ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Jonathan Gysel

Analysis and Modeling of Internet Epidemics

Master Thesis MA-2008-01
December 2007 to June 2008

Tutor: Daniela Brauckhoff
Co-Tutor 1: Thomas Maillart
Co-Tutor 2: Dr. Riley Crane
Supervisor: Prof. Dr. Bernhard Plattner

Abstract

Although there is a lot of literature available exploring the spreading dynamics of cyber worms, only rudimentary information about the detailed prevalence is known. Out of a large unique NetFlow dataset we extract an epidemic archetype called Blaster. Our studies result in several consequences how to describe worms with the so-called Kermack-McKendrick SIR model. In addition we apply the theory of exogenous and endogenous shocks in complex networks to our observed worm example. For both models studied we needed to handle daily and weekly oscillation since Internet traffic is highly affected by consumer caused seasonality. For this purpose we introduce two developed methods. Furthermore we examine the long-timed life cycle of Blaster.

We present an modular framework which allows to analyze anomalies in NetFlow data not just by time series, but also tracks the dynamics in a new IP-based (host centric) approach. This new method gives a much deeper insight into the infection and recovery process and is independent of effects caused by chopping the time into intervals.

Kurzbeschreibung

Obwohl in einschlägiger Literatur die Ausbreitungsdynamik von Cyberwürmern mehrfach behandelt wurde, ist bis heute nur wenig bekannt über die exakte Verbreitung dergleichen. Aus einem immensen NetFlow-Datenarchiv extrahieren wir ein Wurm-Paradebeispiel mit dem Namen Blaster. Unsere Studien legen zahlreiche Konsequenzen für die Modellierung an den Tag und helfen mit, Würmer korrekt mit dem sogenannten Kermack-McKendrick SIR-Modell zu erklären. Zusätzlich verifizieren wir eine in die Mode kommende Theorie namens „Exogene und Endogene Erschütterungen in komplexen Netzwerken“. Für beide Modelle stellte sich das Problem, dass Netzwerkverkehr täglichen und wöchentlichen Schwankungen unterliegt, welche von dem zyklischen Verhalten der Nutzer verursacht wird. Wir führen zwei innovative Methoden ein, um diesen Oszillationen Herr zu werden. Des weiteren untersuchen wir das aussergewöhnlich lange Zeitverhalten unseres Wurmes.
Im technischen Teil führen wir ein Framework ein, mit welchem es nicht nur möglich ist den zeitlichen Verlauf von Anomalien zu studieren, sondern neu auch in einem IP-basierten Ansatz die Dynamik untersuchen kann.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Outline

1.1 Introduction

When in 1986 the first computer worm with the name Morris appeared, no one knew that this would be one of the major security threats in the early years of the next millennium. For instance VBS/Loveletter, CodeRed, Sircam, Nimda, Sobig, and Sasser are just few famous examples causing an economic damage of millions and millions dollars. Record holder concerning spreading speed was SQL/Slammer in early 2003, infecting 90% of the vulnerable population within 10 minutes and slowing down globally the Internet. With the emergence of worms, many papers appeared trying to reconstruct the spreading dynamics. Beneath various theoretical contributions and simulations, there where many attempts to adapt models from other research topics to the case of Internet worms, especially the so called SIR model from the epidemiology area.
Although the interest on large-scale worms has flattened due progresses in both, the security industry and new trends in cyber-crime, our disillusioning finding is that only few of the spreading dynamics has been understood properly. Even for the well studied Blaster worm, we where able to give a much deeper insight than ever before.
Our goal was to verify and if needed improve existing models. Beside the already mentioned SIR model we also applied a framework from physics, named exogenous and endogenous shocks in complex networks. For both models we needed to handle the periodicity in our signals which are omnipresent due day/night cycles. In addition the lower activity on weekends increases the complexity to deconvolve signals. For our findings we developed a modular and extendable c++ framework allowing not only to produce time series, but also collect host-centric informations.

1.2 Overview

For this thesis we studied various papers in order to get an overview what was already done concerning malware spreading dynamics. We summarized our investigations in the related work, Section 1.3. The first part (Section 1.3.1) covers technical papers analyzing merely the source code of worms and the influence of the program design to spreading. We switch then over to

dossiers dealing with the topic of modeling epidemics (Section 1.3.2). In the domain of Internet worms, mainly the SIR model and derivatives of it are adopted. Of course we find networks everywhere. Every research area, from genome to biology to economics, is somehow confronted with networks with similar properties. So we include also documents from other scientific domains.

Chapter 2 deals with the necessary theoretical background needed to understand the SIR model (Sec. 2.1) and the theory of endogenous and exogenous shocks in complex networks (Sec. 2.2). In addition we introduce the main topology of the analyzed SWITCH network, which is mainly an autonomous system as part of the Internet. And we outline in Section 2.4 some important technical details about the investigated Blaster worm.

To explore the dynamics of Blaster, we designed a extensible framework. But the tool set was designed enough flexible to adapt the code to new cases without major transcription issues. Whereas Chapter 3 mainly describes the requirements, the limiting resource factors and basic conditions, Chapter 4 dives into technical details. Beneath the data flow and class descriptions of our NetFlow data extraction tool, we introduce briefly the developed cluster management scripts and mathematical data analysis tools, such as the decay estimation tool.

All results can be found in Chapter 5. Initially we start off with a calculation which fraction of computers become infected (Sec. 5.1). Then we go over to an analysis of the infection process where we state the efficiency of the two used scanning mechanism of blaster, as well as the number of infections within SWITCH (Sec. 5.2). Section 5.3 presents a discussion of the obtained time series and dismantles some alleged inaccuracies in the presented graphs.

Because time series give only a limited view of the complete spreading dynamics, we developed a new approach collecting informations in terms of IPs instead of aggregating traffic in fixed intervals. As in Section 5.4 presented, from this dataset we where able to illuminate many different aspects of the spreading dynamics. For instance we show the fight between spreading and recovery and the user behavior concerning the recovery process. We identify some key values influencing the spreading and disclose the arising consequences for modeling Internet epidemics.

Also crucial is the knowledge of the long time behavior of worms. In Section 5.5 we look at additional weeks after the outbreak (half a year, one year and four years later). The long observed life cycle of worms has also consequences for the SIR-model. We summarize all implications of our findings for the SIR model in Section 5.6. Then we try to create an enhanced SIR model taking into account the consequences we discovered.

Very new and thus still immature is our recently developed method to get rid of daily and weekly oscillations. We recognized that a promising idea of regularly sampling a decay does not work well. Therefore we present a second attempt decoupling the seasonality from the decay (Section 5.7).

We would like to point out the rather extensive Appendix. Beneath a list of abbreviations (Appendix A) you find there many full-page graphic overviews (Appendix B to E). We put them there not to hinder the fluentness of the text part. Whenever necessary there are references in the text pointing to the matching plots.

1.3 Related Work and Further Reading

1.3.1 Internet Worms

Initially we studied many papers about the functionality of various Internet worms. [20] presents a list of the top 20 vulnerabilities which where abused by malware. [11] presents a compilation of bugs found in badly programmed worms and viruses. A in-depth byte code analysis of Bagle was done by Bell Labs [42]. [24] analyzes the code of MyDoom.B. Another article that presents the result from reverse engineered worm code focuses on the SQL/Slammer [7]. Slammer, also known as Sapphire, was the fastest spreading worm ever seen [32]. The history of Sasser [28] shows that it was an inspiration source for other worms like Blaster and Netsky. Obviously there seem to be criminal connections in the malware scene. The spread of Witty worm [44] is also an interesting case for science, since it successfully spreads on a very small target population. [51] evaluates the trends of worm programming and what the future threats are. Yet another assessment about the well studied Code Red is [33]. But their conclusions are creditable, since they can proof that the „DHCP effect“ leads to an overestimation of infected hosts, especially when the considered period is longer than 24 hours.

For our spreading dynamics analysis we relied mostly on Blaster, since the needed technical details of this anomaly are well documented [21, 26]. A resume can be found here [55]. The monitoring of Blaster [2] using a network telescope (aka. black hole) is consistent with our finding about the persistence of worms. In addition they show that the persistence of a worm is due new infections within this second phase. Only 15% of addresses during outbreak are still present in the persistence phase. Interestingly, they propose in [2] to monitor /24 networks instead of single hosts to reduce the effect of overestimation due DHCP. More about anomaly detectability, especially in sampled traffic, can be found here [8, 23].

For our findings, we where obliged to study different network structures, since the topology has a significant influence to the spreading dynamics. Barabasi *et al.* published a paper where they argue to describe the world wide web rather as a scale free network instead of a random graph [1]. The same is also claimed for contacts in instant message networks [46]. Eckmann *et al.* study the response time in email contacts. Since the time until a reply is sent is also dependent on day/night user behavior, they come up with an idea of measuring the time in ticks instead of absolute intervals [22]. During night and weekends, the slowing down of the network becomes eliminated, since the relative time also decreases.

1.3.2 Modeling Epidemics

The SIR model we used and improved originates from epidemiology research in biology. An introduction to the model can be found here [54]. Two interesting extensions to the basic SIR model are mentioned. Inoculation, in computer language called patching, directly allows the transition from susceptible to recovered. Secondly, for non-constant populations with an additional death

and birth rate, an epidemic does not vanish and reaches a steady state. This could also explain the long time observations of computer worms.

[30] presents simulations based on the assumption that immunity is just a temporary state, especially when considering long term measurements. A transition back to susceptible after recovery is their proposed expansion. Analytical Active Worm Propagation (AAWP) is the name of a model developed by Chen *et. al* [13]. In contradiction to the classical SIR, they use a discrete time model. In addition they consider the time needed for an infection and the patching of susceptible hosts. Unfortunately the model is only described qualitatively what makes it difficult to comprehend it. Interesting are the results of taking local subnet scanning into account (LAAWP-Model). Chen *et. al* conclude that hit list scanning, as applied by many worms, rather slows down the spreading if not examined carefully. Based on observations of Code Red, Zou *et al.* corrects two shortcomings not considered in the traditional epidemic model: Human countermeasures like patching, upgrading, and filtering, as well as decreased infection rates due congestion and troubles have a significant influence [60]. The adaptation yields an adequate model for Internet epidemics if the model parameters are chosen carefully. A summarization of the key findings can be found here [31]. The same group also studied the spreading of email worms on different topologies and resulting consequences for selective immunization [18].

To model our epidemics we searched ways how SIR simulations could be applied to worms. [15] gives an introduction of simulating time continuous SIR epidemics. Stochastic time discrete simulations instead are often made with the Gillespie Algorithm (e.g. [41]). We've found only one paper describing how parameter estimation for stochastic epidemics is done [25]. In case of full data availability a maximum likelihood estimation for the spreading and recovery rate is appropriate. A method called Markov Chain Monte Carlo (MCMC) simulation is used for partial observability of an epidemics. The paper introduces a closed-source simulator specialized for diseases in animal reproduction.

We found that the spreading rate when modeling worms must be large. Many studies about the spreading where made by Pastor-Satorras and Alessandro Vespignani. They use the SIS model in order to prove that infinite size scale free networks have an epidemic threshold $\lambda_c = 0$. This means any epidemic event will spread, even when the effective spreading rate λ is very small [38, 6]. This holds also true for a wide range of scale free networks [39] and $\lambda_c = 0$ is close to zero for finite size scale free networks [40]. However their proof is only applicable for scale free networks. This raises the question if worms are still included in the proof, when they operate on a different logical structure rather than the scale free physical network as our Blaster.

We applied also the theory of endogenous and exogenous shocks in complex networks to the case of Blaster. The difference between endogenous and exogenous shocks can be explained well on the example of Amazon book sales rankings [50, 19]. An adaption of the ideas to the case of YouTube gives hope that a distinction between quality content and junk becomes possible [16]. Classifying the chronological sequence of views into exogenous and endogenous events is a robust method and does not rely on qualitative judgment. More mathematical details, e.g. fitting selected data to power laws, are also available [17]. More examples where the model makes

sense are listed here [48], a sample for systems with memory are earthquakes [49].

Chapter 2

Background

The theory block about „Endogenous Versus Exogenous Shocks in Complex Networks“ 2.2, as well as the theory concerning the Kermack-McKendrick SIR-Model, where kept short and cover only the basics needed to understand the upcoming analyses. But, we refer in both sections to further reading. Section 2.3 depicts the topology of the considered SWITCH network and how the used dataset has been recorded. In the last section one can find a description about the programmed functionality of Blaster.

2.1 The S-I-R model

Another model more often used to explain the dynamics of digital viruses is the SIR-Model, having its roots in biological epidemiology research. It was developed in by a Scottish physician called Anderson Gray McKendrick (1876 - 1943), an epidemiology pioneer who tried to use mathematical methods in order to describe contagious illnesses in closed populations. His model was suggested as explanation for the plague in London (1665-66) and cholera (1865).
In its basic form the model consists of three coupled nonlinear ordinary differential equations:

$$\frac{dS(t)}{dt} = -\beta \cdot I(t) \cdot S(t) \tag{2.1}$$

$$\frac{dI(t)}{dt} = \beta \cdot I(t) \cdot S(t) - \gamma \cdot I(t) \tag{2.2}$$

$$\frac{dR(t)}{dt} = -\gamma \cdot I(t) \tag{2.3}$$

In addition, the population is assumed to be constant:

$$N = S(t) + I(t) + R(t) = const \tag{2.4}$$

β is denoted as the infection rate, describing the average number of transmissions from one infected unit. It is composed of the average number of contacts times the probability that such a contact results in a transmission. Similarly we have a recovery rate γ standing for the fraction of infected group members curing per time unit. The reciprocal value of γ is in accordance with the average duration of disease. Initially the whole population is assumed to be susceptible $S(0) = N$. Without loss of generality we further assume that we have one initial Infection ($I(0) = 1$) and no recovered units so far ($R(0) = 0$). Using a value $I(0) > 1$ does not alter the timely evolution, as long as the constant population condition (Eq. 2.4) is still satisfied. It would just imply a shift in time.

A key term describing the dynamics versus time is the so-called epidemiological threshold Θ_0:

$$\Theta_0 = \frac{S \cdot \beta}{\gamma} \tag{2.5}$$

It determines the average number of second generation diseases a first generation unit infects by contact before it is restored, or dead respectively. When $\Theta_0 < 1$ an infected person will transmit the disease to fewer than one second generation unit. Consequently the epidemic vanishes rapidly since it is not self-preserving. We only obtain an outbreak for $\Theta_0 > 1$, where every disease will entail in even more infected units. This threshold, and accordingly the corresponding parameter of the SIS-Model, gave rise to many discussions in the past of security industry, because there is a inconsistence whether Θ must be chosen close to the threshold or rather very large, depending on what condition we want to satisfy. The prevalence level of Internet viruses is always small, which implies to choose Θ rather small. But, the observed life time of viruses can outlast up to decades, indicating a rather large Θ [6, 57]. We already denoted this topic in the last part of Section 1.3.2, where one can find also further reading references.

The model parameters (β, γ) obtained from one disease can not be directly compared with the ones of another. For this one needs to normalize the complete population to 1 (i.e. N=1). Afterwards, we can compare directly the spreading and recovery rate, β and γ, of different diseases with unequal population sizes.

The model as described above has its advantage in the simplicity. But often there are upgrades necessary to match it to real events. In the basic SIR-Model the spread is up to a constant β proportional to the number of susceptible units $S(t)$ and also proportional to the infected population $I(t)$. While this approximation is enough adequate for biological viruses, it does not match the behavior of Internet worms. Typically we reach after an initial undamped expansion a limiting threshold. Depending on the functionality of the malware this can be CPU limitations, network congestion, or the target population simply saturates (for an example look at Fig. B.3c). One possibility to correct this shortcoming is to make β time dependent, for instance as proposed in [60]:

$$\beta(t) = \beta_0 \left[1 - I(t)/N\right]^{\eta} \tag{2.6}$$

where the constants β_0 and η determine the initial infection rate and the sensitivity factor respectively. Similarly as in Equation 2.6, we could also take into account human countermeasures accelerating the recovery process, making the transition from Infection to Recovered also time dependent: $\gamma = \gamma(t)$. However, for slow spreading we should further introduce a transition from the susceptible population directly to recovered. This process, called patching, may be additionally supported by upgrading, setting up filters to block malicious traffic, or even by temporarily disconnecting machines. Such a transition would involve a flow $Q(t) = -\mu \cdot S(t) \cdot (I(t) + R(t))$ in Equation 2.1, taking into account the rising risk awareness of users [60].

In fact there exist many extensions to the basic SIR-Model. For instance we could think of a non-constant population. Especially when we want to consider large time windows, we may have to take the vast growth into account, meaning the birth of many new susceptible machines. One paper, [30], investigates the vulnerability life cycle using an additional transition back from recovered to susceptible. The population groups sizes corresponds in reality always to finite integer numbers. Consequently, the SIR-Model is an continuous approximation appropriate only for large populations. Some papers, particularly when stochastic simulations are deployed, therefore use discretized SIR-Models, like [13] or [18].

2.2 Endogenous Versus Exogenous Shocks in Complex Networks

Extreme events are observed everywhere in nature and in all kind of social systems. One can find them in financial stock markets, power outages, human diseases, earthquakes, social unrests, and so on. This raises the question, if we can link the observed behavior in such complex network systems to epidemics seen in the Internet. Extreme events, in Internet language often referred to anomalies, where not observed just in the basic IP-layer but also in many logical networks routed in higher layers of the Internet structure.

Our field of interest in this thesis is mainly the spreading of Internet worms. In the past we have experienced worms like LoveLetter, Blaster, CodeRed and SQLSlammer which have become a real security threat for both, business and home users. Usually one goal of such malware is a maximal degree of prevalence what requires a fast and aggressive spreading mechanism. This is archived by tree like spreading algorithms. Self-replicating copies of malicious code snowball the network. One initial copy triggers an avalanche and all successfully infected victims aid one another to attack further victims. In order to obtain a flat tree structure superior nodes don't stop to search for vulnerable hosts in the IP address space until they're recovered. So, the spreading strategies of Internet worms are similar to those seen by biological viral epidemics with one difference. Worms have a much more global view, since they also can infect far distant computers and not just physically local neighbors.

Let's illustrate the dynamics of exogenous and endogenous shocks using the example from Amazon book sales (source: [50]). The exact number of sales is confidential, but the timely progress can be reconstructed approximately. Sornette *et al.* tracked the ranking of several thousand books

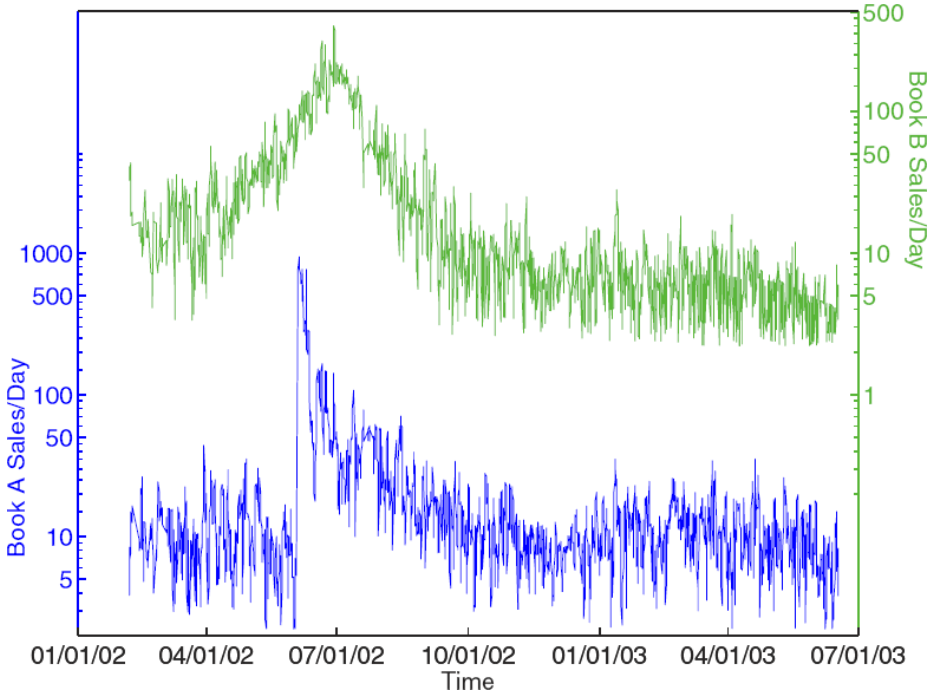


Figure 2.1: Two examples of amazon books, the evolution of sales per day. Book A experiences a exogenous shock in sales whereas the sales for book B are the archetype for an endogenous shock, source: [50]

offered at Amazon and recognized two main prototypes of shocks in the evolution of top sellers over and over again. An article in the New York Times about the phenomenal research of Dr. Miriam Nelson caused a rise from rank 2000 to 6 in less than 12 hours (Book A, Figure 2.1). This book, entitled „Strong Women Stay Young“, is the archetype of an exogenous shock, caused by a published recommendation in a newspaper. Book B, called „Heaven and Earth (Three Sisters Island Trilogy) by N. Roberts“, instead never experienced a major advertising campaign. Its success is merely based on the viral recommendation spreading through a social network. Word-of-mouth propagation entails a slow continuous growth with an almost symmetrical decay.

Sales are a combination of external forces, like advertisement or selling campaign, and of social factors. Consequently, the typical evolution of such sales are a mixture of the two presented prime examples. The sum of all sales can be described by a so-called conditional Poisson branching process with a rate $\lambda(t)$.

$$\lambda(t) = S(t) + \sum_{i,t_i \leq t} \mu_i \cdot \phi(t - t_i) \tag{2.7}$$

The influence of a reader to his next-generation buyers is time dependent. It is believed that the distribution of waiting time defining human activity is power law distributed. The so-called memory kernel $\phi(t - t_i)$ in Equation 2.7 describes this latency, more precisely the distribution of waiting

time between two generations of buys. A person i purchasing a book at time t_i influences μ_i potential new customers, causing a viral spreading. Beneath the described branching process we have a rate of spontaneous sales $S(t)$, not influenced by previous users. They can be modeled as white noise or originate from exogenous sources, like media coverage, either.

The idea of classifying book sales as endogenous and exogenous peaks was also applied to other cases. For instance, the daily views of videos on YouTube reveal exactly the same dynamics [16]. However, just considering the best selling books does not exhibit a third classification. Most products, books as well as videos, never work out and stay fameless. In terms of the exogenous/endogenous model they are referred to as exogenous sub-critical. Activity generated by an exogenous incident at time t_c does not propagate trough the network. After few generations the self-preservation reduces and the initial impulse dies out. Mathematically this is the case when the mean value $\langle \mu_i \rangle$ of potential next-generation customers μ_i is smaller than 1. The activity A_{sc} is then proportional to a power law decay:

$$A_{sc}(t) \sim \frac{1}{(t - t_c)^{1+\theta}} \tag{2.8}$$

For the previously mentioned exogenous critical case, where $\langle \mu_i \rangle$ is close to 1, the spreading reaches many customers over long social network cascades. The Activity can then be described as:

$$A_{exo}(t) \sim \frac{1}{(t - t_c)^{1-\theta}} \tag{2.9}$$

What remains is the endogenous critical spread. The growth is a result of viral word-of-mouth spreading. We can describe the activity proportional to:

$$A_{endo}(t) \sim \frac{1}{|t - t_c|^{1-2\cdot\theta}} \tag{2.10}$$

In the case of YouTube θ was empirically identified to be a universal value close to 0.4 ($\theta = 0.4 \pm 0.1$), leading to exponents near 1.4, 0.6 or 0.2. The three classifications are consistent with a qualitative division into junk, quality and viral videos. Whereas junk videos do not spread beyond few cascades of viewers, the latter two types are similar but the activity growth is different.

Comparing the Amazon book sales versus time (Fig. 2.1 with the dynamics of Internet worm spreading (Fig. 5.10) raises suspicion that the model may be applicable to Internet anomalies. Such an analysis is performed in Chapter 5.7. More about the endogenous and exogenous model and many more examples can be found here [49, 48, 19, 17, 50, 16].

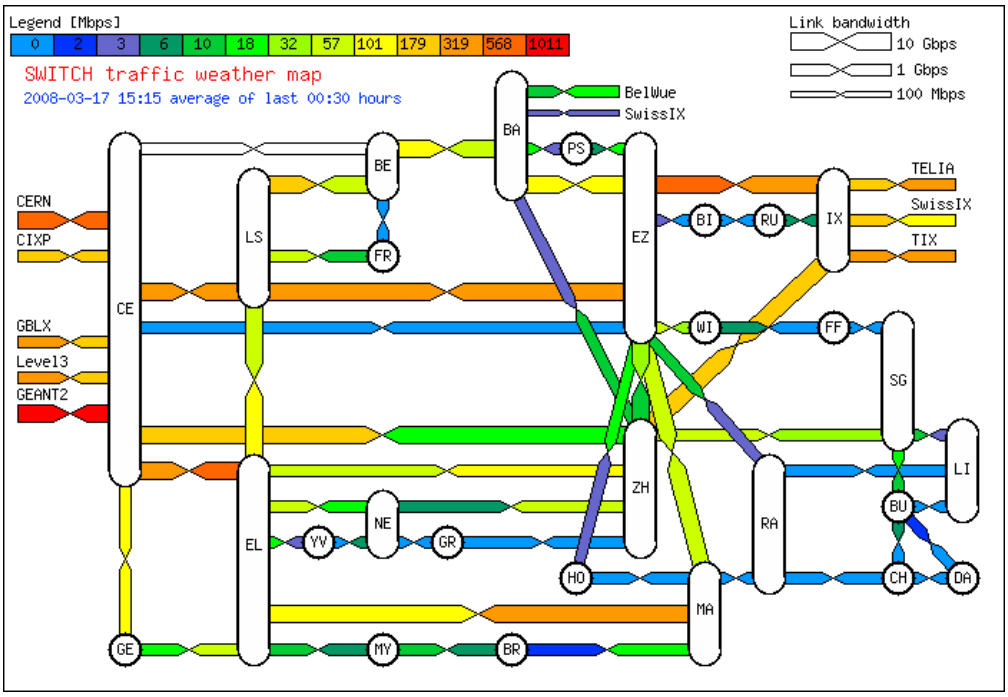


Figure 2.2: The SWITCH traffic weather map, load on an average work day. Beneath the backbone topology we see the four Internet peering interfaces (border routers CE1, CE2, BA, and IX). source [58]

2.3 The SWITCH Network

Beneath all theoretical work we present in this thesis, we verified our outcomes on the basis of real data. All shocks we analyze in our work originate from the traffic recordings done on the borders of the SWITCH network [52]. SWITCH, the Swiss Academic and Research Network, is a mid-sized backbone operator that connects all swiss universities as well as various other research laboratories (e.g. CERN [12] or IBM research lab [27]).

Since early 2003 all flows are recorded in the Cisco NetFlow v5 format. A flow is defined as the 5-tuple source IP, destination IP, source port, destination port, and layer 4 protocol (e.g. TCP, UDP). Bidirectional flows are counted as two separate unidirectional flows. We have *no insight* what is happening within the network. Only flows crossing one of the four border gateways are recorded. Transit traffic is logged twice, but this is only a minor fraction. Currently there are 4 border gateways (see Figure 2.2). Two located at the CERN research lab (CE), one in Basel (BA), and the last one in Zürich (IX). At the end of 2003, approximately 60 million flows per hour where recorded. The assigned IP range contains currently a pool of 2.2 million IPs. The compressed NetFlow dataset, aggregated from 2003 until end of 2007, needs around 30 TByte of space. For more statistics refer to Chapter 5.1.

The SWITCH border routers are configured to export a flow according to the following policies:

- The control flag FIN or RST in a segment is set.

- No packet has been received for 30 seconds.
- A flow lasts longer than 15 minutes. In this case the flow will be divided up in multiple records.
- The router runs out of memory.
- Just one or two packets are sent and followed by a pause longer than 4 second.

2.4 The Spreading Mechanism of W32Blaster

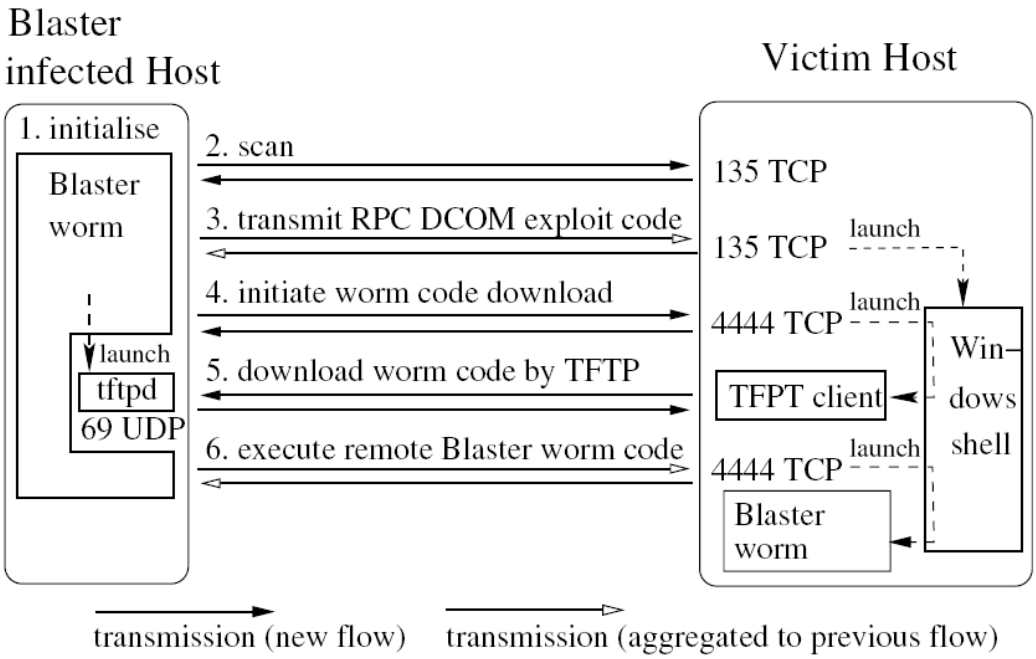


Figure 2.3: Blaster's infection steps. The Blaster worm is a multistage worm, an infection succeeds only if all shown 6 bidirectional transmissions occur, source [55]

Blaster is a multistage worm. This means, more than one step is needed for a successful infection and make this class of worm easier detectable on flow level. Each initialized connection comes along with a (hopefully) unique flow pattern which makes the worm traffic distinguishable from normal traffic. Since there is already a lot of literature available [55, 2, 26, 8] describing the functionality of Blaster, we give here just a brief overview from a network point of view.

Step 1, Initialization: During the period from September to December 2003, Blaster started a TCP SYN flooding attack against `windowsupdate.com`, using a spoofed source IP. This attack was ineffective because Microsoft simply needed to stop the DNS forwarding to the real URL `update.microsoft.com`.

Step 2, Scanning: Blaster scans for a vulnerable service running on destination port 135/TCP. 60% of time is used for random scans using addresses of type A.B.C.0. During the remaining 40% local targets are checked using the first 24 bits (E.F.G.0) of the infected computer's IP E.F.G.H. If G is less or equal to 20, no change is assigned to G. Otherwise a random value between G and G-20 is chosen. After incrementally scanning 20 IPs, Blaster sleeps for 1.8 seconds before it continues to scan the next 20 [4]. A packet probe to port 135 is 40, 44, or 48 bytes.

Step 3, Transmission of exploit: Blaster reuses the connection from scanning to TCP port 135 to transmit the exploit code. With 80% probability Blaster uses the exploit code for Windows XP to produce a DCOM RPC interface buffer overflow, otherwise the Windows 2000 code is taken. Using the wrong exploit code causes the victim to crash and reboot the machine. The exploit opens a remote shell on port 4444/TCP, so that the victim can receive further instructions from the attacker.

Step 4, Initiation of worm download: The attacker connects now to the just opened shell on port 4444/TCP to instruct the victim to download the worm code. The victim uses the by default installed Trivial File Transfer Protocol client, `tftp.exe`, to download a copy of `msblast.exe` from the attacker.

Step 5, Transfer of worm code: If the port 69/UDP on the attacker is open and not blocked by a firewall, the victim downloads the executable blaster file.

Step 6, Starting remotely the worm: As soon as the victim has downloaded a copy of the worm code, the attacker reuses the remote shell connection on port 4444/TCP to start the duplicate. The victim performs the transition to an additional attacker and begins now to scan for further vulnerable victims.

A disadvantage of multistage worm is the error-prone procedure. Each step can go wrong due filtering, unfulfilled race conditions, or differently configured victims. [55] shows that after each stage of Blaster the number of observed flows decreases approximately one order of magnitude. During the initial 12 hours of outbreak, more than 12 million scans per hour are necessary just to find and successfully infect 17 hosts inside AS559[1]. This is one reason why we decided to define an infection as successful only if all 6 steps described above, producing 3 bidirectional connections, are performed. If a quick spreading of a worm is desirable, we can conclude from the observations, a fast and aggressive scanning strategy is important. Taking a certain locality into account, as Blaster did it, and thus scanning in the range of an attacker's IP can help to spread fast but is not necessary. As mentioned, the fastest spreading worm ever seen was SQL/Slammer [32] and it used a completely random scanning strategy. Microsoft claimed at least 8 million fetched systems, whereas the Internet Storm Center estimates the number in the range between 200'000 and 500'000. Compared to the 2^{32} possible IPv4 addresses the probability of finding a vulnerable host is small. Thus the scanning policy should follow the motto „Fire and Forget" since most IPs are unassigned or the target hosts are not vulnerable (due patching or different OS).

[1] AS559 is the abbreviation for autonomous system number 559. This is the major network segment of SWITCH.

Microsoft released a patch for the DCOM RCP vulnerability on July 16, 2003 together with a security bulletin, MS03-026. At the time of the Blaster outbreak, August 11 at 16:00 UTC, becoming infected was a question of having already applied the patch or not. An suggested alternative was also to use a firewall with port 135/TCP closed. Accusing the careless end-users having neglected their duty is not appropriate. In 2003 the necessity of automatic system updates wasn't well known and released security information remained in circles of experts.

Chapter 3

Software Design

In order to be able to extract epidemic events, we created from scratch a new NetFlow data extraction tool called `epim`, standing for "**Epi**demic **M**easurement Tool". As a new object oriented library ([45]) was available which simplified the handling of Cisco NetFlow data, we decided not to rely on existing tools and available c-code examples. The major design goals was easy extensibility in case new epidemic events need to be analyzed. But also simple readability and a short familiarization time where desirable. Experiences we made in the past teached us that some functions, such as determination of source and destination AS, writing results into files, and NetFlow filter mechanisms are needed frequently. To prevent reinventing the wheel all the time, we designed some helper classes which provide all basic services usually required. This should increase the reusability of existing code. Since the framework also standardizes the output, a programmer can really focus on extracting the designated metric. The file handling and merge of results to a single output file will be done automatically.
Beneath simplicity we also spent great efforts to gain an efficient design. Previous work has shown that especially the RAM memory consumption and the execution time are critical parameters. During design phase, we evaluated the memory consumption of various design alternatives and chose the variant with the best trade-off between simplicity, memory consumption, and CPU usage.

Theoretically, it would be possible to analyze the traffic of each single boarder gateway router. Since every router is connected with other carriers we would expect that the devices are affected unequally by the large amount of flows originated from Internet attacks (e.g. see [23]). Our forecast is that the global spreading has not only a timely component but also a spatial dynamics. Nevertheless we decided to consider only the summarized traffic of all boarder gateway routers in order to simplify the interpretation of the results but also to reduce the necessary effort.
We recognized during the process of analyzing Blaster, the time series we extracted so far are valuable and help much to understand epidemic processes. Furthermore, comparing the findings with other papers is possible, since this is a widespread and unpretentious way to exhibit the anomaly progress. But being bound to time series with fixed bin size, as presented in Section 5.3, has its drawbacks. Especially when we want to comprehend the dynamics of spreading,

we need to be independent of effects caused by chopping the time into bins. So we decided to introduce an intermediate step in our data processing system. Instead of directly producing time series, we first collect important information related to observed IPs. For example the time of first scan, time of last scan, time of successful infection, number of scans sent, or the number of scans received. A complete list with detailed description of all fields extracted can be found in Section 4.1.4. This data fields can then be further processed, filtered, and formated. More precisely, we designed a tool called `hm_merge01.pl` which allows to merge multiple output files as obtained when processing data with multiple simultaneously working instances. Afterwards the data can be analyzed with the tools described in Section 4.3.

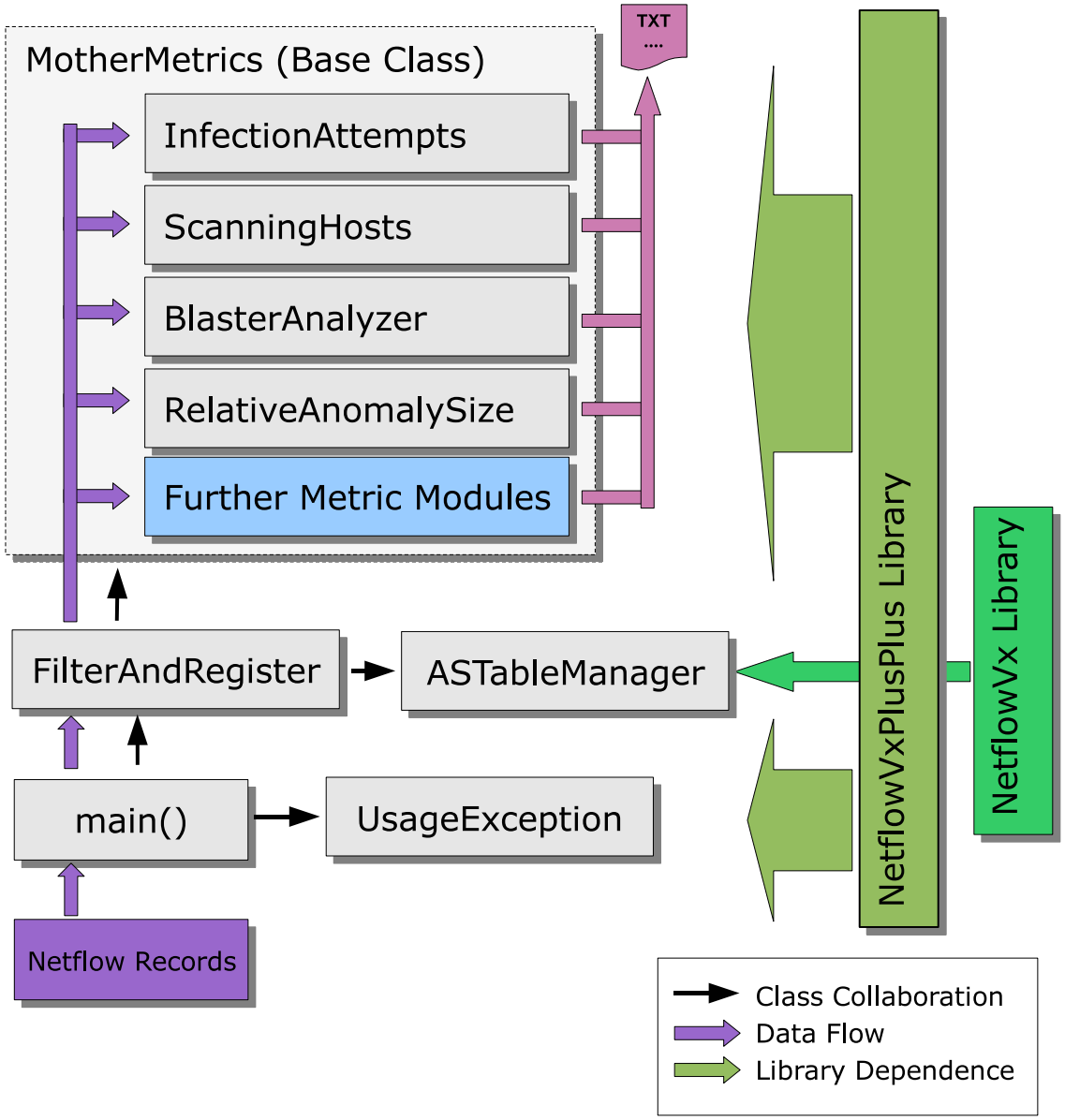


Figure 3.1: Collaboration of classes for the epidemics measurement tool, The library dependence is green whereas pink flows represent the data flow between the modules.

Chapter 4

Implementation

In the previous chapter we studied the requirements for epidemics measurement tool. The first part (Section 4.1) contains the technical documentation of the c++ framework, whereas the other two sections cover the additional scripts written. Perl scripts used for the work flow are in Section 4.2 and the mathematical analysis tools written in Python/SciPy are in the last part (Sec. 4.3).

4.1 Epidemics Measurement Tool

Since the details are well documented using Doxygen, we consider here a more global view. We present the basic concepts behind the implementation as well as the interactions between the involved classes.

4.1.1 Journey of a Record

If you are not familiar with metric extraction it may helps you to know the chronological path of a NetFlow record through all stages in the tool. Otherwise just skip this section. Let's track the path from a packet's point of view. The `NetflowVxPlusPlus` framework transparently reads the records from zipped NetFlow files and we obtain them in the function `main()` of `epim.cpp` by the method `NetFlowInput::getNextRecord(record)`. The start time of the flow will be determined and escorts the record wherever it goes. The record reaches the Object `FilterAndRegister` via method `FilterAndRegister::registerRecord(...)`. There it passes the the method `FilterAndRegister::match_prefilter(...)` in order to sort out unnecessary traffic at the very beginning. Afterwards we determine source and destination AS and pass the record (actually a reference to it) to all metric objects, using the interface `MotherMetrics::registerRecord()`. Most metric objects first apply then their own `YourMetricObj::match_post_filter()` method to reject not needed records. From now on it depends on the metric object, what is done with the record. It may be thrown away or parts of it may be stored in a hash table or other data structure. Storing the complete record (and thus making a copy of it) is not recommended, except if all flag fields are needed.

4.1.2 Class descriptions

main: As usual, the main function starts with command line argument parsing. Afterwards all specified input files become checked for readability and will be sorted according to path and filename if desired. We then loop over all NetFlow files, headers and finally records. All records are passed to the Object `FilterAndRegister` using the method `register-Record(...)`. For the trigger events, also called in the innermost loops, we dedicated a separate chapter (see Chapter 4.1.3).

FilterAndRegister: This class starts and manages all instances of derived `MotherMetrics` classes. So `FilterAndRegister` constructs, destructs, and controls all metrics object and provides all basic services needed. It feeds the metric objects with records, opens an output file handle for them and passes any trigger events to them.

Another service is the determination of source and destination autonomous system of a record. This information is stored in an enumerator called `flowDirection` and accompanies the record from now on to all metric objects. For often needed string manipulations you find some static methods in `FilterAndRegister`.

ASTableManager: All operations relating an AS_TABLE lookup are embedded here. Since the retrieval of autonomous system information was only available via patched `NetflowVx` library, we wrote a wrapper class in order to easily get the necessary information. In case this library code will be ported to c++ it should be facile to deflect AS requests to the new code. This is the only code segment still using the `NetflowVx` library and thus needs linkage to the corresponding library file, as seen in the `Makefile`.

NetflowVxPlusPlus::UsageException Thrown when an error in command line occurs. The Class inherits from `NetflowException` and was added to the `NetflowVxPlusPlus` namespace.

MotherMetrics: To simplify the handling of metric objects we needed a standardized and uniform interface. A good way to achieve such an uniformity is to use an abstract base class from whom all derived classes obtain the same envelope. From the point of view of `Filter-AndRegister` all metric objects are indistinguishable once they are initialized even though they do very different things.

The classes presented so far are all helper classes in order to simplify the measurement of metrics. The following list are now implemented metric modules, inheriting from the base class `MotherMetrics`. Please refer to Fig. 3.1.

InfectionAttempts: The class simply counts the infection attempts, also known as scan traffic, seen in the NetFlow data.

ScanningHosts: This class simply logs unique source IPs of hosts, which sent a blaster packet to port 135. This class is a good example for the usage of `HashTables` without allocated buckets. i.e. we abuse the bucket pointer to store a long int.

BlasterAnalyzer: We tried to obtain as much information as possible about every Blaster infected host hoping that we can track and understand superiorly the spreading. So this class keeps state about all infection steps the multi-stage worm performs. This allows us to generate a bunch of host based time stamps and further information fields.
`BlasterAnalyzer` anonymizes IPs by hashing them and opens its own output file. The standard file handle offered from `FilterAndRegister` is just used for statistics about the memory usage of the module. Since it is impossible to evaluate a long running event on a single calculation node, `BlasterAnalyzer` allows parallelization by splitting up the time axis in shorter intervals, although this is *NOT* perfectly possible. The class catches up trigger events in order to become informed about an ending warm-up and a starting cool-down phase. This information is needed to brief the class on how much the intervals do overlap each other. Since the extracted metrics are strongly dependent on the implementation and chosen definitions we dedicate a separate paragraph to this matter (Chapter 4.1.4).

RelativeAnomalySize: It's often difficult to estimate the real dimension of an anomaly. This class compares the flows generated form an epidemic event to all flows crossing the border of AS559. We also try to determine the number of infected hosts within a specified interval and compare this number to all active hosts. Since we split up this information for flows coming from AS559 and coming from outside, this lets us estimate roughly the percentage of infected hosts inside AS559 (for results see Chapter 5.1).

4.1.3 Triggers - A Message Passing System

Metric objects, derived from the `MotherMetric` class, just obtain plain NetFlow records including start time and AS information. For many measurements this is enough information to calculate the desired time series. But in some cases we really depend on further knowledge of what is going on. For this case we invented a simple message passing system, so that the classes can talk to each other. We entitle such messages as „triggers“ since they activate something at the target. You may have heard the keyword interrupts which work in a similar manner. But our triggers are not able to suspend an ongoing procedure as interrupts may do.
At the moment all triggers are released in the `main()` function. Due efficiency reasons we determine in advance which trigger event will be the the next one, whenever a trigger is blown. If we wouldn't do that, we had to check every single trigger after each incoming record.
Here is a list of currently implemented triggers:

PRINT_HDR: Instructs a metric object to print an identifier for each logged metric, this metrics must be separated by a field separator string which can be specified in the file `epim.cpp`. Usually `FilterAndRegister::delimit` is set to take a simple tabulator as separator.

PRINT_STAT: Evaluate collected information and print them. They must be separated by a delimiter as well. After printing, in most cases an object has to do a cleanup and may reset some counters.

WARM_UP_END: As soon as you intend to distribute your metric calculations to a certain number of calculation nodes, you often are forced to overlap the time intervals you evaluate. This raises the problem, that your metric object does not know, when such an overlap ends. So this trigger transmits the desired time stamp. The current implementation assumes, that the overlap interval is one file containing NetFlow records. In `main()` we call the function `gimme_first_timestamp_in_file()` to determine the time when the warm-up ends. In our case an input file contains the data for exactly one hour. This is usually a sufficiently long interval, since the theoretical maximal misalignment of records is $2 * 18$ minutes in unsorted flows.

COOL_DOWN_START: This trigger does the same as `WARM_UP_END` but provides the time when a new interval on another calculation node starts.

AS_TABLE_UPDATE: As already mentioned, `FilterAndRegister` prompts the utility class `ASTableManager` for AS information. As the AS-Tables change from time to time, they need to be updated. The trigger `AS_TABLE_UPDATE` informs `FilterAndRegister` when it is time to do such an update.

UNKNOWN: A trigger that should never be called. It is here to catch errors.

If your metric implementation wants to be informed about anything else, you can add your own trigger. Usually such an extension is not necessary and it is suggested to avoid this unless inevitable:

1. Go to `GlobalDefs.h` and add the name of your trigger to the `TriggerEvent` enumerator.
2. Start your trigger from the innermost loop in `main()` of `epim.cpp` according to your time desires.
3. In `FilterAndRegister.cpp` you need to adapt the method `FilterAndRegister::triggerEvent()`. Make sure that an appropriate message is shown in case your trigger will be blown. In most cases a trigger is only designated for metric objects, in this case you don't need do change here anything else. But if you intend to influence the management object `FilterAndRegister` you have to catch the trigger event here as it is done e.g. for `AS_TABLE_UPDATE`. If your trigger must be released again in the future, return the time in Unix milliseconds when this should happen so that it can be rescheduled.
4. Catch your trigger in your metric object by reimplementing in your class the virtual method `triggerEvent()` inherited from `MotherMetrics`. For an implementation example see `BlasterAnalyzer::triggerEvent()`. Please notice that a trigger is always sent to all metric objects. A metric object has to decide on its own whether a released trigger is useful or not. By default no triggers are caught.

4.1.4 Extraction of Host-Based Metrics

In order to understand the dynamics of worms we felt impelled to extract host based metrics which help to reconstruct the ongoing within the AS559. Since every shock has its own characteristics,

such an analysis needs to be adapted specifically to every event. A very profound knowledge of an event is inevitable and a meticulous study of all data exchanges via network needs to be done in advance. Otherwise its impossible to construct reliable recognition patterns on flow level which allow tracking an epidemics.
Fortunately in [21] a good documentation of the W32_Blaster event is available and [26] ran a copy of W32_Blaster in a test network. We implemented such a painstaking analysis tool for this event. The object `BlasterAnalyzer` keeps state of all steps this multi stage worm makes. Since many worms show a multistage characteristics from a network point of view, the available class contains all elements needed to analyze other similar events.
The fields we extracted are:

[ ] IP (hash table key, aka primary key, no field in array required)

[0] time(first unsuccessful stage A infection attempt)
Insert guaranteed only if occurred within the last hour before first scan sent.

