to be analysed from the local disks provided satisfactory I/O bandwidth even for large-scale investigations.

Appendix C

Data Capturing System

As part of this thesis, we built a data-capturing system, that collects and stores flow-level traffic data from the SWITCH [6] network. The data format used is Cisco NetFlow v5 [9, 28, 29].

C.1 Objectives

The primary objective of the NetFlow data capturing system is to record the NetFlow packet streams exported by the SWITCH border routers (see below) and to transfer the captured data to long-term storage. As a secondary objective, the data capturing system of the DDoSVax project stores the last four to six weeks of captured data on disk for easy access. Since we only have limited human resources for the operation of the capturing system, primary design goals were reliability and fault-tolerance. As a consequence, the main paradigms used are simplicity and automated fault detection and recovery. An additional complication is that we could not place a host dedicated to data capturing in the SWITCH network and the packet capturing processes have to run with user privileges on a Linux host operated by SWITCH.

C.1.1 Data Flow

The basic data-flow through the capturing system is shown in Figure C.1. The four SWITCH border routers export their data in two streams of NetFlow version 5 [9, 28, 29] UDP packets. One stream is the combined data export

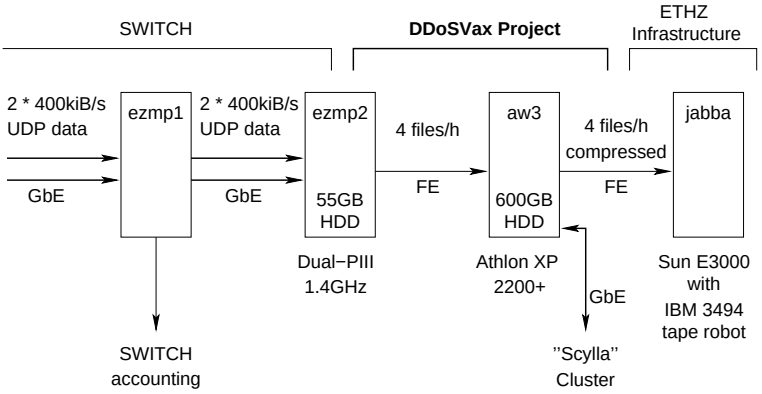

Figure C.1: *Capturing system data flow*

of three routers (`swiCE1`, `swiCE2` and `swiBA2`) mixed together, the other is the data export from `swiIX1`. In the SWITCH topology map (Figure C.2) `swiCE1` and `swiCE2` are on the left (marked together as `CE`). These Routers provide connectivity to CERN, the CERN Internet eXchange Point, Global Crossing, Level 3 and the GEANT2 research network. On top in Figure C.2 is router `swiBA2` (marked `BA`), which connects the University of Basel, the BelWue and the Swiss Internet eXchange point. On the top, right side, there is router `swiIX1` (marked `IX`, which provides connectivity to Telia, the Swiss Internet eXchange point, and the Telehouse Internet eXchange point.

Two data streams arrive via Gigabit Ethernet at the host `ezmp1` (running Linux). `ezmp1` replicates the streams on a packet level. One copy is sent to the host `ezmp2`. The DDoSVax NetFlow packet capturers runs on `ezmp2` as a set of user space processes. They collect the incoming UDP packets into one packet data file and one metadata file per hour and data stream. The metadata contains the sender IP address and a timestamp for each captured packet. It is needed to determine which router sent a specific NetFlow packet, since NetFlow v5 does not identify the exporting router.

The data files are then transferred to the Linux host `aw3`, which resides in the ETH network and is operated by the DDoSVax project. The transfer itself is initiated by `aw3` using the secure shell `ssh` and the secure copy command `scp` (both from the OpenSSH project [78]). The transferred files are compressed and copied to the tape library `jabba`. `Jabba` runs Solaris and is operated by the Information Technology and Electrical Engineering De-

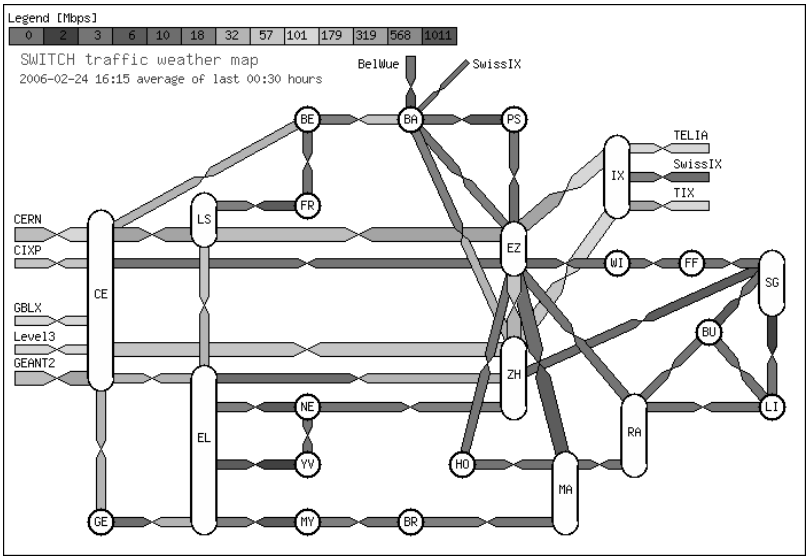

Figure C.2: *SWITCH topology (weather map) from www.switch.ch*

partment (D-ITET) of ETH. In addition, a copy of the last 4 to 6 weeks of compressed data is kept on `aw3` for quick access from the computer cluster "Scylla". "Scylla" forms the secure NetFlow processing infrastructure of the DDoSVax project. See Appendix B for a description of the "Scylla" cluster.

C.1.2 Data Properties

The captured data consists of unprocessed NetFlow v5 packets. The data export from the SWITCH routers is bursty and can reach gigabit speeds. After some modifications to the original SWITCH NetFlow transfer system (see Section C.3), the DDoSVax capturing system experiences only very little data loss. In addition, the SWITCH routers suffer very little flow loss. As a consequence, the captured data forms an accurate and complete description of the SWITCH network border traffic.