[1] time(first seen successful infection attempt over boarder routers)
Successful: All infection steps (and thus the corresponding flows) are present.

[2] time(first scan sent)

[3] time(last scan sent)

[4] time(normal traffic sent)
We take the time stamp closest to [3], if possible [4] > [3], insert guaranteed only if occurred within the last hour after last scan sent.

[5] source IP of first successful attempt, same as attacker IP

[6] number of scans received (after successful infection)

[7] number of scans sent

[8] placement (0 = unknown, 1 = outside, 559 = switch)

[9] recovery time

As mentioned earlier the obtained values are strongly dependent what definitions one chooses. For example for field [1], the first seen successful infection has not to be the very first infection. Because the vulnerability still exists after an infection, a reinfection can occur although the second copy won't be started after download. E.g. when a host becomes infected within the AS559, we won't seen any part of this infection on the border gateway routers of the network. But if a second infection follows across the network borders, we will see and count it as the first *seen* infection. If desired, reinfections can be removed easily by comparing field [1] with [2]. Whenever the latter occurred earlier, it is a reinfection. We give the possibility to do this with the tool described in section 4.2.3. Blaster prevents multiple running instances using mutual exclusion.
Due computation limitations we had to limit the recording of field [0]. To calculate the field, we need to store all Blaster scans in a hash table, which is very memory consuming. Every quarter

hour we do a cleanup of the mentioned table, so in maximum we can look back up to 75min, in minimum we guarantee to store at least the last 60min.
Field [6] is strongly dependent on the overall duration of the input. I.e. distributing the calculation to several nodes will result in a different outcome. The reason for this is, that a host first has to turn out as infected before we can log any scans received. When we split up the observation period into smaller intervals, a machine may not show up in one of these periods because it is is switched off, or becomes recovered. The host then does not denounce itself as infected and thus, due the lack of knowledge what is going on in other intervals, we cannot log the scans received.
Assuming that a host becomes recovered whenever field [4] is greater than [3] allows us to estimate the recovery time [9]. This calculation, as well as a merge of multiple log files, is done in the script `hm_merge01.pl`. The field enumeration is consistent through all programmed application, scripts and analysis tools.
It's important to keep in mind that there exists an exact definition of all fields. All findings have to be put in question in order to find out whether they are artifacts of the used methodology.

4.1.5 Adding a new Metric Module

1. The best way to start with a new metric module is to copy an appropriate class mentioned above and adapt it to the new conditions. Your metric class must inherit and implement the interface defined by the class `MotherMetrics`. For an example with simple counters measuring a metric, have a look at class `InfectionAttempts` in `metrics.h` and `metrics.cpp` . `ScanningHosts` is an example when you intend to use `HashedTable` to store your data. Finally, `BlasterAnalyzer` shows how triggers and self made file output can be employed.

2. Go to `GlobalDefs.h` and specify the number of metrics object you would like to start (preprocessor definition `NUM_METRIC_MODULES_DEF`).

3. To raise the speed of execution go to `FilterAndRegister::match_prefilter()` and specify the traffic parameters all metric objects have in common.

4. Load your object in the constructor `FilterAndRegister::FilterAndRegister()`

In case you want to analyze another epidemic event, you may adapt the existing modules to your case. The code changes needed to observe Witty worm instead of Blaster is directly shown as an example in the code of the class `InfectionAttempts`. The example uses preprocessor directives in order to compile binaries for both events. The classes `ScanningHosts`, and `RelativeAnomalySize` described in section 4.1.2 are also programmed enough generally so that the same adaption should be sufficient for new epidemics:

```
inline bool InfectionAttempts::match_post_filter(
  const NetflowV9Record &r){

  #ifdef BLASTER_WORM  //Blaster 08.2003
```

```
    if( ((r.dOctets == 40) ||
         (r.dOctets == 44) ||
         (r.dOctets == 48)) &&
         (r.prot == TCP) && (r.dstPort == 135) ) {
      return true;
    }
    return false;

  #elif defined WITTY_WORM  //Witty, 03.2004
    if( (r.dOctets >= 796) && (r.dOctets <=1307 ) &&
      (r.prot == UDP) && (r.port == 4000) ){
      return true;
    }
    return false;

  #else
    return true;

  #endif
}
```

Enabling the preprocessor directive `BLASTER_WORM` in the file `GlobalDefs.h`, or `WITTY_WORM` respectively, automatically chooses the correct code segment during compile time.

4.1.6 Limitations

The tool `epim` has some limitations:

- The tool needs flows that are *sorted* according to their start time. Otherwise the output result will not be correct, since for new infections the chronological order of all involved flows is checked. This should not be a considerable drawback any more, because the framework from the Data Mole Project [43] is able to provide the flows in sorted order.

- `Epim` is still a standalone application. For short-term analysis this is sufficient, but for long time segments, let's say 4 weeks and more, an integration of `epim` into the Data Mole Project should be considered. In the design phase of `epim` an eventual integration was bore in mind so that one should not face major difficulties doing it. To simplify the usage of the standalone application when using large datasets, there exists a script named `epim01.pl` which allows to start in parallel multiple instances of `epim`. on a cluster.

- In order to simplify the implementation of new metric modules, the managing class `Filter-AndRegister` determines the source and destination autonomous system of every flow that passes the prefilter (`FilterAndRegister::match_prefilter()`). This is also

done even when the AS information is not necessary. If speed really matters, one should consider switching off the AS determination. The speedup would be up to 15%.

- At the moment, we do *NOT* concatenate flows, which were split by the routers export engine (see Chapter 2.3), because this is connected with a great effort. Previous work has shown that the error should be below 5% ([23]). Packet fragmentation is also only considered where it has a critical impact on the end result.

- Epidemic shocks like W32_Blaster generate a large flood of small flows. We recognized that this massive amount of flow traffic leads to losses in the Cisco record export engine. This makes it difficult to capture all flows involved during a multistage infection. For our case of W32_Blaster we developed a heuristics in order to recognize complete infections over all stages even when some packets get lost. Our constraint is that only one flow per bidirectional connection needs to be present. This results in approximately 20% more detected complete multistage infections.

4.2 Scripts

4.2.1 epim01.pl

Analyzing an anomaly typically involves several hundred data files, a grid of calculation nodes and half a terabyte of disk space. The effort to manage such large calculations cannot be handled manually. Especially when the workload becomes distributed on a cluster. In the header section of `epim01.pl` the path to input files, number of involved nodes, output location, start and end date, etc. can be specified. Running the script first collect all necessary informations, distributes the workload equally to all nodes and generates for each job a shell script using Bourne-family syntax. If desired the jobs can directly become initiated on the remote nodes.

4.2.2 graph_epim01.pl

`graph_epim01.pl` graphically displays a tab-spaced ASCII files containing an array of time series as `epim` provides them. For convenient usage the tool has a command line interface where all currently available features can be enabled or disabled. Typing the command „`./graph_epim01.pl -h`“ in a shell list all the options.
Beneath logarithmic plots, start date, end date you have the option to group series and plot them together on the same coordinate system. To use this feature the column labels need to be indistinguishable until the first underline. Based on pattern recognition the series become grouped. Thus, the first row of the input file must contain a text string for each column, which will be the label for the corresponding series. In addition the first column is reservated for the time values. The preferred time format for dates is `YYY.MM.DD:hhmm`. Optionally you can also use plain digits, but then the time column header must be named `utime`.

The tool manufactures a gnuplot order sequence which will produce pictures in `eps` and `ps` format. You will also obtain a postscript file containing an overview of all generated plots, convenient for printing.

4.2.3 hm_merge01.pl

As described in Section 4.1.2, the application `epim` supports to run in parallel multiple instances to distribute the workload. Each entity of the module `BlasterAnalyzer` will then produce its own output file. These host based metrics, collected for each period, thus need to be merged together afterwards, what will be handled by `hm_merge.pl`. After the assembly the data fields can be further processed if desired. For instance we correct fields which become invalid by merging. In addition we append the field „time of guessed recovery“. Beneath several counters about occurred inaccuracies, a table of interesting statistics will be displayed on `STDERR`. This information allows to estimate the imprecision caused by the deployed data processing chain. As output one obtains a file called `hm.txt` containing the assembled IP centric data fields. Alternatively, the `STDOUT` can be used, without messing up with the statistics.

4.2.4 eps_bbox.py

Many graphic applications, such as the library `matplotlib` in python, are unable to specify the `eps` picture frame correctly. As result embedding in documents and printing becomes difficult, since the picture may be cut in the midst of the visible area. This unplanned by-product corrects the so-called `BoundingBox` to the from ghostview detected edges of a eps picture.

4.2.5 SIR Model Simulations

In Section 5.6 we present an enhanced SIR model taking into account daily seasonality. We ran our simulations in the mathematical environment `Maple`. The files `sir02.mw` – `sir04.mw` contain various simulation runs and auxiliary calculations. The model from Section 5.6 is in file `sir04.mw`.

4.2.6 Further Tools

As usual one builds many helper tools in order to face the problems of daily life:

ren_glue01.pl: Rename a large amount of files.

see_running_procs_on_nodes.sh: Show running applications on all cluster nodes.

do_on_all_nodes.sh: Run a command on all node clients.

kill_on_allnodes.sh: Terminate multiple instances of a process specified by a text string.

4.3 Decay Analysis with Python/SciPy

4.3.1 param_est.py

This tool has been developed to estimate the parameters of a decay. We use the matlab-like python libraries for the numeric calculations. `Numpy` is mainly used for the matrix and array manipulations whereas `pylab` contains plotting capabilities via `matplotlib`. Typically one does specify a tab-spaced ASCII input file with data points on command line: `./param_est.py -f input.txt`. The file structure was already introduced in Section 4.2.2. All other settings are made directly in the source code, since the command line has to many restrictions in order to define a handy user interface. The kernel of mathematical calculations can be found in the following three functions:

get_parameters_power(...): This function performs the deterministic least square fit as derived in Section 5.7. As input an array of time stamps and the corresponding data values have to be given. By specifying the minimal (`rel_shift_min_sec`) and maximal (`rel_shift_max_sec`) range where the power law singularity t_c can be placed, the function chooses the best fit found in `shift_num` estimations. One can select the exponential estimation instead of the default power law fit by setting `ctype` to „`pow`“. One small trick worthy to mention is the singularity shifting. Actually we do not shift t_c but the time array in the opposite direction in order to keep t_c always at time 0.

expanding_window(...): Typically we do not only search for the optimal singularity, but also determine which part of the decay leads to a best fit. `Expanding_window` makes an estimation from the maximum to a specified end value (`xwend1_sec`) which will be incremented in `xwsteps_num` steps until (`xwend2_sec`). Setting `xwend1_sec` or `xwend2_sec` to 0 will set them to the array end time. We strongly recommend always to use `expanding_window` instead of directly involving `get_parameters_power(...)`, since this function also cleans up invalid values and chops the array to remove points before the maximum.

shrinking_window(...) actually does the same as `expanding_window` but uses the decay tail instead of the beginning for the estimation. So the calculation is done from an incrementing value until the end of the array.

Currently we implemented four major tests for our research field. To use them one has to enable them in the `main()` function:

fit_to_noisy_curve(): This function generates an artificial decay by adding Gaussian noise to a deterministic curve. Hence we can test the functionality of `get_parameters_power(...)`.

sampling_analysis(...) allows to regularly sample a decay in order to find likelihood estimations for such sampled curves. Switch to Section 5.7 for examples and results.

decay_analysis(...) is here for the case you want to apply a fit to a simple decay. It provides no bits and pieces and just executes the fit for the complete array of data points.

decouple_analysis(...) Our decays usually where overlayed by seasonality. In this function we analyze the approach of decoupling a decay into a multiplication of a power law and an oscillation term. In addition we try to forecast values in the future to check the reasonability of the estimation.

tilt_analysis(...) We wanted to know the exactness of our estimations. Therefore we measured the tilt (slope) of the seasonality extracted from the power law decay signal.

removed_season_analysis(...) By generating a reference pattern for the daily/weekly oscillation, we where able to remove the oscillation component in our decay signal. This method searches for an optimal stretch factor and DC-offset for the reference pattern in order to erase the oscillation from the power law.

Beneath the described function you will find a couple of small helpers, for instance to detect weekend values or apply an FFT. In addition we make use of the services provided by `supporters.py`. Worth to mention is only the method `least_squares_fitting_simulate(...)`. It utilizes a simulator from the package `scipy.optimize` to estimate parameters of a arbitrary parametric function, also when it is not a power law or an exponential.

4.3.2 host_metrics.py

Most colorized plots in this document are generated by the script `host_metrics.py`. Just like in `param_est.py` one specifies an input file and gets as output dozens of graphs for an Internet epidemics. After all pictures become generated in the `eps` and `png` format, it also produces an overview as `pdf`. Furthermore we call `eps_bbox.py` to correct the BoundingBox which the `matplotlib` does not choose well. While for most plots the same settings need to be made over and over again we developed a general method for plotting, called `general_plot(...)`. One can specify as input one data array or a tuple of them. If the abscissa is omitted the function generates one for you, if only one is specified it is used for all data arrays. It is possible to choose between barplots, step function plots, lines and dots. Setting the `time`-flag to `True` converts the abscissa values, more precisely seconds since epoch, in date values using an adequate axis labeling. As a matter of course, linear, semi logarithmic, and double logarithmic axis are possible, as well as legends, axis labels and a specification for the picture file name. Unfortunately the built-in automatic axis adjustment routine does not work properly, so we avoid this option and implemented a more robust window parameter determination. All those options can be used by just typing in one single line, the function call with the desired options chosen. Check out the examples in `supporters.py`.

Chapter 5

Results

First we determine the fraction of infected hosts compared to the total number of hosts in Section 5.1, before we analyze the infection process in a general approach (Sec. 5.2). Afterwards we look more detailed at the time series and the influences of Nachi (Sec. 5.3.1) Section 5.4 depicts different aspects of the spreading and recovery dynamics. From the long time behavior (Sec. 5.5), but also from the conclusions of all other sections we build at the end an extended and more appropriate SIR model (Sec. 5.6). From this enhanced model we move on and decompose the time series decays with two different approaches in order to find a more accurate explanation for the observed long term life cycle (Sec. 5.7).

5.1 Degree of infection within AS559

In order to see the dimensions of the Blaster anomaly compared to the total network size we performed several measurements. We analyzed the historical routing tables in order to see the allocated IPs in AS559. This is the maximum number of hosts which theoretically could be addressed directly from Internet without special tricks like NAT. Beneath dozens of small network segments the AS559 mainly contained two „/15“[1] and 28 „/16“ segments in August 2003. Due to the sufficient pool of unused IPs the number has not grown much over the years (see Fig. 5.1).
Figure 5.2 compares the number of all flow scans to the number of flows from normal traffic. The sum of both is also shown. After the outbreak of Blaster approximately more than half of all connections are caused by Blaster. This number seems to be huge. However, the number of total packets sent only increases around 10% as shown in [8], indicating that this additional flows contain only few packets. And the additional bandwidth consumption is negligible [23]. We also apportioned the flows according to their origin (not shown). And we find that the ratio (Blaster flows / normal flows) does not show a significant discrepancy between flows leaving AS559 and coming from outside.
In contrast to the flows we see a much greater difference when we consider the unique source IPs (Figure 5.3). We aggregated all active sources within a 15min interval and distinguished

[1] „/15“ stands for a network with an address range of 2^{32-15} IPs

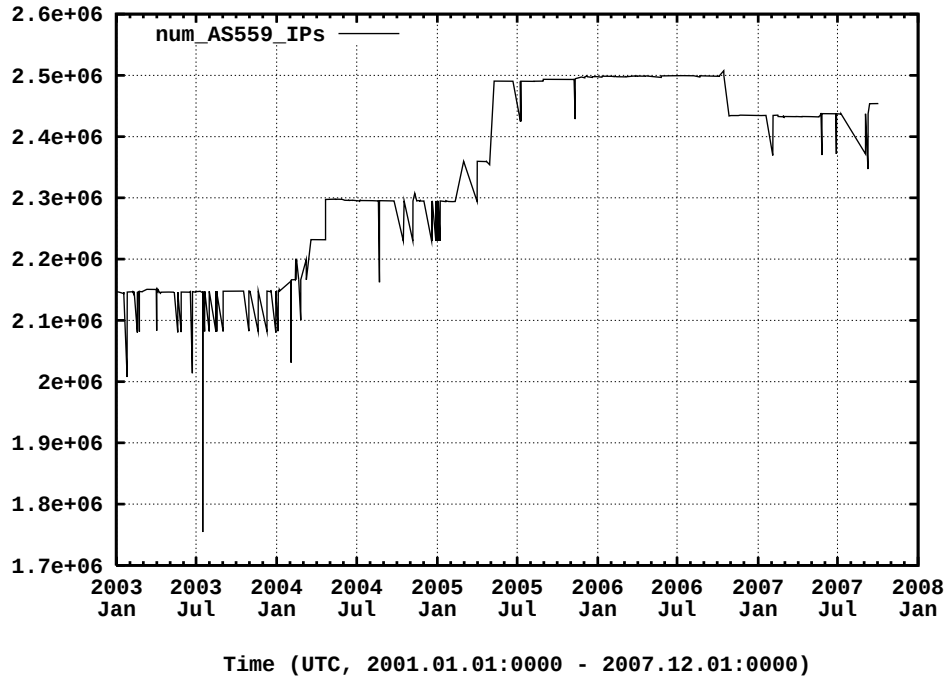


Figure 5.1: Assigned number of IPs for AS559 versus time. The numbers where obtained by evaluating the historical routing table updates and represent the maximum number of hosts that theoretically can be assigned to an AS559.

between Blaster infected and clean hosts. The active infected population seems to be quite small compared to total number of hosts (logarithmic ordinate). After outbreak we have approximately 1000 contaminated hosts versus half a million clean hosts. The curve of the total active unique source IPs covers the curve of normal traffic since the difference is tiny.

As already guessable when we compare Figure 5.2 and 5.3, the ratio of Blaster flows to the total number of flows is much higher than the proportion of (infected hosts)/(all active unique source IPs) (see Figure 5.4). According to our measurements Blaster was not able to capture more than 0.4% of all active hosts in its initial phase. When we just consider unique sources placed within the AS559 as shown in Figure 5.5 we obtain a similar result. Having around 100’000 active unique source IPs within AS559 leads us to a maximal infection population of 1%. Within two weeks the rate shrinks to 0.1%.

Although we can expect a certain concentration of potential susceptible hosts within segments of AS559, for example when an institute has a preference for Windows computers, we can assume that the overall distribution of infected hosts is rather homogeneous in the network.

Knowing that only a very small fraction is infected and knowing the Blaster spreading mechanisms, we clearly can’t assume that the infected host population is a physically continuous region as often assumed in mathematical models. Blaster, as well as all exploit-based worms, operate on a logical layer, the IP layer, and take the underlying physical structure only little (or even not at all) into account. Particularly the often assumed scale-free network structure is not „visi-

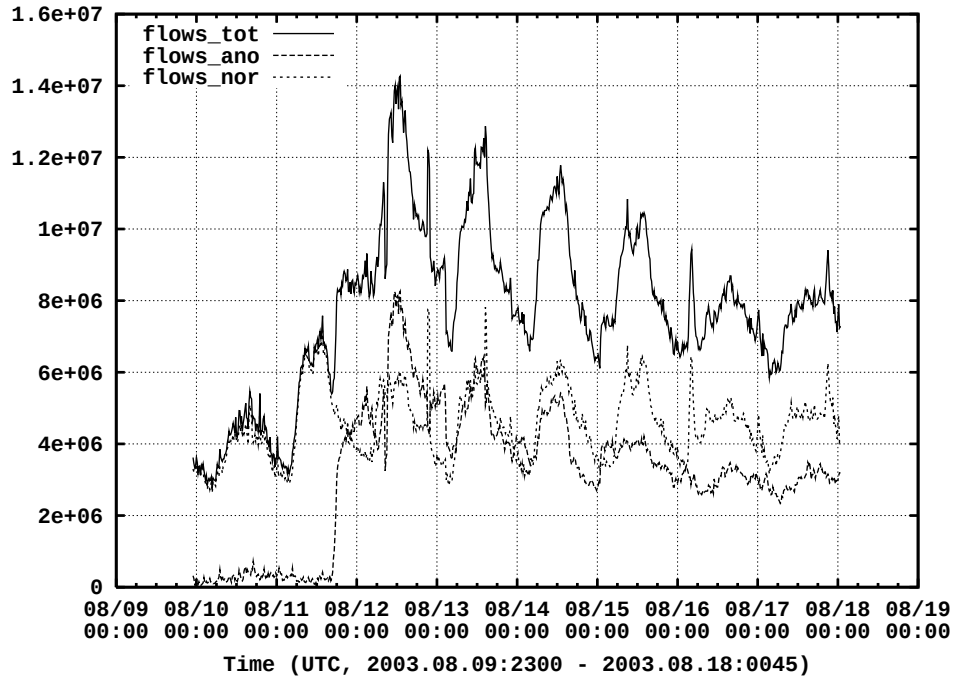


Figure 5.2: Number of total flows (solid line) compared to flows rooting from Blaster scans (dashed line), time series with 15min bins

ble“ on IP layer. Logically every host in the Internet is directly linked with every other host, consequently the topology looks like a complete fully connected graph. This clearly differs exploit-based cyber-worms from the spreading mechanism of human viruses. Applying the model of sexually transmitted diseases (STD) by human contacts, as proposed in [6], may be feasible for other worm categories where only few connections to neighbors are available, but not for exploit-based worms.

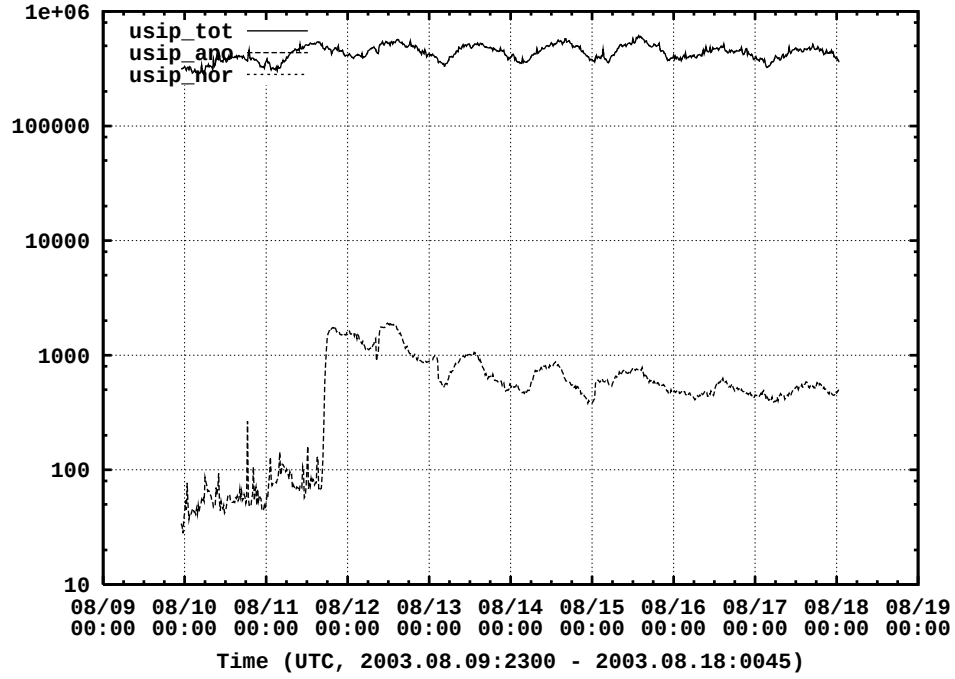


Figure 5.3: Total active unique source IPs (solid line) and the total active unique source IPs of non-infectious traffic (dashed line), time series with 15min bins

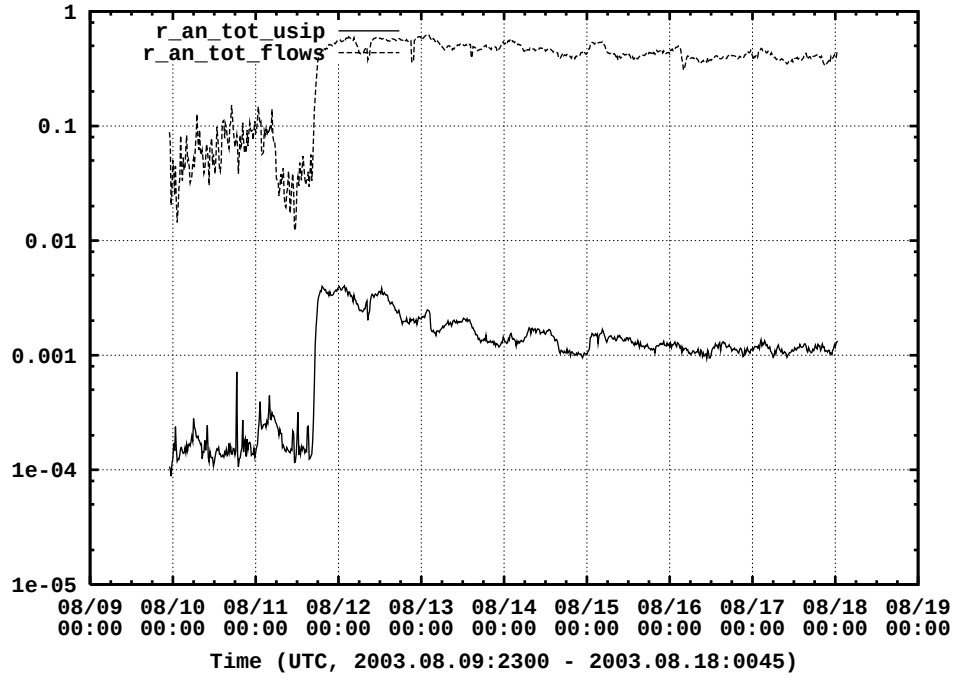


Figure 5.4: Dashed line: Fraction of infected flows rooting from infection vs. total flows per interval (blaster scan flows / total flows). Solid line: ratio of (unique source IPs of infected hosts) / (total unique source IPs), time series with 15min bins

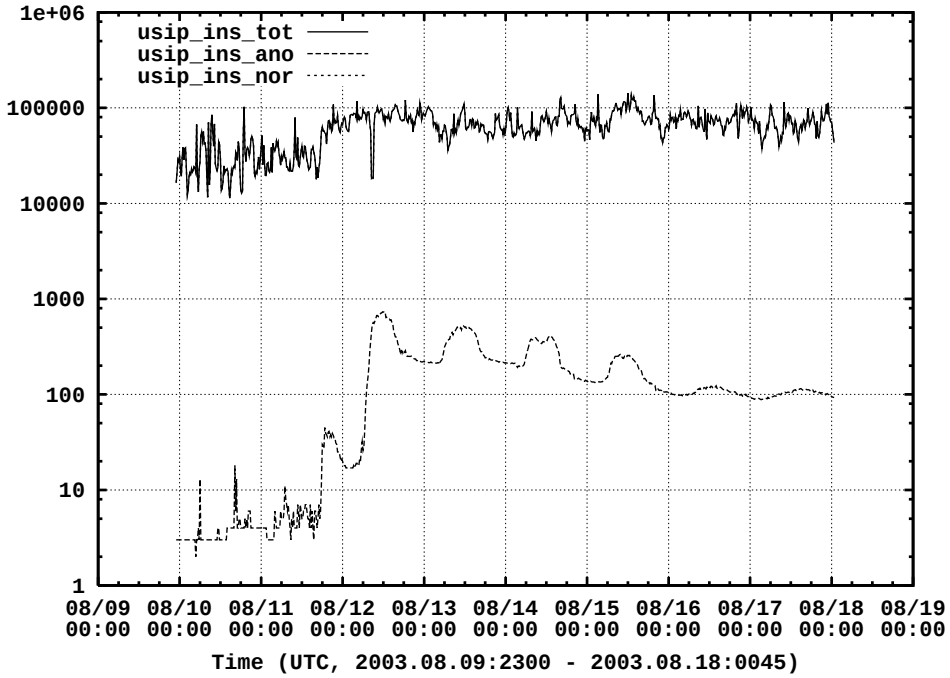


Figure 5.5: Active unique source IPs placed in AS559. Dashed line: unique source IPs sending the Blaster scan signature, solid line: unique source IPs sending non-infectious traffic. Time series with 15min bins

5.2 Analysis of Infection Process

We alluded already in Chapter 3 that beneath time series we intended to account IP centric metrics in order to track single hosts over time. Using the IP as primary key has one drawback. IPs do not need to be static over time. Every host with a non-static IP might get a new one after reboot, or sometimes even when the network gets disconnected for a while. This can lead to two kind of relation errors. A machine obtains a new IP via DHCP that was in use already before. In this case the behavior parameters of two hosts become merged into one tuple. Secondly, a machine may changes its IP to a new one that was not assigned before. This leads to two entities instead of one in our relation. According to [33] the latter case yields an overestimation of an anomaly. Especially when the observed time segment is longer than 24 hours we expect that many machines, in particular office computers, pass an on-off cycle.
Nevertheless, we assume the errors due DHCP-effect is moderate for the SWITCH network for two reasons: Unlike end-customer ISPs with many short-lived dial-up and DSL connections, SWITCH is an academic network with much more static properties. Some universities in AS559 allow direct dial-up connections, but the percentage of such connections is insignificant. In addition it is quite common within SWITCH to assign IPs according to the unique network adapter number, called MAC-address. Consequently a host gets the same IP over and over even after a reboot. For the purpose of simplicity, we consider the AS559 de facto as static and thus use the name host as synonym for IP. As a first approximation this is reasonable.

field no	description	total	inside AS559	outside AS559
0	ti_unsucc_ia	3175	2838	337
1	ti_succ_ia	761	81	680
2	ti_first_scan	144354	3936	140418
3	ti_last_scan	144354	3936	140418
4	ti_normal_tra	30815	3422	27393
5	sIP_succ_ia	761	81	680
6	scans_recv	23680	3955	19725
7	scans_sent	144354	3936	140418
8	AS	145053	3956	141097
9	ti_recovery	7035	2100	4935

Table 5.1: Statistics of Blaster analysis, period from 2003-08-10 00:00 to 2003-08-17 23:59

Table 5.1 lists the number of valid entries for every monitored event recorded in the week after outbreak of Blaster. Splitting up the statistics for the host inside and outside AS559 gives us already precious information about Blaster's spreading dynamics. Within AS559 we observed 3956 infected hosts, but only 81 of them where detected by the random scan mechanism and infected afterwards over the boarder gateway routers. In addition 20 copies never sending a scan (difference of field 8 and 7) belong to the population of the 81 external infections and thus where not working. Consequently the vast majority, more than 98 percent, where infected by the *local*

scanning mechanism.

The chronological distribution (Fig. 5.6) of these 81 infections over the border gateway routers even indicates that only 17 machines where responsible for the initial internal outbreak during the first 12 hours. The implications of this result are dramatic for the security industry. Filtering out an anomaly on backbone level, as tried by SWITCH (see Section 5.3), may damp the spreading. But, missing a single copy of a worm is usually sufficient to make the filtering ineffective.

Another valuable information for modeling the spread of epidemics is the fact that 3955 internal hosts, all but one, have received further infection attempts after their infection. Unlike typically assumed in S-I-R models, this is a strong indication that the complete vulnerable host population becomes infected and not just a small fraction of it. Furthermore this would imply that the spreading rate must be very large, in a first approximation infinite, in order to reach a high degree of prevalence. A high spreading rate in turn would mean an infection occurs almost instantaneously after switching on a machine. Therefore we analyze in Section 5.4 the probability against time of getting infected. Receiving such a large amount of infection attempts is in addition an indication that we reach a saturation due network congestion at some point.

We counted also the unsuccessful infection attempts (field 0) with the intention to relate them to the successful infection. Due memory limitations during the extraction process, an unsuccessful infection becomes only observed if it occurs not earlier than 1 hour before an infection. There are several reasons for an unsuccessful infection. A host may be turned off, hit by the wrong exploit (which is OS dependent), it may be not vulnerable at the arrival time of an infection because of a temporarily enabled defense mechanism, or the target host is not existent at all. Our tool does not consider the latter case, but the other three will be reported. What we can already say from the statistics in Table 5.1 that more than 70 percent of all AS559 hosts experienced an unsuccessful infection attempt (Fig. D.1a). According to our analysis the main reason for an attempt failure is due to unreachable hosts followed by the usage of the wrong exploit.

The good thing of autonomous systems, such as SWITCH, are the well known boundaries, as well as the knowledge of the main physical network structure (called backbone). In network research, we often face the problem that only few is known about the edges between nodes. For example in social networks or sexual networks such containments are difficult and the underlying structure often unknown. But, one has to be very careful with statements about the external hosts since we observe the remaining Internet only partially. Typically an external host posts just few scans by randomly choosing the address range of SWITCH and disappears again on our monitor forever. This is why we have a large discrepancy between the fields observing scanning (i.e. field 2,3,7) and the remaining events. By contrast, the internal machines where observed much more detailed because they are forced to use the border routers whenever a contact to the Internet is necessary.

At least we can estimate roughly the dimension of worldwide infected machines. Knowing that 2.2 million IPs are assigned to AS559 gives a probability of $2.2 * 10^6/2^{32} = 0.051\%$ that a random scan hits the SWITCH address space. Or otherwise stated, we monitor 1 in every 2000 worldwide scans. When we assume that the majority of contaminated hosts sends much more than 2000 scans, as observed for most machines inside, the 140'000 machines counted during first week represent roughly the total number of infections. This is one order of magnitude fewer than

Mic

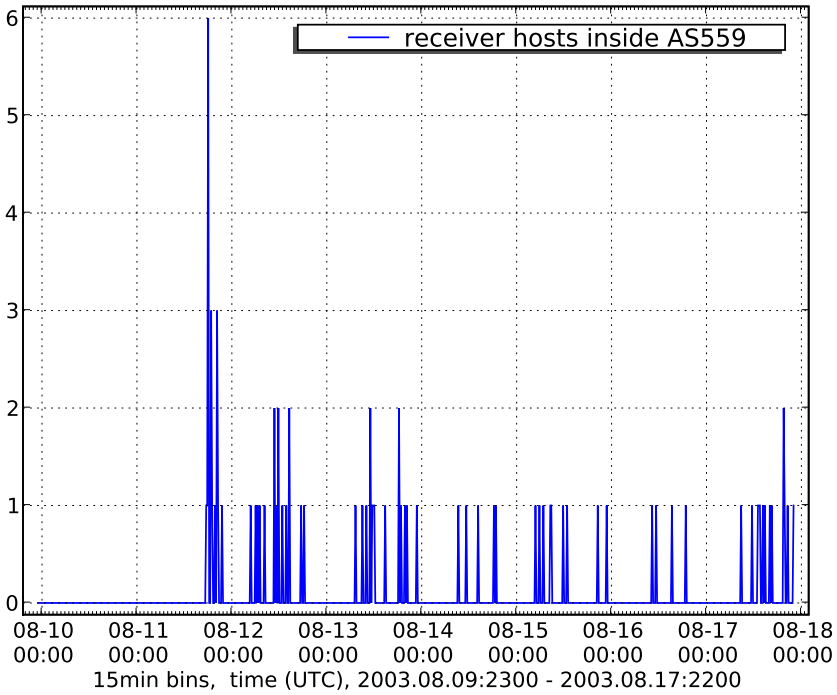


Figure 5.6: Successful multistage infections over boarder routers from sources outside to targets in AS559

5.3 Discussion of time series

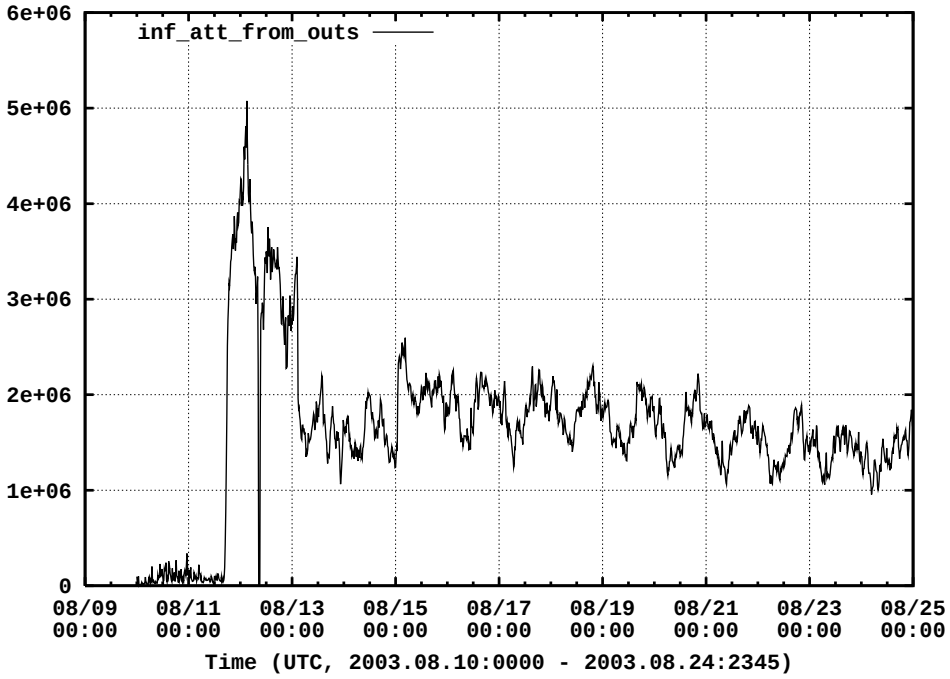


Figure 5.7: Number of scans to AS559 aggregated in 15min bins (The number of TCP SYNs sent to destination port 135 aggregated in 15min intervals. Only scans from Internet to AS559 are shown)

Initially we generated rudimentary time series to get familiar with the Blaster epidemics. We filtered out all scan traffic in the border gateway logs of AS559 using the following heuristics: The destination port is 135, packets per flow = 1, bytes per flow = 40, 44, or 48 bytes, protocol = TCP. Since Blaster always scans in series of 20 we used this also as criterion for our filter. The availability of the time dependent autonomous system information allowed us the distinction between traffic coming from SWITCH network, mainly composed of AS559, and the remaining Internet. We determined for each source IP whether it can be located in AS559 or not. Within intervals of 15 minutes we counted all scans crossing the border gateway routers and plotted them versus time (Fig. 5.7 and 5.8). Be aware, that we don't see the 40 percent traffic generated from the local subnet scanning, thus only the 60 percent pure random scans are seen.
Interesting is the large drop of scans in the morning of August 12 (Fig. 5.7). The maintainer of SWITCH set up a filter for Blaster scans on backbone level hoping to confine the spreading. Knowing that the internal spreading continued without serious reductions proofs the ineffectiveness of such countermeasures. The AS559 encompasses only a geographically small area and is located in just one time zone (UTC+1). This becomes manifested in the clear day/night rhythm (Fig. 5.8) whereas for the remaining Internet(Fig. 5.7) the traffic suffers from averaging over many time zones. In addition, the same holds true for the weekend with clearly visible reduced traffic on Saturday and Sunday for AS559. The user behavior, especially the working time, has

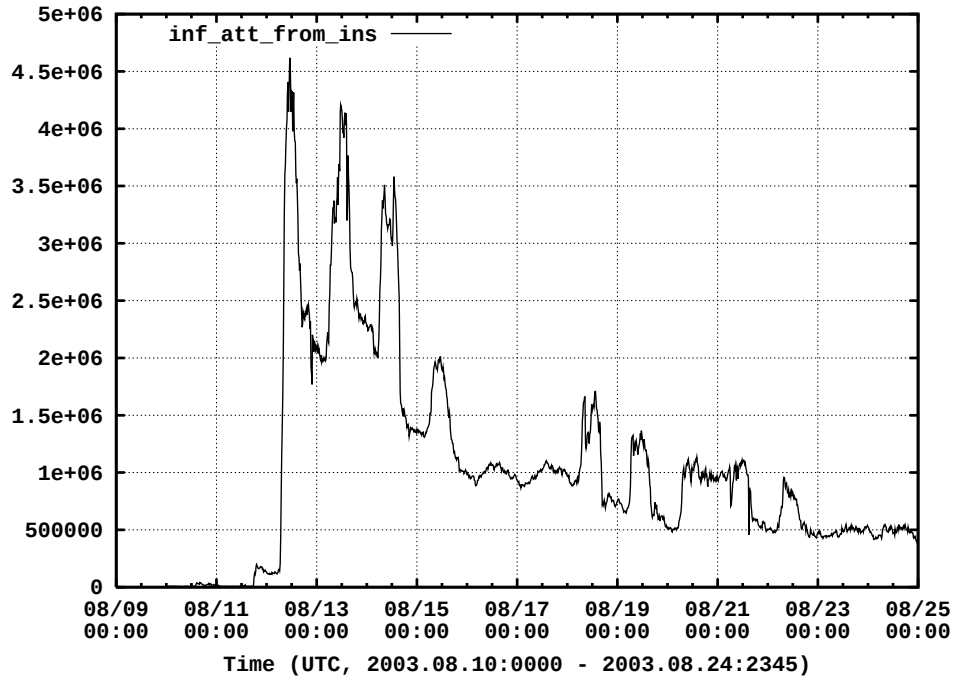


Figure 5.8: Number of scans from AS559 aggregated in 15min bins (The number of TCP SYNs sent to destination port 135 aggregated in 15min intervals. Only scans from AS559 to Internet are shown)

a great influence to the spreading dynamics of worms. Without susceptible targets no spreading can occur and this gives us an explanation why the internal outbreak happened with a delay. Figure 5.9 states the main outbreak on August 11 around 4pm, whereas the number of internally infected hosts stayed on a moderate level at this time until the next morning (Fig. 5.10). It is not surprising to see the number of successful infections exploding right after the start of work at 7 am local time.

5.3.1 Nachi - Hunting Blaster for a Good Aim?

Figure 5.9 and 5.10 display the unique IPs which generated the scans displayed in the previous two plots. We could characterize this time series also as the „instantaneous number of infected hosts“ in case the probability of generating scans within every 15 min interval is fairly high. While most reports focus on the decay during the first week after outbreak, we discovered an increase of externally scanning hosts (Fig. 5.9) after August 19. As we will see in Section 5.4, it is very unlikely that the spreading experiences a revival. So why does the number of externally scanning host rise again? A search in virus databases disclosed that another worm is the reason for it. On August 18th, the first copycat of Blaster was released. Nachi, often also referred to as Welchia.a, exhibited two interesting attributes. First, it searched locally for Blaster installations, ended the process and removed it. In addition it downloaded and installed several patches from the official Microsoft update center in order to close the DCOM RPC vulnerability on port 135. So, why did

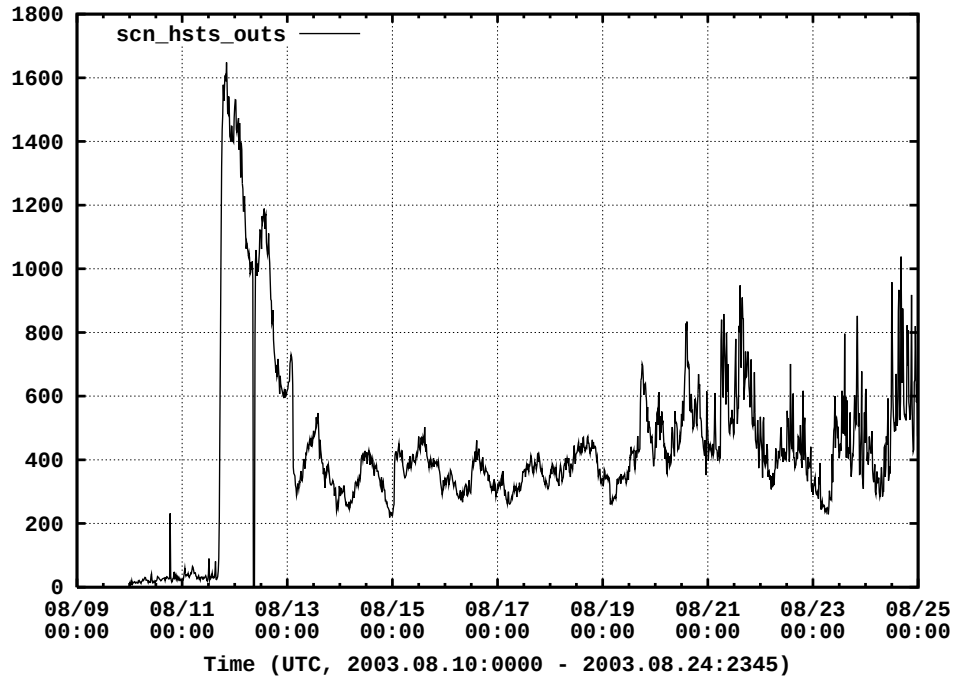


Figure 5.9: Unique source IPs sending scans from outside to AS559 aggregated in 15min bins (Unique source IPs generating TCP SYN scans to destination port 135 aggregated in 15min intervals. Only hosts outside AS559 are counted when they send more than 20 scans.)

Nachi such good things? Was it programmed by good guys, accomplishing a good mission? We believe the reason for this nice side of this worm was in fact a clever economic maneuver. Since vulnerable machines are a limited resource and sharing a host with other malware is not desired, the patches where applied to keep the exclusive sovereignty.

The second interesting property of Nachi was the usage of two exploits. It increased the probability to infect a victim using a second exploit for the WebDav vulnerability in Microsoft IIS 5.0 described in Microsoft Security Bulletin MS03-007. One may ask, why the number of scans did not rise meaningfully (see Fig. 5.7). Knowing the exact scan behavior gives us the answer. New victims become discovered with ICMP pings instead of using TCP-SYNs to port 135. If they respond Nachi first tries the WebDav exploit on port 80 and only when the attempt fails a single TCP SYN to port 135 is sent. Our filter for Blaster just catches up the latter event. Most activity of Nachi is thus not observed. Nachi was programmed to self-destruct when the system date changes from year 2003 to 2004.

Let's bring to mind again, the scans seen on border gateway routers are all purely random. Consequently it is no miracle that the number of fabricated scans within AS559 (Fig. 5.8) is a scaled version of the hosts generating them (Fig. 5.10), since all infected hosts send in average a constant number of scans per time unit. An exception is of course the first peak after outbreak in Figure 5.8, where we reached a saturation limit and the proportionality was not given any more. The scaling property does not hold for true for the hosts scanning from outside, because the

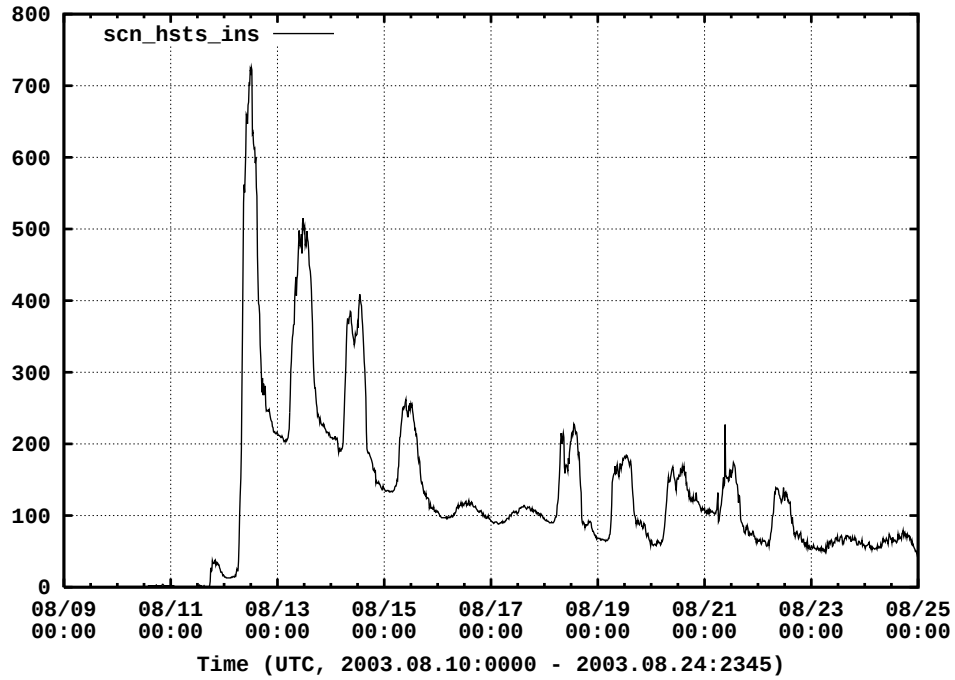
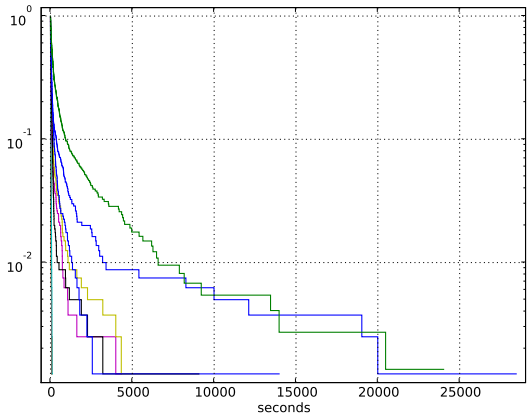


Figure 5.10: Unique source IPs sending scans from AS559 to the Internet aggregated in 15min bins (Unique source IPs generating TCP SYN scans to destination port 135 aggregated in 15min intervals. Only hosts placed inside AS559 are counted when they send more than 20 scans.)

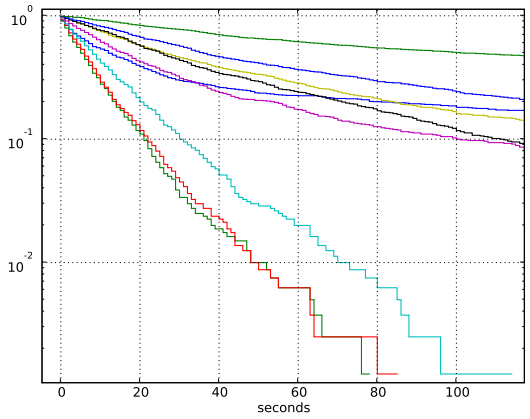
probability of striking the small IP range of AS559 at a constant rate is unlikely on the time scale we're considering.

More time series, also in other resolutions and longer time frames, about the event discussed so far can be found in the appendix B. A much more fine granular decomposition of Blaster and the arising consequences for the security industry can be found in the PhD thesis of Dübendorfer [21].

5.4 Host Based Metrics



(a) CCDF of time between two infections



(b) Detail view of Fig. 5.11a (zoomed to the first 100 seconds)

Figure 5.11: CCDF of waiting time between two infections (first scans), lin(t) log(y), 800 chronological consecutive hosts per curve, 400 points overlapping windows, only hosts within AS559 are counted.

In Section 5.2 we have interpreted the statistics about the IP centric fields we monitor. While we barely scratched there the surface of Blaster's spreading mechanism, we will now crawl into the data and graphically present different aspects of the infection process. Exploiting this reach source will give us a insight about the dynamics of Blaster as never before. We restrict our discussion to the period until August 18 in oder to avoid unnecessary complexity coming up with Nachi's birth. Furthermore we set the focus on AS559, since we have here knowledge about the complete boundary traffic in contrast to the all other parts of the Internet. And it is difficult to make valid statements for the latter when too many parameters remain unknown.

First discovery is that a host gets infected almost instantaneous as soon as it becomes connected to the Internet. But let us explain this step by step. We analyzed the distribution of waiting times between two infections, in order to understand the propagation process. Therefore we sorted chronologically the time of all infections within AS559 and calculated the difference time between two consecutive infections in a resolution of seconds. As in Figure 5.11a seen, we evaluated then the complementary cumulative distribution function (CCDF) of 800 successive waiting points. We kept this window size constant, but shifted it over the waiting time sequence in steps of 400 points. Using such windows helps us to see the timely evolution of the process. It starts with the mountainside-shaped green curve that becomes more and more flat In the detail view (Fig. 5.11b) we see that at the time of maximal prevalence the curves get a straight line shape (red, light blue, and green lines). Afterwards the curves transform back to the an mountainside-shape, ending with the blue curve in Fig. 5.11a. In Appendix D the graphs are also displayed in linear and double logarithmic scale (Fig. D.9). We obtained a similar behavior for the „waiting time between two last scans“ (Fig. D.10). Beneath the pictured window size 800/400, we played

around with other value combinations. Independent of the chosen window sizes the qualitative characteristics does not alter. The CCDF's for the complete window can also be found in Appendix D.

Our conclusion is before the outbreak we find a log-normal like behavior that fades into a Poisson process (almost straight line) and evolves back to the shape before the outbreak. Both distributions have a finite moment, and the Poisson process signifies the spreading is completely random at the point of culmination. This implies that the probability of getting instantaneously infected is very high. We believe that this behavior is a showcase for many further Internet epidemics, which has far reaching consequences, especially for modeling the dynamics of worms. For instance, in the SIR-model introduced in Section2.1 this would imply to assign very high values to the spreading rate β. Furthermore it is consistent with the long observed lifetime of viruses and worms (for Blaster refer to Section5.5).

Furthermore, we believe there are only few transitions from susceptible directly to recovered, also known as patching, will occur. This is due the short reaction time that remains after the boot process finishes. Therefore, an user must be aware of the worm in advance, boot a machine without Internet connection and patch it off line in order not to become infected and skip this state

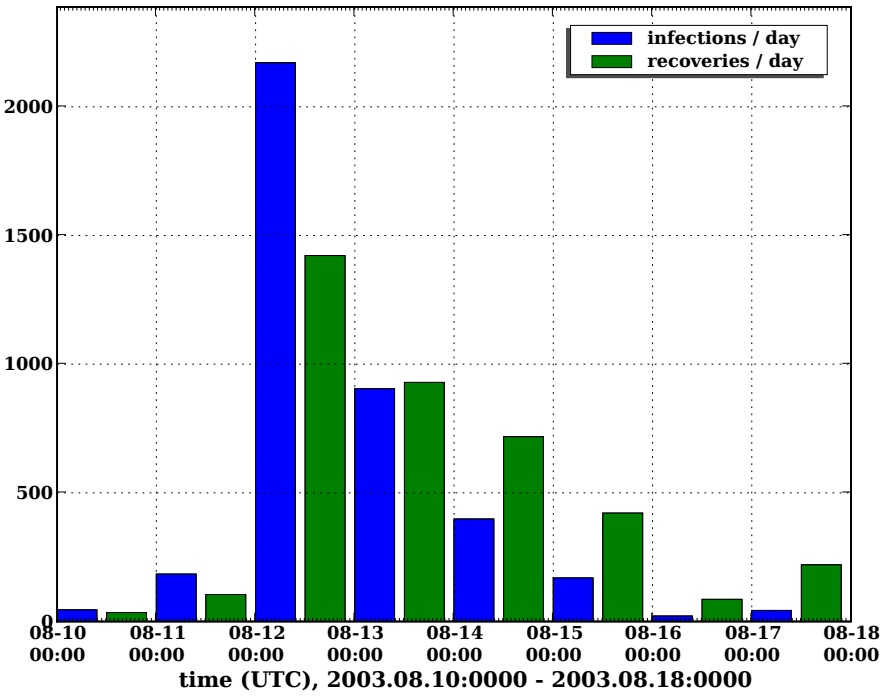


Figure 5.12: Infections / day, recoveries / day for Blaster. Only hosts (unique source IPs) within AS559 are considered.

Another consequence is that the spreading is a matter of when people turn on their machine. The new infections in Figure 5.12 on day 2, day 3, and so on, are thus fresh machines never turned on since outbreak. Roughly half of the vulnerable population, which is 3956, in AS559 gets online

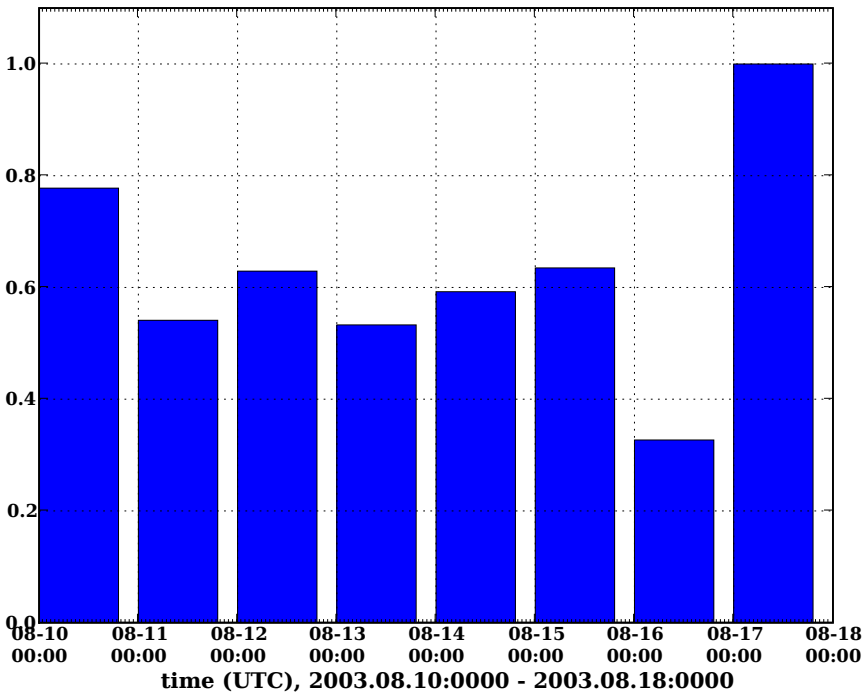


Figure 5.13: recovery ratio = (#last scans (recoveries)/day) / (#infections / day + #unrecovered hosts from yesterday). Only hosts (unique source IPs) within AS559 are considered.

and immediately infected on the first day, a quarter on the second day, an eighth on the third day, et cetera. On the same plot, one can also see the number of recoveries per day. From this two measurements we also determine the fraction of hosts that become recovered, more precisely that stop scanning, from the total infections counted on a day. In such a calculation one has also to consider the number of unrecovered hosts from yesterday, since they still belong to the infected population. What we calculated in Figure 5.13 is a kind of recovery ratio:

$$daily\ recovery\ ratio = \frac{\frac{last\ scans}{day}}{\frac{infections}{day} + unrecovered\ hosts\ from\ yesterday} \in [0..1] \tag{5.1}$$

The interesting point is that this ratio seems to be an uniform factor, at least for the first week. Even before the internal outbreak, which is on August 12, as well as afterwards the ratio is around 0.6, meaning that the recovery process of people is approximately proportional to the number of infections. One exception is the weekend where we observe a decreased recovery activity (August, 16). The last bar is at one because all hosts, even if unpatched, exhibit a last scan somewhere close to the end of an observation period. Thus all remaining unrecovered hosts pretend to become recovered on the last viewed day advancing the ratio to 1. In contrast to our finding, [60] modeled an accelerating recovery process for CodeRed because the awareness of people increases over time. Although this argument seems plausible, we cannot approve them in the case of Blaster.

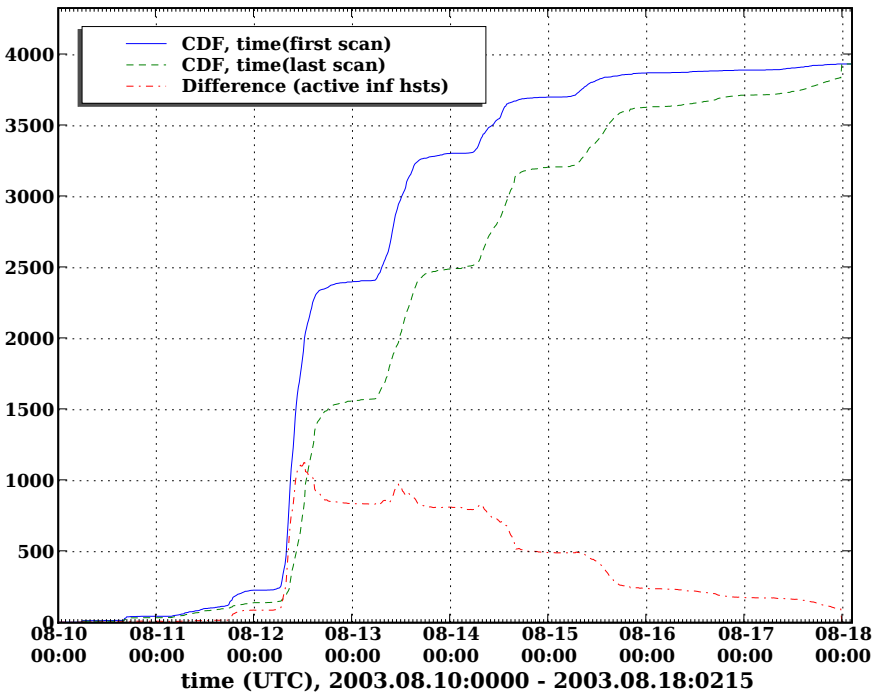


Figure 5.14: Unnormalized CDF for hosts getting infected (solid line) and stop scanning (dashed line) versus time. Only hosts within AS559 are considered.

While the „instantaneous number of infected hosts“ in Figure 5.10 clearly exhibits daily/weekly oscillation, we searched a way to get a seasonally adjusted decay. Therefore we charted the cumulative timely evolution of new infections, as well as the evolution for the last scan activity of all IPs. As seen in Figure 5.14 we have a visual representation of the fight between infection and recovery. The difference (dash dotted line) of first activity (solid line) and last activity (dashed line) must consequently be the infected number of hosts, but without seasonality. A detail view of the red dash-dotted line (Fig. 5.15) confirms what we already discovered earlier in this section. Infections occur only in the morning hours, when fresh machines become available. And the patching process is continuously done until 6pm (local time) when workers exit their office. During night the infected population remains constant until the next morning, where a new day cycle starts. By contrast, the instantaneous number of scanning hosts shows a large reduction in infected population at night. So this must be an effect of machines switched-off during night. From day to day the patching defeats more and more the spreading and gets the upper hand. We can also say that not more than 1200 host are infected at each moment.

Unfortunately we have no clue how much the DHCP-effect falsifies the result. When two infected hosts, as well as two non-vulnerable hosts, exchange their IP overnight, this has no implication to the difference curve in Figure 5.15. But imagine an infected and non-vulnerable host swapping their IP. In this case this is equivalent to an imaginary recovery in the evening and an imaginary infection during the next morning. Overnight we thus have one machine less. The same also holds true if a vulnerable machine gets a so far unused IP assigned on the next day. Consequently the

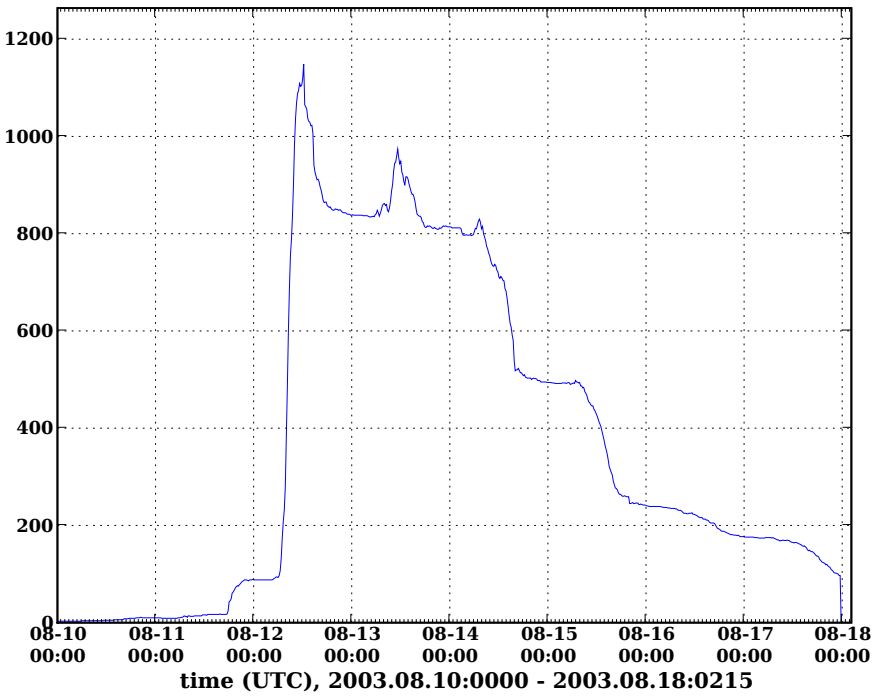


Figure 5.15: Number of infected hosts (difference of the CDFs in Fig. 5.14). In contrast to Fig 5.10 this evolution is seasonally adjusted. Only hosts within AS559 are considered.

two valleys between day 1 and day 3 in reality might be mildly less deep than they appear. In the worst case we might obtain a strictly monotonic decreasing function. This would also imply a slightly overestimated number of new infections on subsequent days in Figure 5.12.

In the following last part we would like to concentrate on the recovery process and the associated user behavior. Let us suppose that from time to time some machines get recovered not by hand, but by an automated mechanism. For instance an administrator will presumably not service manually complete farms of windows computers but rather trigger an automated update. For us this would mean that we should be able to see such automated recoveries. Figure 5.16 pictures the process of new infections and recoveries versus time during the first three weeks. Unlike in Figure 5.10 where we see the instantaneous number of scanning hosts, a machine is only counted once at the moment it initially appears. As expected we discovered such automated recovery bumps, namely whenever the green dashed line rises suddenly, e.g. on August 14, 18 or 21. The largest bump on August 27 is not due recovery because the number of new infections (solid blue line) accompanies the recovery line. We suspect this needle rather to be a large but short probe attempt.

We where also interested in how long it takes until humans recover their computers. Our prediction was that we might get a power-law behavior, meaning that most users intervene immediately while small fraction waits a long time. We expected a similar behavior as Eckmann *et al.* obtained for the reply time of e-mails [22]. For all infected hosts in AS559 we determined the individual waiting time from first scan to last scan. As in Figure 5.17 presented, we calculated then the

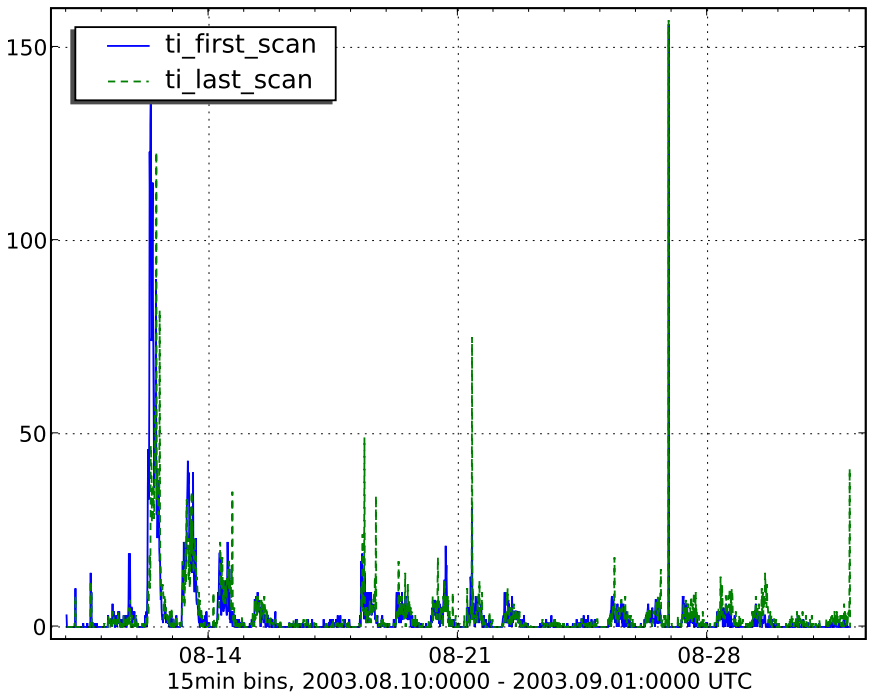


Figure 5.16: New infections (first scans) and last scans versus time. Whenever a new IP starts to spread the Blaster signature it increments the blue solid line. When the same IP stops sending the signature, it is counted by the green dashed line. The interval length is 15min. Only hosts within AS559 are considered.

probability to be still infected after a given time, so to speak the survival function of infected computers. Plotting the ordinate in logarithmic scale yields, aside from the usual daily ripple, a straight line until 480 kilo seconds. This cutoff is due the limited observation period of one week. Our conclusion is that at least during the first week the CCDF decay is exponential, meaning the recovery process is completely random. According to our measurements the half-life is close to 24 hours which would agree with the findings in Figure 5.13 where we obtained similar values for the daily recoveries.

After this decomposition of Blaster, we believe to have identified the key values of the spreading dynamics. And we dare now to list the key components producing the time series in Figure 5.10. The instantaneous number of scanning hosts is mainly a superposition of the following four core processes:

- The infection process is primarily a question of when people turn on their machines, since an invasion will occur almost immediately.

- Responsible for the seasonality in Fig. 5.10 is mainly the working cycle of humans. In a first approximation Formula 5.16 can be used, where beneath a constant fraction of continuously working machines an oscillation term is included. More difficult would be the description of weekends.

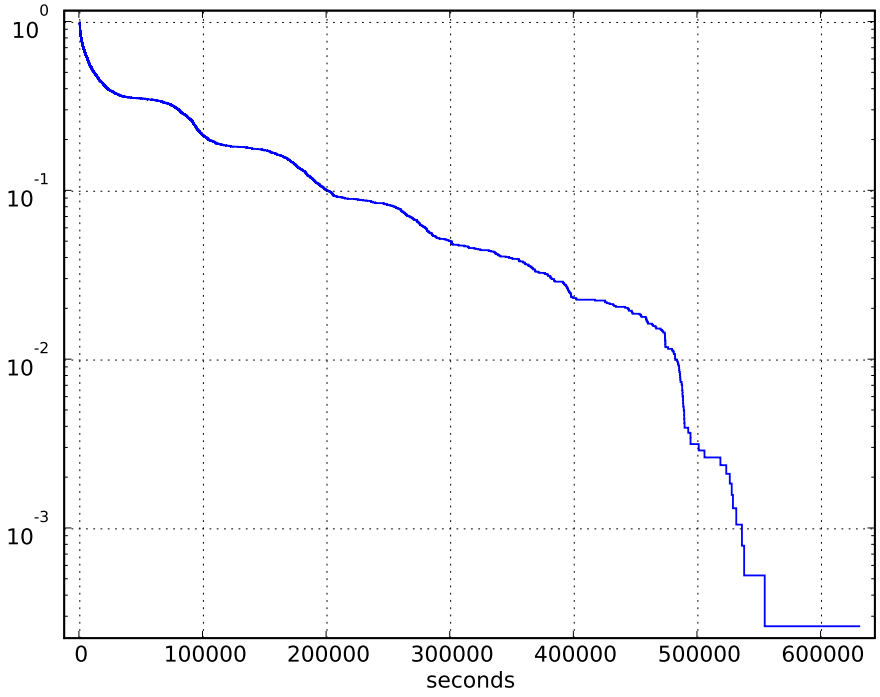


Figure 5.17: CCDF of „time to recovery“ (time to last scan), logarithmic ordinate, AS559 (CCDF of time span from first scan to last scan, aka the total time of being infected for each IP. We see here how fast hosts get recovered (user behavior). Only hosts within AS559 are considered.)

- For the recoveries we have to consult the statistical distribution describing the disease duration versus time. According to our results this could be exponentially distributed.
- Only few hosts become patched an move directly from the vulnerable to recovered population after outbreak and can therefore be neglected.

We encourage the reader to visit the appendix where one can find a more comprehensive collection of graphs. In addition there are all presented plots also for a 3 week duration. Studying them has to be done with great care, since Nachi is now appearing on the screen. Our database can only handle one infection and one recovery per IP. Suppose a host is infected by Blaster, recovered afterwards and Nachi intrudes again. This would inaccurately be combined to one single infection with recovery. For instance this is one reason why the number of infected hosts in Figure D.18c decays not as fast as it is supposed to.

5.5 Long Time Analysis

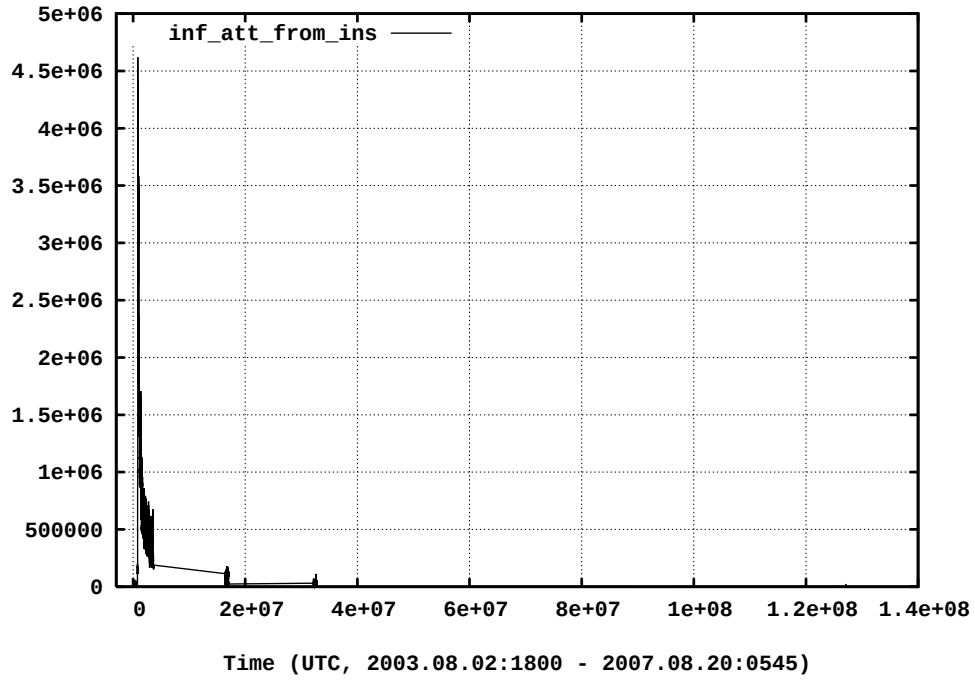


Figure 5.18: Blaster, long time analysis, infection attempts from AS559, 15min bins (4 year overview in different axis scales of the TCP SYN packets sent to destination port 135 aggregated in 15 min intervals. Only traffic from AS559 is shown.)

Besides all the information we can gain from short term analysis, it is also crucial to know the long term dynamics. For instance a long lifetime has consequences for the security industry. Current virus and malware scanners use so called signatures to identify malicious code. When we know that in general viruses can outlast for decades this implies to keep recognition patterns also for years. They do not expire. The problem of such long lasting measurements is the huge dataset that needs to be analyzed. It is infeasible to process approximately 25 TB of NetFlow data for the relevant time space from 2003 until 2007. As workaround we content ourself with four well chosen additional weeks in the just mentioned time window. More precisely we've selected the weeks 6 months (Fig. C.2), 1 year (Fig. C.3) and 4 years (Fig. C.4) after the outbreak, as well as one week before. All time windows start on a Monday in order to simplify the comparison. During the week before outbreak (Fig. C.1) there was an increased amount of traffic rooting from security interested users probing with the proof of concept sample code. For all host related time series we applied a threshold of at least 20 necessary scans in order to become counted as infected. We introduced this to reduce the noise level coming from machines accidentally using the same scan signature.

As in Figure 5.18 stated, we find that the RCP-DCOM vulnerability signature yields not much activity after 4 years within AS559. There is merely one host still generating some activity (see Fig. C.4). In contrast to AS559 we count on our boarder routers an average population of more

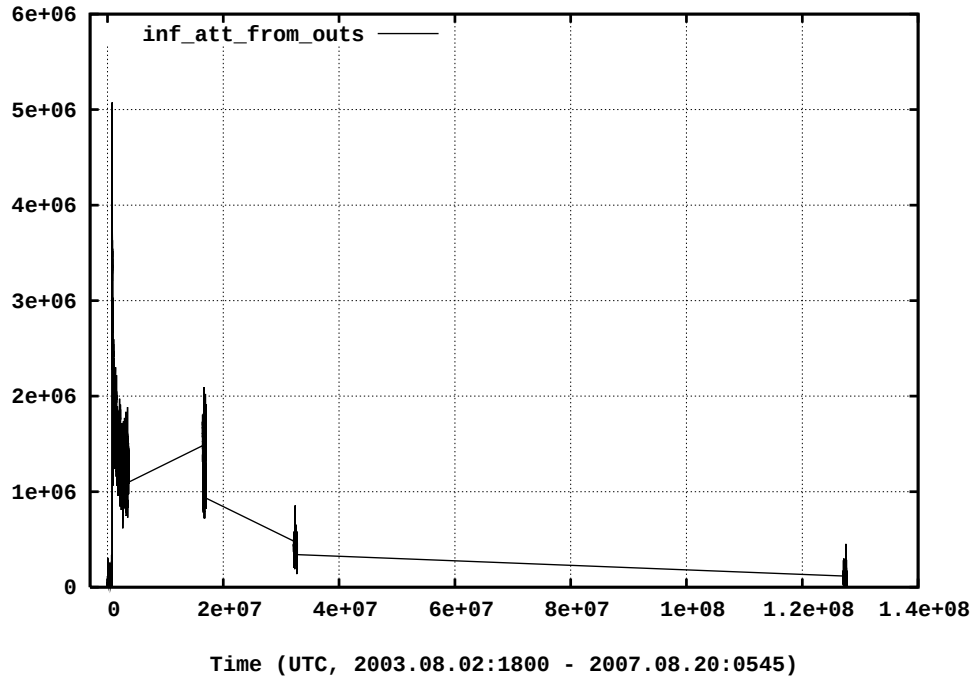


Figure 5.19: Blaster, long time analysis, infection attempts from outside AS559, 15min bins (4 year overview in different axis scales of the TCP SYN packets sent to destination port 135 aggregated in 15 min intervals. Only traffic from Internet to AS559 is shown.)

than 25 hosts which still send the scan signature to AS559. This is a remarkable amount and it is in accord with the assumption of long lifetimes of worms and viruses. Also noticeable is the activity 6 months later (at $\sim$1.8e+07 seconds since outbreak on Fig. 5.19) which is on a much higher level than expected. We assume this to be an effect of a worm mutation, a successor of Blaster and Nachi, raising the height of activity. Especially worms like Nachi which involve multiple vulnerability exploits possess the capability to scan for security holes for which we would not expect much activity due a high degree of patching. The interesting point of such secondary peaks is that we suspect there to be a relation to aftershocks encountered in earthquakes (see [49]). Further graphs of Figure 5.18 and Figure 5.19 in logarithmic scales are in Appendix C.

5.6 Improved SIR Model for Worms

In the result chapter we have gathered many qualitative informations that have a direct impact to the modeling of worm dynamics. Let's recapitulate them:

- Since the hosts get almost instantaneously infected, the spreading rate must be large.
- Nearly all hosts get infected. This is also an implication of the high spreading rate.
- The infection rate is mainly dependent on the number of available machines and thus when the people turn on their machines.
- Human interactions impact the recovery process and are the key parameter for cure.
- Only few transitions from susceptible directly to recovered occur after outbreak (aka patching).
- The overall lifetime of an epidemic can be up to decades.

Our goal was then to incorporate the findings into an improved SIR model. We tried many variants and possibilities to formulate an adequate model. One of them we would like to present here:

$$\frac{dS(t)}{dt} = -\beta m(t) \cdot I(t) \cdot S(t) \tag{5.2}$$

$$\frac{dI(t)}{dt} = \beta m(t) \cdot I(t) \cdot S(t) - \gamma h(t) \cdot I(t) \tag{5.3}$$

$$\frac{dR(t)}{dt} = -\gamma h(t) \cdot I(t) \tag{5.4}$$

where $m(t)$ depicts the machine availability and $h(t)$ the human recovery activity respectively:

$$m(t) = m_0 + (1 - m_0) \cdot sin\left(\frac{\omega}{2}t\right)^2,\ m_0 \in [0..1] \tag{5.5}$$

$$h(t) = h_0 + (1 - h_0) \cdot sin\left(\frac{\omega}{2}t\right)^2,\ h_0 \in [0..1] \tag{5.6}$$

We tried to assume realistic parameters for a qualitative evolution of the infected host population. While we know that the fraction of machines in continuous operation m_0 has to be roughly at 40%, we suppose the nocturnal human population h_0 to be small (2%). Altogether we obtained for the tuple (β = 5000, γ = 0.6, m_0 = 0.4, h_0 = 0.02, ω = 5.6) the numerical solution shown in Figure 5.20. As a first approximation the qualitative evolution matches the result we obtained in Figure 5.15. But we discovered a weakness in the SIR model which is independent of the chosen parameters: The initially needed fast decay seems to be incompatible with a long lifetime. The envelope of the timely progress is always composed of exponentials, as long as we employ an differential equation system like Eq. 5.2- 5.4. However, the suggested power law behavior as proposed in Section 5.7 can reach both, a fast initial descent and a long life cycle. Consequently we believe

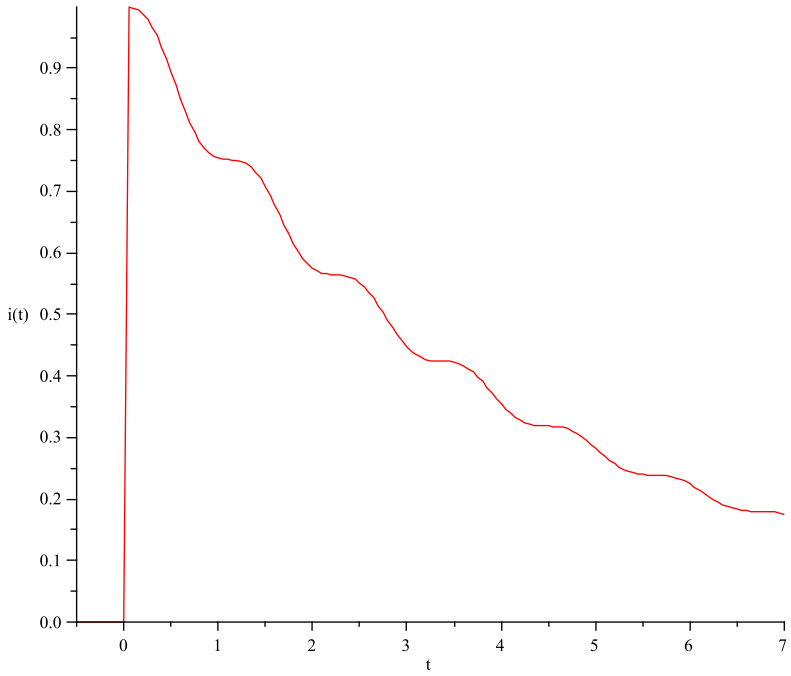


Figure 5.20: Qualitative evolution of the infected host population for our SIR model described with Eq. 5.2 – 5.4. The realistically large chosen spreading rate causes the epidemic to reach almost 100% hosts in the vulnerable population.

that the Kermack-McKendrick SIR model, as well as our improved one, can be used at best as an approximation during the initial outbreak phase, but is not able to explain the sustained lifetime. A second problem is the dominance of the spreading rate β, which implies a strictly monotonic decreasing function after culmination in our model. No secondary peaks on subsequent days are possible (like those in Fig. 5.15), as the modeling with $m(t)$ and $h(t)$ in general would allow it.

5.7 Shock Decay Analysis

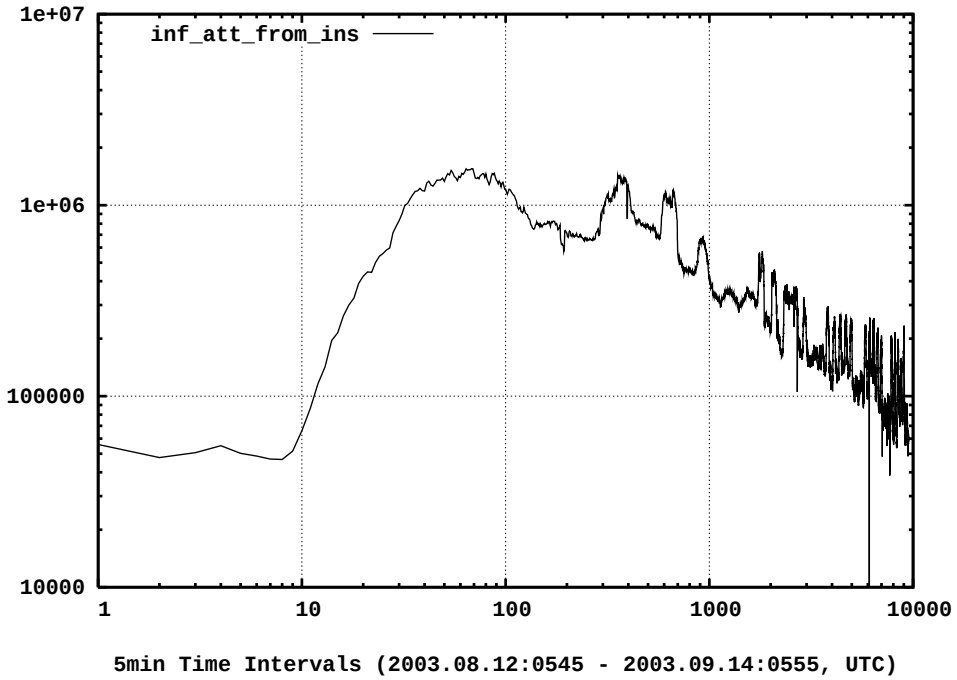
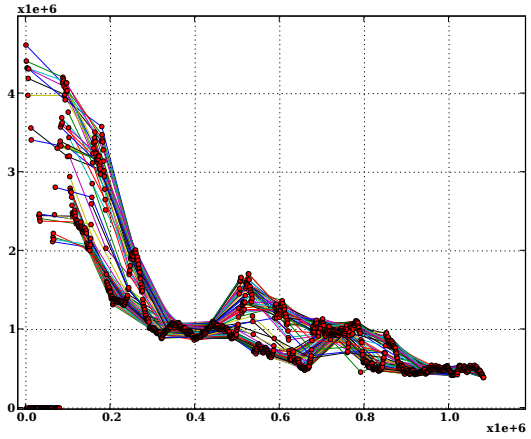


Figure 5.21: Qualitative evolution of infection attempts from AS559 versus time, both axis logarithmic (The number of TCP SYNs sent to destination port 135 aggregated in 15min intervals. Only scans from AS559 to Internet are shown)

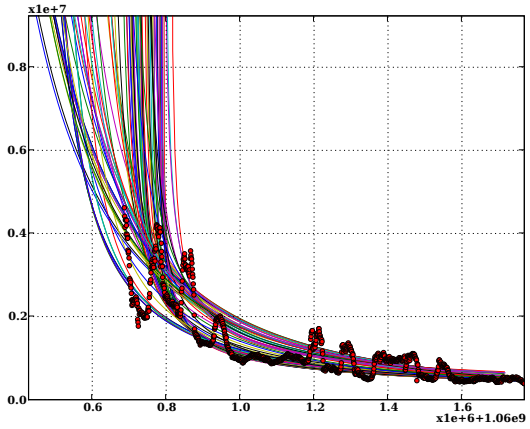
In the previous Section 5.6 we have investigated the SIR model and we concluded the fast initial decay is in conflict with the long life time of the worm. This raises the question if we can find a better explanation for the long time behavior. In Section 5.3 we have presented a plot pointing out the infection attempts from inside (Fig. 5.8). Out of nowhere we observe an almost instantaneous rise in the number of scans sent, reaching the maximum after few hours. And during the following days the number decreases slowly, but stays far above the level before the outbreak. This shock clearly reminds us to the picture seen from Amazon book sales, showing a exogenous shock for Book A, (Fig. 2.1). Drawing the infection attempts with double logarithmic axis (Fig. 5.21) raises further the suspicion, because the straight-line-like rise and fall match a power-law behavior, as described in Formula 2.9. So, we considered it worth to analyze the dynamics more detailed. Particularly we are interested in the power-law parameters (A, t_c, α) describing the decay:

$$y(t) = \frac{A}{(t - t_c)^{\alpha}} \tag{5.7}$$

The idea how to obtain the power-law decay parameters is simple. First we delete all point before the maximum of the curve (as done in Fig. E.2 and Fig. 5.22a). For a given window size (a limited number of points) of the decay, we used the deterministic least square fit formulas minimizing the error between the estimated curve and the real data points:



(a) Sampled decays of „infection attempts from inside“ (Fig. 5.8). Every line connects two points if the time between is exactly 24h. Weekends are included, 15min bins.



(b) Estimated power law decays for the sampled curves of „infection attempts from inside“ , Figure 5.22a. A large variety of estimations is obtained.

Figure 5.22: Sampled decays of „infection attempts from inside“ and the corresponding least square fit estimations

$$y_{est}(x) = A \cdot x^B \tag{5.8}$$

$$B = \frac{n\sum_{i=1}^{n}(\ln(x_i)\cdot\ln(y_i)) - \sum_{i=1}^{n}\ln(x_i)\cdot\sum_{i=1}^{n}\ln(y_i)}{n\sum_{i=1}^{n}(\ln(x_i))^2 - (\sum_{i=1}^{n}\ln(x_i))^2} \tag{5.9}$$

$$A = \exp\left(\frac{\sum_{i=1}^{n}\ln(y_i) - B\cdot\sum_{i=1}^{n}\ln(x_i)}{n}\right) \tag{5.10}$$

Where n is the number of points in the window. To compare different fits, we need a benchmark „how good“ the fit is. As measure for the error, we evaluated the summarized squared fluctuations of the noisy points y around the best fit y_{est}:

$$\tilde{\chi}^2 = \sum_{i=1}^{n}((y_i - y_{i,est})/y_{i,est})^2 \tag{5.11}$$

Our used error measurement $\tilde{\chi}^2$ has its root in the Chi Square test looking similarly:

$$\chi^2 = \sum_{i=1}^{n}((y_i - y_{i,est})^2/y_{i,est}) \tag{5.12}$$

A fit using Formula 5.8 – 5.10 assumes that the singularity t_c is at time 0. Since we process real data and our maximum is far below infinity, we have to find the correct t_c, which will be somewhere

before the time of the maximum. Playing around with the data has shown that t_c values with a distance to the maximum of more than 6 days makes no sense. So we used this as upper bound for t_c. In our analysis tool, we move around t_c measuring $\tilde{\chi}^2$ as output. The optimal parameters are found, when $\tilde{\chi}^2$ becomes minimal.

We could also think of an exponential decay. Our implementation uses the following formulas to estimate the exponential parameters:

$$y_{est}(x) = A \cdot e^{B \cdot x} \tag{5.13}$$

$$A = \exp\left(\frac{\sum_{i=1}^{n} \ln(y_i) \cdot \sum_{i=1}^{n} (x_i)^2 - \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} x_i ln(y_i)}{n \sum_{i=1}^{n} (x_i)^2 - (\sum_{i=1}^{n} x_i)^2}\right) \tag{5.14}$$

$$B = \frac{n \sum_{i=1}^{n} x_i \ln(y_i) - \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} \ln(y_i)}{n \sum_{i=1}^{n} (x_i)^2 - (\sum_{i=1}^{n} x_i)^2} \tag{5.15}$$

Be aware that for an exponential decay of the form $y(t) = A \cdot e^{-\alpha \cdot (t - t_c)}$ we can find a best fit for any t_c, because $y(t) = A \cdot e^{-\alpha \cdot t} \cdot e^{\alpha \cdot t_c} = \hat{A} \cdot e^{-\alpha \cdot t}$. Therefore we have chosen to lock our t_c at t_{max}.

One big difference between the Amazon and YouTube data analysis, but also all other groups researching worm dynamics, is our much fine granular resolution. Our plots clearly outline the daily and weekly oscillations, while in the previously mentioned cases only daily counts are available. We have been confronted with the problem how to handle this periodic behavior, since it cannot just be neglected and considered as noise. Averaging over a day is also not an option, because we would loose too much information and obtain too few data points. Our first Idea was to sample regularly the decay and then connect the point at a certain time with the points at the same time on subsequent days. That is to say, we connect a point with its successor when the time between is exactly 24 hours (see Fig. 5.22a). Our hope was that at best the whole array of curves would produce similar estimated parameters.

This is not the case as Figure 5.22b impressively shows. We obtained a large variety of parameter tuples. Therefore we tried to identify groups with similarities, for instance clusters for daily and nightly decays (Fig. E.4b and E.4d). The best matching group was at night (Fig. E.4d) from 21 p.m. to 04 a.m. While most α-values where in a comparable range, the standard deviation for t_c and A remained rather high (first two rows in Table 5.2).

The success was rather moderate because of two reasons. Lower traffic on weekends leads to indentations on Saturday and Sunday. And because we needed to cut off all data points before the maximum, many curves missed completely the initial part of the outbreak, especially the nightly decays (best seen on Fig. E.4d). This leads to wrong estimations for the singularities t_c, but also falsifies α. By removing the weekend data points we tried to defuse the former, with moderate success (last two rows in Table 5.2). As mentioned above, we could also think of an exponential decay. So we applied our decay estimation method using an exponential fit as described in formula 5.13 – 5.15. The results (not shown) where even worser than for the power law. Therefore

average for	w/e[a]	A	t_c	α	sdev(α)	sdev(t_c)	sdev(A)
0800 - 1400	yes	2.23E+021	08-07 13:14:04	1.70	0.54	73.5h	3.16E+19
2100 - 0400	yes	4.80E+009	08-13 00:37:39	0.61	0.30	39.1h	1.37E+15
0700 - 1600	no	5.71E+018	08-08 16:13:48	1.38	0.66	88.82h	4.76E+21
1600 - 0600	no	2.39E+014	08-12 19:01:01	0.59	0.24	33.7h	2.02E+14

[a]weekend data considered or not

Table 5.2: Average estimations for daily and nightly decay clusters

we discarded the sampling method, since it complicated the issue rather than giving us a consistent picture for the decay behavior.

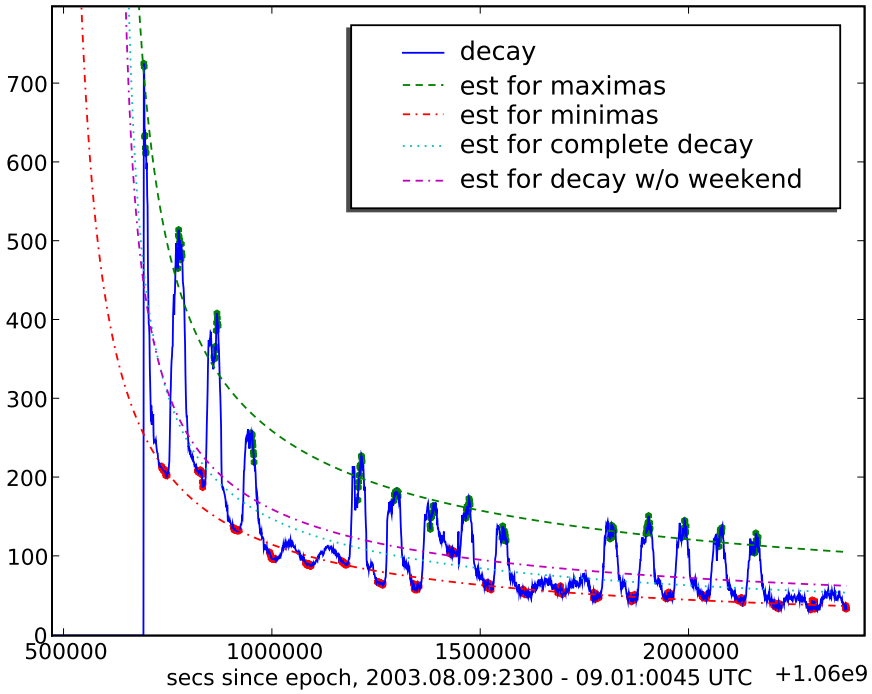
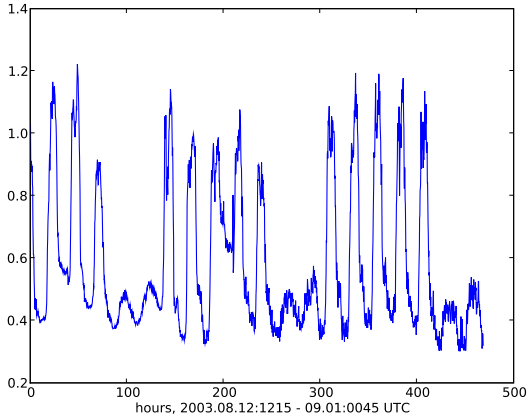
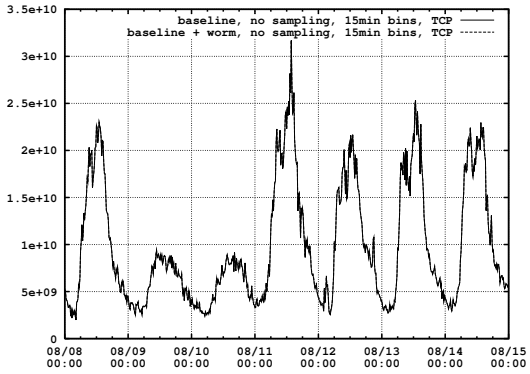


Figure 5.23: Maxima, minima decay estimations for the signal in Fig. 5.10. The dashed green line is the estimation for the green dots (weekend not included) whereas the red dot-dashed line is an estimation for the red dots. The light blue and pink lines are estimations for the complete blue decay where the latter does not consider tuples of the weekend.

Another idea was the following: The instantaneous number of infected hosts in Figure 5.10 could be the composition of two or more signals. First we have the effects of an epidemic dying out, producing some kind of decay as described in Formula 5.7. Secondly, there is the a non-constant host population $m(t)$ with some machines showing a cyclic on-off property. This could be approximately modeled by:



(a) The daily/weekly oscillation in the signal of Fig. 5.10 after removing the power law component (green dashed line in Fig. 5.23)



(b) Counted bytes over router swiCE3.switch.ch, once with and once without the Blaster infection attempts. The difference of baseline and (baseline+worm) is insignificant, since only few additional bytes are generated by Blaster.