In order to understand the data handling effort needed, we measured typical NetFlow data volumes the SWITCH border routers generate. The figures given in Table C.1 are based on the data captured in the second week of January, 2004. The given values represent a typical situation. They re-

	Raw NetFlow Packets	Compressed with `bzip2 -1`
1 hour	2.1 GiB	730 MiB
1 day	50 GiB	17 GiB
1 month	1.5 TiB	510 GiB
1 year	18 TiB	6.1 TiB

Table C.1: *Typical SWITCH data volume (scaled, start of 2004)*

Compressor	Compressed Size	Compression CPU Time	Decompression CPU Time
bzip2	660 MiB	1100 s	410 s
bzip2 -1	730 MiB	730 s	200 s
gzip	810 MiB	280 s	32 s
lzo	1.1 GiB	41 s	13 s
no compression	2.1 GiB	0 s	0 s

Table C.2: *Compressor comparison (1h data, Athlon XP 2800+ CPU)*

mained fairly constant during the observation period (start of 2003 until start of 2006). The observed volumes are scaled to different time intervals to provide an overview of the date volumes to be expected.

C.2 Compression and Long-Term Storage

Since we have space constraints for data storage, all data is compressed before being transferred onto long-term storage. Originally data was compressed using `bzip2` with default parameters. However, during some network events, most notably the Nachi.a worm, with its massive increase in observed flow numbers due to ICMP target probing, the system load due to compression became too high and had to be reduced. Currently, `bzip2` with parameter `-1` is used.

We considered the following compressors (all of which are available under the GNU public license):

- **bzip2** [24], a modern compressor based on the Burrows-Wheeler block sorting algorithm and Huffman coding. It compresses well, but is slow.

- **gzip** [50] the GNU zip compressor. It is based on the Lempel-Ziv method. Its performance and resource requirements are average in all regards.

- **lzo** [66] the Lempel-Ziv-Oberhumer compressor. This compressor family is extremely fast at the cost of compression ratio.

Table C.2 gives the expected compression performance and CPU times to be expected when processing typical data from the SWITCH network. Main memory consumption is low for all compressors (below 10MiB) and not listed. The data sample is the same as in Table C.1. Typically, data analysis will involve decompression but not compression. Compression is done by the capturing system. The CPU load values due to compression are given as reference values for capturing system design.

One impact of using a CPU intensive compressor is that most measurements done on stored data in the DDoSVax project are CPU bound. Without compression, NetFlow data analysis is usually I/O bound, i.e. reading data from disk is slower than processing it. With `bzip2` compression, a single file server can deliver data to 8…10 nodes doing analysis in the "Scylla" computer cluster, before I/O becomes the limiting factor. (See Appendix B for a discussion of the characteristics of the DDoSVax computer cluster "Scylla".) As a consequence, a positive effect on software creation was observed, namely a weaker tendency to optimise analysis algorithms for speed prematurely. Decompression of an hour of data takes around 200 CPU seconds. Optimising analysis algorithms to perform better than this number is subject to fast diminishing returns. This curbs the often observed implementors impulse to write highly optimised code, that hard to modify and maintain, too early in a project. Observed positive effects include cleaner software structure, higher willingness to work with clean interfaces and checked containers and generally a higher willingness to experiment, since implementation work becomes easier and less time-consuming.

If very fast repeated processing of the same data is needed, decompression and storage of the raw data on disks local to the Scylla nodes is possible. Experience has shown that this is rarely done in the DDoSVax project. Using more cluster nodes is less work than writing more optimised code, and seems to be conceptually and administratively easy to do for our data users. It should be taken into account, however, that most DDoSVax analysis work is done by sequentially processing the data in a set of hour files in order to obtain a global view of some data characteristics. For more forensics-oriented work,

i.e. following the activities of a single or a smaller set of hosts in detail, a different approach to data storage would be advisable. It should also be noted that our agreement with SWITCH does not allow us to do any forensics work without explicit permission by SWITCH for each individual case.

Performance is not the only consideration for compressor selection. An additional question is how much data is lost in the case of bit errors introduced after compression. With the processed data volume this becomes a concern. The `bzip2` compressor loses one raw data block (100 kiB ... 900 kiB, depending on the selected compression parameter) per bit-error. May other compressors cannot recover any data following a bit error. In the course of the DDoSVax project, 5 bit-errors were found in the stored data for the year 2004. This indicates that a different compressor choice would also have been reasonable with regard to bit-error behaviour, at least as long as data is stored in relatively small files.

C.3 Scalability, Bottlenecks

C.3.1 Network and Operating system

The primary capturing bottleneck that can cause packet loss is the CPU scheduler on the capturing system. When data is received, the socket-buffers start to fill up. If the scheduler takes too long to assign the CPU to the capturing process, the socket buffer becomes full and the kernel starts to drop packets.

When we started the DDoSVax project, SWITCH experienced some data loss in its NetFlow export streams, in the order of up to several percent. We found that the capturing and replication was done with the default socket buffer size of 64kiB. At the same time, the routers exported data in bursts that could theoretically reach gigabit-speed and could fill up the socket buffers in a bit more than one millisecond. Our measurements (Table C.3) show real data bursts of up to 320kiB in a 10ms window. Due to kernel restrictions on the host capturing the UDP NetFlow data stream, we could not obtain measurements for shorter intervals. It is quite possible that the maximum observed speed for a 10ms interval is not the top export speed, since shortening the measurement interval leads to significant burst speed increase for all measurements given in the table.

We observed idle times of up to 2 seconds between the bursts. Due to the 10ms scheduling interval of the Linux kernel on the host capturing the NetFlow packets, the socket buffers could frequently not be emptied fast enough

and packets were lost. To solve this problem, the socket buffer size on `ezmp1` and `ezmp2` was increased to 2MiB. This is enough to contain several seconds of data and thus solved the loss problem completely.

C.3.2 CPU and Main Memory

The CPU requirements of the capturing system are low. In December 2005, we observed a total CPU load of about 2.4% with an Intel Xeon 3.60GHz CPU. Since the capturing system runs on a host with two of these CPUs, the overall CPU load is about 1.2%. Main memory consumption is low as well. Besides the socket buffers, the main consumer is the buffer cache used for disk writes. Neither are critical in our set-up.

C.3.3 Disk Storage Speed

Since we capture only two data streams on a lightly loaded system, we receive disk write performance comparable to the linear write performance of the filesystem. The capturing system uses a fast SCSI disk. Even slower 7200 rpm ATA disks can write in excess of 10MiB per second in realistic scenarios today. This exceeds the NetFlow data delivery speed significantly and hence disk storage speed is not a significant concern in the DDoSVax capturing set-up.

C.3.4 Scaling Up Observations from SWITCH Data

Data export characteristics are not available to us from fast routers other than those in the SWITCH network. We can only speculate about scalability based on scaled-up figures derived from our measurements. The obtained numbers can serve as a first approximation and give hints on how to evaluate a concrete capturing situation.