Figure 5.24: Comparison between the calculated oscillation signal and common router traffic

$$m(t) = m_0 + (1 - m_0) \cdot \cos\left(\frac{\alpha}{2} \cdot t\right)^2, \quad m_0 \in [0..1] \tag{5.16}$$

where m_0 is the fraction of hosts in continuous operation and $m(t)$ must always be positive. If the signal $s(t)$ of Figure 5.10 is the product of $y(t)$ (from Eq. 5.7) and $m(t)$, we should be able to recover either $y(t)$ or $m(t)$ if the other one beneath $s(t)$ is given. For instance, the oscillation is the result of dividing $s(t)$ and $y(t)$: $m(t) = s(t)/y(t)$. For $y(t)$ we used the data points around the peaks, more precisely from 11am – 14pm, and applied a power-law least square fit (green dashed line in 5.23). Dividing the signal $s(t)$ (blue solid line) by $y(t)$ (green dashed line) gives us the seasonality line as shown in Figure 5.24a. This daily and weekly oscillation look exactly as we would expect it according to our experiences from previous work ([23], see Fig. 5.24b). Furthermore, we can infer from the DC-offset that around 40 percent of the infected population is permanently up.

what	A	t_c	α	ymax
day / max	439045	2003.08.11:1806	0.579	2003.08.12:1215
night / mins	9692196	2003.08.10:0331	0.862	2003.08.13:0000
complete incl w/e	578046	2003.08.11:1842	0.645	2003.08.12:1215
complete excl w/e	383663	2003.08.11:1647	0.605	2003.08.12:1215

Table 5.3: Parameters for estimations in Fig. 5.23

It is interesting to find our power law exponent to be around -0.6 (Table 5.3). This matches exactly the empirically determined exponent for the exogenous critical case in YouTube [16]. And we thus

also suspect our Blaster decay to be exogenous critical. But also the fact that during the night before the outbreak in AS559 a small nightly population was infected in advance and caused the shock in the upcoming morning does also vote for an exogenous shock. Looking at a single worm decay does not proof to be the exponent generally at -0.6 for all kinds of exogenous events in the Internet. However, we have now an example that there may be a consistence with the YouTube case.

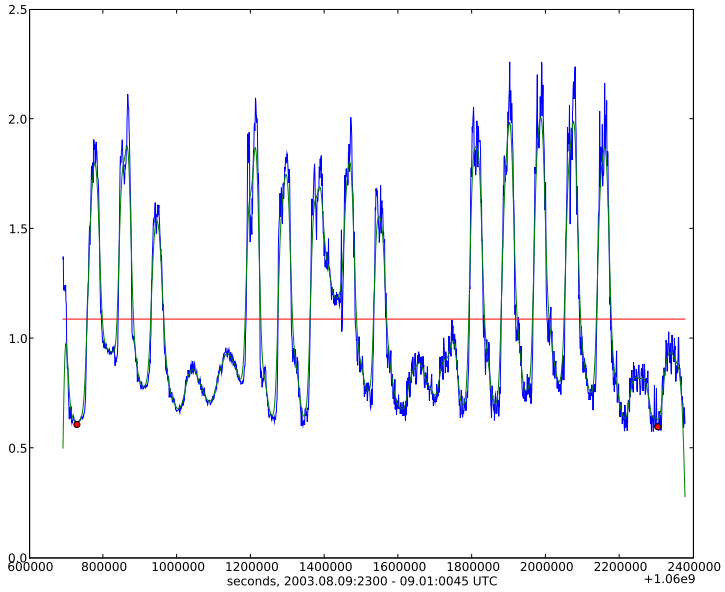
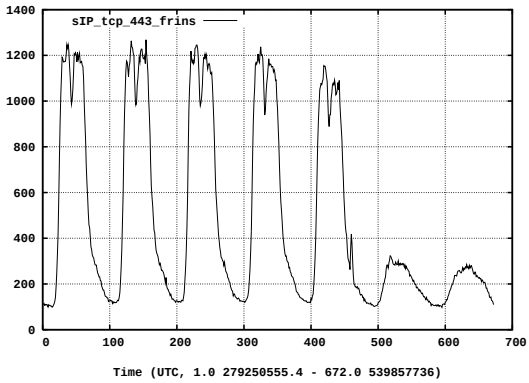


Figure 5.25: Blue line: Same daily/weekly oscillation as in Fig. 5.24a, but in addition we shifted t_c and α until the slope of linear regression (red line) becomes 0. Green line: convolution of the blue signal with a Gaussian low pass filter to remove noise. Red line: Result of the linear regression estimation using the blue curve between the two red dots

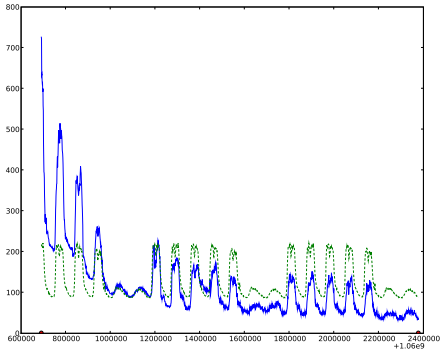
In order to be sure that the estimations of Table 5.3 are in an acceptable range of accuracy, we build a procedure to verify the power law parameters. Dividing the initial signal $s(t)$ by an imprecisely estimated power law y_{est} (Eq. 5.8) leads inevitably to a small tilt in the extracted daily oscillation $m(t)$. As measure how good an estimation is, we consequently can use the slope of the seasonality signal (see Figure 5.24a).

From an initial estimation of the complete signal (light blue dotted line in Fig. 5.23) we started to vary t_c in a iterative procedure until the linear regression's slope converged to 0. For the estimation of the linear regression (red line in Fig. 5.25), we used just the points from the first nightly minimum to the last one, indicated with red dots. Since it is difficult to find local extremal values in distorted signal, we searched for a way to eliminate the high frequency noise. Every valley rooting from noise is a potential candidate for our daily minima. By convolving the signal (blue line) with a linear Gaussian low pass filter, we obtained the smoothed green line which allowed to determine

the wanted nightly minima. This search for minima was necessary because boundary values at the beginning and end need to be at the same level. Otherwise we would obtain an error in the slope as side effect from the different levels of the initial and final data points.
In the case of the decay in Figure 5.23, t_c moved from 2003.08.11 18:42 to 2003.08.12 01:10 and the initially determined exponent α = 0.645, which is highly dependent on the chosen t_c, displaced only slightly to 0.601. Of course we verified also the tilt error of other available decays. In average, the error of α was around 0.05, but never more than 0.1. This gives us confidence that the obtained α values come along with an error below $\pm$0.1.



(a) Used reference signal. Unique source IPs located in AS559 sending traffic to destination port 443 (HTTPS) on nodes outside AS559.



(b) Blue solid line: Signal from Fig. 5.23 (instantaneously infected hosts within AS559). Green dashed line: Scaled and upward shifted reference signal of Fig. 5.26a which removes the seasonality the best when we divide the blue by the green line (see Fig. 5.27)

Figure 5.26: The used reference seasonality signal to remove daily and weekly oscillations in the power law decay of Fig. 5.23

Until now we have always looked for ways to remove the power law decay in our signals in order to obtain the daily seasonality. As a last point we tried it the other way round. We thought about generating a reference oscillation pattern which should allow us to clean up the daily/weekly pattern in power laws. Since no such patterns are publicly available, we produced ourselves such samples. Since it remained unclear what kind of metric creates the best archetype for our case, we evaluated 90 different combinations out of the following fields:

- metric (number of flows, number of packets, number of bytes, unique source IPs, unique destination IPs per interval)
- protocol (TCP, UDP, remaining (ICMP, OSPF, ...))
- destination port (80/HTTP, 443/HTTPS, all ports)
- flow direction (from AS559, from outside AS559)

To avoid inaccuracies due the timely evolution of the Internet usage, we wanted to reuse the same NetFlow data which we already employed for Blaster. Therefore we filtered out all Blaster related

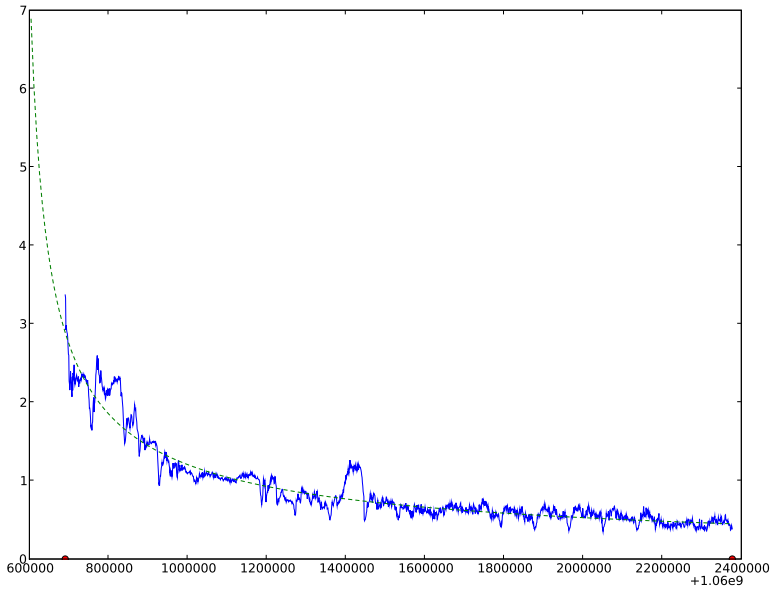


Figure 5.27: Blue solid line: Signal from Fig. 5.23 (instantaneously infected hosts within AS559) with *removed* daily/weekly oscillation. It can be obtained from the division of the blue line by green one of Fig. 5.26b. Green dashed line: Best found power law fit for the blue solid line.

traffic in order to obtain the so called baseline for the metrics mentioned above. Afterwards we averaged the patterns over five weeks of data to reduce the noise level in the samples. For the decay of instantaneously infected hosts within AS559 (Figure 5.23), we have chosen to take the reference pattern depicted in Figure 5.26a. It also uses unique source IPs per interval placed within AS559 as measurement, but the metric considers flows to destination port 443 (HTTPS) instead of port 135. Consequently it should be clear that this pattern must be stretched by a factor η_s and maybe incremented by a DC offset ϱ_{dc} for the purpose of serving as a replacement for the unknown real oscillation.

$$y_{clean}(t) = \frac{s(t)}{\eta_s \cdot m_{osc}(t) + \varrho_{dc}} \tag{5.17}$$

By varying the parameter space of η_s and ϱ_{dc}, we cleaned up our decay signal $s(t)$ with the reference pattern $m_{osc}(t)$. We looked for solutions $y_{clean}(t)$ which resemble most suitable an ideal power law. To be more precisely, we minimized the deviation error $\tilde{\chi}^2$ of Equation 5.11 between $y_{clean}(t)$ and an applied power law estimation $y_{est}(t)$ (from Eq. 5.8).
Figure 5.27 shows the cleaned signal $y_{clean}(t)$ (blue solid line), as well as the resulting estimation $y_{est}(t)$ (green dashed line), when we apply the best found stretch parameters η_s = 0.115 and ϱ_{dc} = 75.8 to the pattern $m_{osc}(t)$. Although not perfect, the oscillation in $y_{clean}(t)$ has mainly vanished. This confirms our assumption that the measured signal $s(t)$ roughly is the multiplication of a

power law and an oscillation therm. If we involve maximum likelihood estimation, we get for the decay parameters the following triple: (A = 8925, α = 0.687, t_c = 2003.08.11 02:49). It should be clear that the amplitude A changes compared to the signal including the daily/weekly seasonality because of the applied division. But α and t_c are still very close to what we obtained above for the complete decay estimation still including the oscillation (Fig. 5.23, A = 578046, α = 0.645, t_c = 2003.08.11 18:42).

Chapter 6

Conclusions and Outlook

The goal of this thesis was to test and verify existing models which explain the propagation and immunization dynamics of epidemic processes.

To hit this target we developed a couple of tools to analyze Internet epidemics in NetFlow datasets (Chapter 4). An initially written module allowed to visualize simple time series by aggregating traffic in regular intervals. Our interest where mainly focused on the number of scans crossing the borders of the AS559, as well as the number of infected hosts generating these scans. As an example we used the W32_Blaster event, because this is a well analyzed worm. We could gain a much deeper insight than studies already made.

We recognized that time series are precious but provide only a limited view of the complete dynamics of epidemics. This encouraged us to collect information about each infected host, such as time of first scan, time of last scan, number of scans sent, and AS affiliation. The collected host centric data allowed us a better comprehension of Blaster's spreading dynamics. Beneath plots like the number of new infections per interval (FigD.3) or the occurrences of unsuccessful infection attempts (Fig. D.1a) where able to provide a better understanding of the propagation, such as in the spreading dynamics (Fig. 5.11b), the recovery behavior (Fig. 5.17), scans sent per host (Fig. D.7), or the fight between spreading and recovery (Fig. 5.15). We believe coming away from simple time series and instead extracting the key features of epidemics, as we did, is the forward-looking approach. A great potential could be retrieved by collecting key information directly in a database which seem to be most suitable also for further processing.

A problem we faced was the rarely considered ongoing of Nachi, a worm that started to spread 8 days after Blaster, as well as further aftershocks. Nachi hunted and tried to remove Blaster from infected hosts, using a similar scanning routine, since vulnerable hosts are a limited resource. This procedure accelerated the removal of Blaster and falsified the expected immunization process (e.g. see Fig. 5.9). We believe that such secondary outbreaks can be related to aftershocks of earthquakes (Sec. 5.5) and thus are interesting for further research.

Chapter 5.1 presents several measurements which give us a picture of the dimension the Blaster anomaly affected the SWITCH network. Although up to 60% of all flows where caused by Blaster, less than 1% of all active hosts where infected at each time. The infected population remained small within the network, which is an important fact for modeling the spread of epidemics (Sec. 5.3.1).

Hoping that we can classify the evolution of infected hosts (endogenous or exogenous shock), we analyzed the decay of various plots (Sec. 5.7). Graphical representations of some decays arouse suspicion that we might could extract the parameters, especially the exponent, of a power law or exponential decay (see Fig. B.5 – B.11, D.3 –D.4). To get rid of the daily oscillation, which we always have when we consider Internet traffic, we brought up the idea of sampling the curve in 24 hour intervals (Fig.5.22a). We searched for daily and nightly time ranges which show a similar decay behavior and compared the parameters we obtained from these set of curves. Although this approach seemed to be promising, the obtained range of parameters was huge and not conformable (Fig. 5.22b). More fruitful was the attempt to separate the decays into a power law and oscillation term. Dividing by the estimated power law decay produced a well known oscillation pattern with a daily/weekly characteristics (Fig. 5.24a).

Our results also have an influence on the modeling Internet epidemics with the SIR model (Sec. 5.6). The most significant outcome is our finding that infections occur almost instantaneous, which has several implications. Almost all hosts get infected what further implies to choose the spreading rate very high. Otherwise the vulnerable host population gets not infected entirely. The spreading dynamics is heavily dependent on the availability of vulnerable machines and thus when humans switch them on and off. Since computers get contaminated quickly, the probability to patch a vulnerable machine before an infection is quite low. For success a machine needs to be rather patched when it is in disconnected mode (although patches typically are merely online obtainable). Consequently, we believe that only few hosts evade an infection and pass directly to the recovered population. We recognized that the long lifetime of worms can hardly be explained with the Kermack-McKendrick SIR model. For short term analysis it can be used as an approximation. But the fast initial decay after culmination is in conflict with the long lifetime. The exogenous/endogenous shock model can explain much better such enduring life cycles.

Unfortunately the research topic of large Internet anomalies has been mostly abandoned. Automatic update functions and firewalls have become widespread, for instance both are now active by default in current Windows versions. This has forced the malware industry to move continuously to the application layers, such as intrusions via ActiveX or e-mail. The epoch of large mass worms seems to reach an endemic state. Nowadays the trend goes clearly towards silent spreading and hidden intrusion (buzzword botnet).

Nevertheless we can state that the exact Internet epidemic spreading is still not completely understood until now, despite the release of many papers covering this topic. Especially modeling in presence of daily/weekly oscillation was hardly considered so far although we identified such seasonality as an absolutely crucial parameter for the spreading dynamics. Aggregating the traffic for complete days to remove the seasonality is certainly not a considerable solution because we loose too much information.

Since the propagation is intrinsically linked to the underlying network structure we strongly assume that the topology must be visible in the spreading dynamics. We believe here is a great potential to bring forward the research about the assumed scale free characteristics of the Internet. Similarly there exist many different topology types in the upper layers about which we know

only few.
Also interesting to us are the analogy to many distant research topics. It is not only that we use the SIR model rooting from biology, but the worm spreading process has also similarities to nuclear heating processes. We already mentioned the analogy to aftershocks of earthquakes. Multidisciplinary approaches would help to improve the understandability of computer networks.
Beneath all topics mentioned so far, we compiled a catalog of open questions:

- As mentioned in Section 5.5, we observed parts of an aftershock in the long time analysis. In addition we expect there to be more secondary avalanches. Finding them and study some of them would help to approve our current view of the dynamics.
- Beneath the number of infections we have also recorded some reinfections over the boarder routers, which we have not analyzed in detail.
- In Section 5.4 we determined that the probability of getting immediately infected is very likely. It would be great to empirically determine the probability density function versus time to get infected.
- Until now we talked not much about the so called saturation effect. More precisely around 2.5 hours after the external outbreak the population experiences a reduction in the so far unhindered growth. Beneath an overload of the record export engine we could also imagine congestion problems or any other kind of physical limitation. We have not found a satisfying explanation for this damping impressively seen on Figure B.3 and B.4.
- A further unknown is how many traffic was rejected by the router's record export engine. From the heuristics we used to recognize multi stage infections from Blaster, we have hints that the losses are larger than assumed until now (see Section 4.1.6).
- We believe that we have not exploited all informations we gathered with our tool set described in Section 4.1. Although we presented many results in Section 5.4, there remain information we even did not look at.
- In order to understand the dynamics better, it may makes sense to track some single hosts. Particularly the individual scanning behavior could then be studied.
- For many applications it would be relevant to have a standard pattern for the weekly cycle. Such a sample would help to identify and remove the seasonality in all kind of data.
- In the section about the SIR Model (Sec. 5.6) we have assumed that the total host population remains constant. Since the growth of the Internet during the long observation period is claimed to be exponential in a first approximation, it may makes sense to introduce a birth and death rate to find a more appropriate model. Theoretically the birth as well as the death of nodes can occur in the susceptible, infected, and recovered population. For the sake of simplicity it would be sufficient to add a birth rate just to the susceptible population. The nodes usually don't have pre-installed viruses and thus rarely appear firstly in the infected population. Hosts born directly to the recovered population make sense, but have no

influence to the spreading dynamics and therefore can also be neglected. If the death and birth rate are on a enough high level, it is possible to obtain a *steady state* after the main shock which could explain the long time behavior. The question is then if it is more suitable to model the long life with the just mentioned enhanced SIR differential equations or if the suggested power law decay (Sec. 5.7) matches better.

- It is possible to find differential equation systems which are very similar to the ones from the SIR model, but they have a power law instead of exponential decays as solution. The idea is to introduce exponents larger than one to the system variables $S(t)$, $I(t)$, and $R(t)$ in the coupled equations. How this works in detail is described here [29].
- It would also be helpful to know the precise amount of used IPs within SWITCH, not just the ones instantaneously in use. We know from the routing tables of AS559, that this number must be smaller than 2.5 Million (see Sec. 5.1). Consequently it should be feasible keep track of all IPs in a HashTable.
- Is there a correlation between the unsuccessful infections and successful infection attempts? (see Fig. 5.6 and D.1a)
- While real-life viruses only can reach a small number of neighboring hosts (e.g. when looking at humans, animals), electronic Worms have a much more global view (as mentioned in Section 5.1). It would be useful to further determine what influence this difference has on the spreading dynamics.
- In Section 5.7 we calculated the power law exponent for Blaster to be at -0.6. Unlike empirically proven for the YouTube case, we don't know yet, if this is also an universally valid value for all epidemic shocks in the Internet. This would imply to analyze a larger number of anomalies.
- Our results indicate that the recovery process is Poisson-like (Sec. 5.4). But we don't know much more about the user behavior. We believe that one could gain more about the human interaction.
- According to our paper review, we found so far no paper seriously trying to empirically estimate the key parameters of the SIR model for cyber worms. Setting up a hypothesis test based on the deviation of the fit would be a contribution to check the plausibility of the model in the case of Internet anomalies.
- One way to handle the seasonality could be to choose a relative time scale. When we oscillate the speed of time, more precisely when we slow down the day and boost up the night, we may would be able to remove the seasonality.

6.1 Future Work

We also collected a brief overview to several worms which could be investigated in future to empirically check and compare power law exponents. As a matter of course this list is non-

exhaustive.

W32.Witty.Worm

Date: 20.03.2008

Description: Exploits the ICQ parsing vulnerability of ISS products to gain unauthorized remote execution access to a machine using a buffer overflow. When doing so, the worm overwrites system memory and runs in the same security context as the ISS product being exploited [53].

Special: Uses a vulnerability of an infrequently used security software (ISS).

Spreading: Sends itself to 20000 randomly generated IP addresses with random destination ports and a UDP source port of 4000.

Recognition: Easy

Difficulties: NetFlow data of corresponding month may be corrupt.

Sources: `www.symantec.com/security_response/writeup.jsp?docid=2004-032009-1441-99&tabid=2`
`xforce.iss.net/xforce/alerts/id/166`
`www.secureworks.com/research/threats/witty/`
`www.cc.gatech.edu/~akumar/witty.html`

W32.Sobig.F@mm

Date: 18.04.2003

Description: W32.Sobig.F@mm is a mass-mailing, network-aware worm that sends itself to all the email addresses it finds in certain local files [53].

Special: The worm uses its own SMTP engine to propagate. Top 1 of Sophos malware List 2003.

Spreading: SMTP/mass-mailing

Recognition: Medium. SMTP, destination port 25, size 103000-125000 bytes, unusual large size.

Difficulties: No better pattern match produceable than port and file size.

Sources: `www.symantec.com/security_response/writeup.jsp?docid=2003-081909-2118-99`
Dissertation of Duebendorfer, p. 80ff

W32/SQLSlammer

Date: 24.01.2003

Description: W32.SQLExp.Worm is a worm that targets the systems running Microsoft SQL Server 2000, as well as Microsoft Desktop Engine (MSDE) 2000 [53].

Special: Fastest known spreading. Population doubles in 8.5 $\pm$1 sec.

Spreading: Random scan, small bug in random generator.

Recognition: Easy. The worm sends 376 bytes to UDP port 1434, the SQL Server Resolution Service Port.

Difficulties: Not available in SWITCH NetFlow dataset

Sources: `www.symantec.com/security_response/writeup.jsp?docid=2003-012502-3306-99`
`www.tecchannel.de/sicherheit/grundlagen/402050/index.html`
`www.caida.org/publications/papers/2003/sapphire/sapphire.html`

myETH DDoS Attack

Date: 09.01.2005

Description: Distributed denial of service attack to `http://www.myeth.ethz.ch`.

Special: TCP SYN

Recognition: Medium, should be feasible

Difficulties: Maybe no infected hosts, just an attack.

Sources: Diploma Thesis of Daniel Reichle, (CSG, D-ITET, ETHZ, DA-2005-06)

W32.Sasser.Worm

Date: 30.04.2004

Description: W32.Sasser.Worm is a worm that attempts to exploit the vulnerability described in Microsoft Security Bulletin MS04-011 [53].

Special: Top 3 of Sophos malware list 2004

Spreading: It spreads by scanning the randomly selected IP addresses for vulnerable systems.
25% of the time, the last two octets of A.B.C.D, if successful retrieval of host. Using the Windows API, gethostbyname().
23% of the time, the last three octets of A.B.C.D
52% Completely random

Recognition: Medium.

- Request to TCP port 445
- Command shell on TCP port 9996
- (FTP server on TCP port 5554)

Difficulties: 8 similar variants. No detailed information available so far, additional analysis necessary.

Sources: `www.symantec.com/security_response/writeup.jsp?docid=2004-050116-1831-99&tabid=2`
`www.viruslist.com/en/viruses/encyclopedia?virusid=50204`

W32.Dabber.A

Date: 14.05.2004

Description: W32.Dabber.A is a worm. This worm is based on an available exploit code. Uses a vulnerability in „Local Security Authority Subsystem Service (LSASS)“ [53]

Special: This worm propagates by exploiting vulnerability of another worm, which is in the FTP server component of W32.Sasser and its variants. Removes Sasser infection on host. Slow spreading.

Spreading: The worm begins to scan sequential IP addresses in random subnets for systems infected with W32.Sasser.Worm variants.

Recognition: Medium

- scan to TCP port 5554, (41 Bytes on IP layer)
- send exploit to TCP port 5554
- shell on port 8967
- FTP download of worm > 29696 Bytes
- (Backdoor server listening on port 9898)

Difficulties: Conflicting descriptions in available sources. Closer analysis necessary.

Sources: `secure.enterasys.com/support/security/incidents/2004/05/10899.html`
`www.golem.de/0405/31310.html`
`www.secureworks.com/research/threats/dabber/`
`www.network-secure.de/content/view/2200/1350/`

W32.Mydoom.A@mm

Date: 26.01.2004

Description: W32.Mydoom.A@mm (also known as W32.Novarg.A) is a mass-mailing worm that arrives as an attachment with the file extension .bat, .cmd, .exe, .pif, .scr, or .zip. The worm sets up a backdoor into the system by opening TCP ports 3127 through 3198 [53].

Special: Top 7 of Sophos malware List 2004. Between 01.02. and 12.02.2004 it performs a DoS attack against `www.sco.com`

Spreading: Mass-mailing worm, blacklist of „dangerous domains“ and account names. Spreading also via Kazaa.

Recognition: Difficult, but may be possible.
Sources: `www.symantec.com/security_response/writeup.jsp?docid=2004-012612-5422-99&tabid=2`
`www.tik.ee.ethz.ch/~ddosvax/attacks/mydoom/`

CodeRed Worm

Date: 16. July 2001

Description: The CodeRed Worm affects Microsoft Index Server 2.0 and the Windows 2000 indexing service on computers running Microsoft Windows NT 4.0 and Windows 2000, which run IIS 4.0 and 5.0 Web servers. The worm uses a known buffer overflow vulnerability contained in the Idq.dll file [53].

Special: Disabling itself on 21.07.2001. Deface of WebSites. Successor: CodeRedII.

Spreading: Vulnerability scanning.

Recognition: The worm sends its code as an HTTP request.

Difficulties: Not avialable in NetFlow dataset.
Sources: `www.symantec.com/security_response/writeup.jsp?docid=2001-071911-5755-99&tabid=2`
`www.caida.org/publications/papers/2002/codered/codered.pdf`

W32.Zotob.E

Date: 16.08.2005

Description: W32.Zotob.E is a worm that opens a back door and exploits the Microsoft Windows Plug and Play Buffer Overflow Vulnerability (described in Microsoft Security Bulletin MS05-039) [53].

Spreading: Sends packets to IP addresses generated at random based on the IP address of the compromised computer. The IP addresses generated use the first 2 octets of the compromised computer, and randomly generated values for the third and fourth octets. The worm will begin to generate entirely random IP addresses after 32 failures on local IPs or after 512 failures, if it was successful at least once.

Recognition: Easy.

- connect IRC-Server 72.20.27.115 on TCP port 8080

- opens UDP port 69 to initiate TFTP
- Scan to TCP port 445
- (open a back door using TCP port 8594)

Difficulties: Conflicting descriptions in available sources. Closer analysis necessary.

Sources: `www.symantec.com/security_response/writeup.jsp?docid=2005-081615-4443-99&tabid=2`
`www.bsi.de/av/vb/ZotobE.htm`
`www.heise.de/security/news/meldung/62802`
`www.f-secure.com/v-descs/zotob_a.shtml#details`

W32.Welchia.Worm (W32/Nachi.worm, Net-Worm.Win32.Welchia.a)

Date: 18.08.2003

Description: W32.Welchia.Worm is a worm that exploits multiple vulnerabilities, including: The DCOM RPC vulnerability (first described in Microsoft Security Bulletin MS03-026) using TCP port 135. The worm specifically targets Windows XP machines using this exploit. Users are recommended to patch this vulnerability by applying Microsoft Security Bulletin MS03-039. The WebDav vulnerability (described in Microsoft Security Bulletin MS03-007) using TCP port 80. The worm specifically targets machines running Microsoft IIS 5.0 using this exploit. As coded in this worm, this exploit will impact Windows 2000 systems and may impact Windows NT/XP systems [53].

Special: Uses more than one vulnerability. Attempts to download the DCOM RPC patch from Microsoft's Windows Update Web site, install it, and then restart the computer. Attempts to remove W32.Blaster.Worm.

Spreading: The worm uses either A.B.0.0 from the infected machine's IP of A.B.C.D and counts up, or it will construct a random IP address based on some hard-coded addresses.

After selecting the start address, the worm counts up through a range of class B-sized networks. For example, if the worm starts at A.B.0.0, it will count up to at least A.B.255.255.

Recognition: Medium-Difficult.

- Checks for active machines to infect by sending an ICMP echo request, or PING, which will result in increased ICMP traffic.
- Once the worm identifies a machine as being active on the network, it will either send data to TCP port 135, which exploits the DCOM RPC vulnerability, or it will send data to TCP port 80 to exploit the WebDav vulnerability starting with the latter one.
- Creates a remote shell on the vulnerable host, which reconnects to the attacking computer on a random TCP port, between 666 and 765, to receive instructions.
- Download Dllhost.exe (about 10 KB when compressed through UPX) and if not available Svchost.exe via TFTP.

Difficulties: Reboot in infection process. 2 IPs may belong to one host.
Sources: www.symantec.com/security_response/writeup.jsp?docid=2003-081815-2308-99&tabid=2
www.viruslist.com/en/viruses/encyclopedia?virusid=24946
www.securityfocus.com/infocus/1802

W32.Beagle.A@mm

Date: 18.01.2004

Description: W32.Beagle.A@mm is a mass-mailing worm that sends itself to all email addresses it gathers from certain files on the compromised system. The worm also attempts to access scripts from a website.

Special: Approximately 50 variants where seen in wild.

Spreading: Mass-mailing

Recognition: Difficult (mail worm), 15 KB attachment
Sources: www.symantec.com/security_response/writeup.jsp?docid=2004-011815-3332-99&tabid=2
home.arcor.de/idapalace/papers/bagle_analysis_v.1.0.pdf

W32.Nimda.A@mm

Date: 18.09.2001

Description: W32.Nimda.A@mm is a mass-mailing worm that uses multiple methods to spread itself. The name of the virus came from the reversed spelling of "admin." Sends itself by email. Searches for open network shares. Attempts to copy itself to unpatched or already vulnerable Microsoft IIS web servers. Is a virus infecting both, local files and files on remote network shares.

Spreading: Mass Mailing

Recognition: Not available in NetFlow dataset
Sources: www.symantec.com/security_response/writeup.jsp?docid=2001-091816-3508-99&tabid=2

W32.Erkez.B@mm (Zafi-B)

Date: 10.06.2007

Description: W32.Erkez.B@mm is a mass-mailing worm that sends itself to the email addresses found on an infected computer. It also copies itself to the folders that are likely to be shared on file-sharing networks.

Special: Top 2 of Sophos malware List 2004. Sends numerous HTTP get requests to well chosen sites to perform Denial of Service (DoS) attacks. Blacklist for some domains.

Spreading: Mass Mailing

Recognition: It is written in Assembler and packed using FSG. It is 12800 bytes in packed form, and 33292 in unpacked form.

Sources: `www.symantec.com/security_response/writeup.jsp?docid=2004-061110-4018-99&tabid=2`
`www.viruslist.com/en/viruses/encyclopedia?virusid=51876`

Netsky-P

Date: 21.03.2004

Description: W32.Netsky.P@mm is a mass-mailing worm that sends itself to email addresses it gathers from certain files on the system. The worm also tries to spread itself via various file-sharing methods by copying itself into directories using enticing filenames [53].

Special: Top 1 of Sophos malware List 2004. Top 2 of Sophos malware List 2005.

Spreading: Mass mailing and p2p

Recognition: Difficult

Sources: `www.symantec.com/security_response/writeup.jsp?docid=2004-032110-4938-99&tabid=2`
`www.sophos.de/security/top-10/200512summary.html`

W32.Sober@mm, Sober-A to Sober-Z

Date: 24.10.2003 and later

Description: W32.Sober@mm is a mass-mailing worm that uses its own SMTP engine to spread itself. The subject of the email varies, and it will be in either English or German [53].

Special: Many similar variants. Top 6 of Sophos malware List 2003 (Sober-A). Top 8 of Sophos malware List 2004 (Sober-I). Top 3 of Sophos malware List 2005 (Sober-Z). Top 3 of Sophos malware List 2006.

Spreading: Mass mailing

Recognition: It is written in Visual Basic and is 63KB in size.

Difficulties: Many mutantions

Sources: `www.symantec.com/security_response/writeup.jsp?docid=2003-102410-5713-99`
`www.viruslist.com/en/viruses/encyclopedia?virusid=23067`

Mytob

Date: 26.02.2005

Description: This network worm infects computers running Windows. The worm spreads via a vulnerability in the Windows LSASS service (MS04-011). It also spreads via the Internet as an attachment to infected messages. It sends itself to email addresses harvested from the victim machine [56].

Special: Top 1 of Sophos malware List 2006.

Spreading: Mass-Mail, LSASS vulnerability

Recognition: It is a Windows EXE file, approximately 43KB in size, packed using FSG. The unpacked file is approximately 143KB in size.

Difficulties: Approx 162 Variants seen in wild.

Sources: `www.symantec.com/security_response/writeup.jsp?docid=2005-022614-4627-99&tabid=2`
`www.viruslist.com/en/viruses/encyclopedia?virusid=74564`

Netsky.P

Date: 21.03.2004

Description: W32.Netsky.P@mm is a mass-mailing worm that sends itself to email addresses it gathers from certain files on the system. The worm also tries to spread itself via various file-sharing methods by copying itself into directories using enticing filenames.

Special: Top 1 of Sophos malware List 2004 (Netsky-P). Top 2 of Sophos malware List 2005 (Netsky-P). Top 2 of Sophos malware List 2006.

Spreading: Mass mail and p2p

Recognition: difficult (size)

Sources: `www.symantec.com/security_response/writeup.jsp?docid=2004-032110-4938-99&tabid=2`

Appendix A

Abbreviations for Measured Metrics

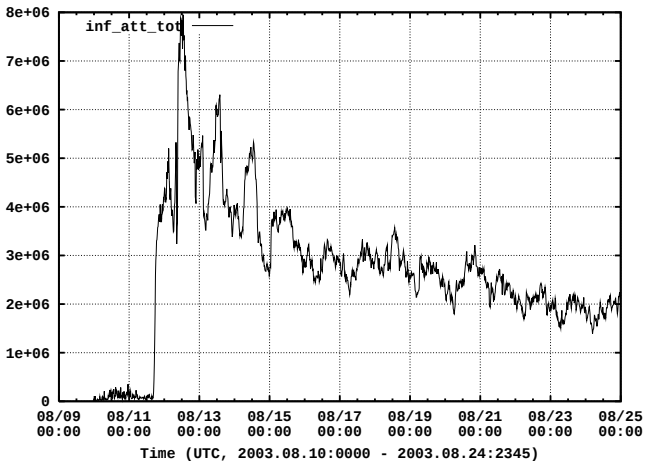
inf_att_from_ins	number of infection attempt flows coming from AS559
inf_att_from_outs	number of infection attempt flows coming from outside AS559
inf_att_tot	Infection attempt flows total
scn_hsts_ins	Number of scanning hosts placed inside AS559, using a threshold of 20 necessary scans per interval
scn_hsts_outs	Number of scanning hosts placed outside AS559 using a threshold of 20 necessary scans per interval
scn_hsts_tot	Total number of scanning hosts using a threshold of 20 necessary scans per interval
sscn_hsts_ins	Number of scanning hosts placed inside AS559, no scan threshold
sscn_hsts_outs	Number of scanning hosts placed outside AS559, no scan threshold
sscn_hsts_tot	Total number of scanning hosts, no scan threshold
usip_ins_ano	Unique source IPs of anomaly traffic inside AS559
usip_ins_nor	Unique source IPs of normal traffic inside AS559
usip_ins_tot	Unique source IPs of total traffic

flows_ano	Number of anomaly flows
flows_nor	Number of normal traffic flows
flows_tot	Total number of flows per interval
num_AS559_IPs	Number IPs assigned to AS559
r_ano_tot_flows	Ratio of anomaly / total flows
r_ano_tot_usip	Ratio of anomaly / total unique source IPs

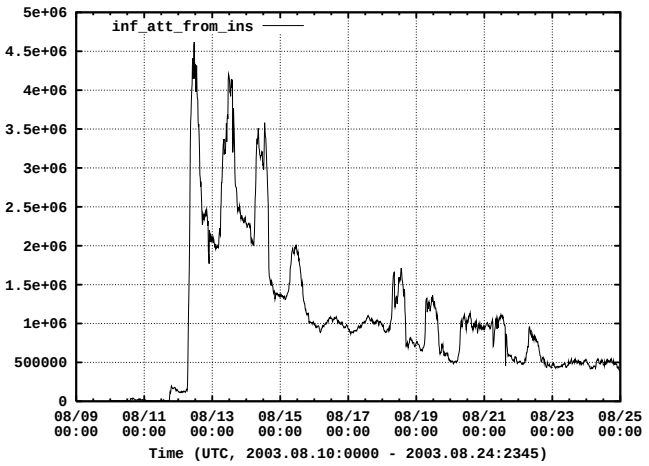
hashedIP	Hashed (anonymized) IP
ti_unsucc_ia	Time(first unsuccessful stage A infection attempt)
ti_succ_ia	Time(first seen successful infection attempt over boarder routers)
ti_first_scan	Time(first scan sent)
ti_last_scan	Time(last scan sent)
ti_normal_tra	Time(normal traffic sent)
sIP_succ_ia	Source IP of first successful attempt, attacker IP
scans_recv	Number of scans received (after successful infection)
scans_sent	Number of scans sent
AS	Autonomous system
ti_recovery	Time(guessed recovery)

Appendix B

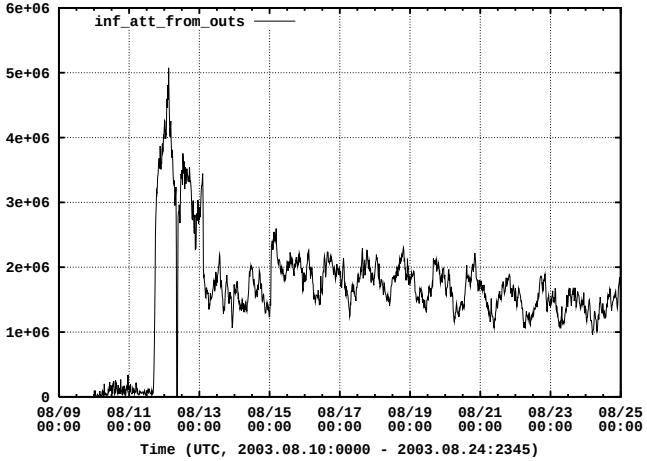
W32_Blaster time series



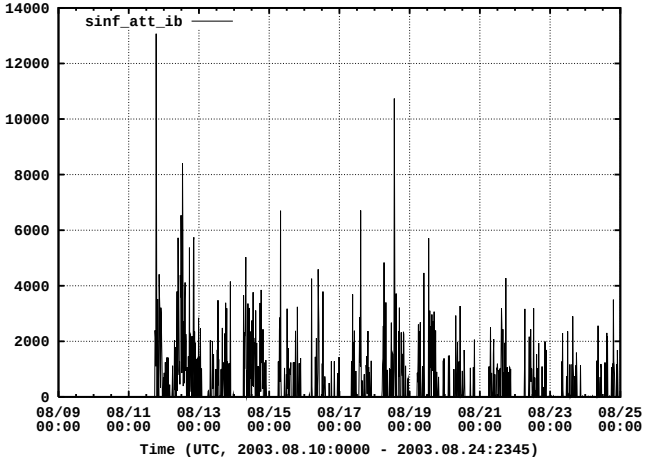
(a) Infection attempts total



(b) Infection attempts from AS559

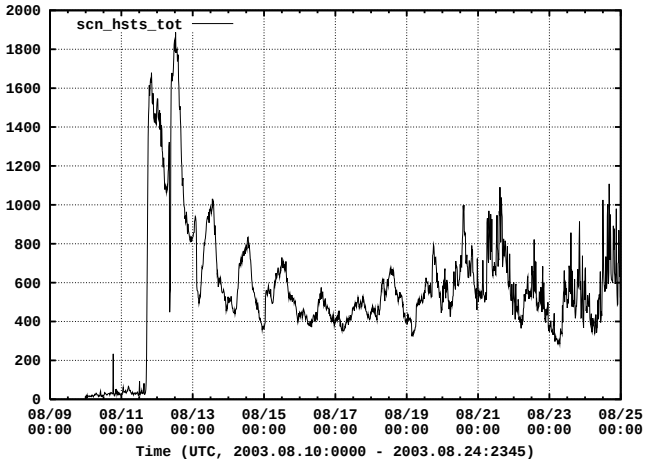


(c) Infection attempts from outside

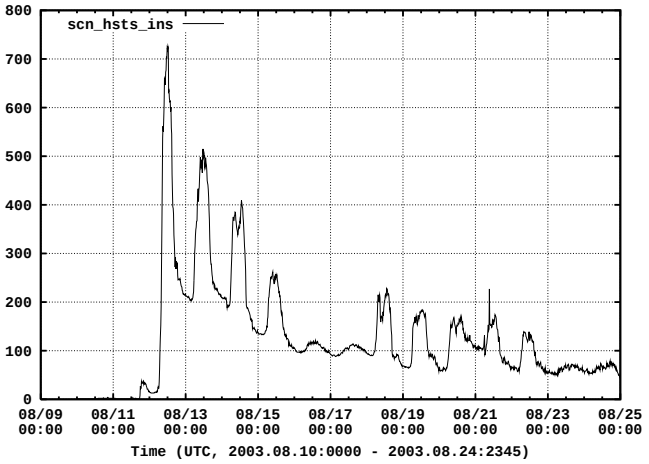


(d) Infection attempts routed externally (from AS559 to AS559 over boarder routers)

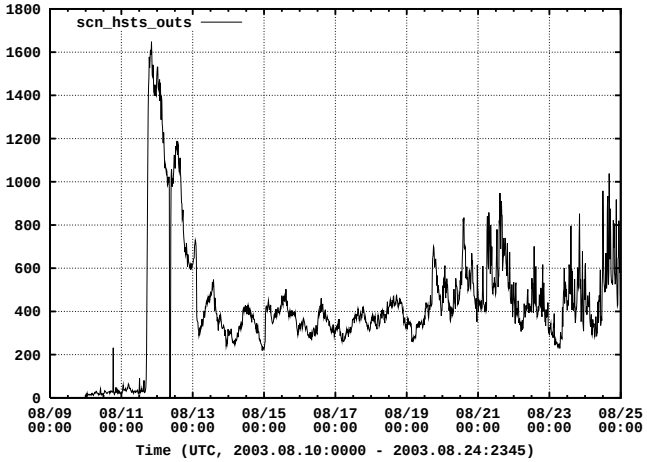
Figure B.1: Blaster infection attempts, 15min bins (The number of TCP SYNs sent to destination port 135 aggregated in 15min intervals. Beneath the total traffic over the boarder routers, we distinguish between traffic coming from AS559 and the remaining Internet)



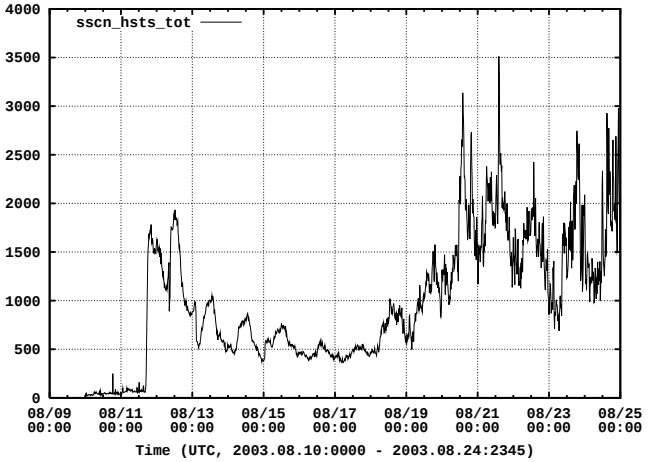
(a) Scanning hosts total



(b) Scanning hosts located in AS559

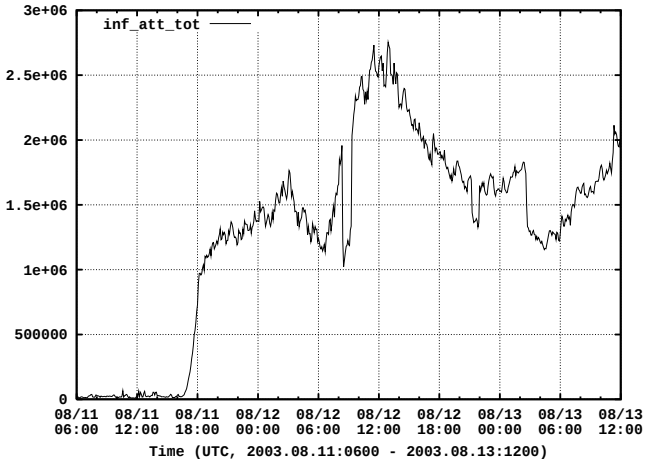


(c) Scanning hosts located outside AS559

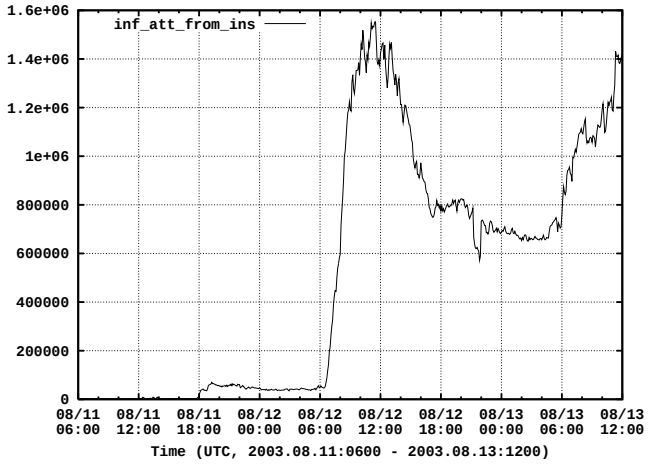


(d) Scanning hosts total, simpler detection algorithm

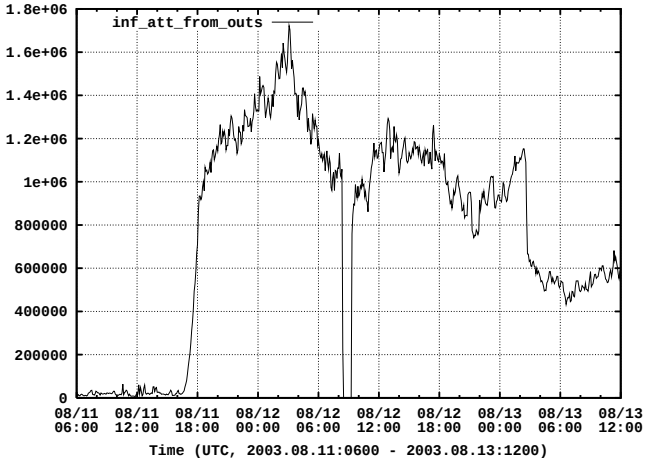
Figure B.2: Blaster, scanning hosts, 15min bins (Unique source IPs generating TCP SYN scans to destination port 135 aggregated in 15min intervals. Beneath the total traffic over the boarder routers, we distinguish between traffic coming from AS559 and the remaining Internet. Source IPs generating less than 20 scans are not counted (except Subfig (d).)