We assume that the user-space capturing process can remove data from the socket-buffer much faster than it arrives as soon as the process has the CPU and gets executed. From the observed CPU consumption of the Perl-script used for data capturing, this is a realistic assumption.

Table C.3 gives the maximum export speed for the most bursty NetFlow data stream from the SWITCH routers (`swiIX1`), versus observation window length. Window shift is a uniform 10ms. The measurements were taken over two weeks of December 2005 (5th - 19th). Figure C.3 gives the minimum

buffer sizes required versus the maximum capturing process reaction times (i.e. maximum scheduling delay). The long-term average data rate for the observation interval was 150kiB/s for the observed router.

Window	10 ms	100 ms	1 s	10 s	100 s
Data	320 kiB	1.1 MiB	2.0 MiB	15 MiB	61 MiB
Speed	32 MiB/s	11 MiB/s	2 MiB/s	1.5 MiB/s	0.61 MiB/s

Table C.3: *Maximum export burst of swiIX1 (5.12.2005-18.12.2005)*

Note that these observations cannot directly be used to scale the given figures for another capturing situation, since the maximum burst speed is only weakly dependent on the average data rate, but strongly dependent on the network and the router buffer memory and export algorithm. As a consequence, these figures may change completely if a different router type is used. Our observations can give a general idea about "reasonable" figures, but are no substitute for individual and careful evaluation of each capturing situation.

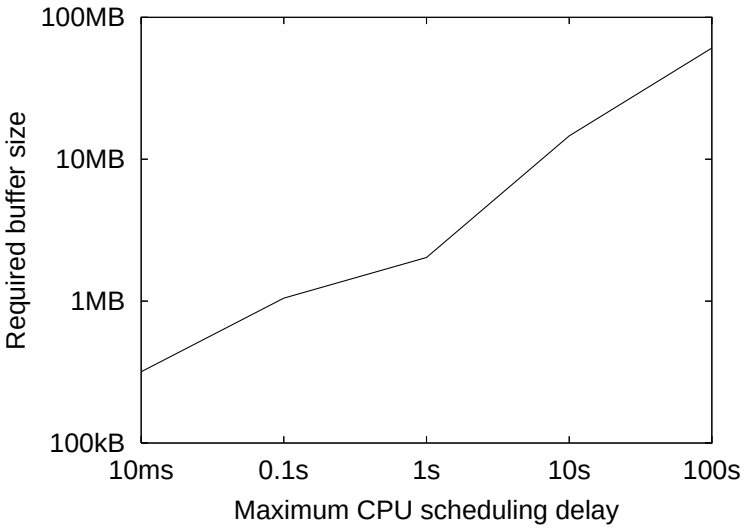

Figure C.3: *Minimal needed socket buffer size vs. maximum CPU scheduling delay (150kiB/s average data volume)*

C.3.5 Performance Improvement

The most important bottleneck is the CPU scheduler latency, together with the socket buffer capacity. It is possible to use a real-time capable scheduler or a real-time kernel extension to reduce the maximum time that the socket buffers must hold data before it is processed. The main socket buffer functionality for de-bursting moves then to a buffer in the capturing software or to the filesystem buffer-cache. The primary advantage is that the buffer can be made larger than the current kernel-limit of 16MiB. The main disadvantage of a real-time extension is that the implementation of the capturing software becomes more complicated and error-prone. We believe that using a lightly loaded system and generously sized socket buffers is the preferable solution, except when the available maximum socket-buffer size is too small to ensure reliable NetFlow packet capturing.

C.4 Fault Tolerance

Because the DDoSVax NetFlow capturing system runs around the clock without human supervision, the system needs to recover from most problems without manual intervention. At the same time, problems should be reported to the system administrator with reasonable delay. All components are kept simple in order to make design and implementation errors less likely.

The error recovery mechanisms on the data-capturing host (`aw3`), uses a 2-layer supervision mechanism. The individual components of the mechanism and their relation are shown in Figure C.4. The capturing software itself takes NetFlow packets and writes them into files. It does nothing else in order to be as simple as possible. The first fault tolerance layer consists of a supervisor process, implemented as a Python script with a single loop that is run once every minute. At the beginning of each loop a timestamp is written to disk. The supervisor checks periodically that the capturer processes are still running and that a minimum of free disk-space is left. If a capturer is not running, it is restarted and an email notification is sent. If free disk space gets low, an email is sent to the administrator, but no mitigation is done, since nothing can be done without manual intervention.

The supervisor is in turn monitored by a `cron`-job, i.e. `cron` provides basic reliable and periodic execution service. The `cron` service was chosen because even if the `cron`-job hangs or crashes, it will be run again at the next preselected time. The `cron`-job checks periodically that the supervisor still

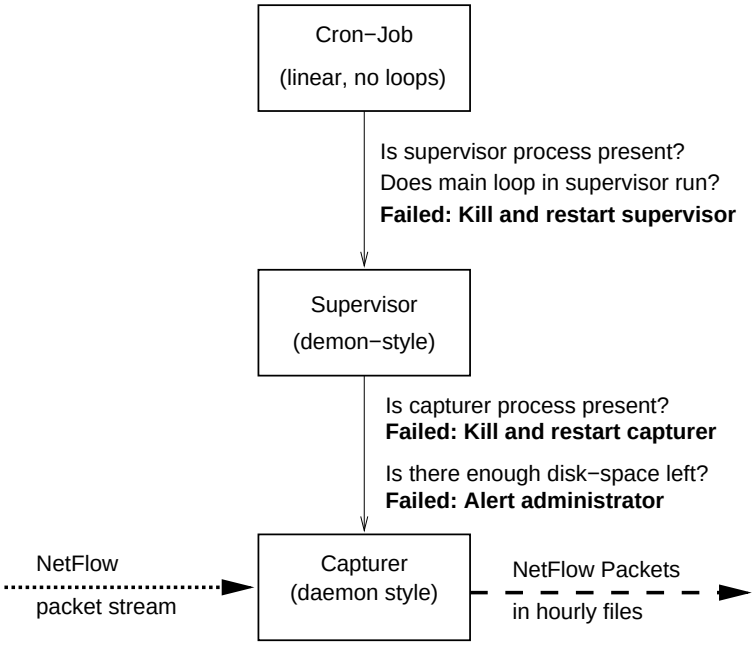

Figure C.4: *Fault tolerance mechanism on flow capturer*

runs in exactly one instance. It also checks whether the main loop of the supervisor is still running by looking at the loop timestamp on disk. If either test fails, the supervisor is restarted and an email notification is sent. The `cron`-job itself is a Phyton script that has a very simple linear structure, i.e. does not contain any loops. It performs very few and only simple operations in order to provide maximum reliability. After the tests are finished, the script exits and is started again by `cron` at the beginning of the next 10 minute interval.