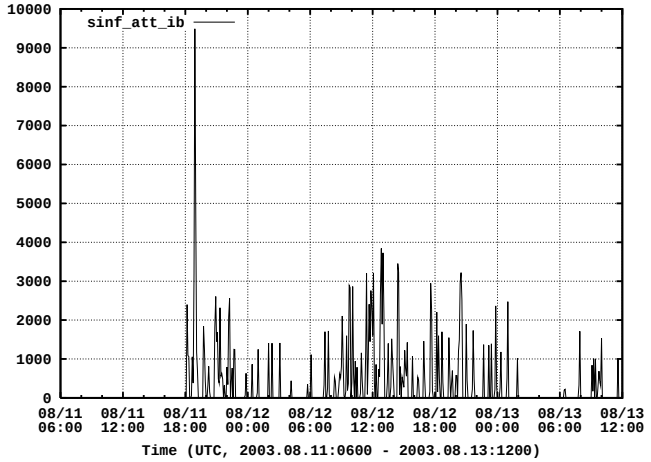
(a) Infection attempts total



(b) Infection attempts from AS559

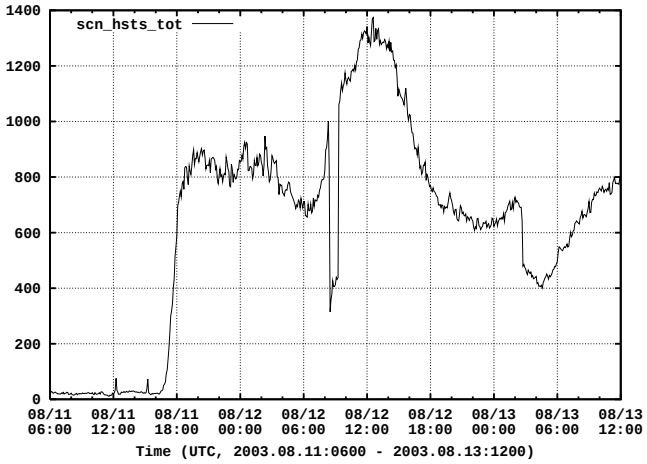


(c) Infection attempts from outside

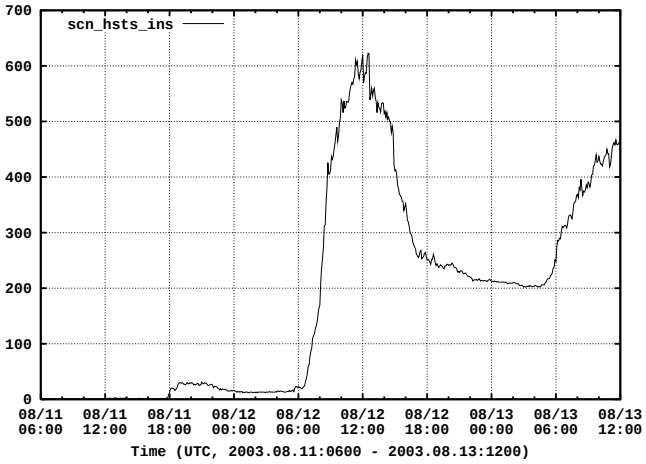


(d) Externally routed infection attempts (from AS559 to AS559

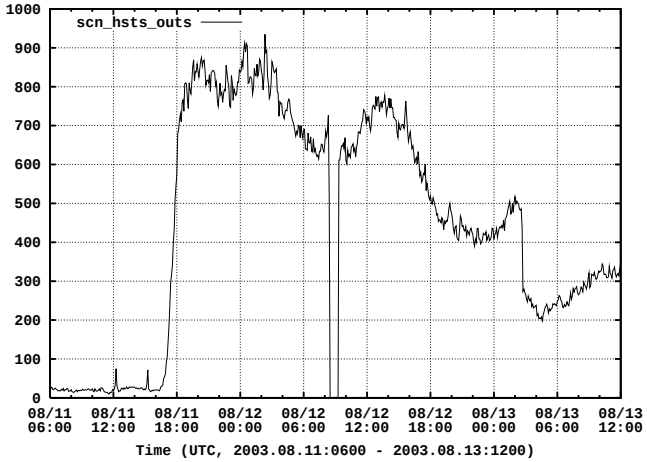
Figure B.3: Blaster infection attempts, 5min bins (The number of TCP SYNs sent to destination port 135 aggregated in 5min intervals. Beneath the total traffic over the boarder routers, we distinguish between traffic coming from AS559 and the remaining Internet)



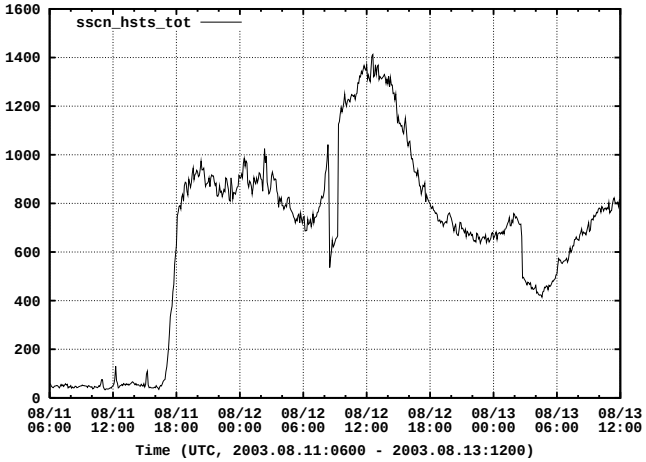
(a) Scanning hosts total



(b) Scanning hosts located in AS559

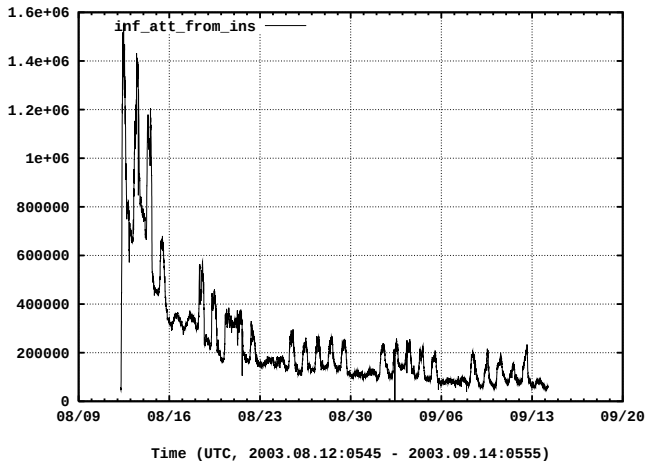


(c) Scanning hosts located outside AS559

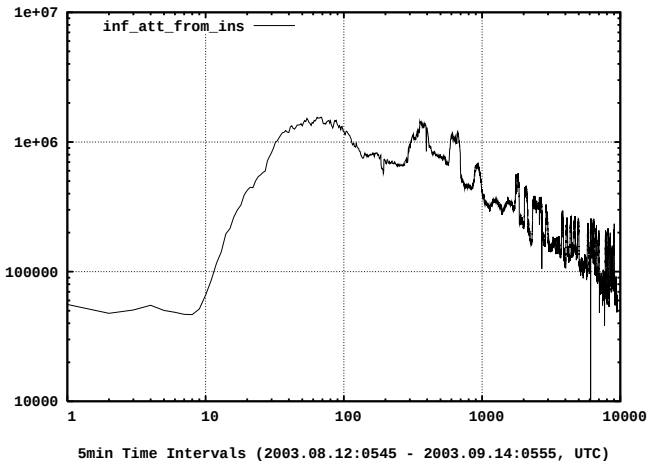


(d) Scanning hosts total, simpler detection algorithm

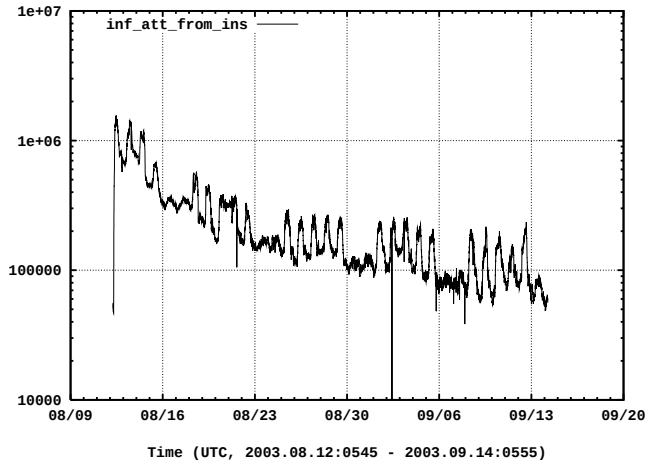
Figure B.4: Blaster, scanning hosts, 5min bins (Unique source IPs generating TCP SYN scans to destination port 135 aggregated in 5min intervals. Beneath the total traffic over the boarder routers, we distinguish between traffic coming from AS559 and the remaining Internet. Source IPs generating less than 20 scans are not counted (except Subfig (d).)



(a) lin(t) lin(y)

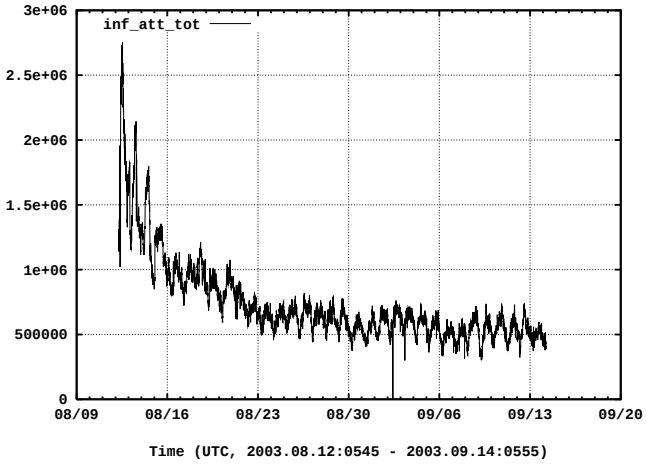


(b) log(t), log(y)

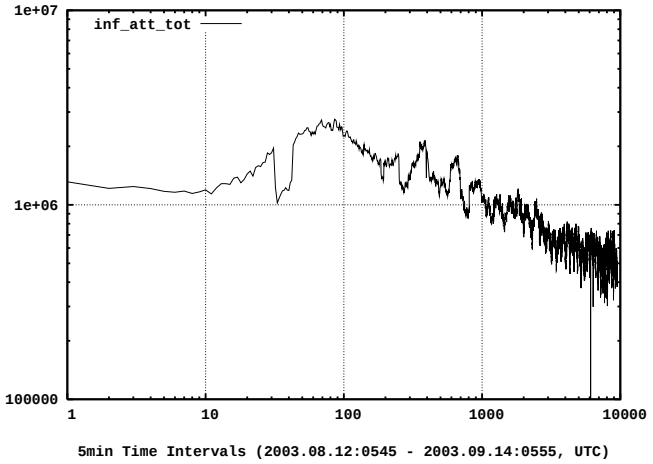


(c) lin(t), log(y)

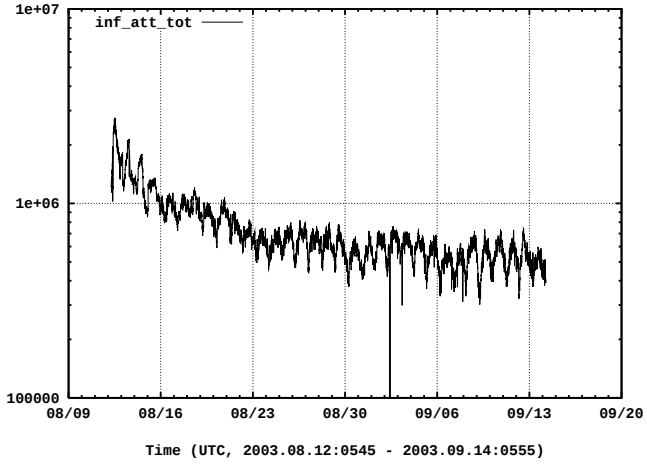
Figure B.5: Blaster infection attempts from AS559 , 5min bins, linear / logarithmic scale compare (The number of TCP SYNs sent to destination port 135 aggregated in 5min intervals in different scales. Only traffic coming from AS559 is shown)



(a) lin(t) lin(y)

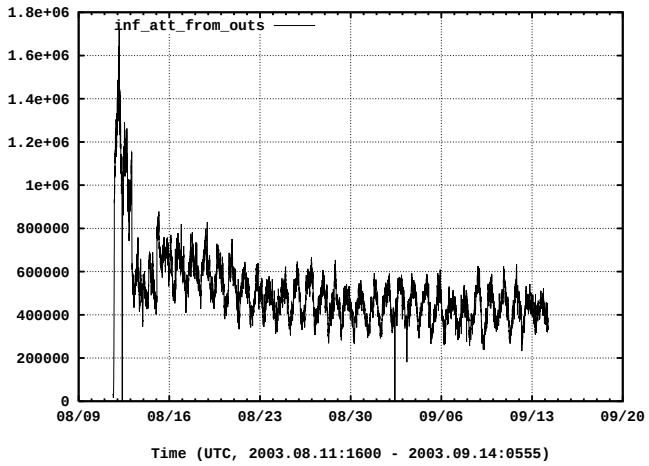


(b) log(t), log(y)

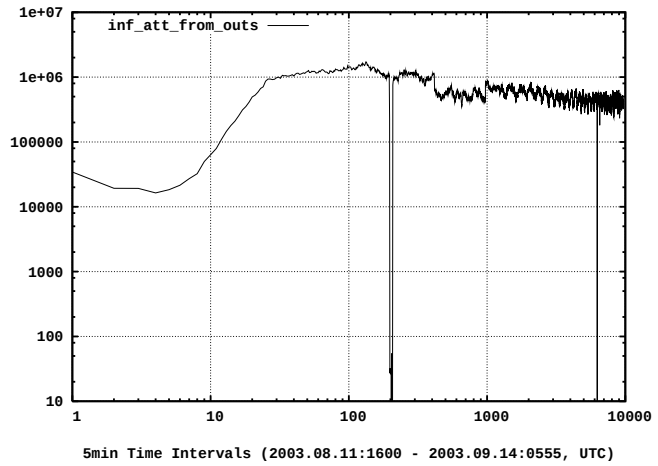


(c) lin(t), log(y)

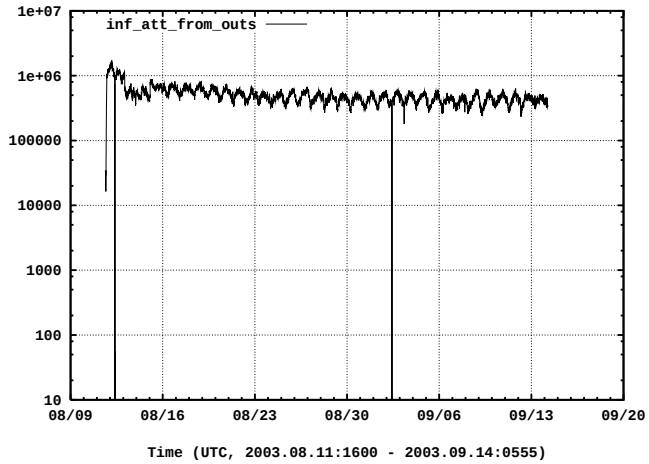
Figure B.6: Blaster infection attempts total, 5min bins, linear / logarithmic scale compare (The number of TCP SYNs sent to destination port 135 aggregated in 5min intervals in different scales. All traffic over boarder routers is shown.)



(a) lin(t) lin(y)

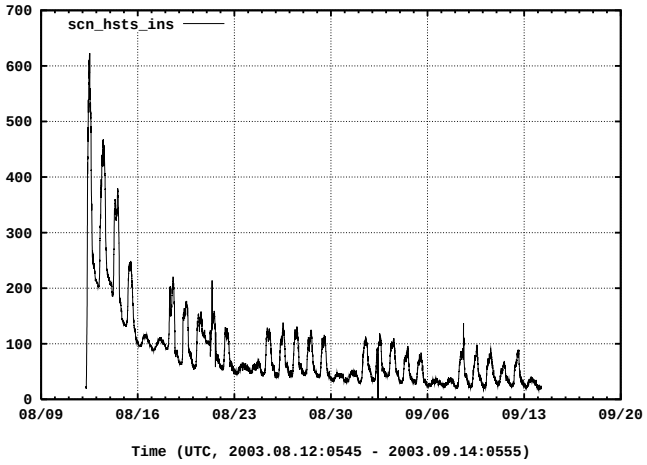


(b) log(t), log(y)

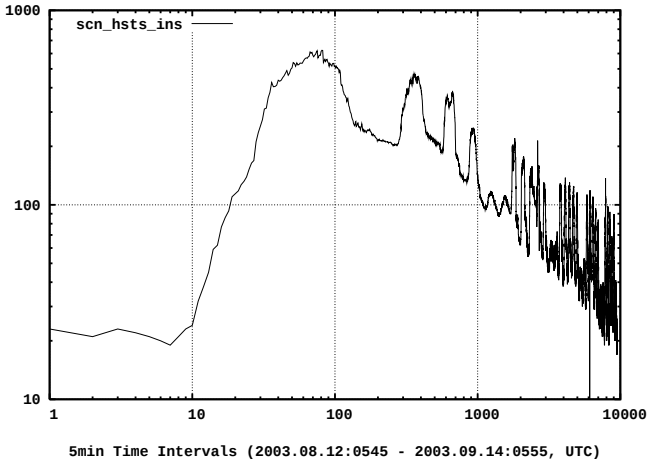


(c) lin(t), log(y)

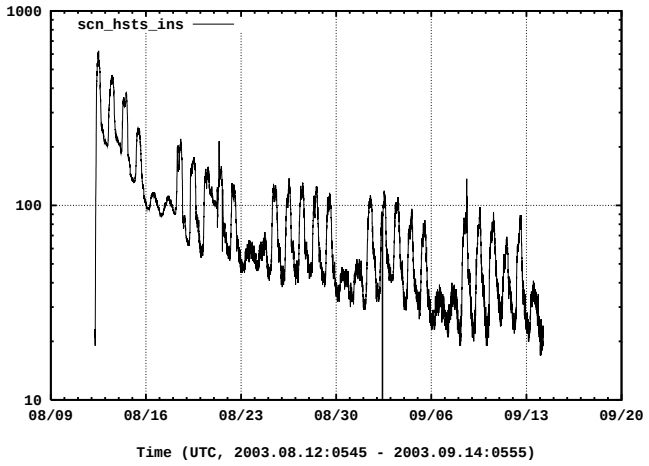
Figure B.7: Blaster infection attempts from outside AS559, 5min bins, linear / logarithmic scale compare (The number of TCP SYNs sent to destination port 135 aggregated in 5min intervals in different scales. Only traffic coming from Internet to AS559 is shown.)



(a) lin(t) lin(y)

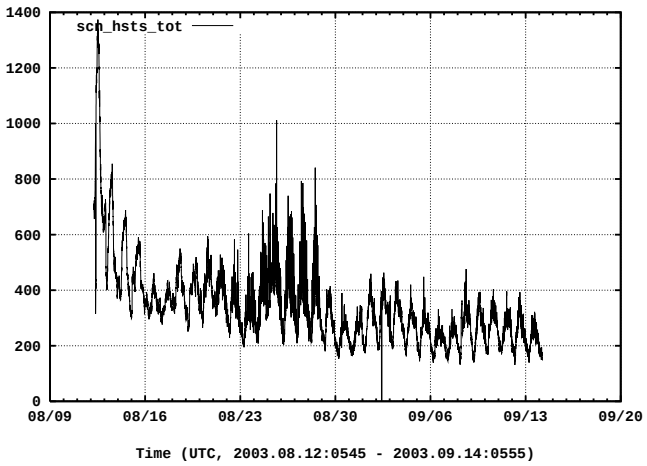


(b) log(t), log(y)

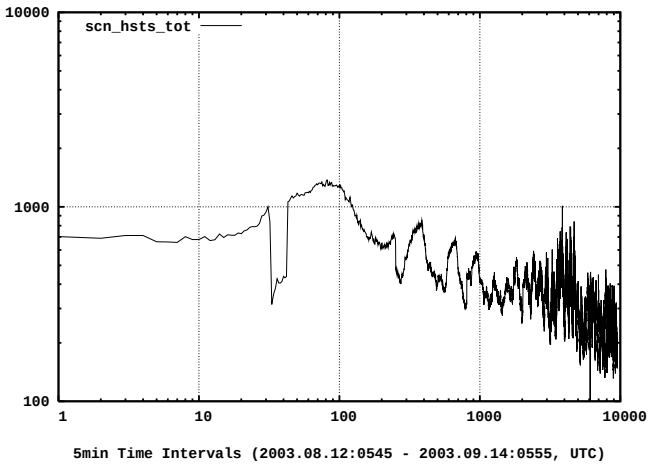


(c) lin(t), log(y)

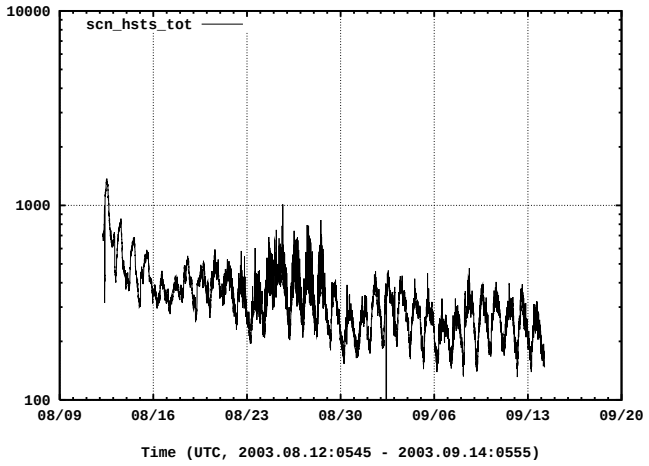
Figure B.8: Blaster, scanning hosts located in AS559 , 5min bins, linear / logarithmic scale compare (Unique source IPs generating TCP SYN scans to destination port 135 aggregated in 5min intervals in different axis scales. Only host within AS559 are shown. Source IPs generating less than 20 scans are not counted.)



(a) lin(t) lin(y)

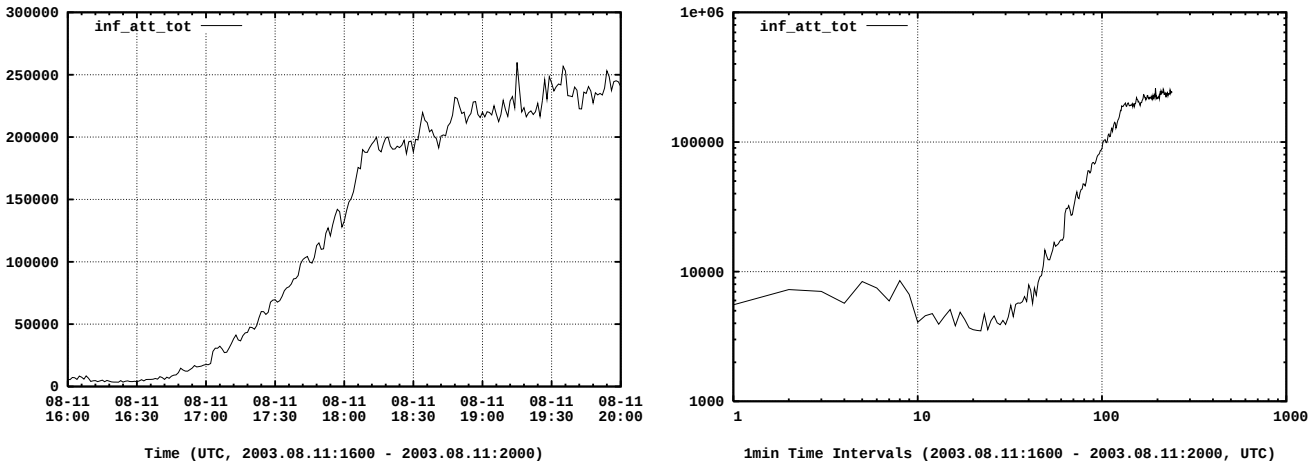


(b) log(t), log(y)



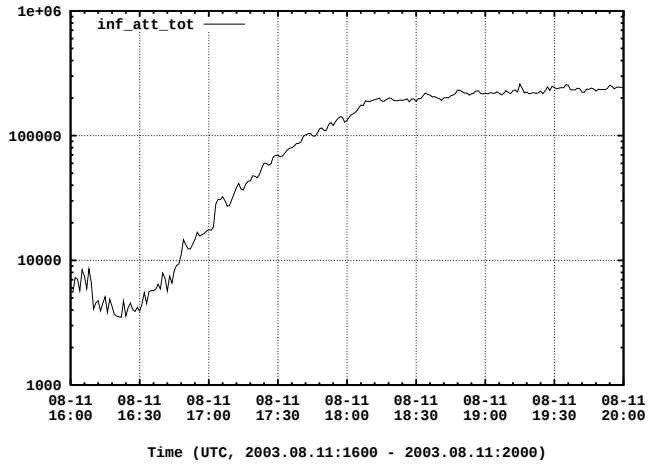
(c) lin(t), log(y)

Figure B.9: Blaster, scanning hosts total, 5min bins, linear / logarithmic scale compare (Unique source IPs generating TCP SYN scans to destination port 135 aggregated in 5min intervals in different axis scales. All source IPs seen on boarder routers are counted. Source IPs generating less than 20 scans are not counted.)



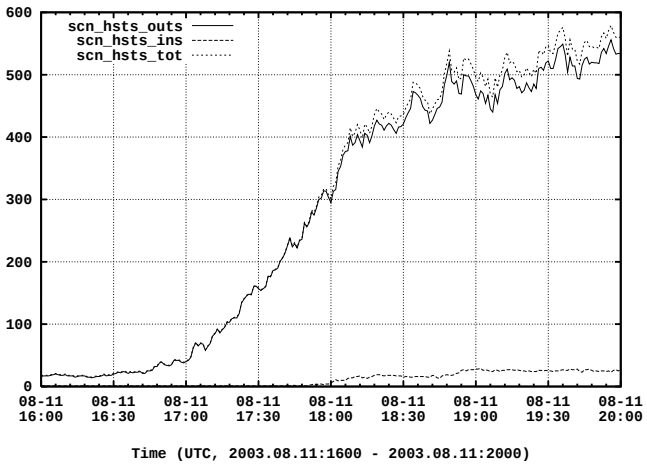
(a) lin(t) lin(y)

(b) log(t), log(y)

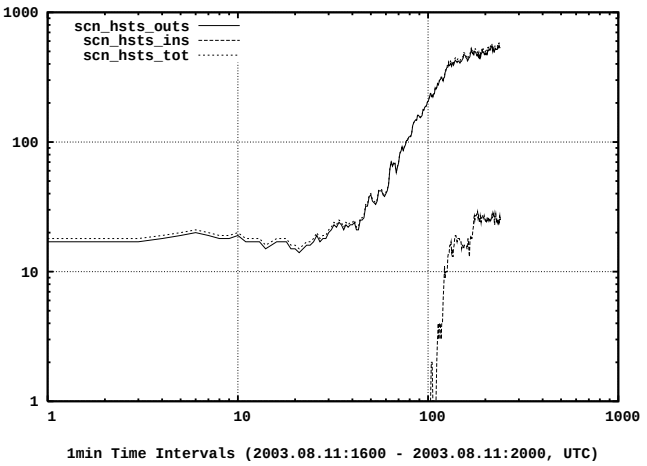


(c) lin(t), log(y)

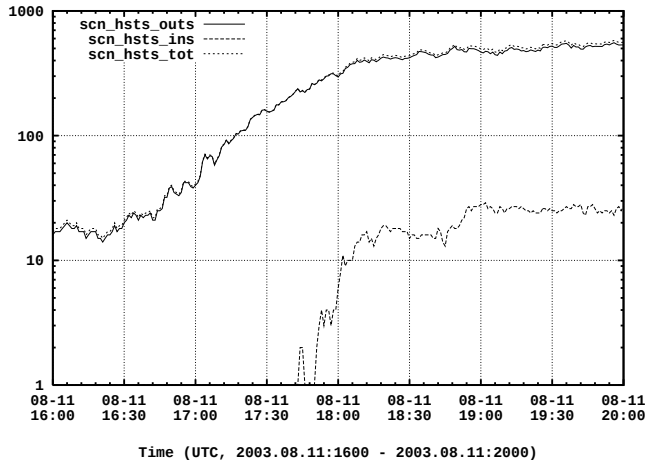
Figure B.10: Blaster, infection attempts, external outbreak, 1min bins, linear / logarithmic scale compare (Detail view of the external outbreak. The number of TCP SYNs sent to destination port 135 aggregated in 1min intervals in different scales. Only traffic coming from Internet to AS559 is shown.)



(a) lin(t) lin(y)



(b) log(t), log(y)

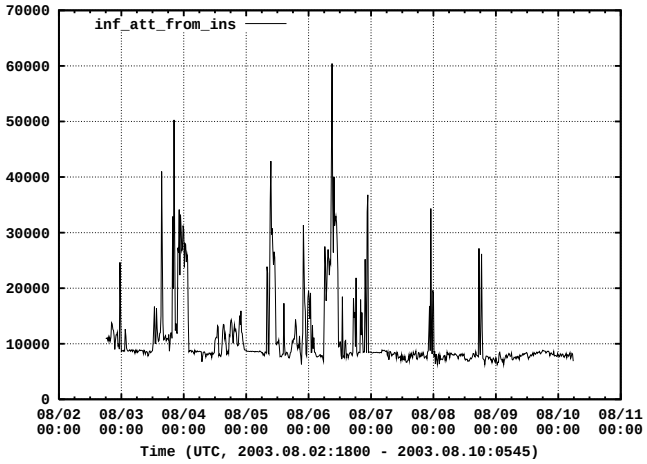


(c) lin(t), log(y)

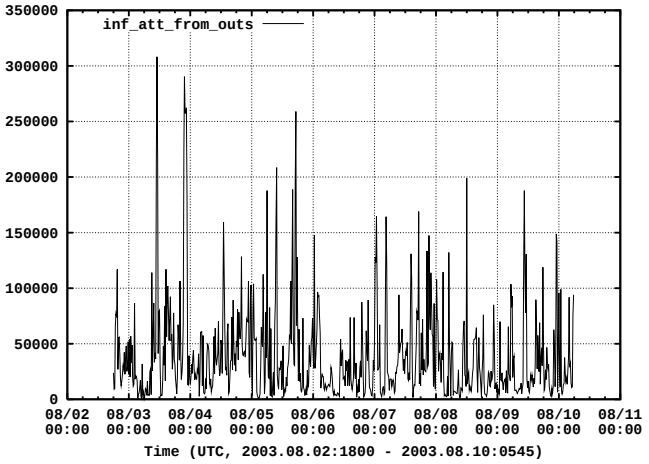
Figure B.11: Blaster, scanning hosts, external outbreak, 1min bins, linear / logarithmic scale compare (Detail view of the external outbreak. Unique source IPs generating TCP SYN to destination port 135 aggregated in 1min intervals in different scales. We distinguish between traffic coming from Internet (solid line) and AS559 (dashed line).)

Appendix C

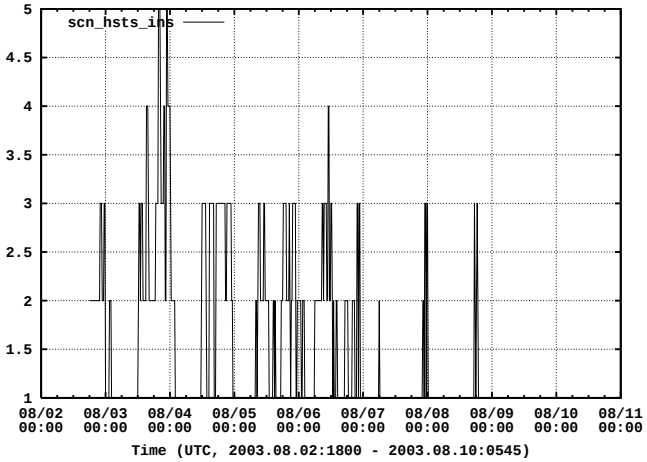
Observations of W32_Blaster one Week Before Outbreak and in Later Periods



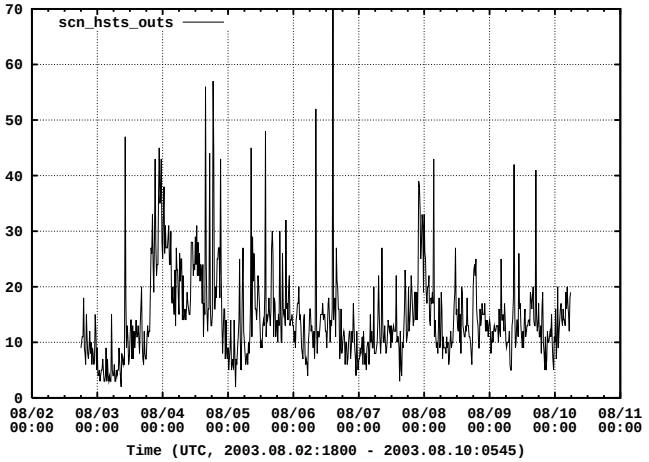
(a) Infection attempts from AS559



(b) Infection attempts from outside AS559

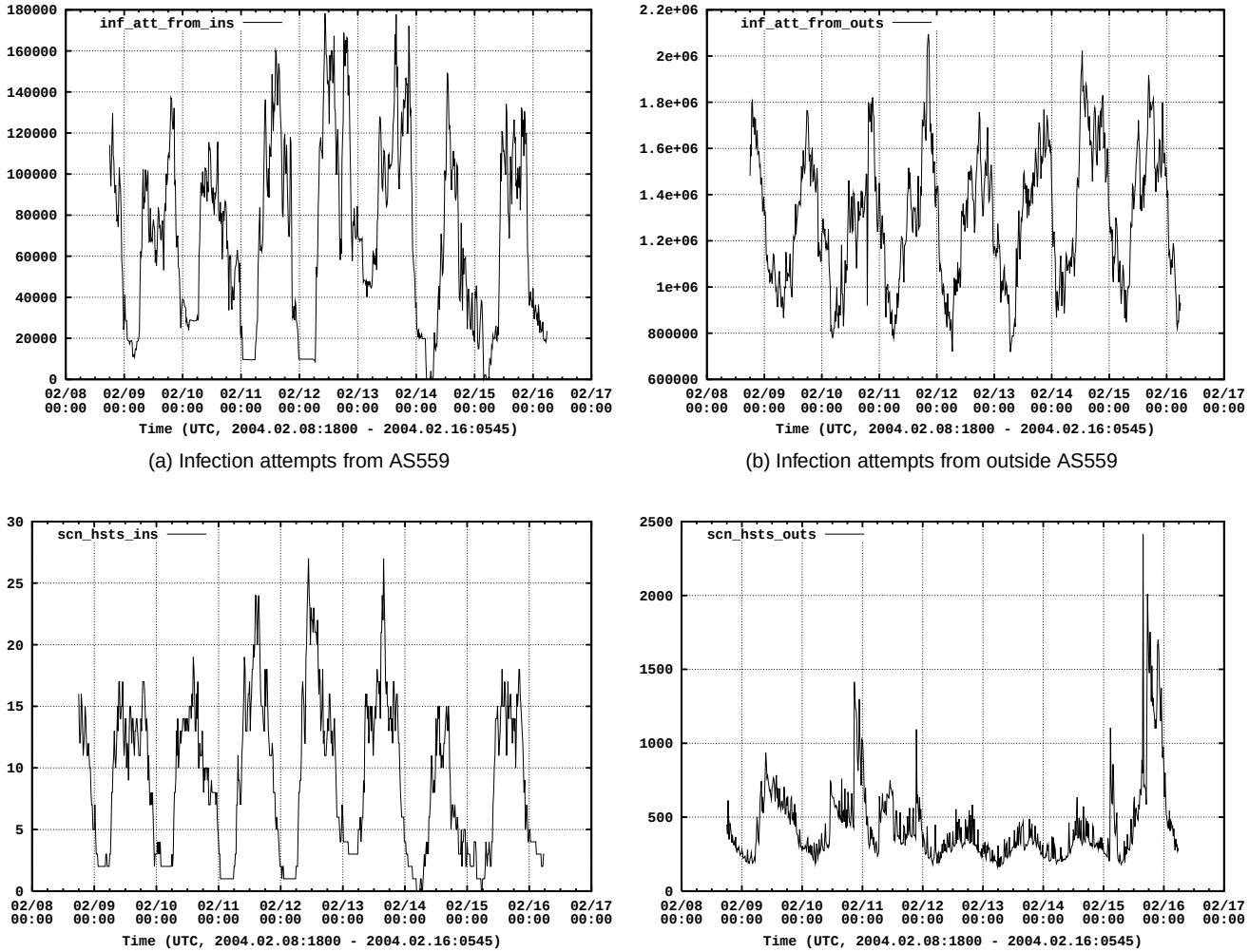


(c) Scanning hosts located inside AS559



(d) Scanning hosts located outside AS559

Figure C.1: Blaster time series, one week before outbreak, 15min bins (The number of TCP SYN packets to destination port 135 aggregated in 15 min intervals, as well as the unique source IPs generating them. We distinguish between hosts located within AS559 and the remaining Internet)



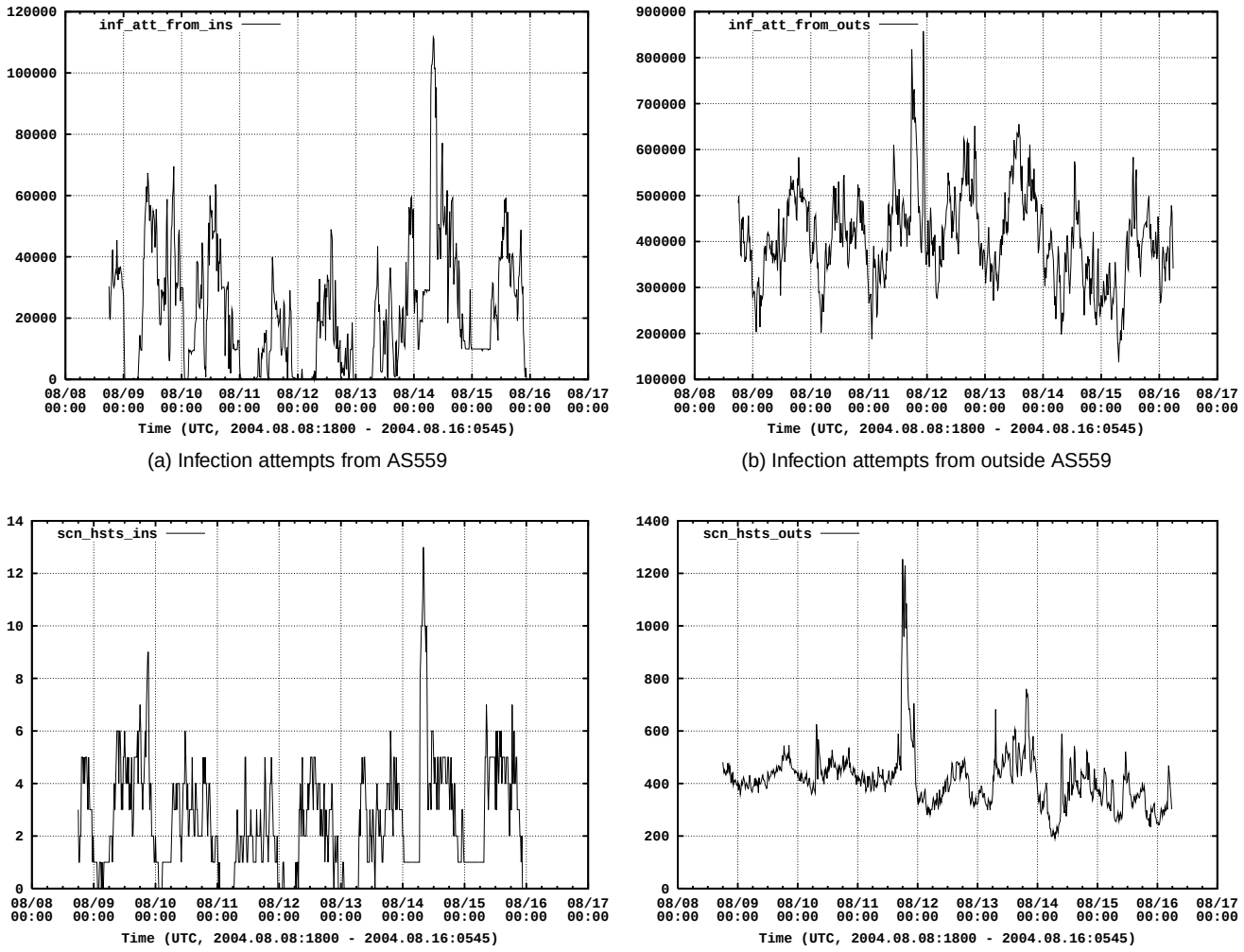
(a) Infection attempts from AS559

(b) Infection attempts from outside AS559

(c) Scanning hosts located inside AS559

(d) Scanning hosts located outside AS559

Figure C.2: Blaster time series, 6 months after outbreak, 15min bins (The number of TCP SYN packets to destination port 135 aggregated in 15 min intervals, as well as the unique source IPs generating them. We distinguish between hosts located within AS559 and the remaining Internet)



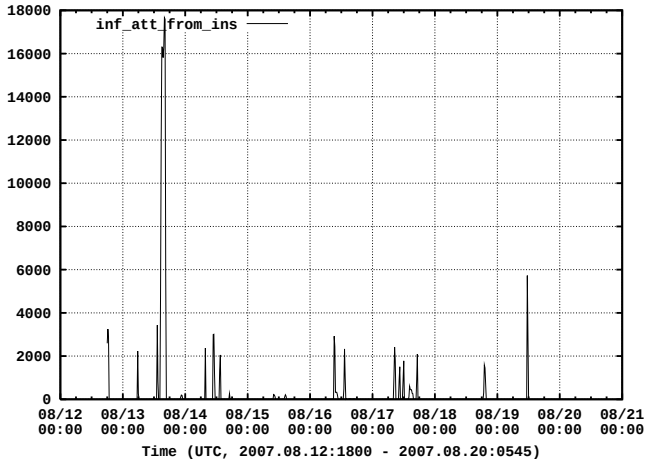
(a) Infection attempts from AS559

(b) Infection attempts from outside AS559

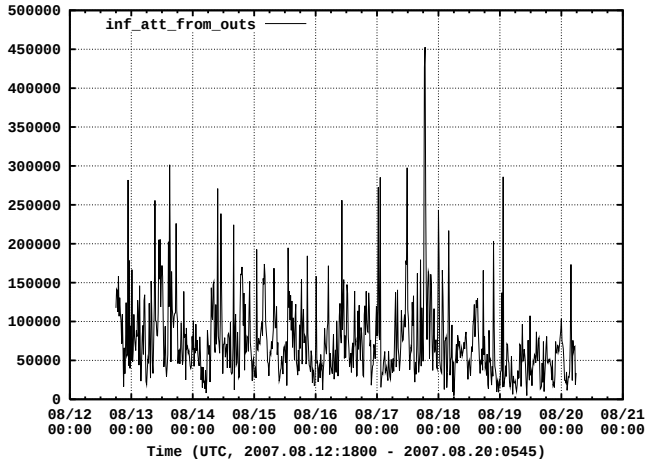
(c) Scanning hosts located inside AS559

(d) Scanning hosts located outside AS559

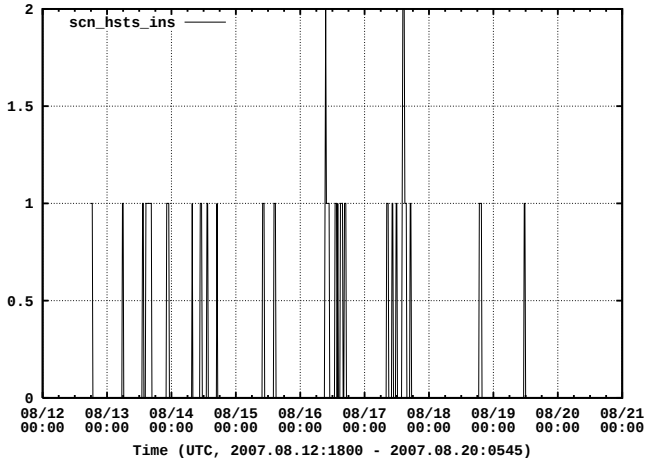
Figure C.3: Blaster time series, 12 months after outbreak, 15min bins (The number of TCP SYN packets to destination port 135 aggregated in 15 min intervals, as well as the unique source IPs generating them. We distinguish between hosts located within AS559 and the remaining Internet)



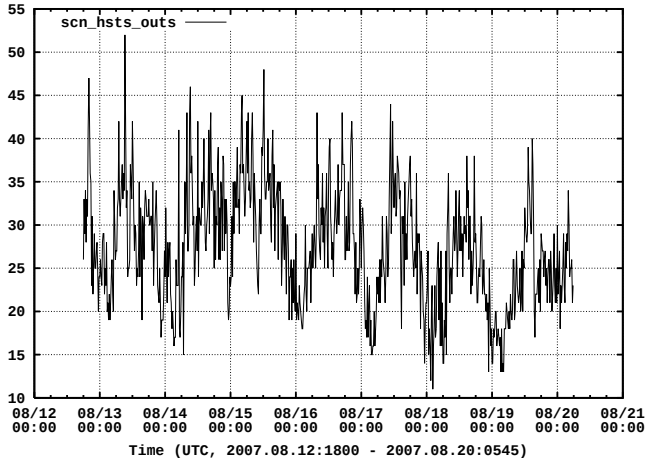
(a) Infection attempts from AS559



(b) Infection attempts from outside AS559

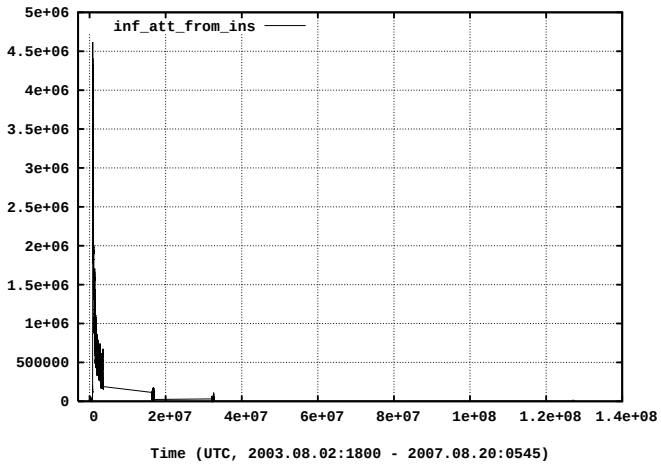


(c) Scanning hosts located inside AS559

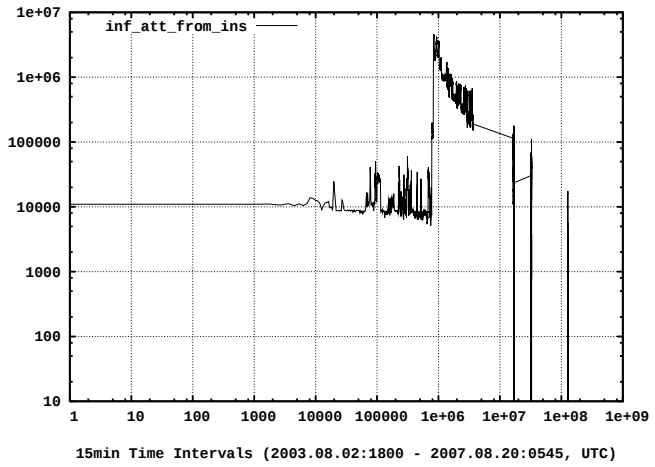


(d) Scanning hosts located outside AS559

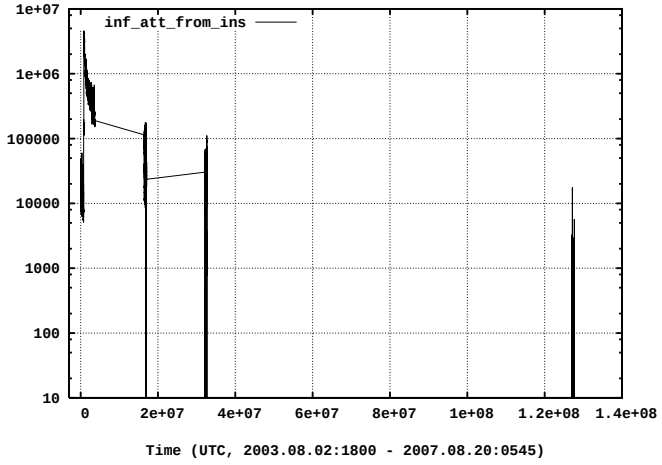
Figure C.4: Blaster time series, 4 years after outbreak, 15min bins (The number of TCP SYN packets to destination port 135 aggregated in 15 min intervals, as well as the unique source IPs generating them. We distinguish between hosts located within AS559 and the remaining Internet)



(a) lin(t), lin(y)

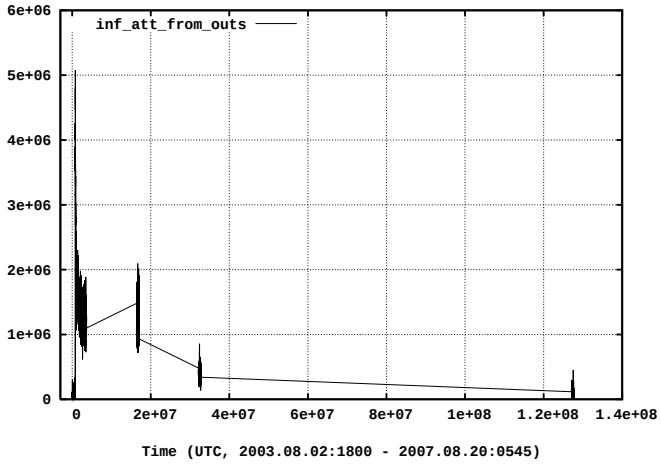


(b) log(t), log(y)

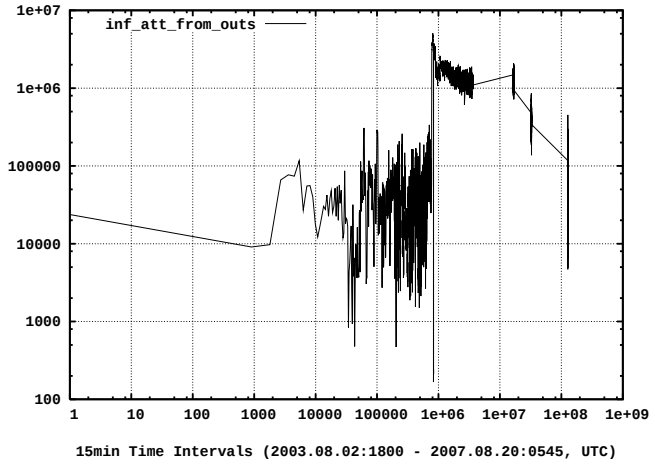


(c) lin(t), log(y)

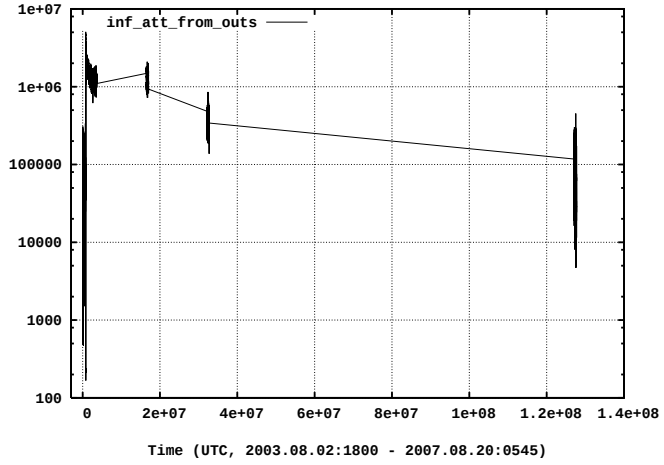
Figure C.5: Blaster, long time analysis, infection attempts from AS559, 15min bins (4 year overview in different axis scales of the TCP SYN packets sent to destination port 135 aggregated in 15 min intervals. Only traffic from AS559 is shown.)



(a) lin(t), lin(y)

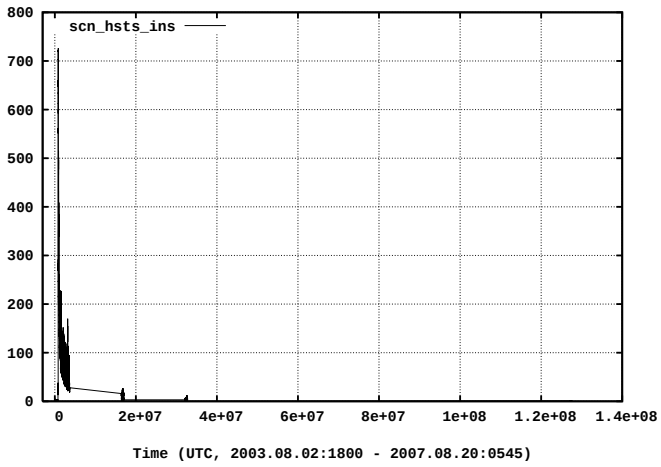


(b) log(t), log(y)

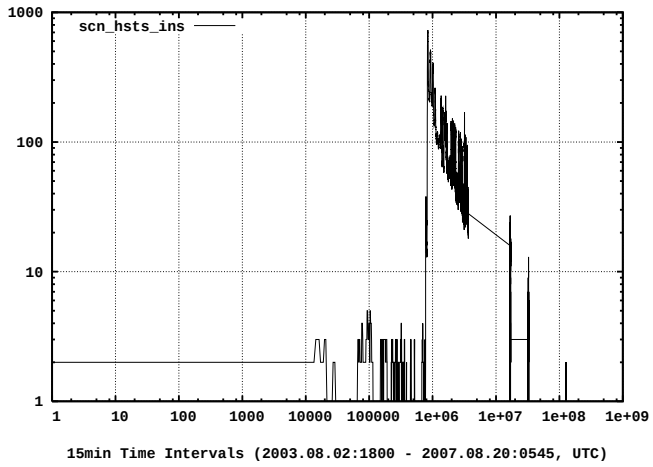


(c) lin(t), log(y)

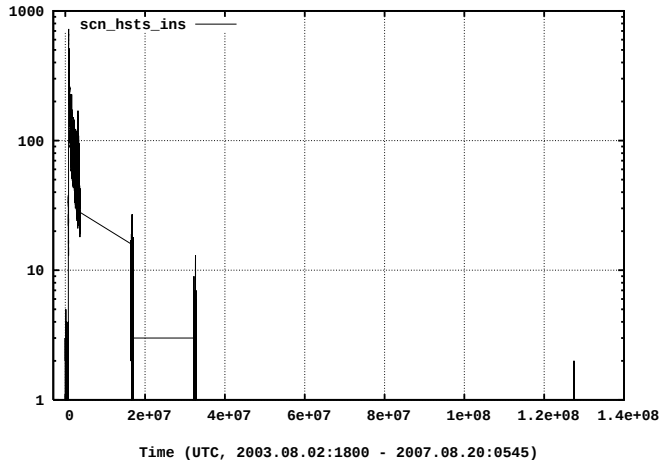
Figure C.6: Blaster, long time analysis, infection attempts from outside AS559, 15min bins (4 year overview in different axis scales of the TCP SYN packets sent to destination port 135 aggregated in 15 min intervals. Only traffic from Internet to AS559 is shown.)



(a) lin(t), lin(y)

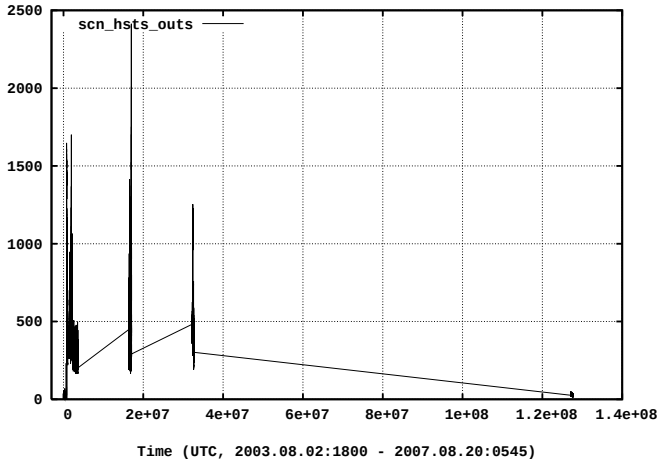


(b) log(t), log(y)

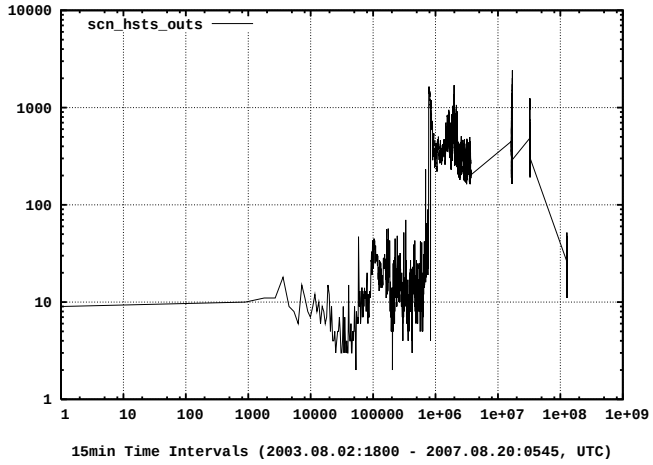


(c) lin(t), log(y)

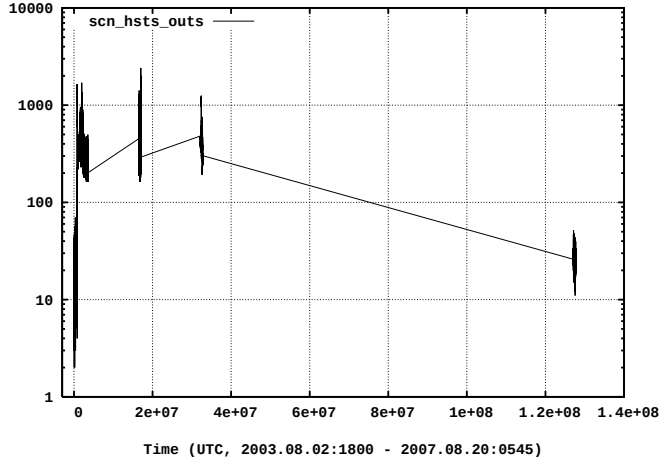
Figure C.7: Blaster, long time analysis, scanning hosts placed inside AS559, 15min bins (4 year overview in different axis scales of unique source IPs sending TCP SYN packets to destination port 135 aggregated in 15 min intervals. Only hosts within AS559 are shown.)



(a) lin(t), lin(y)



(b) log(t), log(y)

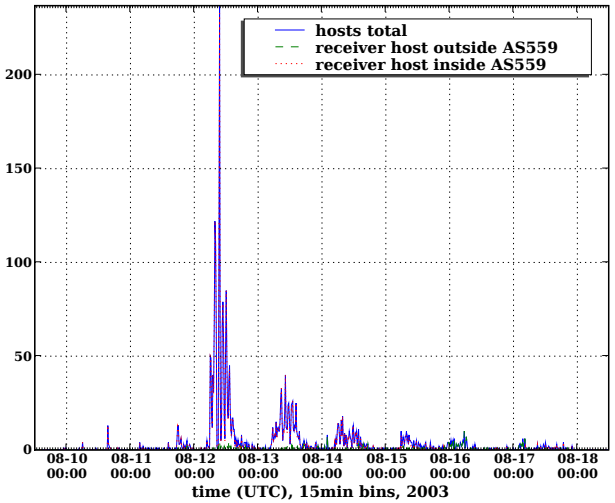


(c) lin(t), log(y)

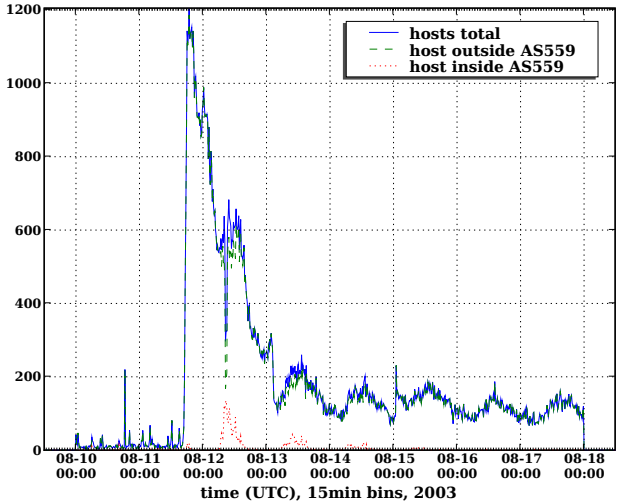
Figure C.8: Blaster, long time analysis, scanning hosts placed outside AS559, 15min bins (4 year overview in different axis scales of unique source IPs sending TCP SYN packets to destination port 135 aggregated in 15 min intervals. Only hosts outside AS559 are shown.)

Appendix D

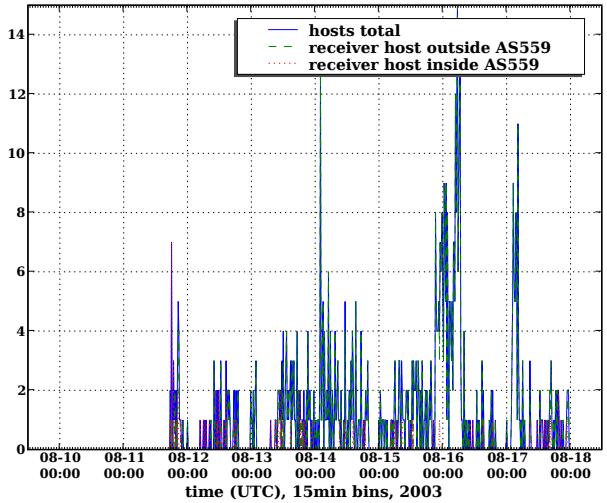
Plots, host based metrics



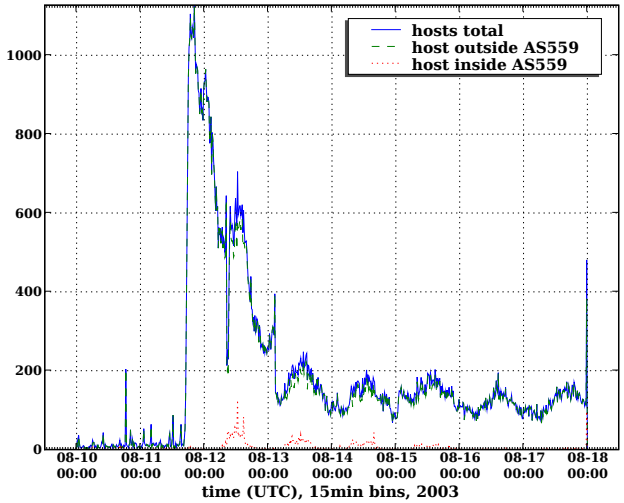
(a) time of first unsuccessful infection attempts (target not existent, switched off, wrong exploit)



(b) Time of first scan (number of *new* infected hosts per interval)

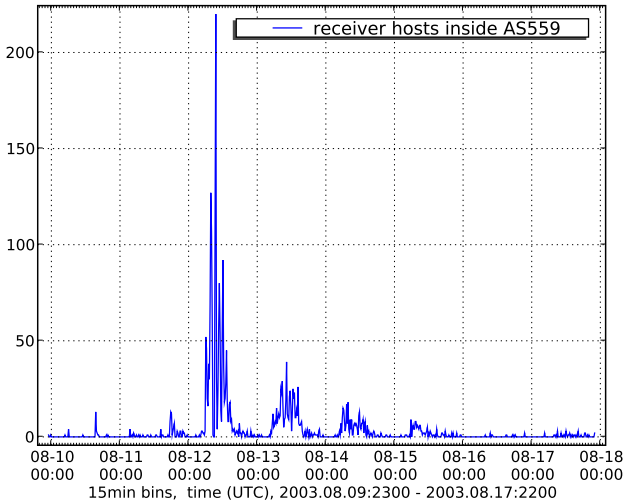


(c) Time of (complete multistage) infection over AS559 borders

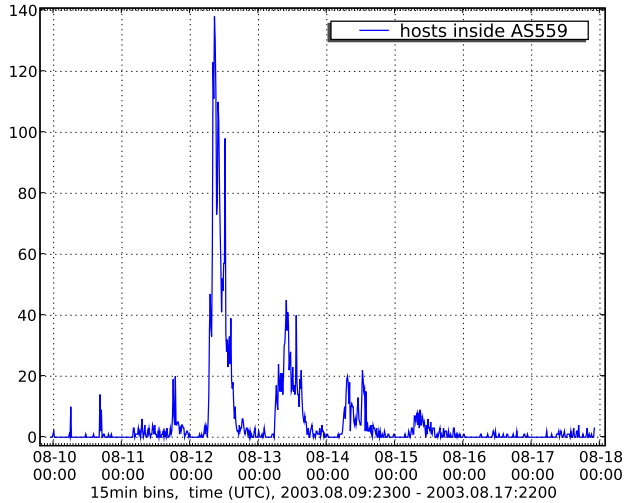


(d) Time of last scan

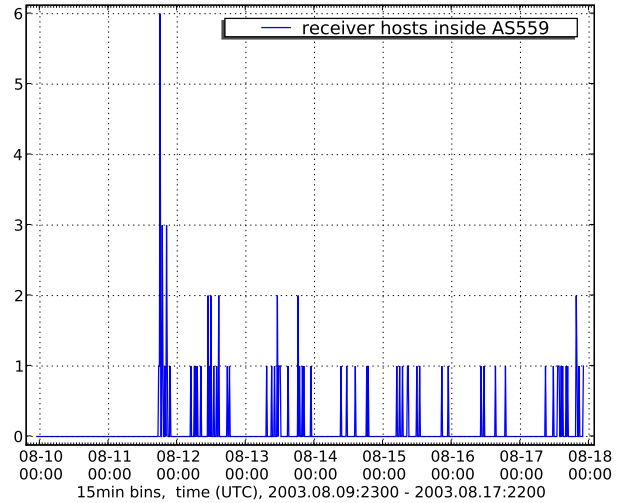
Figure D.1: Blaster, time series of unique events, 15min bins (Time of first and last occurrence of a TCP SYN to destination port 135 coming from a given IP, as well as the time of the first unsuccessful TCP SYN connection. Subfigure (c) counts the hosts successfully infected over the boarder routers.)



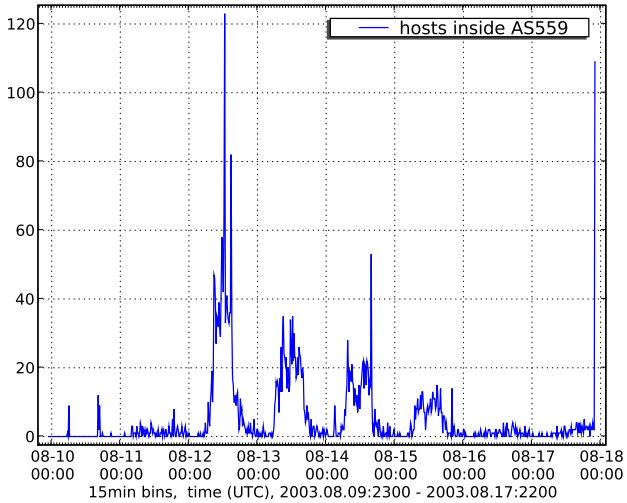
(a) time of first unsuccessful infection attempts (target not existent, switched off, wrong exploit)



(b) Time of first scan (number of *new* infected hosts per interval)

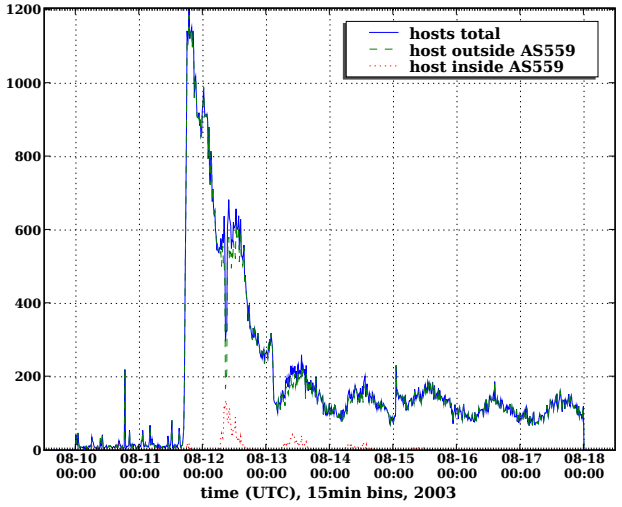


(c) Time of (complete multistage) infection over AS559 borders

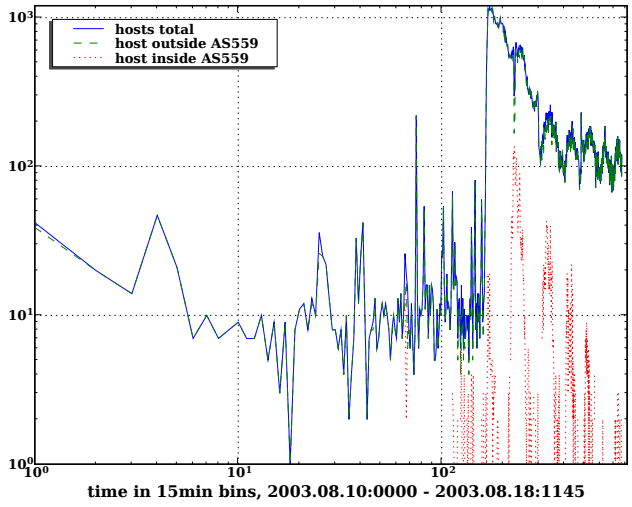


(d) Time of last scan

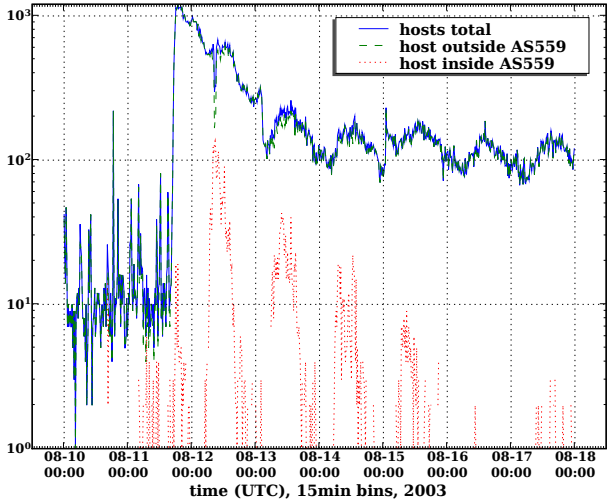
Figure D.2: Blaster, time series of unique events, 15min bins, AS559 (Time of first and last occurrence of a TCP SYN to destination port 135 coming from a given IP, as well as the time of the first unsuccessful TCP SYN connection. Subfigure (c) counts the hosts successfully infected over the boarder routers. The statistics is only shown for hosts belonging to AS559.)



(a) lin(t) lin(y)

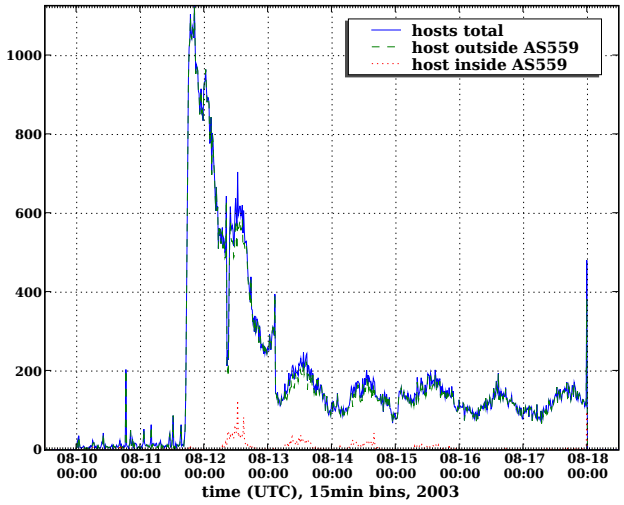


(b) log(t), log(y)

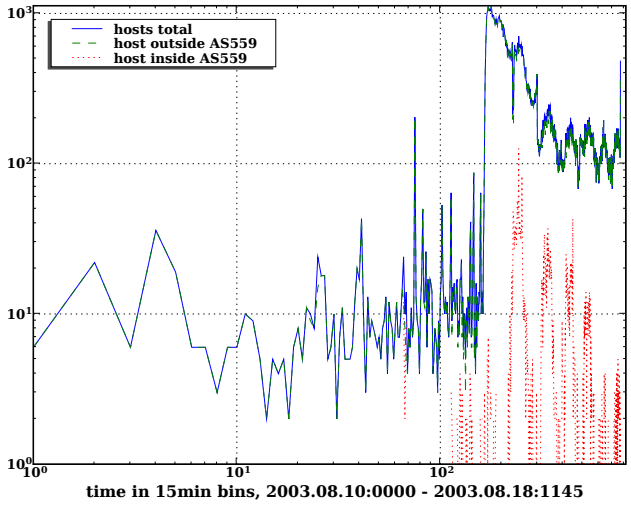


(c) lin(t), log(y)

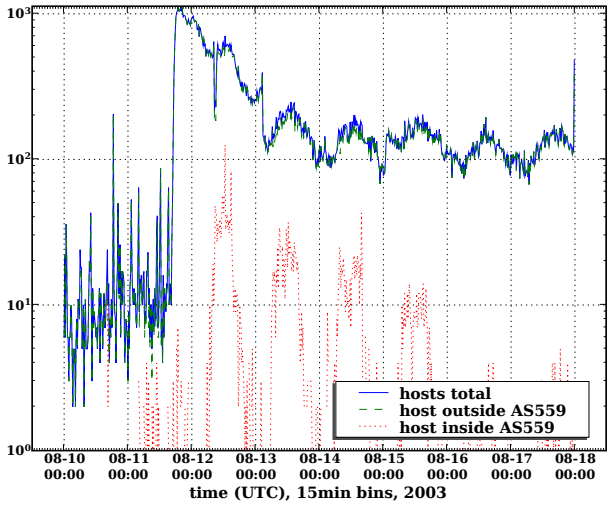
Figure D.3: Time of first scan (number of *new* infected hosts per interval), 15min bins, linear / logarithmic scale compare (Time of first occurrence of a TCP SYN to destination port 135 coming from a given IP in different axis scales. We distinguish between hosts inside and outside AS559.)



(a) lin(t) lin(y)

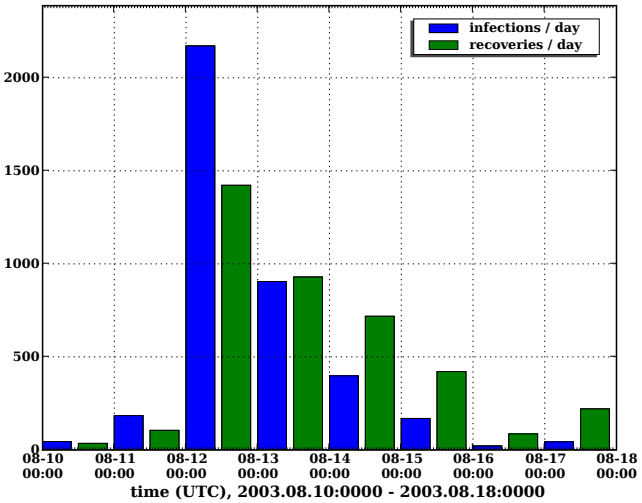


(b) log(t), log(y)

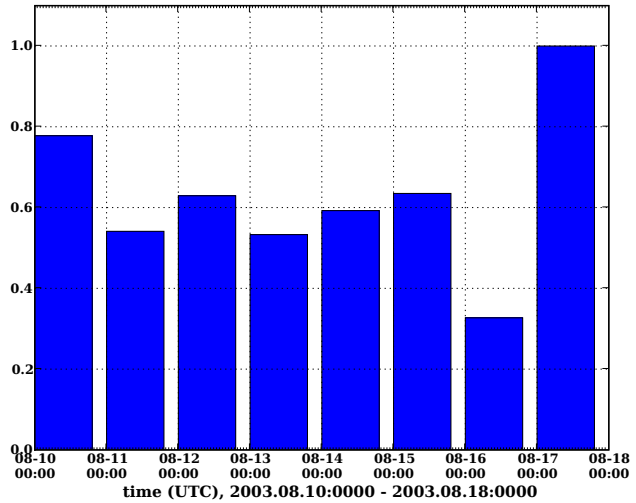


(c) lin(t), log(y)

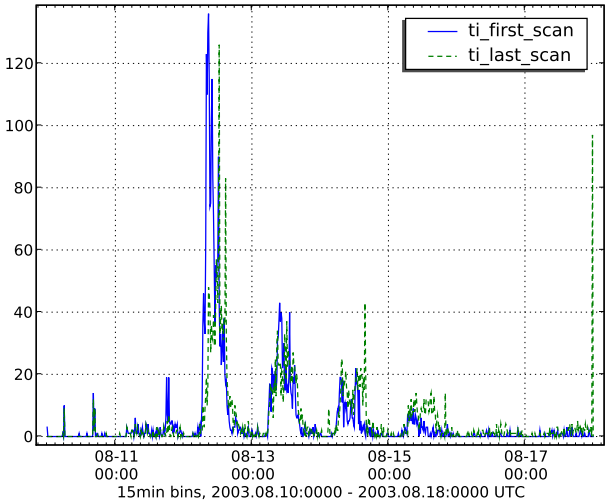
Figure D.4: Time of last scan, 15min bins, linear / logarithmic scale compare (Time of last occurrence of a TCP SYN to destination port 135 coming from a given IP in different axis scales. We distinguish between hosts inside and outside AS559.)



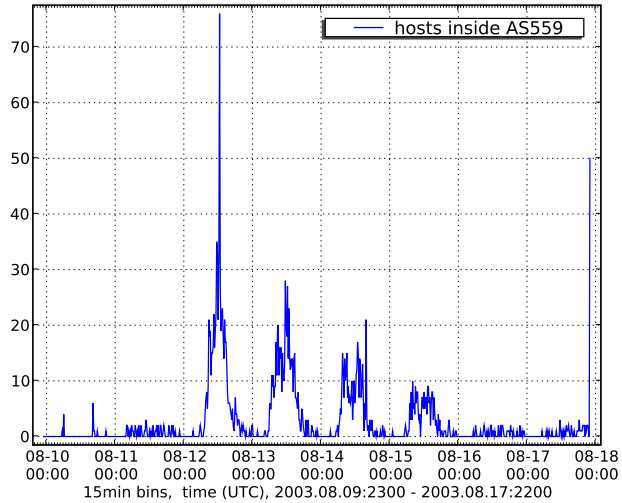
(a) infections / day, recoveries / day, AS559



(b) recovery ratio = #recovieries/day / (#infections / day + #unrecovered hosts from yesterday), AS559

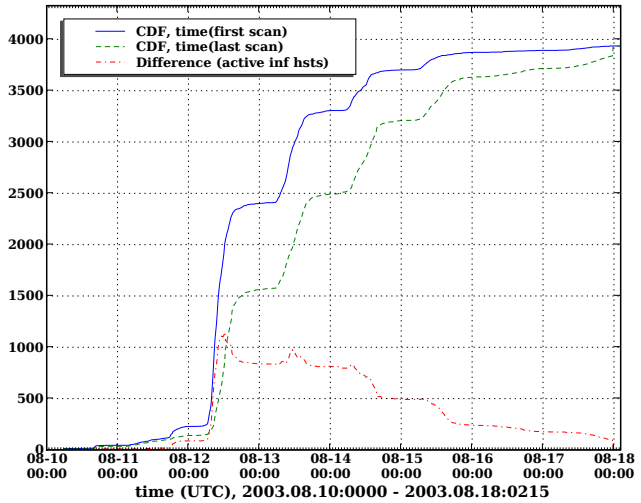


(c) time of first scans and time of last scans per interval for each infected source IP, 15min bins

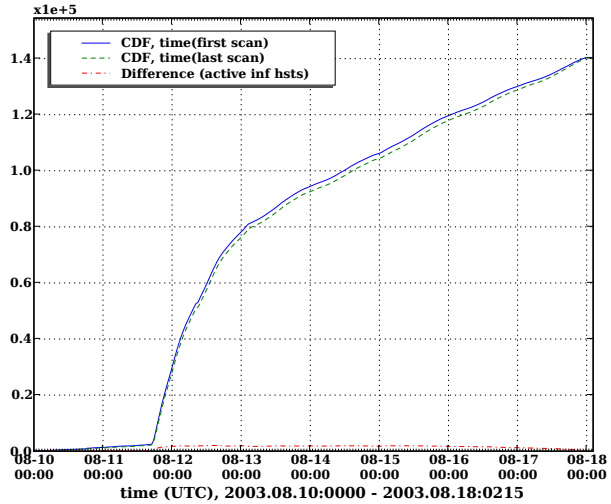


(d) time of guessed recoveries per interval, 15min bins

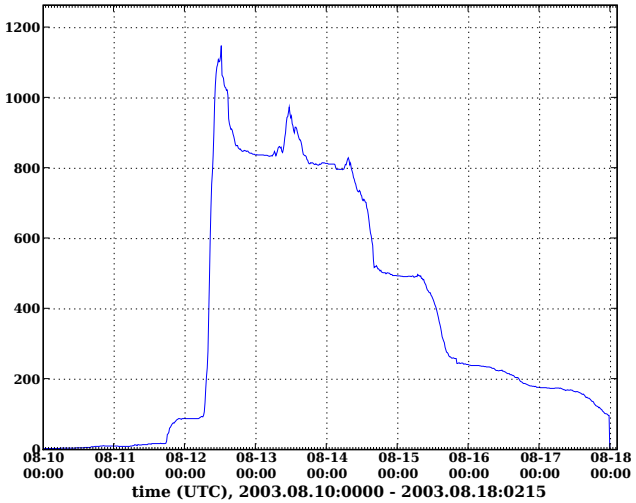
Figure D.5: Several statistics for Blaster about spreading and recovery behavior in AS559



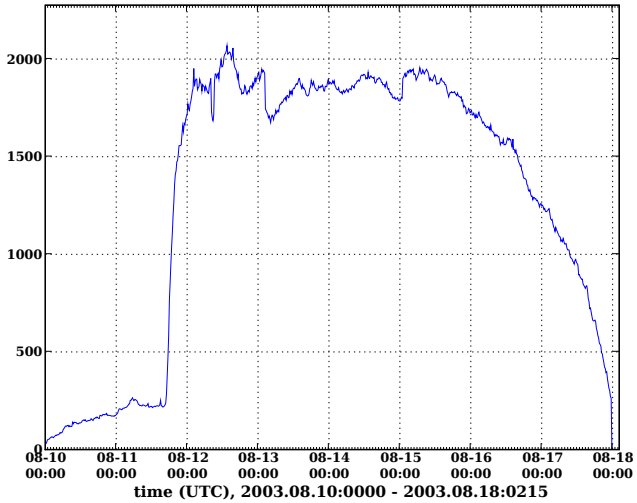
(a) Unnormalized CDF for hosts getting infected (solid line) and recovered (dashed line), AS559



(b) Unnormalized CDF for hosts getting infected and recovered , hosts outside AS559

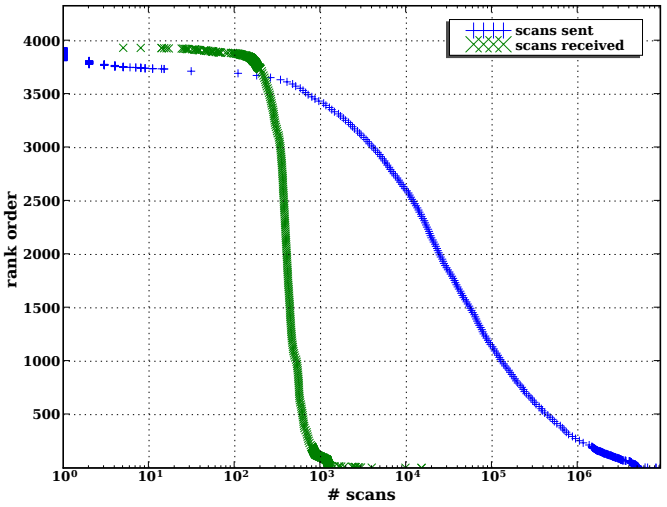


(c) Number of infected hosts (difference of the CDFs above)

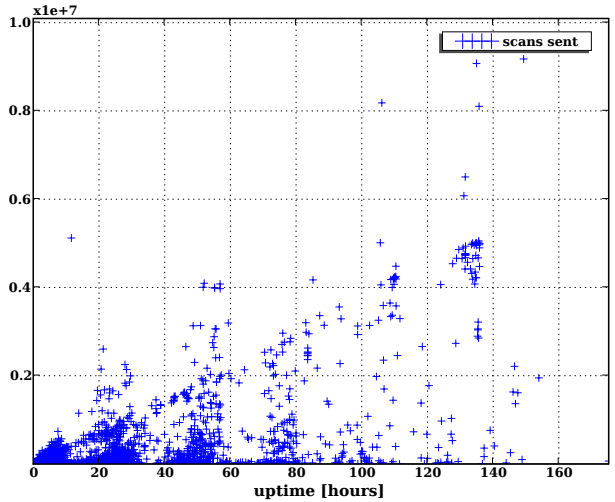


(d) Number of infected hosts (difference of the CDFs above)

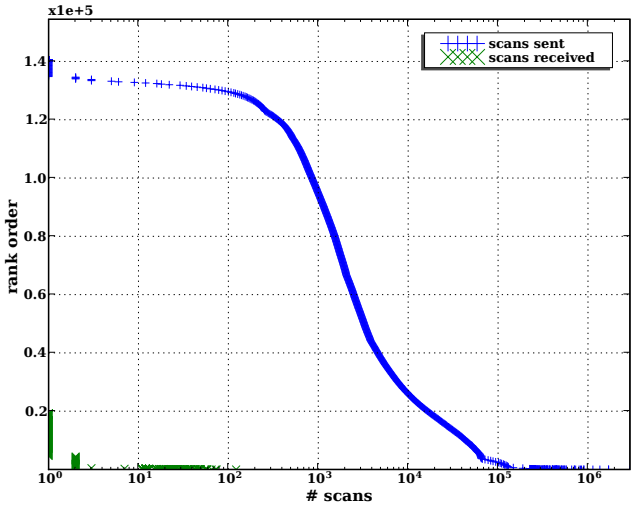
Figure D.6: Fight between spreading and recovery versus time for Blaster worm. We distinguish between hosts inside and outside AS559.



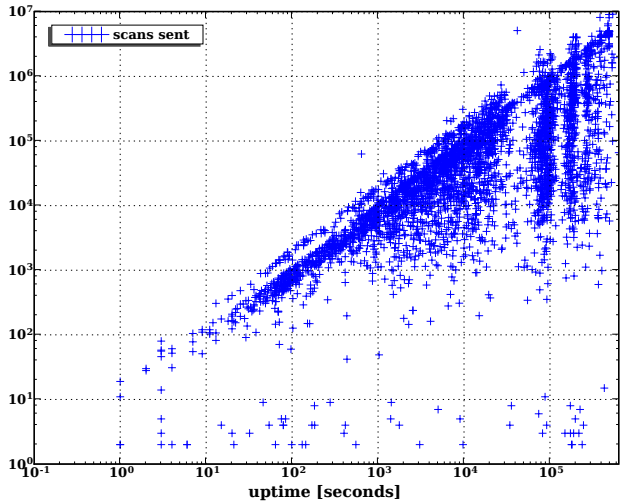
(a) Rank order of sent and received scans, host in AS559, log(t) lin(y)



(b) Scans sent in function of uptime, hosts in AS559, lin(t) log(y)



(c) Rank order of sent and received scans, host outside AS559, log(t) lin(y)



(d) Scans sent in function of uptime, hosts in AS559, log(t) log(y)

Figure D.7: Blaster, number of sent and received scans

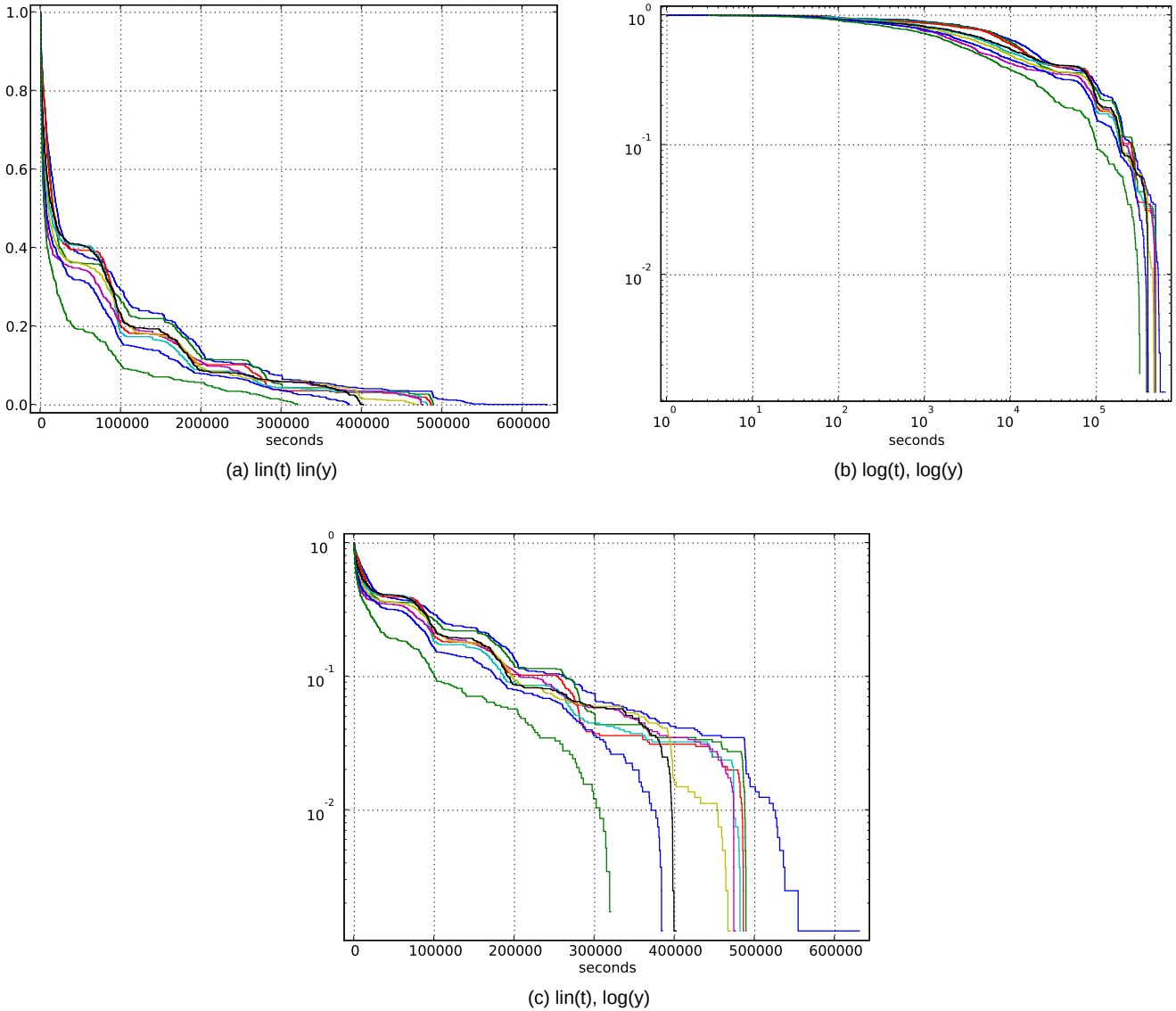


Figure D.8: CCDF of time(last scan) - time(first scan), 800 chronological consecutive hosts per curve, 400 points overlapping windows, AS559 (CCDF of time span from first scan to last scan, aka the total time of being infected for each IP. We see here how fast hosts get recovered (user behavior). Each line represents the CCDF for 800 hosts infected at the same epoch. Only hosts within AS559 are considered.)