A third monitoring mechanism is implemented on the host that pulls the data into the ETH network (`aw3`). The transfer to ETH, compression and transfer to tape storage is done hourly with a `cron`-executed script. All data files, that were not modified for more than 66 minutes, are transferred to ETH and deleted on `ezmp2`. The script tests whether there are enough hour-files, whether they have reasonable size and whether they actually have a minimum number of flow-records in them. Transferred files are checked for correct

sizes. The script also checks whether the system clock on the capturing host is running correctly, since inaccuracies could lead to transfer and deletion of not yet closed hourly files. In case of deviations from pre-set limits, an email is sent.

Since the whole software installation on aw3 consists of a single script called from a cron-job and because the host requirements are limited to a standard PC with enough storage space, aw3 could be replaced with minimal effort in case of hardware problems. In addition, a redundant installation where a secondary transfer host pulls files from the primary capturing system on ezmp2 if they have been there for a longer time (several hours) is easy to establish. Replicating the software installation on aw3 and changing one parameter in the transfer script is enough. The primary transfer system removes all data files from ezmp2 within less than three hours (taking into account some time for the actual transfer). The redundant transfer system could then be set to pull only files modified at three hours old from ezmp2 and would activate when aw3 fails.

For the data transfer from SWITCH to aw3 and, after compression, onwards to tape storage (on the jabba system), there are two safety mechanisms. Every file is checked whether it has the correct file size after transfer. Files too small can be observed when transfers are interrupted. Files too large can be observed with filesystem problems on the tape library. In this case, the file size is rounded up to the next full block size, but the file is empty. As second test, all files are tested whether they are correctly structured.

There is space for about half a day of data storage on the data capturing host (in case aw3 becomes unreachable or unavailable). In addition aw3 stores the last 4-6 weeks of data locally, so that outages of the jabba tape system will not result in data loss. So far we have observed one unannounced outage of the ETH network that prohibited transfer from the SWITCH network to aw3 and lasted 6 hours. We have experienced unannounced jabba outages of up to a day and announced outages of several days during upgrades of the tape library hardware. Neither of these problems caused any data loss.

C.5 Privacy Concerns and Collaboration Possibilities

Swiss privacy laws do not allow collection and storage of network payload data and exceptions are only made when it is absolutely necessary for net-

work maintenance and operation. Even then the data has to be deleted as soon as it is not needed anymore or after a very short time. All captured data is to be treated as confidential and misuse or publication can be subject to criminal penalties. Identifying individual users and inspection of the payloads of their data requires either a court order or permission of the sender or receiver of the data. To a limited degree this can be done in employment contracts, but employers cannot suspend privacy completely that way. The exact legal situation with regard to data payloads is still in flux. A summary regarding the present situation in the workplace can be found in [103].

Since we are only capturing flow level data and not payloads, the legal restrictions are lighter. The data still needs to be kept confidential, since there might be possibilities to identify individual behaviour. We also have a contractual agreement with SWITCH, that does prevent us from identifying behaviour of individuals and publishing any data that could be sensitive with regard to privacy, such as unanonymised IP addresses. There is one valid possibility to work with data attributed to a user: Individuals can identify and inspect the data they sent or received over the network.

The NetFlow data archive and processing infrastructure is used in the DDoSVax project to support research activities, such as this thesis. Access is restricted to individuals that have a need to access the data. Researchers are already bound to secrecy by their contract. Students doing thesis work for semester- and master-theses have to sign a confidentiality agreement, and are reminded that they could be personally subject to criminal prosecution if they misuse the data. They are also disallowed from keeping any of the data or possibly privacy relevant results in their possession after their thesis work is finished. thesis data is instead archived by the DDoSVax project.

Publications and student theses have to be inspected by the DDoSVax core staff before they can be published. If there is even a remote possibility of a privacy violation, SWITCH has to be consulted as well and has to clear the publication.

In order to allow new research projects on the DDoSVax NetFlow data archive and to create the possibility of scientific collaboration based on the data, the contract between the CSG (Communication Systems Group, the DDoSVax project is a research activity of the CSG) at ETH and SWITCH provides the possibility of submitting research proposals to a review board that is jointly formed by the CSG and SWITCH. This review board has the primary obligation to ensure that privacy laws and confidentiality of the NetFlow data is kept. Research proposals must describe how this is ensured and

the review board may impose additional restrictions or reject project proposals. A specific concern is data access given to researchers not affiliated with ETH or not subject to swiss privacy laws. In case of such collaborations, strong contractual agreements might be required and requested by the review board. An other option would be to limit the access of the external researchers to anonymised data and results. This option would still allow meaningful collaboration where, e.g., algorithms could be developed jointly and then tested and evaluated by DDoSVax or CSG staff members.

C.6 Lessons Learned

We now discuss our experiences made with the capturing system and its design while operating it for several years.

- The increased socket buffer size of 2 MiB was large enough for the SWITCH data and we observed no significant loss of data in transit or capturing after the socket buffers were enlarged.

- Capturing the data into hour-files did not cause any significant problems in processing or storage. However, it turned out that the original file-boundary selection method, that just counted 3600 seconds before starting a new file, had a small but noticeable skew compared to real-time. While not a problem when operating the capturing system for a month or so, the accumulated skew became a concern after longer operation periods, since it made finding specific data more difficult. The original capturing script was therefore successfully upgraded with a time-synchronisation mechanism that can vary the length of a captured file by up to two minutes in order to bring its boundaries closer to the hour boundaries on the system clock. The two-minute limit allows resynchronisation after a restart within 15 hours. while keeping the files still approximately one hour long. The latter is convenient, because we frequently experienced situations where the system-clock on `ezmp2` was off by an arbitrary amount of time after reboot, but was corrected within a few hours. This was due to such factors, as for example an unreachable NTP host and other issues outside of our control. With fast synchronisation, this situation could lead to files that contain between a few seconds and up to nearly two hours of data. These files would trigger the alerting mechanisms that monitor file counts, file sizes and

flow counts, which in turn would cause a need for manual inspection. With the used slow convergence method the characteristics of the data files stay in the normal parameter range, even when resyncing to a new clock setting. Directly using the system clock as the time base was also considered, but rejected because it would again cause problems when the clock on the capturing host was set to a wrong value. The synchronisation mechanism has run accurately and without problems for several years by now.

- Whenever error messages are sent automatically via email, it is important to have a rate-limiting mechanism. Without rate-limiting unintended DoS attacks on the capturing system administrators email account are a real possibility.

- In order to verify that a data file has been transferred from or to a remote system correctly, we found it sufficient to compare the file sizes. Errors we found included files both too small and too large as a result of transfer and file-system problems. A data comparison using checksums would find bit-errors, but would cause significant additional I/O load on the involved hosts. The primary concern is bit-errors in the compressed data, since each leads to a loss of one data-block of about 100kB (raw size). Given the low number of observed bit-errors in compressed data (about 5 in the stored data for 2004), use of checksums for transfer would not have improved stored data quality significantly.

  Bit errors in the raw data can happen in the capturing chain up to the point when the data is compressed. Note that these errors would only impact a single flow record in most cases and at maximum a complete flow packet. We do not know the number of bit errors in the raw data, but can estimate an upper bound for the possible randomly placed errors. Finding non-randomly placed errors introduced by the router would likely need some kind of plausibility test, which is rather infeasible in our set-up. Since the 16 bit NetFlow version field is checked by our NetFlow processing library on each packet read, we know that very few of these fields are corrupted. In the whole 2004 data we saw less than 10 packets with incorrect version fields that were potential bit-errors. Scaling this linearly up (by assuming the error position is random in the packet) gives less than 7700 packets in the whole of 2004 with bit errors introduced in the raw data, as the typical NetFlow packet in the SWITCH network is around 1440 bytes long. We believe

that these numbers do not justify adding checksums to the transfer of the raw data. In addition, such checksums would not help against errors introduced in the routers, during network transfer and in the capturing host before the data is written to disk.

Curriculum Vitae

Arno Wagner, Dipl. Inform.
arno@wagner.name
Born January 7th, 1969
Citizen of Austria

Education

2000 - 2008	ETH Zürich, PhD Studies in Network Security
2003	ISC^2 CISSP certificate
1990 - 1996	University of Karlsruhe, Germany, Diploma (M.S.) in Computer Science
1989	Abitur (German general university entrance qualification), Staudinger-Gesamtschule Freiburg, Germany

Work Experience

2006 - today	Lecturer, ETH Zürich
2000 - 2006	ETH Zürich, research and teaching assistant
1997 - 2000	Researcher at the Institute for Telematics, Trier, Germany
1996 - 1997	University of Karlsruhe, Teaching assistant

Honours

2006	Best paper award at ICISP 2006 for **"Flow-Based Identification of P2P Heavy-Hitters"**
2005	Best paper award at WETICE 2005 for **"Entropy Based Worm and Anomaly Detection in Fast IP Networks"**
2004	Best paper award at WETICE 2004 for **"An Economic Damage Model for Large-Scale Internet Attacks"**
1997	**Award of the sponsor's association of the FZI** for exceptional performance in the Diploma Exam in Computer Science

Document License

Unmodified distribution of this work for free is allowed. For derived work and other uses, please consult the license below or contact the author.
Author address:

Email: arno@wagner.name or arno.wagner@acm.org
WWW: http://www.tansi.org

Creative Commons - Attribution-NonCommercial-ShareAlike 3.0
License URL:
http://creativecommons.org/licenses/by-nc-sa/3.0/legalcode

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. **Definitions**

    a. **"Adaptation"** means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image "synching") will be considered an Adaptation for the purpose of this License.

b. **"Collection"** means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(g) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

c. **"Distribute"** means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.

e. **"Licensor"** means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

f. **"Original Author"** means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

g. **"Work"** means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

h. **"You"** means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

i. **"Publicly Perform"** means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

j. **"Reproduce"** means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. **Fair Dealing Rights.** Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. **License Grant.** Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

   a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;

   b. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";

   c. to Distribute and Publicly Perform the Work including as incorporated in Collections; and,

   d. to Distribute and Publicly Perform Adaptations.

   The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights described in Section 4(e).

4. **Restrictions.** The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

   a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(d), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(d), as requested.

b. You may Distribute or Publicly Perform an Adaptation only under: (i) the terms of this License; (ii) a later version of this License with the same License Elements as this License; (iii) a Creative Commons jurisdiction license (either this or a later license version) that contains the same License Elements as this License (e.g., Attribution-NonCommercial-ShareAlike 3.0 US) ("Applicable License"). You must include a copy of, or the URI, for Applicable License with every copy of each Adaptation You Distribute or Publicly Perform. You may not offer or impose any terms on the Adaptation that restrict the terms of the Applicable License or the ability of the recipient of the Adaptation to exercise the rights granted to that recipient under the terms of the Applicable License. You must keep intact all notices that refer to the Applicable License and to the disclaimer of warranties with every copy of the Work as included in the Adaptation You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Adaptation, You may not impose any effective technological measures on the Adaptation that restrict the ability of a recipient of the Adaptation from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section 4(b) applies to the Adaptation as incorporated in a Collection, but this does not require the Collection apart from the Adaptation itself to be made subject to the terms of the Applicable License.

c. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.

d. If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and, (iv) consistent with Section 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(d) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

e. For the avoidance of doubt:
   i. **Non-waivable Compulsory License Schemes.** In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;
   ii. Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License if Your exercise of such rights is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(c) and otherwise waives the right to collect royalties through any statutory or compulsory licensing scheme; and,
   iii. Voluntary License Schemes. The Licensor reserves the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License that is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(c).

f. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. **Representations, Warranties and Disclaimer**

   UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING AND TO THE FULLEST EXTENT PERMITTED BY APPLICABLE LAW, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THIS EXCLUSION MAY NOT APPLY TO YOU.

6. **Limitation on Liability.**

   EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. **Termination**

   a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

   b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. **Miscellaneous**

   a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

   b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

   c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

   d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

   e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

   f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

Bibliography

[1] http://www.ripe.net.

[2] CAIDA Analysis of Code-Red. http://www.caida.org/research/security/code-red/. Last visited May 2008.

[3] Capacitor plague. http://en.wikipedia.org/wiki/Capacitor_plague. Last visited May, 2008.

[4] SANS Top-20 2007 Security Risks. http://www.sans.org/top20/. Last visited May 2008.

[5] Sasser worm author arrested in Germany. ZDNet.co.uk. May 2004, avaliable from http://news.zdnet.co.uk/security/0,1000000189,39154196,00.htm. Last visited May 2008.