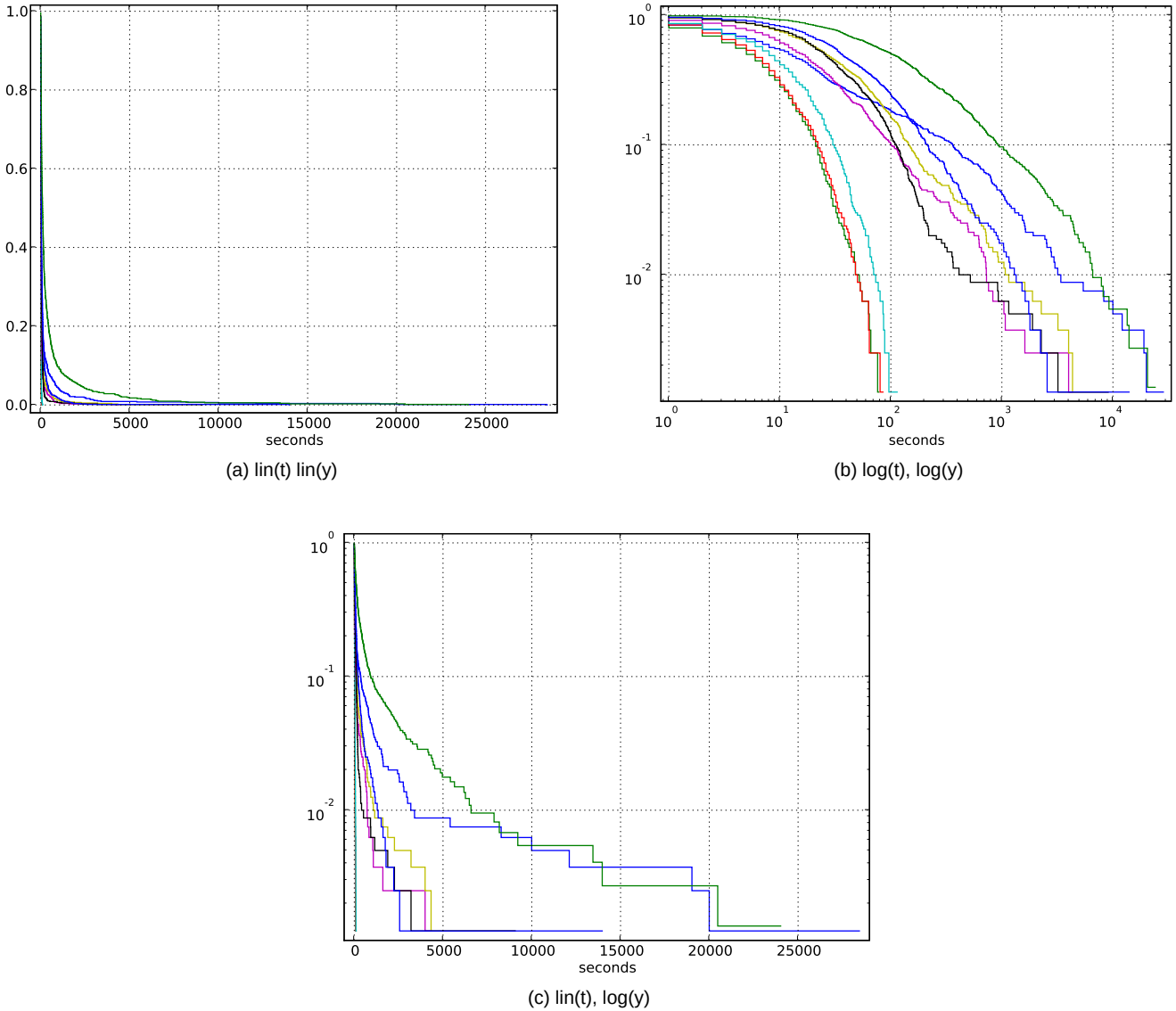


Figure D.9: CCDF of waiting time between two infections (first scans), 800 chronological consecutive hosts per curve, 400 points overlapping windows, only hosts within AS559 are counted

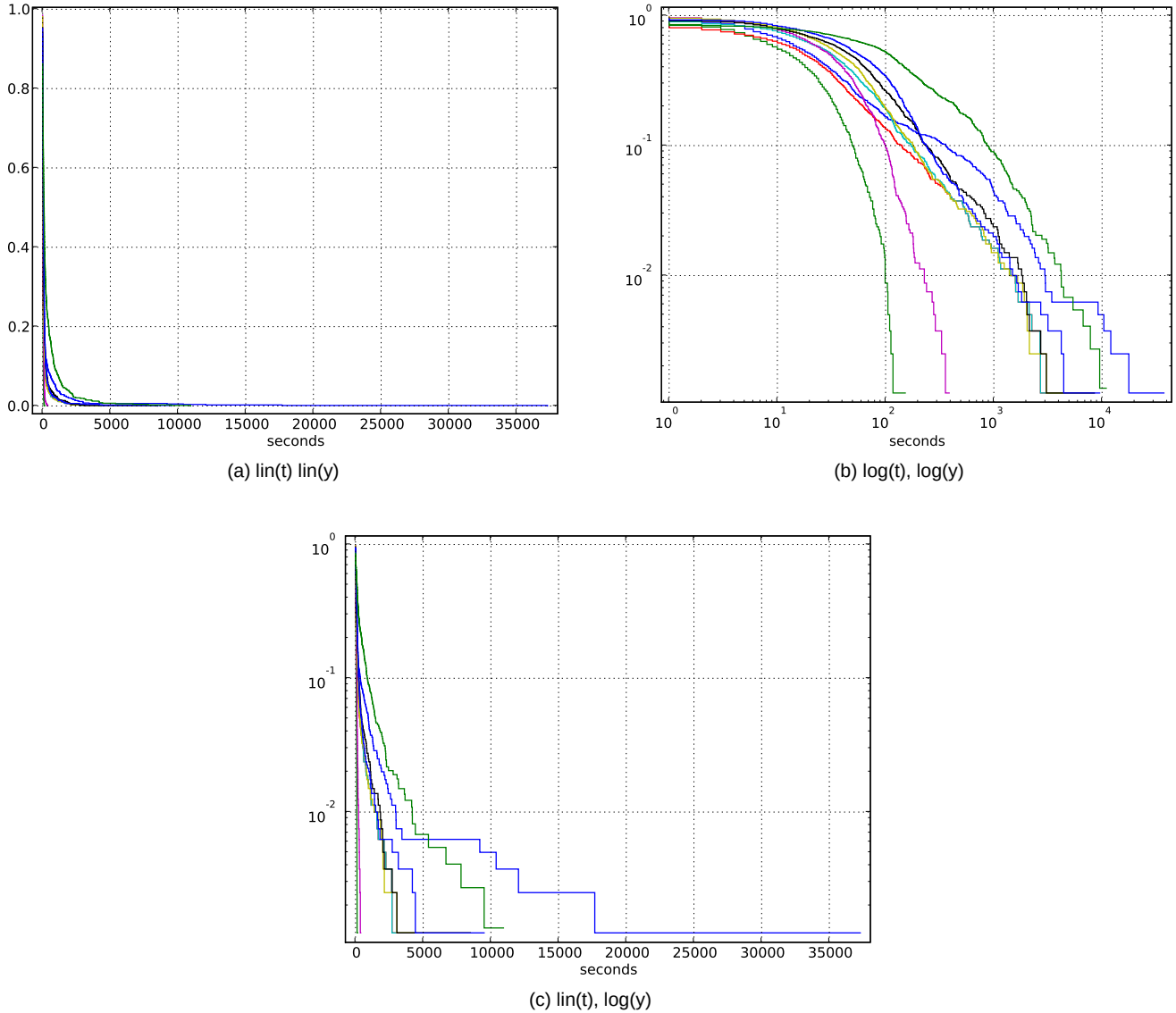


Figure D.10: CCDF of waiting time between two recoveries (last scans), 800 chronological consecutive hosts per curve, 400 points overlapping windows, only hosts within AS559 are counted

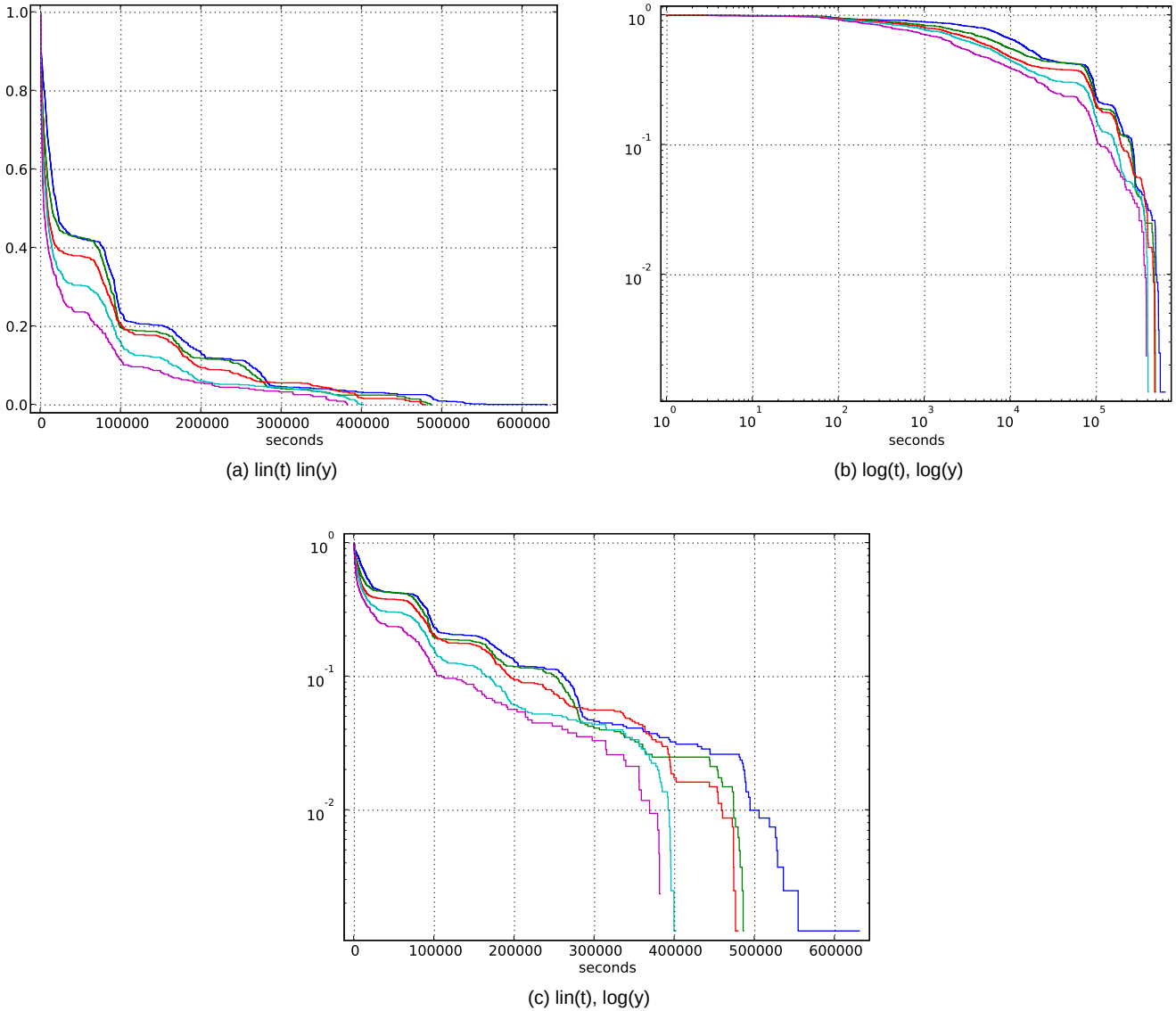
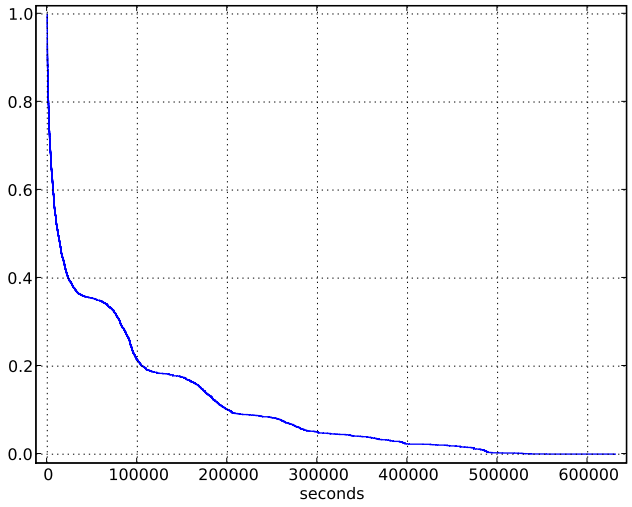
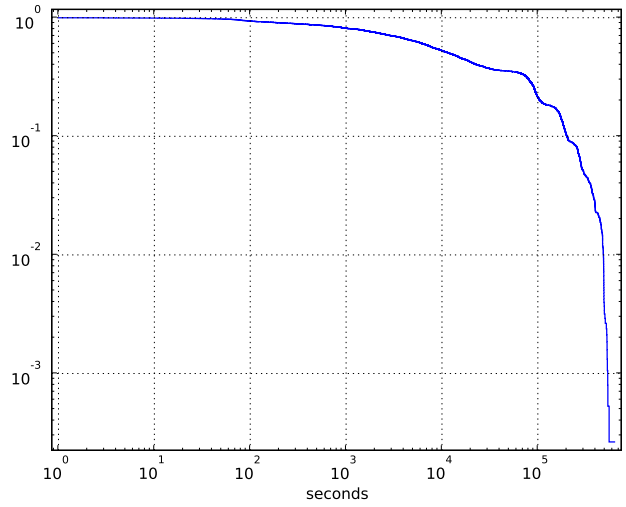


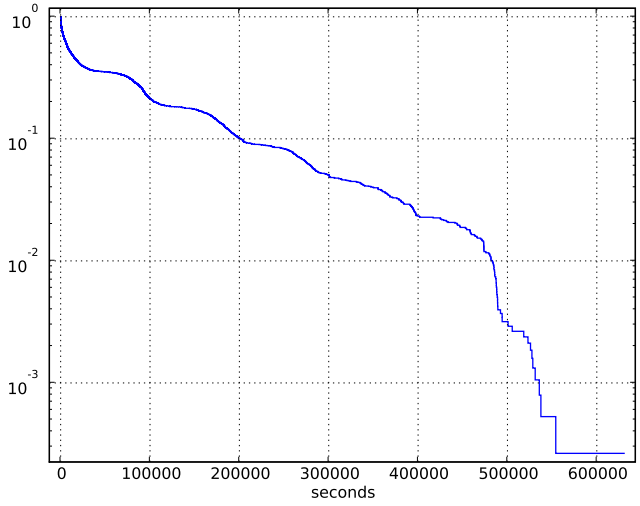
Figure D.11: CCDF of waiting time between two guessed recoveries, 800 chronological consecutive hosts per curve, 400 points overlapping windows, only hosts within AS559 are counted



(a) lin(t) lin(y)

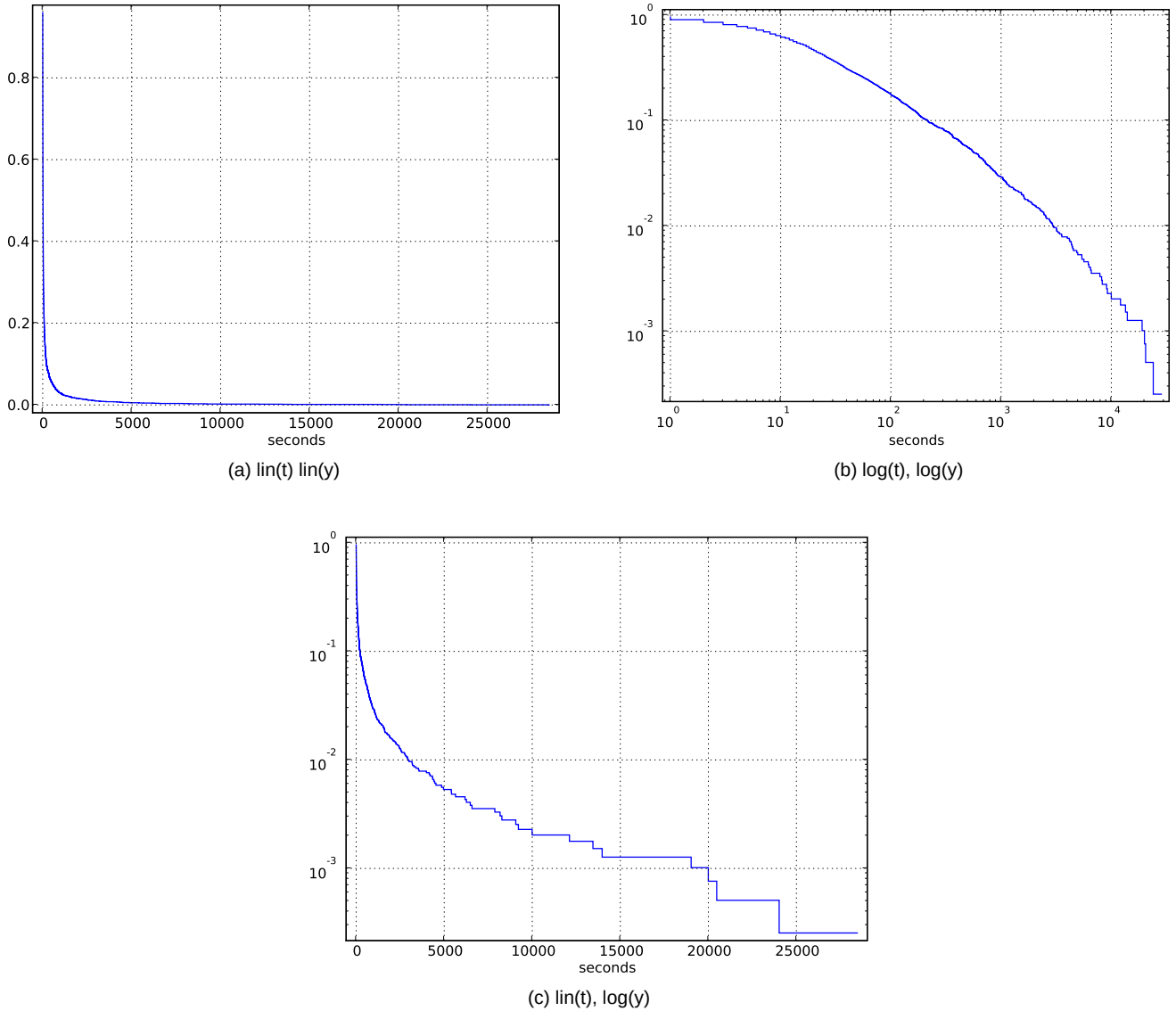


(b) log(t), log(y)



(c) lin(t), log(y)

Figure D.12: CCDF of time(last scan) - time(first scan), AS559 (CCDF of time span from first scan to last scan, aka the total time of being infected for each IP. We see here how fast hosts get recovered (user behavior). Only hosts within AS559 are considered.)

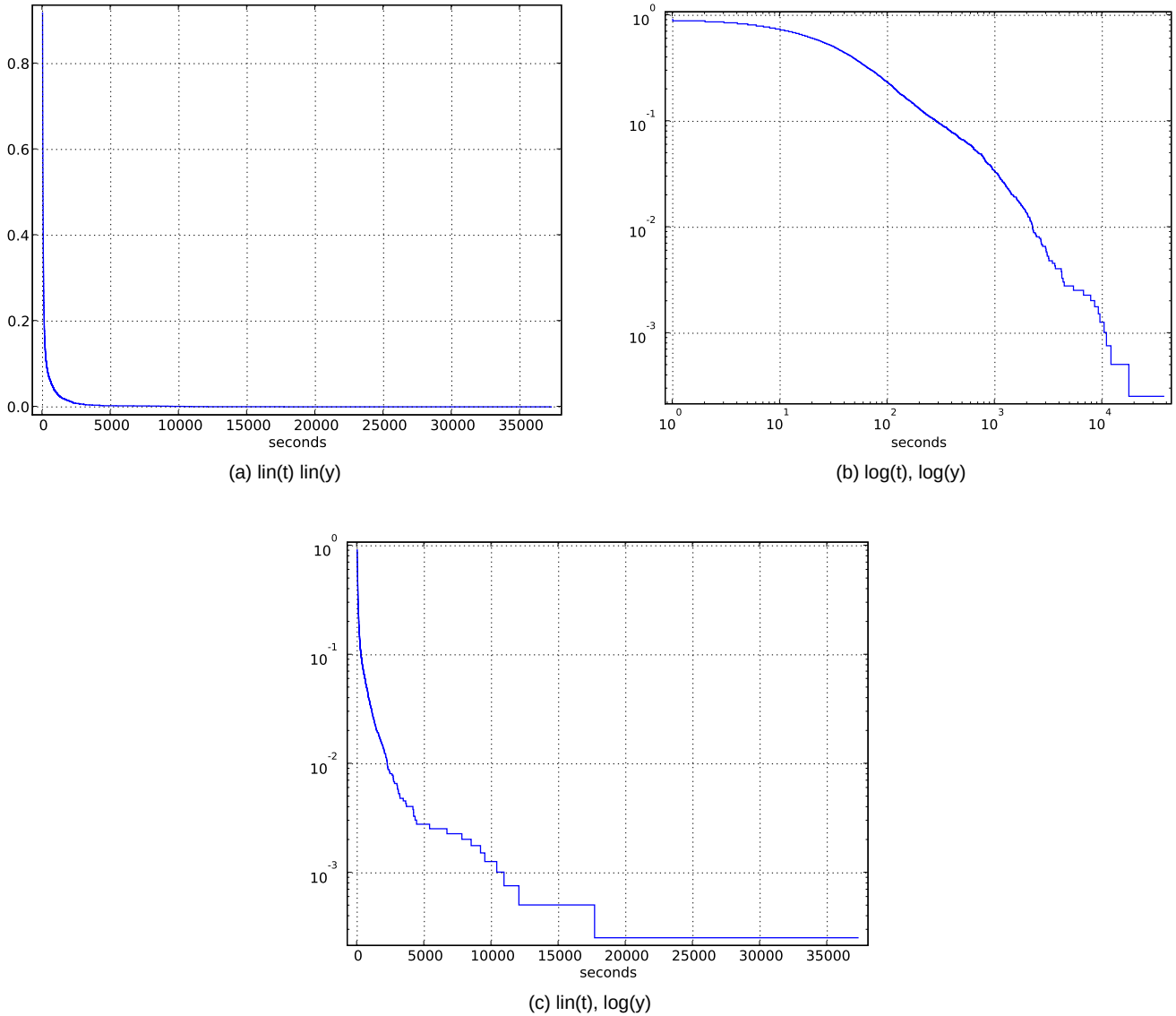


(a) lin(t) lin(y)

(b) log(t), log(y)

(c) lin(t), log(y)

Figure D.13: CCDF of waiting time between two infections (first scans), Only hosts within AS559 are considered.

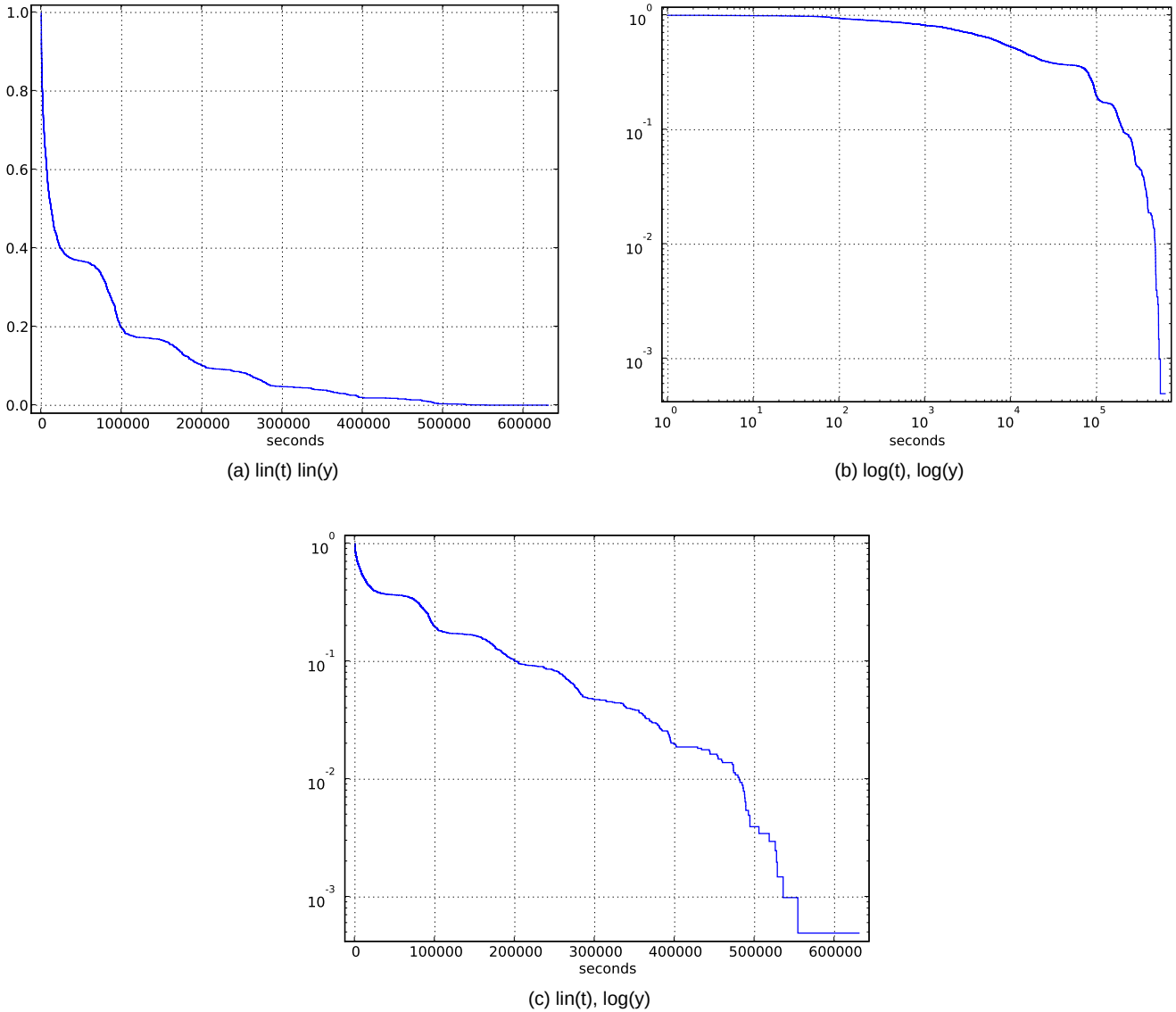


(a) lin(t) lin(y)

(b) log(t), log(y)

(c) lin(t), log(y)

Figure D.14: CCDF of waiting time between two last scans, Only hosts within AS559 are considered.

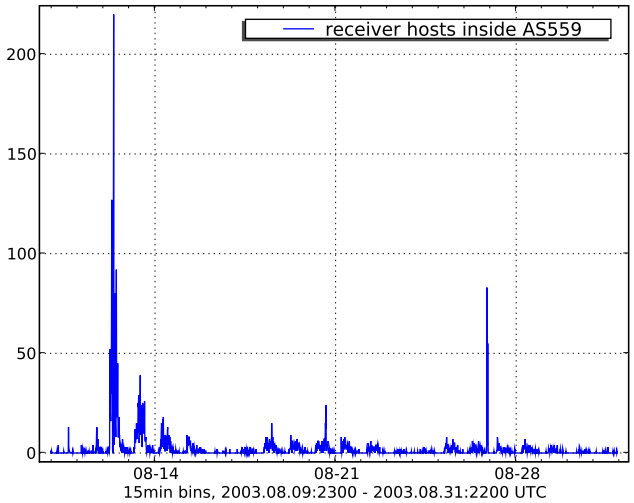


(a) lin(t) lin(y)

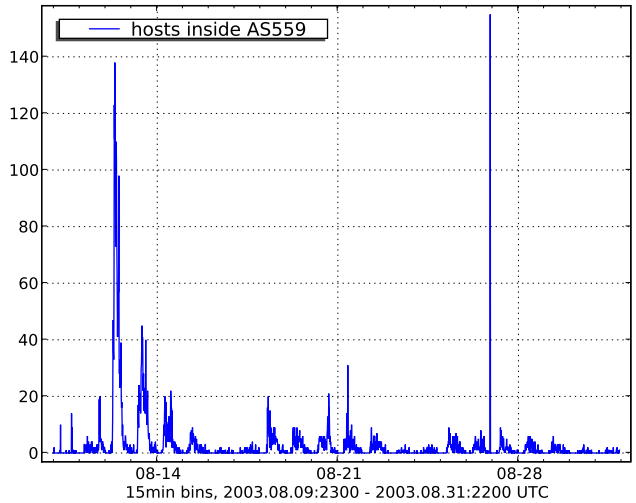
(b) log(t), log(y)

(c) lin(t), log(y)

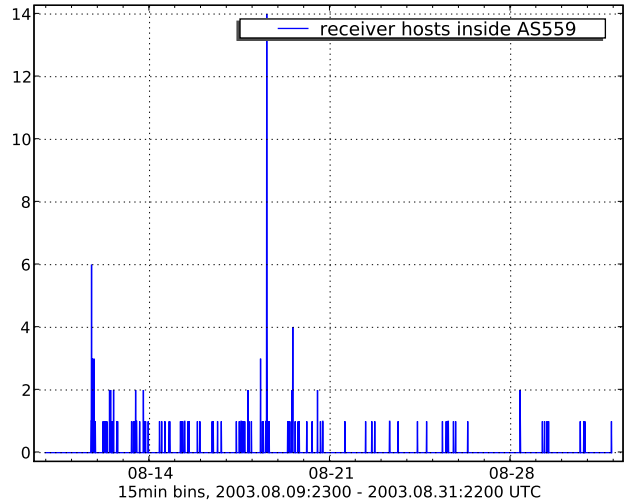
Figure D.15: CCDF of waiting time between two guessed recoveries, Only hosts within AS559 are considered.



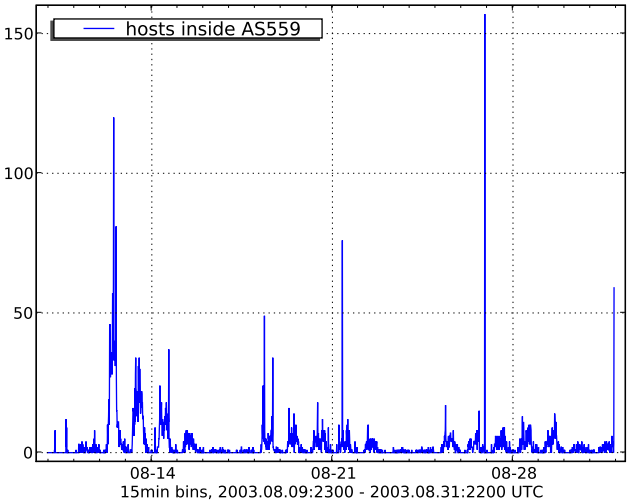
(a) time of first unsuccessful infection attempts (target not existent, switched off, wrong exploit), AS559



(b) Time of first scan (number of *new* infected hosts per interval)

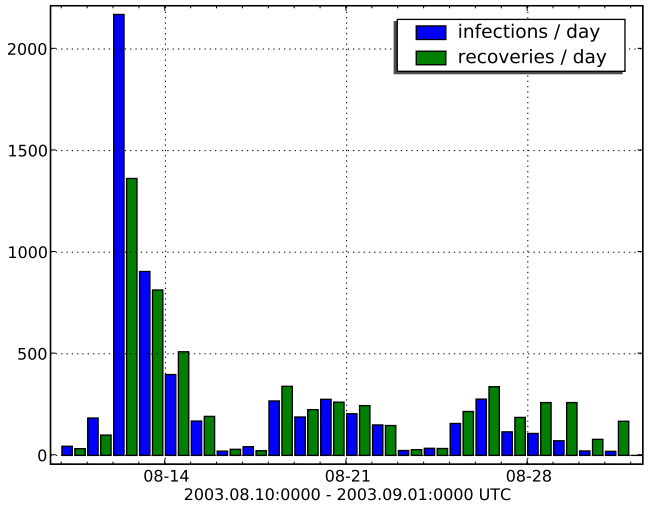


(c) Time of (complete multistage) infection over AS559 borders

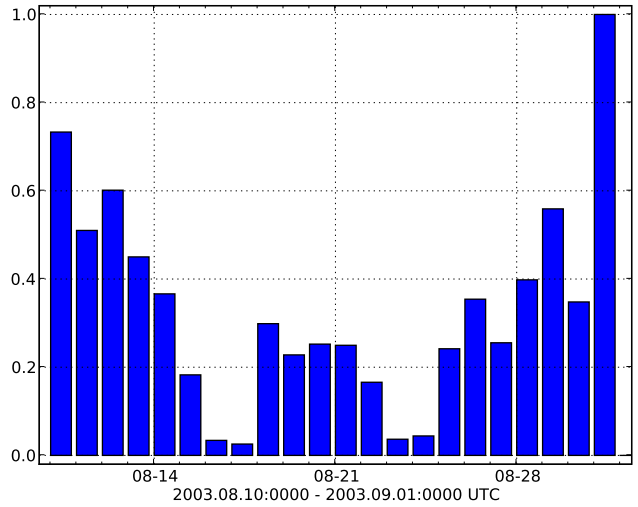


(d) Time of last scan

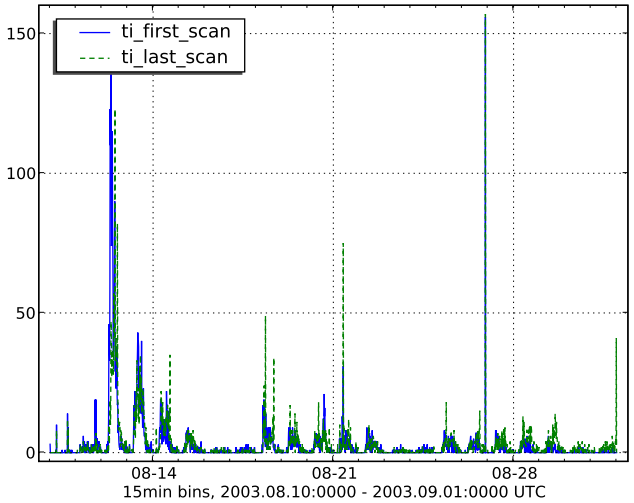
Figure D.16: Blaster, time series of unique events, 15min bins (Time of first and last occurrence of a TCP SYN to destination port 135 coming from a given IP, as well as the time of the first unsuccessful TCP SYN connection. Subfigure (c) counts the hosts successfully infected over the boarder routers. The statistics is only shown for hosts belonging to AS559. First 3 weeks after outbreak.)



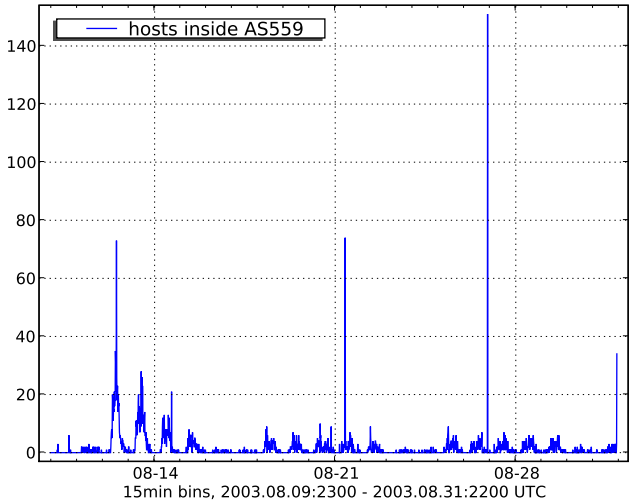
(a) infections / day, recoveries / day, AS559



(b) #recovieries/day / (#infections / day + #unrecovered hosts from yesterday), AS559

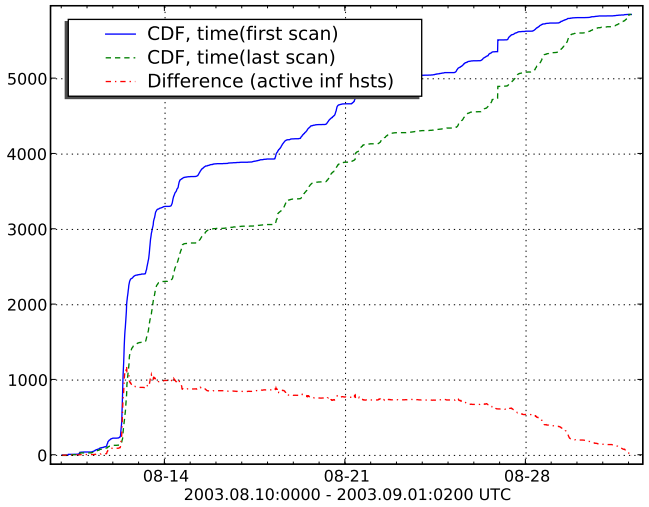


(c) time of first scans and time of last scans per interval, 15min bins

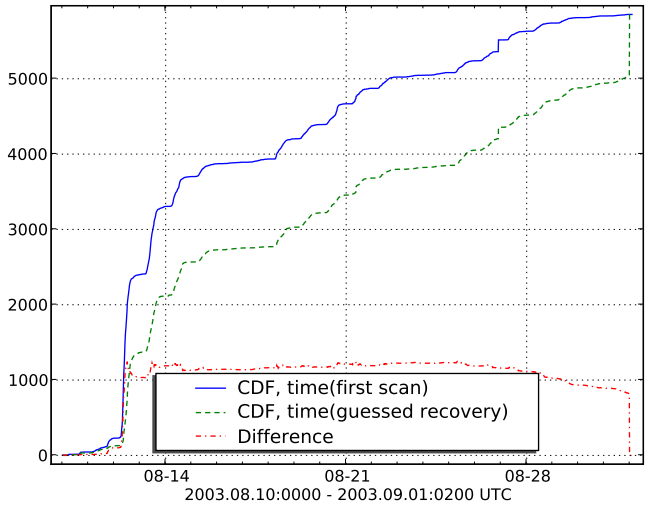


(d) time of guessed recoveries per interval, 15min bins

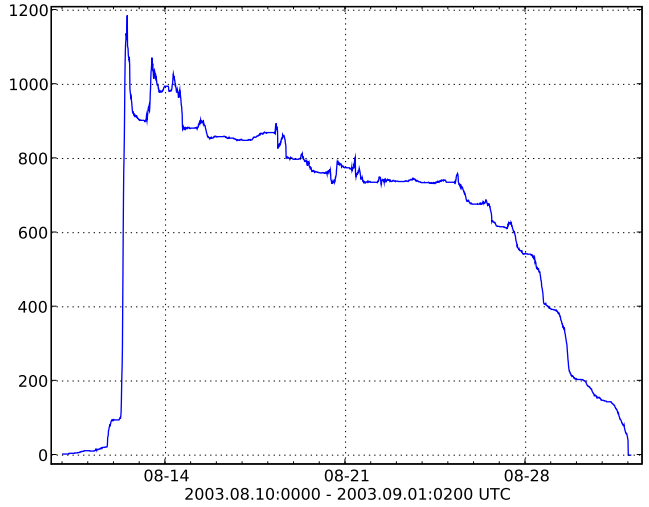
Figure D.17: Several statistics for Blaster about spreading and recovery behavior during the first 3 weeks. We distinguish between hosts inside and outside AS559.



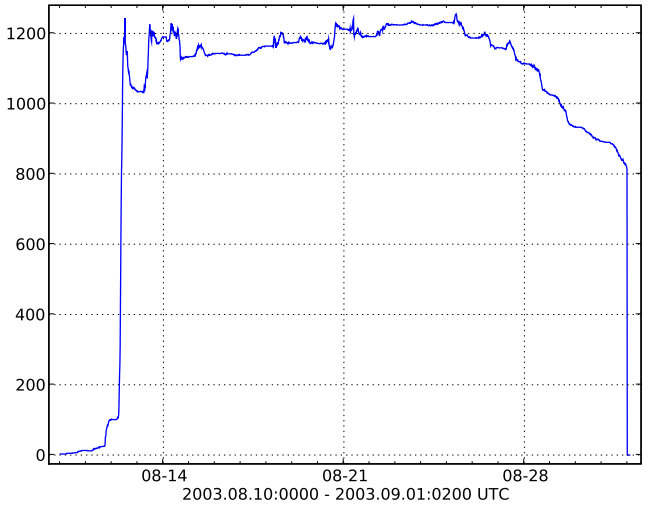
(a) Unnormalized CDF for hosts getting infected (solid line) and recovered (dashed line), AS559



(b) Unnormalized CDF for hosts getting infected and recovered, unknown recoveries are set to epoch end, AS559

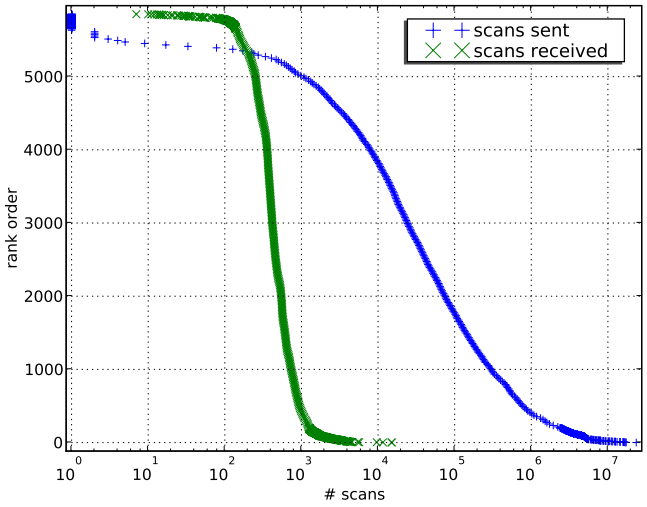


(c) Number of infected hosts (difference of the CDFs above)

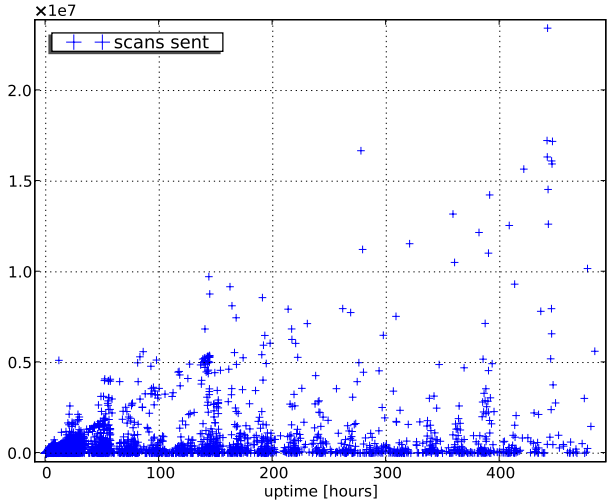


(d) Number of infected hosts (difference of the CDFs above)

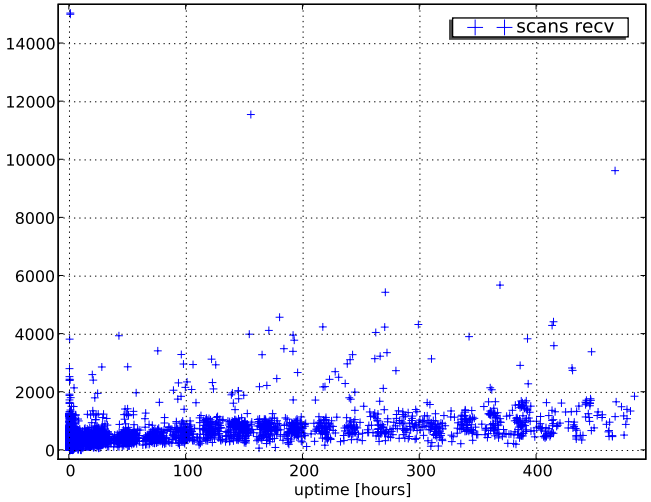
Figure D.18: Several statistics for Blaster about spreading and recovery behavior in AS559 during the first 3 weeks



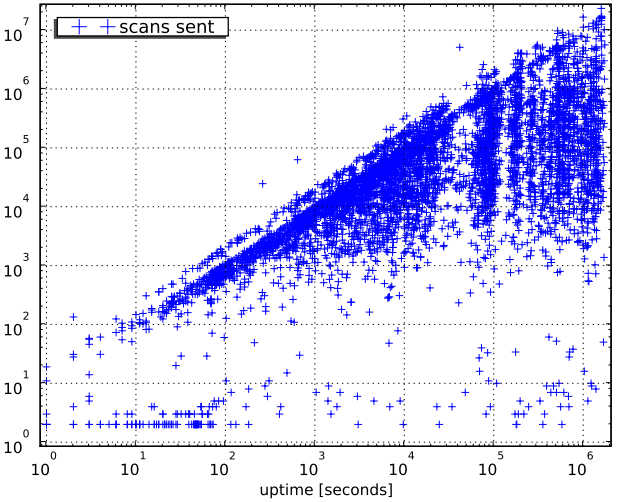
(a) Rank order of sent and received scans, host in AS559, log(t) lin(y)



(b) Scans sent in function of uptime, hosts in AS559, lin(t) log(y)

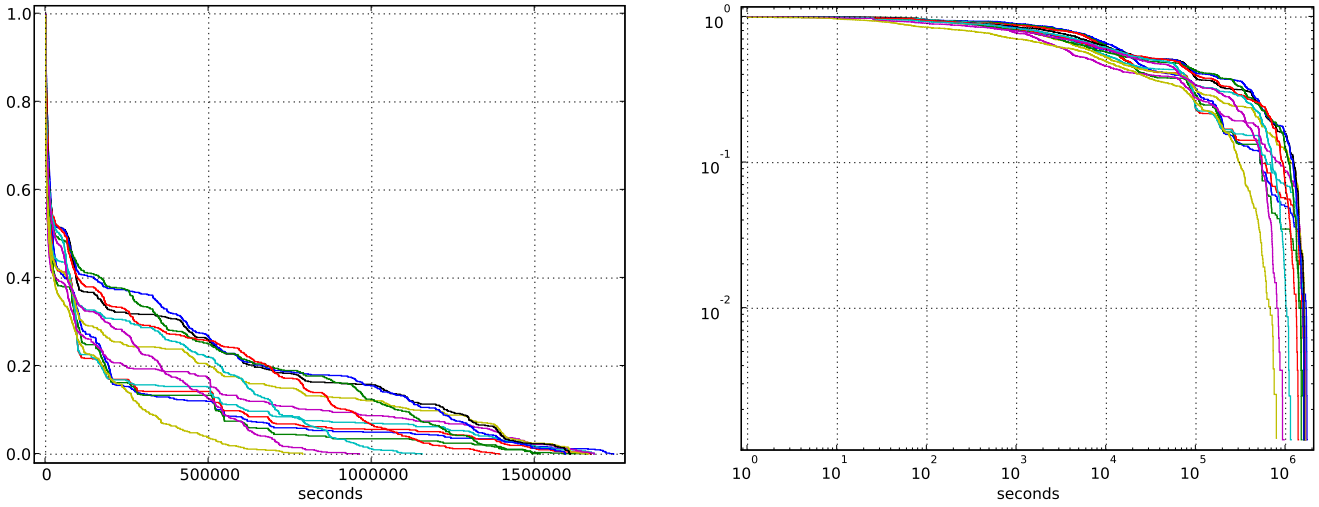


(c) Scans received since infection, AS559, lin(t) lin(y)



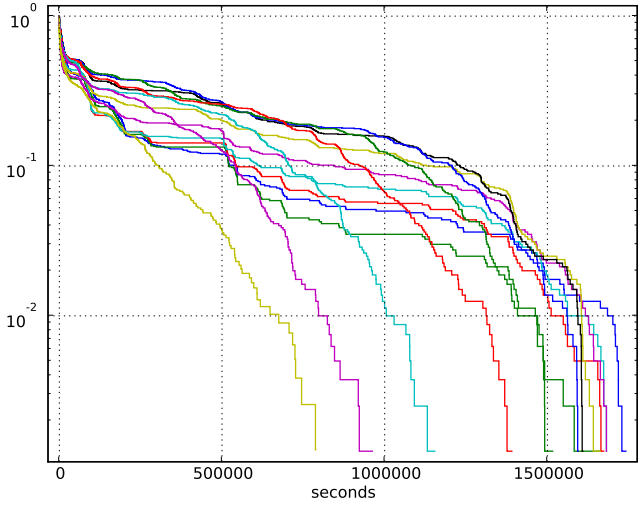
(d) Scans sent in function of uptime, hosts in AS559, log(t) log(y)

Figure D.19: Blaster, number of sent and received scans



(a) lin(t) lin(y)

(b) log(t), log(y)



(c) lin(t), log(y)

Figure D.20: CCDF of time(last scan) - time(first scan), 800 chronological consecutive hosts per curve, 400 points overlapping windows, AS559 (CCDF of time span from first scan to last scan, aka the total time of being infected for each IP. We see here how fast hosts get recovered (user behavior). Each line represents the CCDF for 800 hosts infected at the same epoch. Only hosts within AS559 are considered. First 3 weeks after outbreak.)

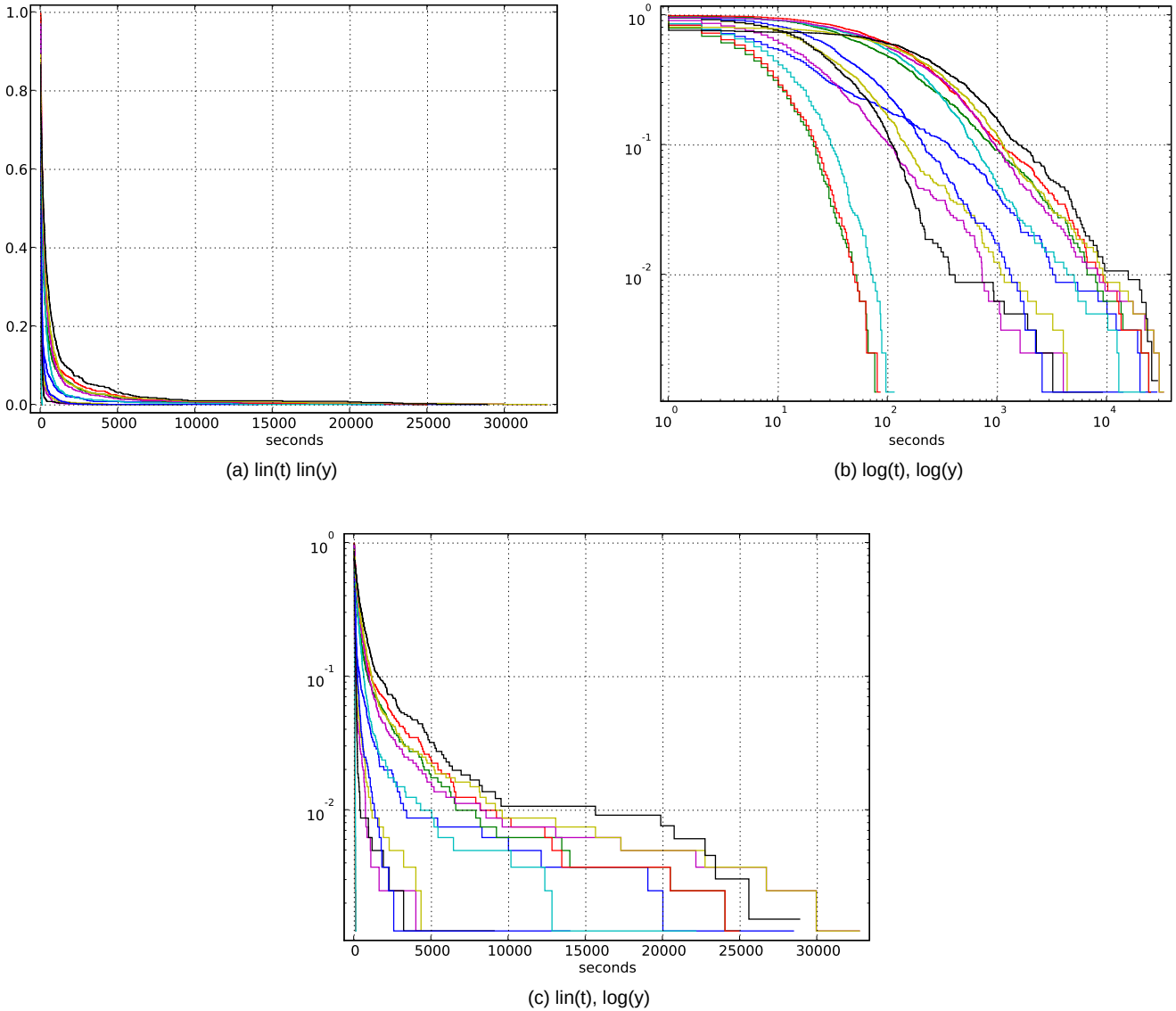
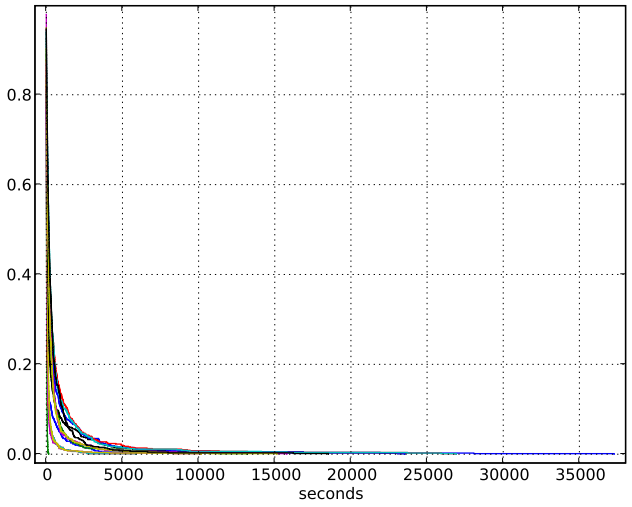
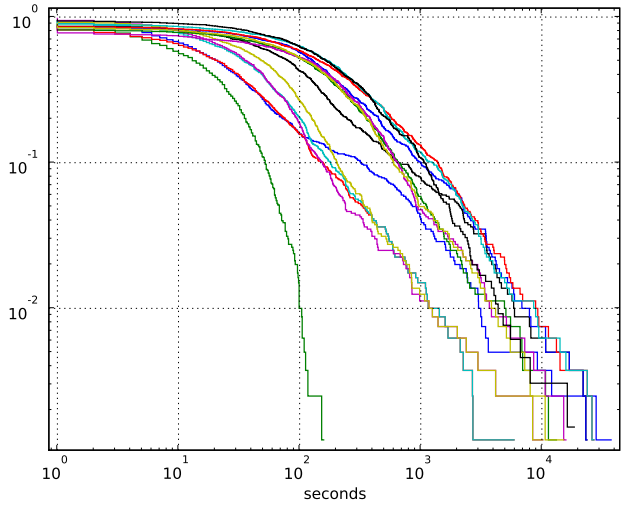


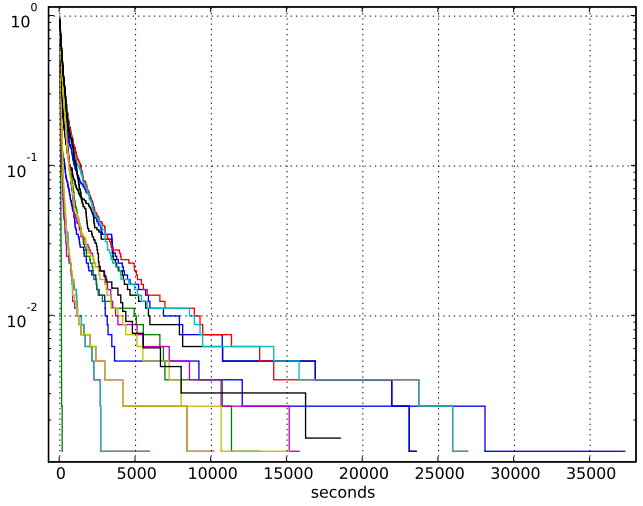
Figure D.21: CCDF of waiting time between two infections (first scans), 800 chronological consecutive hosts per curve, 400 points overlapping windows, only hosts within AS559 are counted. First 3 weeks after outbreak.