[6] The Swiss Education & Research Network. http://www.switch.ch.

[7] Wikipedia: LZO. http://en.wikipedia.org/wiki/LZO. Last visited May 2008.

[8] Wikipedia: Move-to-front transform. http://en.wikipedia.org/wiki/Move-to-front_transform. Last visited May 2008.

[9] Cisco White Paper: NetFlow Services and Applications. http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm, 2002. Not avaliable anymore as of June, 2006.

[10] Blaster Worm Update. http://isc.sans.org/diary.php?storyid=26, August 2003. Last visited May, 2008.

[11] McAfee: W32/Nachi.worm. http://vil.nai.com/vil/content/v_100559.htm, August 2003. Last visited January, 2008.

[12] Symantec Security Response - W32.Blaster.Worm. http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.worm.html, 2003. Last visited May, 2008.

[13] W32.Welchia.Worm. http://securityresponse.symantec.com/avcenter/venc/data/w32.welchia.worm.html, August 2003. Last visited May, 2008.

[14] Jay R. Ashworth. The Risks Digest - The Great Capacitor Scare of 2003. http://catless.ncl.ac.uk/Risks/22.73.html, May 2003.

[15] Paul Barford and David Plonka. Characteristics of Network Traffic Flow Anomalies. In *Procceedings of the ACM SIGCOMM Internet Measurement Workshop*, 2001.

[16] J. L. Bentley, D. D. Sleator, R. E. Tarjan, and V. K. Wei. A Locally Adaptive Data Compression Scheme. *Communications of the ACM*,

29(4), 1986.

[17] CERT Security Advisory CA-2003-20 MS.Blaster. http://www.cert.org/advisories/CA-2003-20.html, 2004.

[18] Daniela Brauckhoff, Martin May, and Bernhard Plattner. Comparison of Anomaly Signal Quality in Common Detection Metrics. In *Procceedings of ACM SIGMETRICS 2007, MineNet Workshop*, June 2007.

[19] Daniela Brauckhoff, Martin May, and Bernhard Plattner. Flow-Level Anomaly Detection - Blessing or Curse? *IEEE INFOCOM 2007*, Student Workshop, 2007.

[20] Daniela Brauckhoff, Bernhard Tellenbach, Arno Wagner, Anukool Lakhina, and Martin May. Impact of Packet Sampling on Anomaly Detection Metrics. In *Procceedings of the ACM Internet Measurement Conference 2006*, Rio de Janeiro, Brazil, October 2006.

[21] L. Briesemeister, P. Lincoln, and P. Porras. Epidemic Profiles and Defense of Scale-Free Networks. In *Proceedings of the ACM CCS Workshop on Rapid Malcode (WORM'03)*, October 2003.

[22] M. Burrows and D. J. Wheeler. A Block-Sorting Lossless Data Compression Algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994.

[23] The Burrows-Wheeler transform. http://en.wikipedia.org/wiki/Burrows-Wheeler_transform. Last visited May 2008.

[24] The bzip2 and libbzip2 official home page. http://sources.redhat.com/bzip2/. Last visited May 2008.

[25] Wikipedia: Bzip2. http://en.wikipedia.org/wiki/Bzip2. Last visited May 2008.

[26] CAIDA: Cooperative Association for Internet Data Analysis. http://www.caida.org/.

[27] Senthilkumar G. Cheetancheri, John Mark Agosta, Denver H. Dash, Karl N. Levitt, Jeff Rowe, and Eve M. Schooler. A Distributed Host-based Worm Detection System. In *Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense*, 2006.

[28] Cisco IOS NetFlow - White Papers. http://www.cisco.com/en/US/products/ps6601/prod_white_papers_list.html. Last visited May, 2008.

[29] Cisco NetFlow Services Solutions Guide. http://www.cisco.com/en/US/products/sw/netmgtsw/ps1964/products_implementation_design_guide09186a00800d6a11.html. Last visited May, 2008.

[30] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC

3954, October 2004.

[31] Robert X. Cringely. Calm Before the Storm. http://www.pbs.org/cringely/pulpit/2001/pulpit_20010730_000422.html. Last visited May, 2008.

[32] David Dagon, Cliff Zou, and Wenke Lee. Modeling Botnet Propagation Using Time Zones. In *In Proceedings of the 13th Network and Distributed System Security Symposium NDSS*, February 2006.

[33] R. Danyliw and A. Householder. 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, 2001.

[34] R. Danyliw and A. Householder. CERT Advisory CA-2001-23 Continued Threat of the Code Red Worm. http://www.cert.org/advisories/CA-2001-23.html, 2001.

[35] DDoSVax. http://www.tik.ee.ethz.ch/~ddosvax/.

[36] Thomas Dübendorfer. *Impact Analysis, Early Detection and Mitigation of Large-Scale Internet Attacks*. PhD thesis, Department for Information Tchnology and Electical Engineering, ETH Zurich, 2005.

[37] Thomas Dübendorfer, Arno Wagner, Theus Hossmann, and Bernhard Plattner. Flow-level Traffic Analysis of the Blaster and Sobig Worm Outbreaks in an Internet Backbone. In *Proceedings of DIMVA 2005, LNCS 3548*, Springer's Lecture Notes in Computer Science, 2005.

[38] Matthew Dunlop, Carrie Gates, Cynthia Wong, and Chenxi Wang. SWorD - A Simple Worm Detection Scheme. In *OTM Conferences (2)*, pages 1752–1769, 2007.

[39] eEye Digital Security. Blaster Worm Analysis. http://www.eeye.com/html/Research/Advisories/AL20030811.html, 2003.

[40] Wikipedia: Entropy. http://en.wikipedia.org/wiki/Entropy. Last visited May, 2008.

[41] FAI - Fully Automatic Installation. http://www.informatik.uni-koeln.de/fai/. Last visited May, 2008.

[42] S. Floyd and V. Paxson. Difficulties in Simulating the Internet. *IEEE/ACM Transactions on Networking*, 2001.

[43] Stefan Frei, Martin May, Ulrich Fiedler, and Bernhard Plattner. Large-scale Vulnerability Analysis. In *LSAD '06: Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense*, pages 131–138, New York, NY, USA, 2006. ACM.

[44] Hungary Gabor Szappanos VirusBuster. Virus Bulletin: Virus information and overview - W32/Welchia. http://www.virusbtn.com/resources/viruses/welchia.xml, April 2004. Last visited Jan-

uary, 2005.

[45] Carrie Gates, Michael Collins, Michael Duggan, Andrew Kompanek, and Mark Thomas. More Netflow Tools for Performance and Security. In *Proceedings of the 18th Conference on Systems Administration (LISA 2004), Atlanta, USA, November 14-19, 2004*, pages 121–132. USENIX, 2004.

[46] Wikipedia: Gibbs' inequality. `http://en.wikipedia.org/wiki/Gibbs'_inequality`. Last visited May, 2008.

[47] J. Willard Gibbs. *The Collected Works of J. Willard Gibbs*. Yale University Press, 1957.

[48] GNU GENERAL PUBLIC LICENSE, Version 2. `http://www.gnu.org/copyleft/gpl.html`, June 1991.

[49] G. Gu, Z. Chen, P. Porras, and W. Lee. Misleading and Defeating Importance-Scanning Malware Propagation. In *Proceedings of the 3rd International Conference on Security and Privacy in Communication Networks (SecureComm 2007), Mice, France*, 2007.

[50] The gzip home page. `http://www.gzip.org/`.

[51] Lukas Haemmerle. P2P Filesharing Population Tracking. Master's thesis, ETH Zurich, 2004.

[52] L. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, and D. Wolber. A Network Security Monitor. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1990.

[53] Prof. Dr. K. Hinderer. *Stochastik für Infomatiker und Ingenieure*. Prof. Dr. K. Hinderer, 1989. Skript zu Vorlesung an der Universität Karlsruhe.

[54] Wikipediea: Huffman Coding. `http://en.wikipedia.org/wiki/Huffman_coding`. Last visited May, 2008.

[55] David A. Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098–1101, 1952.

[56] `http://www.ipv6.org/`.

[57] S. M. Kay. *Fundamentals of Statistical Signal Processing: Estimation Theory*. Prentice-Hall PTR, Englewood Cliffs, NJ, 1993.

[58] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining Anomalies Using Traffic Feature Distributions. In *Proccedings of ACM SIGCOMM, Philadelphia, PA, August 2005*, 2005.

[59] Frank Lambert. A Student's Approach to the Second Law and Entropy. `http://www.entropysite.com/students_approach.html`, August 2005. Last visited May, 2008.

[60] Wanke Lee and Dong Xiang. Information-Theoretic Measures for Anomaly Detection. IEEE Sumposium on Security and Priovacy, Oakland, CA, May 2001.

[61] Robert Lemos. MSBlast epidemic far larger than believed. http://news.com.com/MSBlast+epidemic+far+larger+than+believed/2100-7349_3-5184439.html, 2004. Last visited May 2008.

[62] Abraham Lempel and Jacob Ziv. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, May 1977.

[63] Ming Li and Paul Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Verlag, second edition edition, 1997.

[64] Richard Lippmann, David Fried, Isaac Graf, Joshua Haines, Kristopher Kendall, David McClung, Dan Weber, Seth Webster, Dan Wyschogrod, Robert Cunningham, and Marc Zissman. Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, Los Alamitos, CA, 2000. IEEE Computer Society Press.

[65] Wikipedia: LZ77 and LZ78. http://en.wikipedia.org/wiki/LZ77. Last visited May, 2008.

[66] http://www.oberhumer.com/opensource/lzo/. LZO compression library. Last visited May, 2008.

[67] Mohammad Mannan and Paul C. van Oorschot. On Instant Messaging Worms, Analysis and Countermeasures. In *WORM '05: Proceedings of the 2005 ACM workshop on Rapid malcode*, pages 2–11, New York, NY, USA, 2005. ACM.

[68] M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. In *ACM Trans. on Modeling and Computer Simulation*, volume 8, pages 3–30, 1998.

[69] Makoto Matsumoto. Mersenne Twister Home Page. http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html. Last visited May, 2008.

[70] John McHugh. The 1998 Lincoln Laboratory IDS Evaluation. In *RAID '00: Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection*, pages 145–161, London, UK, 2000. Springer-Verlag.

[71] J. Mirkovic, J. Martin, and P. Reiher. A Taxonomy of DDoS Attacks

and DDoS Defense Mechanisms. http://www.lasr.cs.ucla.edu/ddos/ucla_tech_report_020018.pdf, 2002. Last visited May, 2008.

[72] (Modified) BSD license. http://en.wikipedia.org/wiki/BSD_License, 1988. Last visited May, 2008.

[73] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer Worm. *IEEE Security and Privacy*, 4(1):33–39, July 2003.

[74] D. Moore, C. Shannon, and J. Brown. Code-Red: a case study on the spread and victims of an Internet worm. In *Proceedings of the ACM/USENIX Internet Measurement Workshop*, Marseille, France, November 2002.

[75] O. Müller, D. Graf, A. Oppermann, and H. Weibel. Swiss Internet Analysis. http://www.swiss-internet-analysis.org/, 2004.

[76] Estimation of entropy and information of undersampled probability distributions. Workshop followinf NIPS'03. http://www.menem.com/~ilya/pages/NIPS03/, 2003. Last visited May, 2008.

[77] The openMosix Project. http://openmosix.sourceforge.net/. Last visited March, 2008, project officially shut down as of March 1, 2008.

[78] http://www.openssh.com/.

[79] R. Pastor-Satorras and A. Vespignani. Epidemic Spreading in Scale-Free Networks, 2001.

[80] Ryan Permeh, Marc Maiffret, and Ryan Permeh. eEye Digital Security Advisory .ida Code Red Worm. http://research.eeye.com/html/advisories/published/AL20010717.html, July 2001. Last visited May, 2008.

[81] Georgios Portokalidis and Herbert Bos. SweetBait: Zero-Hour Worm Detection and Containment Using Low- and High-Interaction Honeypots. *Elsevier Computer Networks, Special Issue 'From Intrusion Detection to Self-Protection'*, 51(5):1239–1255, April 2007.

[82] J. Quittek, T. Zseby, B. Claise, and S. Zander. Requirements for IP Flow Information Export (IPFIX). RFC 3917, October 2004.

[83] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. My Botnet Is Bigger Than Yours (Maybe, Better Than Yours): Why Size Estimates Remain Challenging. In *Proceedings of HotBots 2007*, April 2007.

[84] RFC 1951: DEFLATE 1.3 specification.

[85] RFC 1952: GZIP 4.3 Specification.

[86] RFC 3513: Internet Protocol Version 6 (IPv6) Addressing Architecture.

[87] Haakon Ringberg, Augustin Soule, Jennifer Rexford, and Christophe Diot. Sensitivity of PCA for Traffic Anomaly Detection. In *SIGMETRICS '07 Conference Proceedings*, pages 109–120, 2007.

[88] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.

[89] S. Schechter, J. Jung, and A. Berger. Fast Detection of Scanning Worm Infections. In *In Proceedings of the Seventh International Symposium on Recent Advances in Intrusion Detection, French Riviera, France, September 2004*, 2004.

[90] C. Shannon and D. Moore. CAIDA: The Spread of the Witty Worm. `http://www.caida.org/research/security/witty/`, 2004.

[91] C. E. Shannon. A Mathematical Theory of Communication. The Bell System Technical J. 27, 1948.

[92] C.E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27, 1948. available from `http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html`.

[93] Colleen Shannon and David Moore. The Spread of the Witty Worm. *IEEE Security and Privacy*, 2(4):46–50, July/August 2004.

[94] SILK - System for Internet-Level Knowledge. `http://tools.netsa.cert.org/silk/`. Last visited May, 2008.

[95] Augustin Soule, Fernando Silveira, Haakon Ringberg, and Christophe Diot. Challenging the Supremacy of Traffic Matrices in Anomaly Detection. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, 2007.

[96] Eugene H. Spafford. The Internet Worm Program: An Analysis. Technical Report CSD-TR-823, Purdue University, 1988.

[97] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in Your Spare Time. In *Proc. USENIX Security Symposium*, 2002.

[98] Stuard Staniford, David Moore, Vern Paxson, and Nicholas Weaver. The Top Speed of Flash Worms. In *Proceedings of the Workshop on Rapid Malcode (WORM) 2004*, 2004.

[99] Stephens, Curtis E, ed. Information technology - AT Attachment 8 - ATA/ATAPI Command Set (ATA8-ACS), working draft revision 3f, December 2006.

[100] Standard Template Library Programmer's Guide. `http://www.sgi.`

com/tech/stl/. Last visited December, 2005.

[101] Bernhard Plattner Thomas Dübendorfer. Host Behaviour Based Early Detection of Worm Outbreaks in Internet Backbones. In *Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WET ICE 2005), STCA security workshop, Linköping, Sweden*, 2005.

[102] Yuji Ukai and Derek Soeder. ANALYSIS: Sasser Worm. http://research.eeye.com/html/advisories/published/AD20040501.html, 2004. visited May, 2008.

[103] Eidgenössischer Datenschutz und Öffentlichkeitsbeauftragter (EDÖB). Leitfaden Internetüberwachung am Arbeitplatz. http://www.edoeb.admin.ch/dokumentation/00445/00472/00532/index.html?lang=de, 2003. Art.-Nr. 410.054.d, Bundesamt fr Bauten und Logistik BBL. Last visited May, 2008.

[104] US-CERT. Vulnerability Note: Witty (VU#947254). http://www.kb.cert.org/vuls/id/947254, 2004.

[105] P. Vitanyi and R. Cilibrasi. Clustering by compression. http://arxiv.org/abs/cs.CV/0312044, 2003. Last visited May, 2008.

[106] A. Wagner and B. Plattner. Peer-to-Peer Systems as Attack Platform for Distributed Denial-of-Service. In *ACM SACT Workshop, Washington, DC, USA*, 2002.

[107] Arno Wagner. NetFlow Data Capturing and Processing at SWITCH and ETH Zurich. Contribution to the Architectural Panel at FloCon, 2004.

[108] Arno Wagner. Entropy Based Detection of Fast Internet Worms. *The Mediterranean Journal of Computers and Networks*, 4(1), January 2008. ISSN: 1744-2397, Special Issue on Network Measurement and Data Mining.

[109] Arno Wagner, Thomas Dübendorfer, Lukas Hämmerle, and Bernhard Plattner. Flow-Based Identification of P2P Heavy-Hitters. In *International Conference on Internet Surveillance and Protection (ICISP),*, Cap Esterel, France, 2006.

[110] Arno Wagner, Thomas Dübendorfer, Bernhard Plattner, and Roman Hiestand. Experiences with Worm Propagatiopn Simulations. In *Proceedings of the Workshop on Rapid Malcode (WORM) 2003*, 2003.

[111] Arno Wagner and Bernhard Plattner. Entropy Based Worm and Anomaly Detection in Fast IP Networks. STCA security workshop, WET ICE 2005 Linköping, Sweden, 2005.

[112] L. Wall, T. Christiansen, and R. L. Schwarz. *Programming Perl, 2nd*

Edition. O'Reilly, 1996.

[113] Y. Wang, D. Chakrabarti, C. Wang, and C. Faloutsos. Epidemic Spreading in Real Networks: An Eigenvalue Viewpoint, 2003. 22nd Symposium on Reliable Distributed Computing, Florence, Italy, Oct. 6-8, 2003.

[114] N. C. Weaver. `http://www.cs.berkeley.edu/~nweaver/warhol.html`, 2001. visited June, 2003.

[115] Nicholas Weaver and Dan Ellis. Reflections on Witty: Analyzing the Attacker. *login The USENIX Magazine*, June 2004. Avaliable at `http://www.usenix.org/publications/login/`.

[116] Stephanie Wehner. Analyzing Worms and Network Traffic using Compression. `http://arxiv.org/pdf/cs.CR/0504045`, April 2005. Last visited May, 2008.

[117] Stephanie Wehner. Analyzing Worms and Network Traffic using Compression. *Journal of Computer Security*, 15(3):303–320, 2007.

[118] S. Wei, J. Mirkovic, and M. Swany. Distributed Worm Simulation with a Realistic Internet Model. In *Proceedings of the 2005 Symposium on Modeling and Simulation of Malware*, June 2005.

[119] Diego Zamboni, James Riordan, and Yann Duponchel. Building and Deploying Billy Goat: a Worm-Detection System. In *Proceedings of the 18th Annual FIRST Conference*, 2006.

[120] C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and Early Warning for Internet Worms, 2003. In *Proceedings of the 10th ACM conference on Computer and communication security*.

[121] C. C. Zou, W. Gong, and D. Towsley. Code Red Worm Propagation Modeling and Analysis. In *Proceedings of the 9th ACM conference on Computer and communications security, Washington, DC, USA*, November 2002.

[122] Cliff C. Zou. Internet Worm Propagation Simulator. `http://tennis.ecs.umass.edu/~czou/research/wormSimulation.html`, 2004. Last visited December, 2007.

[123] Cliff C. Zou, Don Towsley, and Weibo Gong. On the Performance of Internet Worm Scanning Strategies. *Perform. Eval.*, 63(7):700–723, 2006.