(a) lin(t) lin(y)



(b) log(t), log(y)



(c) lin(t), log(y)

Figure D.22: CCDF of waiting time between two recoveries (last scans), 800 chronological consecutive hosts per curve, 400 points overlapping windows, only hosts within AS559 are counted. First 3 weeks after outbreak.

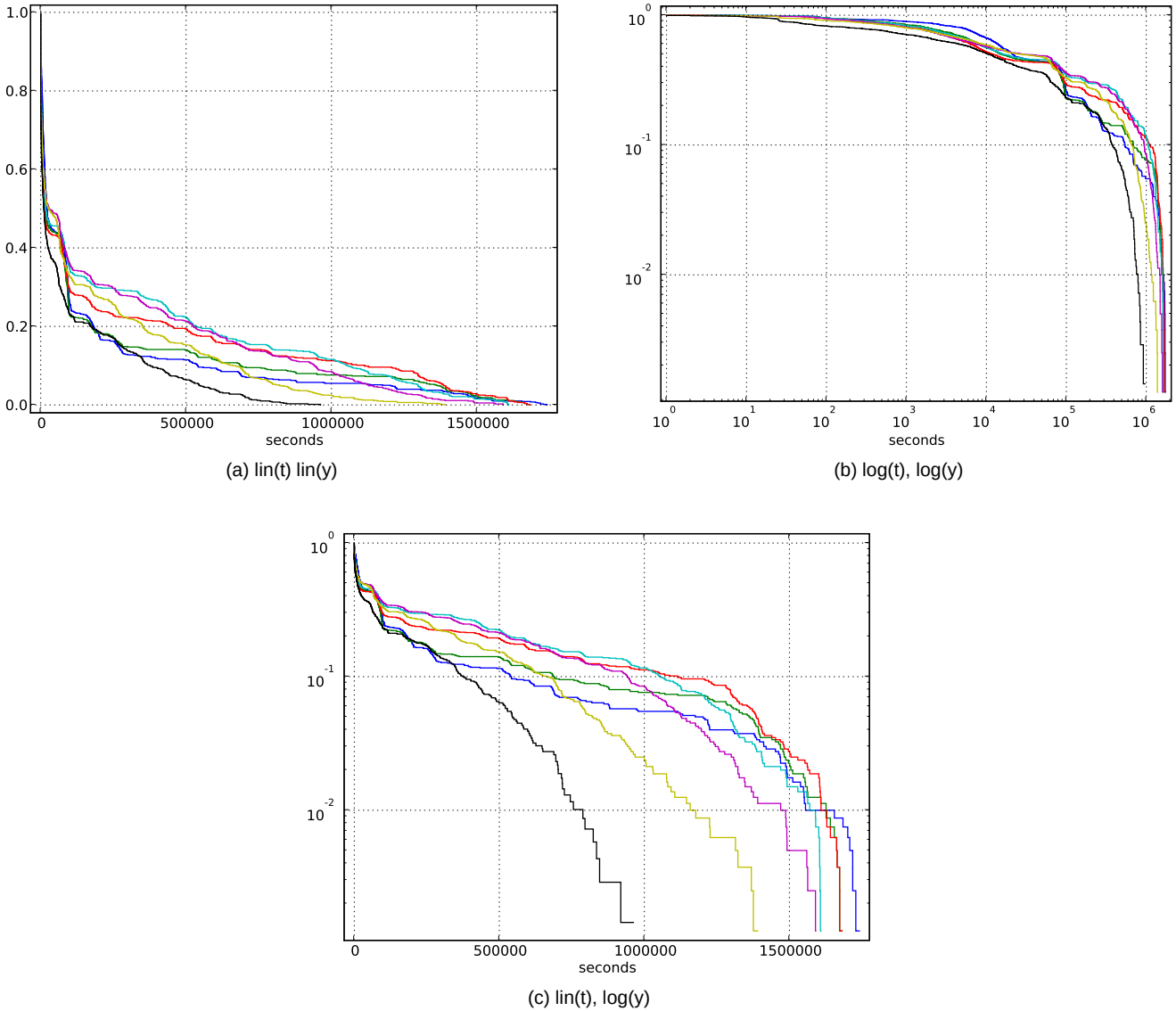
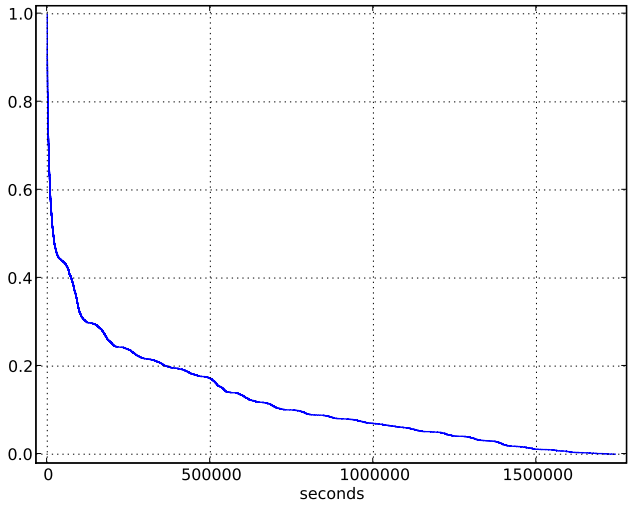
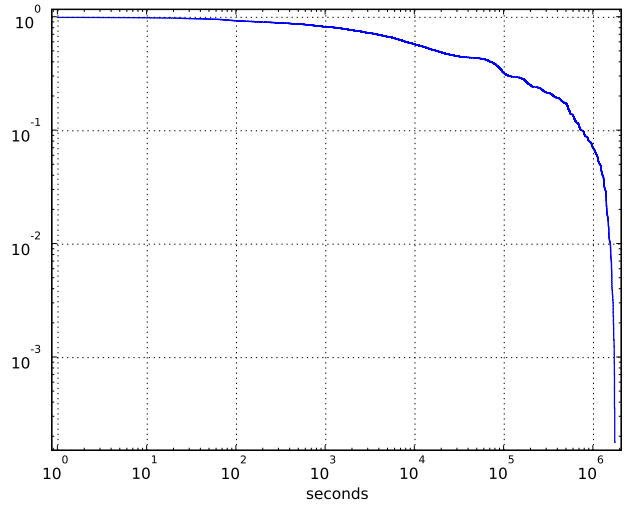


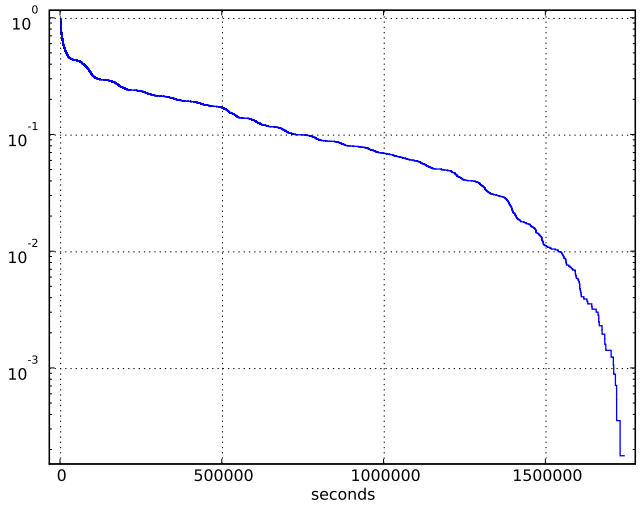
Figure D.23: CCDF of waiting time between two guessed recoveries, 800 chronological consecutive hosts per curve, 400 points overlapping windows, only hosts within AS559 are counted. First 3 weeks after outbreak.



(a) lin(t) lin(y)

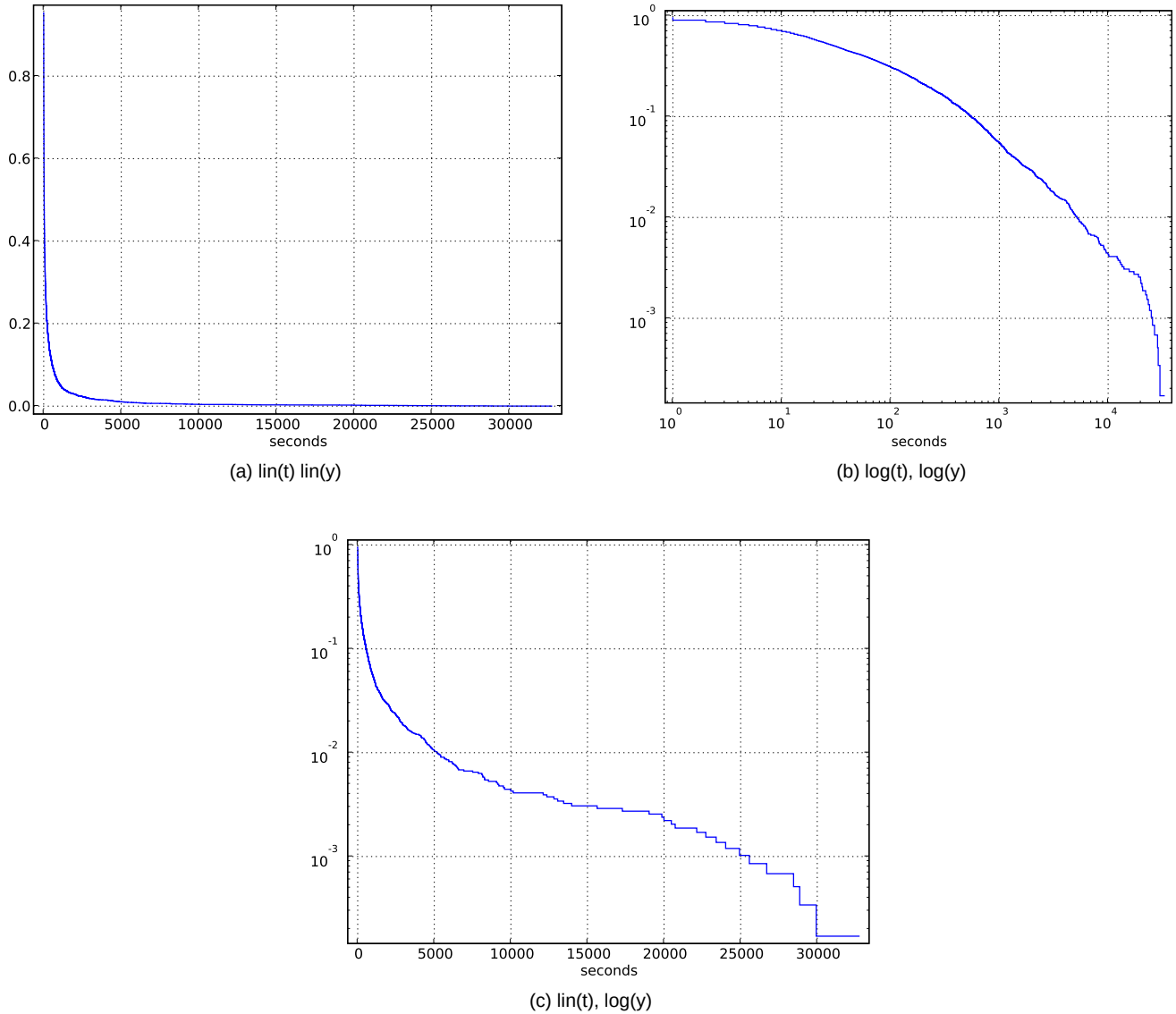


(b) log(t), log(y)



(c) lin(t), log(y)

Figure D.24: CCDF of time(last scan) - time(first scan), AS559 (CCDF of time span from first scan to last scan, aka the total time of being infected for each IP. We see here how fast hosts get recovered (user behavior). Only hosts within AS559 are considered. First 3 weeks after outbreak.)

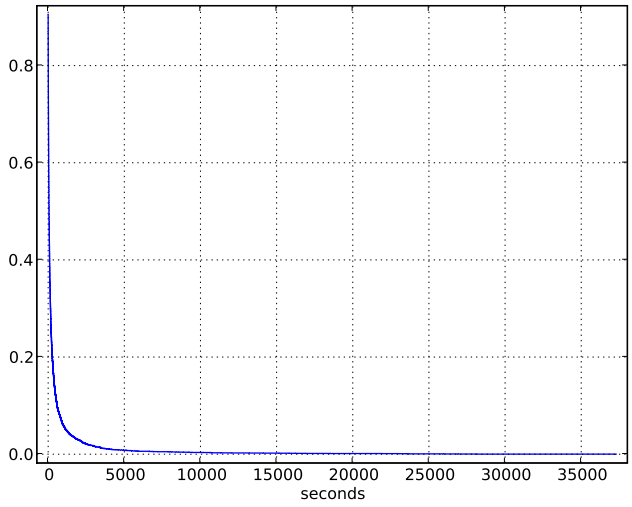


(a) lin(t) lin(y)

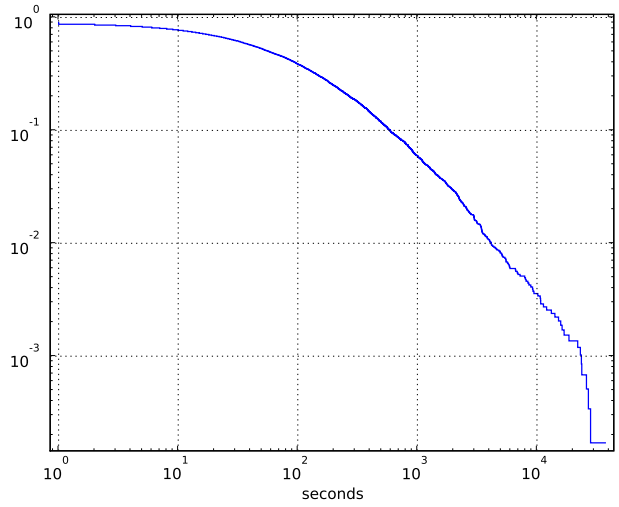
(b) log(t), log(y)

(c) lin(t), log(y)

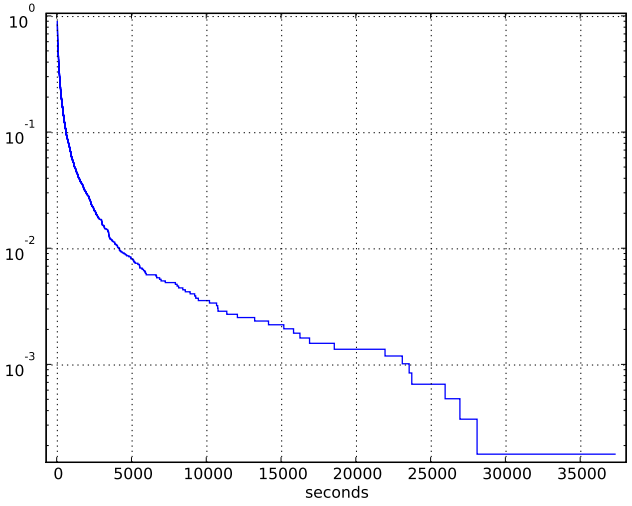
Figure D.25: CCDF of waiting time between two infections (first scans). Only hosts within AS559 are considered. First 3 weeks after outbreak.



(a) lin(t) lin(y)

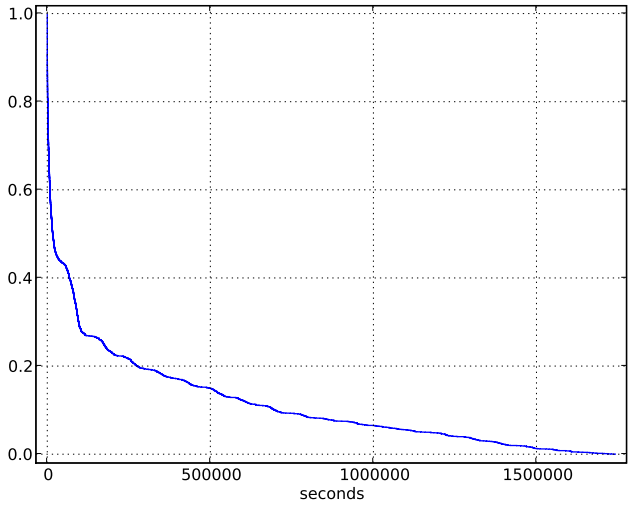


(b) log(t), log(y)

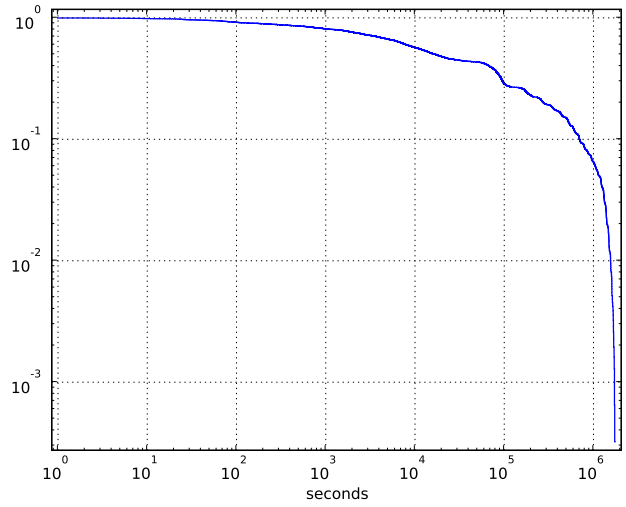


(c) lin(t), log(y)

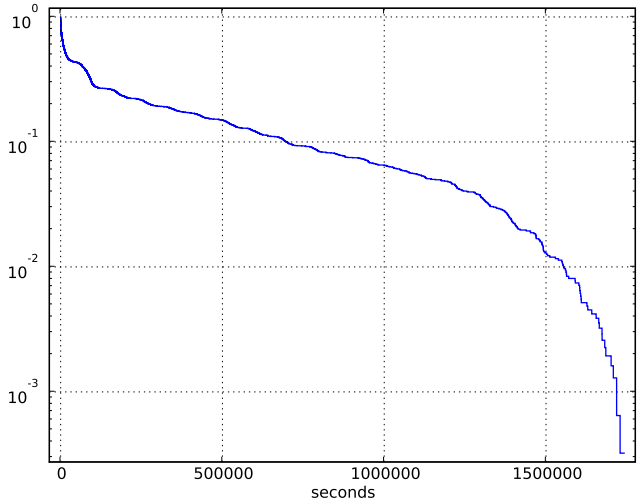
Figure D.26: CCDF of waiting time between two recoveries (last scans). Only hosts within AS559 are considered. First 3 weeks after outbreak.



(a) lin(t) lin(y)



(b) log(t), log(y)



(c) lin(t), log(y)

Figure D.27: CCDF of waiting time between two guessed recoveries, Only hosts within AS559 are considered. First 3 weeks after outbreak.

Appendix E

Plots, Least Square Fitting

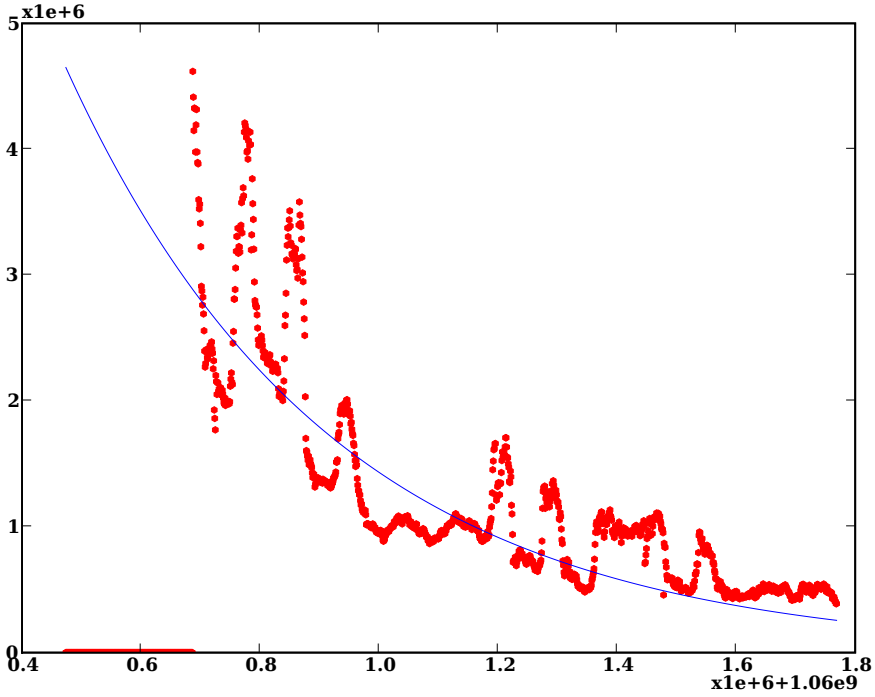


Figure E.1: Best found exponential fit for infection attempts from inside

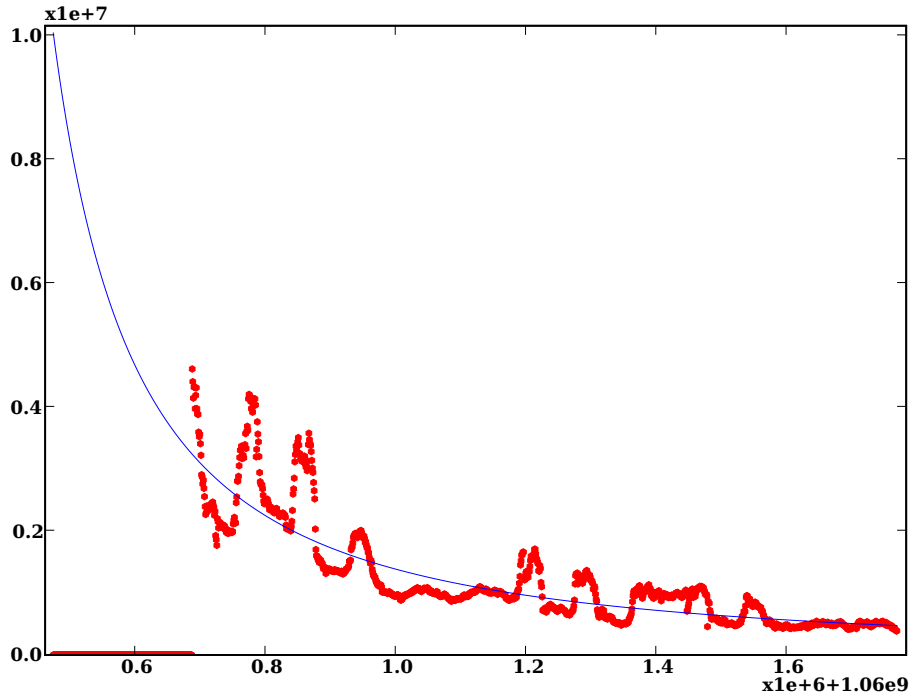
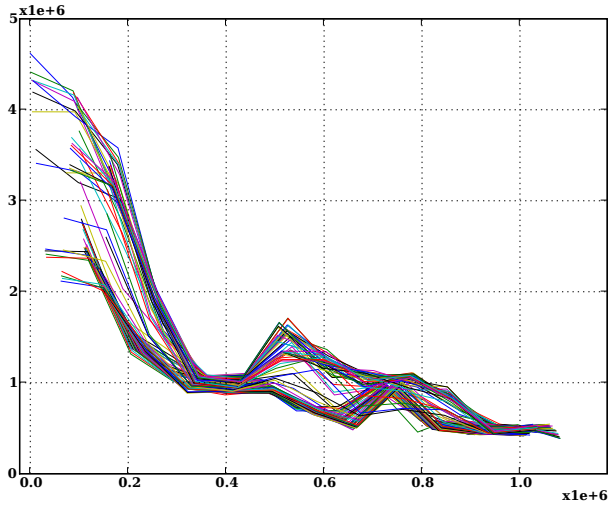
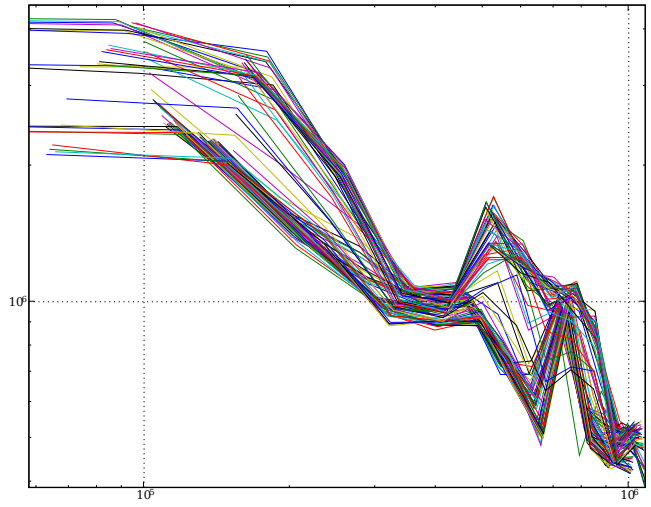


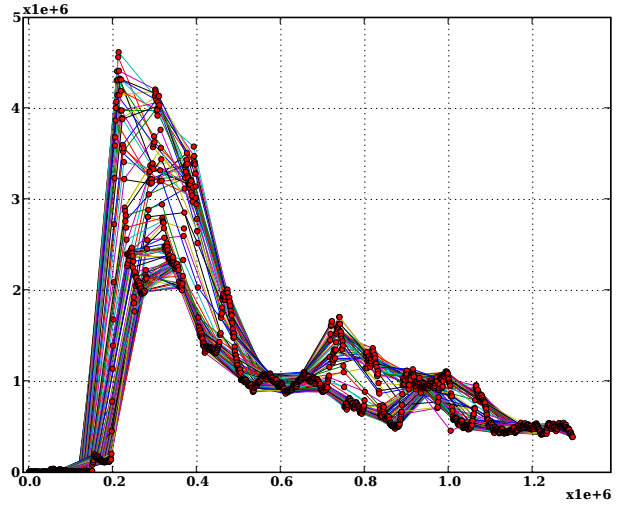
Figure E.2: Best found power law fit for infection attempts from inside



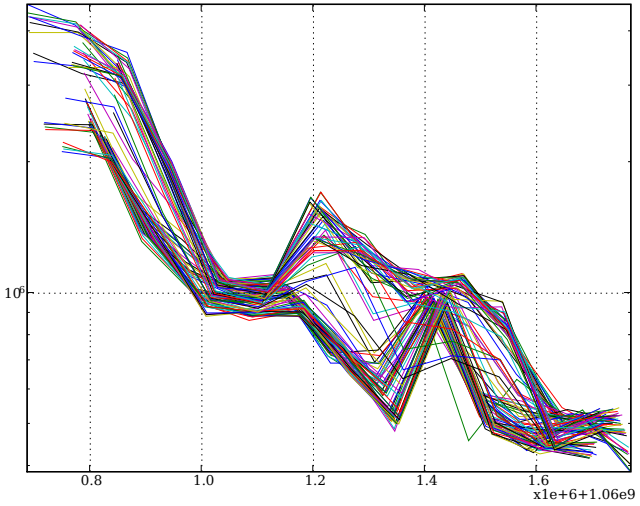
(a) lin(t), lin(y)



(b) log(t), log(y)

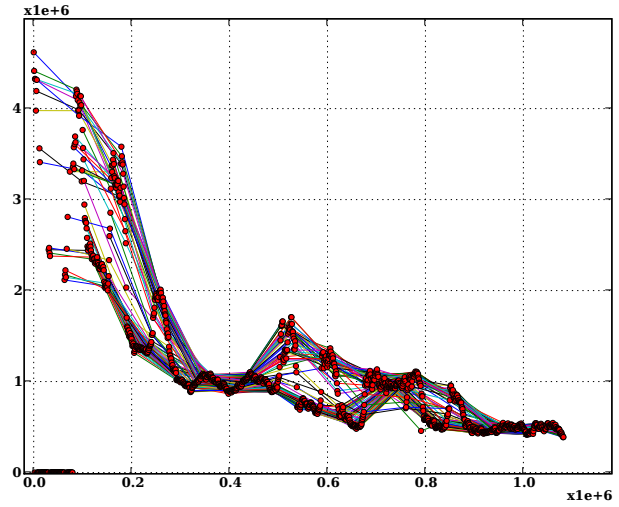


(c) complete curves of sampled Blaster anomaly

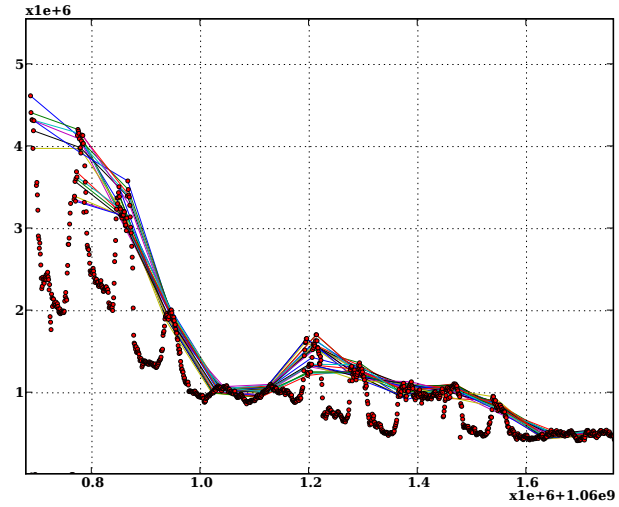


(d) lin(t), log(y)

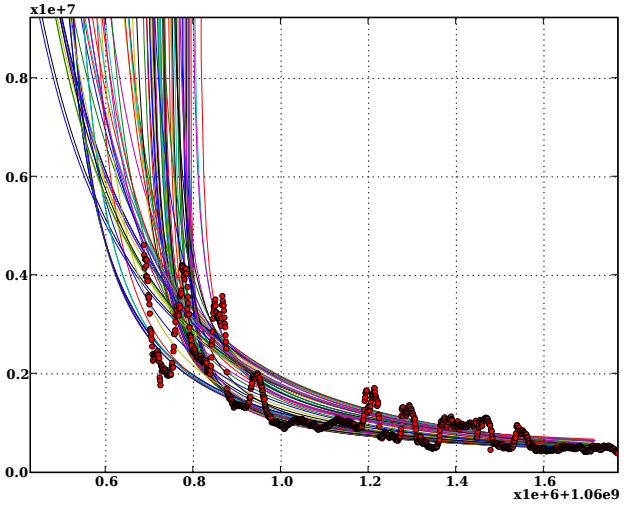
Figure E.3: Sampled decay of infection attempts (TCP SYN to destination port 135) for sources in AS559, 24h sampling, linear/logarithmic compare (We tried to remove seasonality by regularly sampling the decay of and estimated the parameters for each curve. The same graph is represented with different axis scales.)



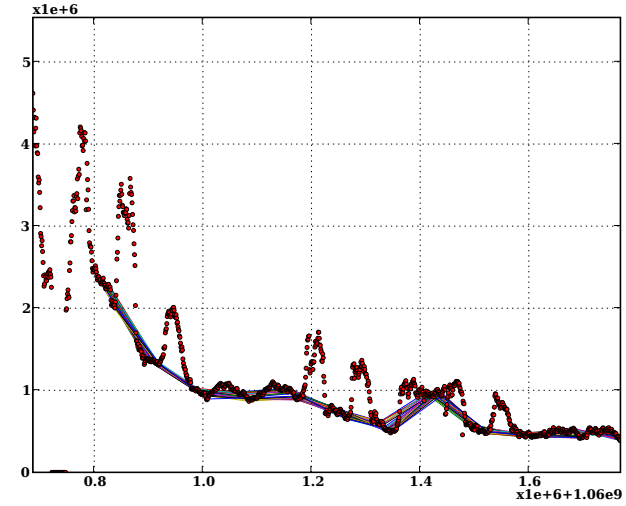
(a) All 96 lines of sampled decay



(b) Similar lines for the hours 08:00 - 14:00

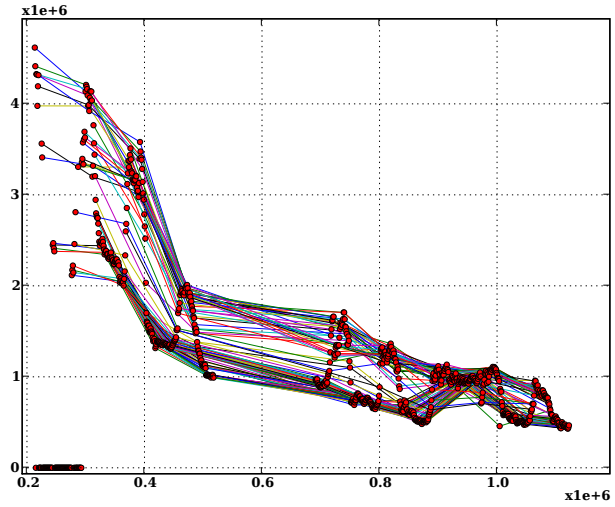


(c) Fits for all 96 decays

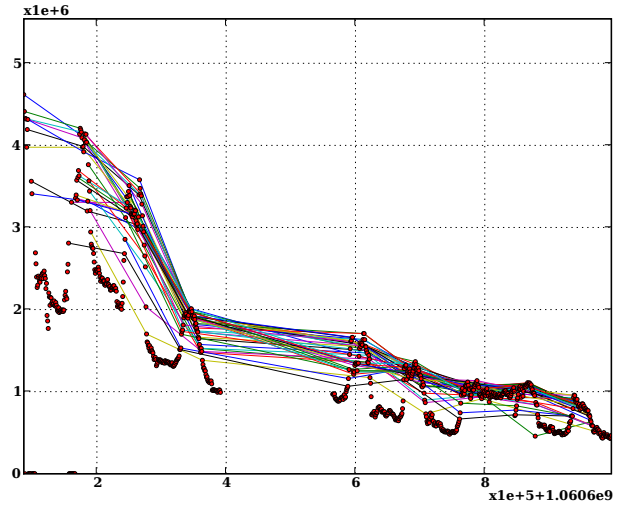


(d) Similar lines for the nightly hours (21:00 - 04:00)

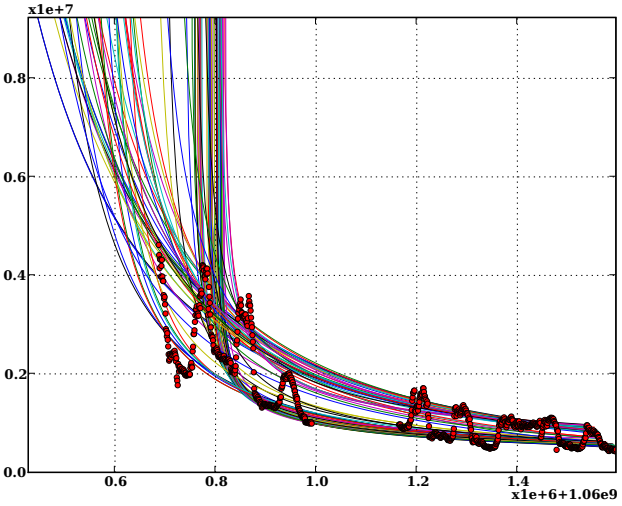
Figure E.4: Sampled decay of infection attempts from inside, 24h sampling, including weekend days, 15min bins. Subfigure (c) shows the resulting estimations for (a). (We tried to remove seasonality by regularly sampling the decay of and estimated the parameters for each curve.)



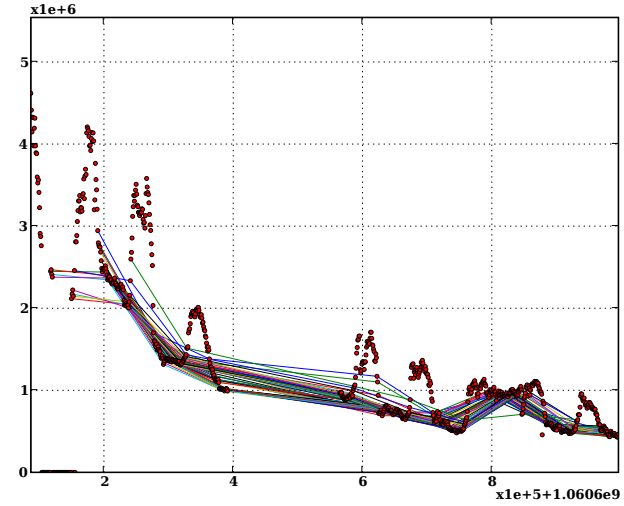
(a) All 96 lines of sampled decay



(b) Similar lines for the hours 07:00 - 16:00



(c) Fits for all 96 decays



(d) Similar lines for the nightly hours (16:00 - 06:00)

Figure E.5: Sampled decay of infection attempts from inside, 24h sampling, *no* weekend days, 15min bins (We tried to remove seasonality by regularly sampling the decay of and estimated the parameters for each curve.)

Appendix F

Schedule

Week	Calendar Week	Tasks	Deliverables
1	49	paper reading, analyze new frameworks, set up environment, learn svn	
2	50	writing fundamental code snippets, object oriented measurement tool design, paper reading	schedule
3	51	programming of metric measurement tool	
4	52	**X-Mas**, paper reading	ToC
5	1	visualization of extracted data (Perl scripts)	
6	2	evaluation of other 5-10 epidemic events	
7	3	sort flows, metric measurement, visualization of data	
8	4	sort flows, metric measurement, visualization of data	
9	5	learning R / Matlab	
10	6	fit scripts to our model	informational presentation
11	7	endogenous versus exogenous shock analysis	
12	8		
13	9		
14	10		
15	11		
16	12		
17	13		informational presentation
18	14		
19	15		
20	16		
21	17		
22	18		
23	19		
24	20	finishing documentation	
25	21	presentation preparing	presentation
26	22	reserve	
27	23	-	

Table F.1: planned schedule

Appendix G

Task Description

G.1 Introduction

Epidemics and immunization have been widely studied. Most of this work has proposed models for explaining the propagation of epidemics in the absence or presence of immunization (e.g., [9, 13, 59]. Moreover, organizations such as CAIDA [10] have conducted several measurement studies on observed worms.

The goal of this thesis is to study the propagation and defense strategies of several worm epidemics within the SWITCH network. Therefore, the student first needs to select worms that fulfill two requirements: they should be clearly visible in the available traces, and they should have a distinct pattern, which helps to identify the population of infected hosts. Further, the student computes several statistics (e.g., number of infected hosts, infection rate, immunization rate) for the selected worms, particularly longitudinal analysis of infected hosts. In a second step, the student will analyze the gathered statistics with standard tools such as Matlab or R. The student will verify existing physical models which explain the observed propagation and immunization dynamics based on cascading effects and epidemic processes [36, 37, 5, 34, 3]. These models are rarely confronted with extensive monitoring datasets. The goal is to uncover possible weaknesses in these models. If applicable, a particular model will be developed, using the framework on Endogenous versus Exogenous origins of Crises [19, 48].

The data used during this Master thesis is NetFlow [14] data which is collected on the SWITCH (Swiss Education Backbone Network) [52] border routers. This data comprises all aggregated packet headers (flows) which are sent between SWITCH-internal hosts, and hosts which are not part of the SWITCH backbone network. Consequently, ETH-internal traffic and traffic between SWITCH-internal sites is not observed.

G.2 The Task

This thesis is conducted at ETH Zürich. The task of this Master's Thesis is three-fold: First, the student has to select suitable candidate epidemics for further study. Second, the student will

implement C/C++ code for extracting the interesting statistics from the measurement data. Third, the student will verify existing physical models for the infection and immunization process on the extracted statistics.

Select Worms for Further Study

Select one or two email- or exploit-based worms to study. Examples are Sobig, MyDoom, Netsky, Witty [44], Blaster, Sasser, Nachi/Welchia, Nimda. Selection criteria are the visibility of the respective worm in our traces, as well as the possibility of identifying infected hosts with our traces.

Analysis of Infection and Immunization Dynamics

Define a methodology for extracting the required statistics from NetFlow traces. Implement C/C++ tools to do this, and use existing code where applicable. Analyze the number of infected hosts vs time for these worms, regarding contamination, saturation, and immunization/repair processes. See [35, 47] and references therein for further details on percolation theory.

Verify Physical Model(s) for Infection and/or Immunization Process

Conduct a statistical analysis of the computed statistics and verify selected physicals models based on the extracted data (see references above). If time permits, develop a model based on Endo/Exo Dynamics. Use existing statistical tools in Matlab or R for this task.

G.3 Deliverables

The following results are expected:

- Comparison of existing biological and physical models for epidemics and immunization.
- Concise description of the selected epidemics and methodology for determination of infected and immunized systems.
- Implementation of C/C++ tools for extracting the required statistics from the NetFlow data.
- Verification of existing models on the computed statistics.
- Proposal for an endo/exo-based model (if time permits).
- A concise description of the work conducted in this thesis (motivation, related work, own approach, implementation, results and outlook). The survey as well as the description of the prototype and the testing results is part of this main documentation. The abstract of the documentation has to be written in both English and German. The original task description is to be put in the appendix of the documentation. One sample of the documentation needs to be delivered at TIK. The whole documentation, as well as the source code, slides of the

talk etc., needs to be archived in a printable, respectively executable version on a CDROM, which is to be attached to the printed documentation.

G.4 Organizational Aspects

Documentation and presentation

A documentation that states the steps conducted, lessons learned, major results and an outlook on future work and unsolved problems has to be written. The code should be documented well enough such that it can be extended by another developer within reasonable time. At the end of the thesis, a presentation will have to be given at TIK that states the core tasks and results of this thesis. If important new research results are found, a paper might be written as an extract of the thesis and submitted to a computer network and security conference.

Dates

This Master's thesis starts on December 3rd 2007 and is finished on June 3rd 2007. It lasts 6 months in total. At the end of the second week the student has to provide a schedule for the thesis. It will be discussed with the supervisors.
After a month the student should provide a draft of the table of contents (ToC) of the thesis. The ToC suggests that the documentation is written in parallel to the progress of the work.
Two intermediate informal presentations for Prof. Plattner and all supervisors will be scheduled 2 months and 4 months into this thesis.
A final presentation at TIK will be scheduled close to the completion date of the thesis. The presentation consists of a 15 minute talk plus 5 minutes for questions. Informal meetings with the supervisors will be announced an organized on demand.

Supervisors

Daniela Brauckhoff, brauckhoff@tik.ee.ethz.ch, +41 44 632 70 50, ETZ G93

Thomas Maillart, maillart@er.ethz.ch, +41 44 632 82 40, KPL F18

Bibliography

[1] Reka Albert, Hawoong Jeong, and Albert-Laszlo Barabasi. Diameter of the World-Wide Web. *Nature, Macmillan Magazines Ltd.*, 401:130, September 1999.

[2] Michael Bailey, Evan Cooke, Farnam Jahanian, David Watson, and Jose Nazario. The Blaster Worm: Then and Now. *IEEE Security & Privacy Magazine*, 05:26–31, 2005.

[3] J. Balthrop, S. Forrest, MEJ Newman, and M.M. Williamson. COMPUTER SCIENCE: Technological Networks and the Spread of Computer Viruses, 2004.

[4] Kaspersky lab virus list, net-worm.win32.lovesan.a, 2008. URL: `http://www.viruslist.com/en/viruses/encyclopedia?virusid=24773`.

[5] M. Boguna, R. Pastor-Satorras, and A. Vespignani. 8 Epidemic Spreading in Complex Networks with Degree Correlations.

[6] Stefan Bornholdt and Heinz Georg Schuster. *Handbook of Graphs and Networks, from the Genome to the Internet*. WILEY-VCH GmbH, 2003.

[7] Paul Boutin. Slammed!: An inside view of the worm that crashed the internet in 15 minutes. *Wired Magazin*, 2003. URL: `http://www.wired.com/wired/archive/11.07/slammer.html`.

[8] Daniela Brauckhoff, Bernhard Tellenbach, Martin May, and Anukool Lakhina. Impact of Packet Sampling on Anomaly Detection Metrics. *IMC 2006: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, 2006.

[9] Linda Briesemeister, Patrick Lincoln, and Phillip Porras. Epidemic profiles and defense of scale-free networks. *WORM '03: Proceedings of the 2003 ACM workshop on Rapid malcode*, pages 67–75, 2003.

[10] CAIDA - the Cooperative Association for Internet Data Analysis, 2008. URL: `http://www.caida.org/publications/papers/bydate/`.

[11] John Canvan. Me code write good: The l33t skillz of the virus writer. *White Paper, Symantec Security Response, Dublin*, 2006.

[12] European Organization for Nuclear Research, 2008. URL: `http://www.cern.ch/`.

[13] Zesheng Chen, Lixin Gao, and Kevin Kwiat. Modeling the spread of active worms. *IEEE Infocom*, 2003.

[14] Cisco Systems Inc. NetFlow White Papers, 2008. URL: `http://www.cisco.com/en/US/products/ps6601/prod_white_papers_list.html`.

[15] A. Clark. S-i-r model of epidemics, basic model and examples. *Department of Mechanical Engineering, University of Rochester*, 2005.

[16] R. Crane and D. Sornette. Searching with viral dynamics on social networks: the case of youtube. ., 2007.

[17] R. Crane and D. Sornette. Viral dynamics on youtube: how social networks renormalize human activity. ., 2007.

[18] Cliff C.Zou, Don Towsley, and Weibo Gong. Email worm modeling and defense. *Department of Electrical and Computer Engineering, University of Massachusetts*, 2004.

[19] F. Deschatres and D. Sornette. The Dynamics of Book Sales: Endogenous versus Exogenous Shocks in Complex Networks. *Arxiv preprint physics/0412171*, 2004.

[20] Rohit Dhamankar. Die sans top 20 internet security vulnerabilities. *TippingPoint, a division of 3Com*, 2005.

[21] Thomas P. Dübendorfer. *Impact Analysis, Early Detection and Mitigation of Large-Scale Internet Attacks*. Shaker Verlag, 2005. ISBN 3-8322-4739-4.

[22] Jean-Pierre Eckmann, Elisha Moses, and Danilo Sergi. Entropy of dialogues creates coherent structures in e-mail traffic. *PNAS*, 101:14333–14337, October 2005.

[23] J. Gysel, D. Brauckhoff, and B. Tellenbach. Anomaly Visibility and Detectability in Sampled Traffic, 2007. Semester Thesis SA-2007-12.

[24] Eric S. Hines. Mydoom.b worm analysis. *Applied Watch Technologies Inc, Glenview, IL*, 2004.

[25] Michael Hoehle and Erik Jorgensen. Estimating parameters for stochastic epidemics. *Dina Research Report No. 102, Department of Animal Science and Animal Health, Royal Veterinary and Agricultural University, Denmark*, 2002.

[26] Theus Hossmann. Analysis of Sobig.F and Blaster Worm Characteristics. *Communication Systems Group, ETH Zurich, Semester Thesis SA-2004.23*, 2004.

[27] IBM Zurich Research Laboratory, 2008. URL: `http://www.zurich.ibm.com`.

[28] Trend Micro Inc. The sasser event: History and implications. *Whitepaper, Trendlabs Research*, 2004.

[29] A. Johansen and D. Sornette. Finite-time singularity in the dynamics of the world population, 2001. URL: `citeseer.ist.psu.edu/johansen00finitetime.html`.

[30] Sudarshan K.Dhall Jonghyun Kim, Sridhar Radhakrishnan. Measurement and analysis of worm propagation. *School of Computer Science, University of Oklahoma*, 2004.

[31] Stefan Misslinger. Internet worm propagation. *Department of Computer Science, Technische Universität München*, 2003.

[32] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. The spread of the sapphire/slammer worm. *Technical report, support by NSF, DARPA, Silicon Defense, Cisco Systems, AT&T, NIST, and CAIDA members*, 2008. URL: `http://www.caida.org/publications/papers/2003/sapphire/sapphire.html`.

[33] David Moore, Colleen Shannon, and Jeffery Brown. Code-red: a case study on the spread and victims of an internet worm. In *Proceedings of the Internet Measurement Workshop (IMW)*, 2002. URL: `http://www.caida.org/outreach/papers/2002/codered/codered.pdf`.

[34] Adilson E. Motter and Ying-Cheng Lai. Cascade-based attacks on complex networks. *Phys. Rev. E, 66(6):065102*, Dec 2002.

[35] MEJ Newman. The Structure and Function of Complex Networks. *SIAM Review*, 45(2):167–256, 2004.

[36] MEJ Newman, S. Forrest, and J. Balthrop. Email networks and the spread of computer viruses. *Physical Review E*, 66(3):35101, 2002.

[37] R. Pastor-Satorras and A. Vespignani. Epidemic dynamics and endemic states in complex networks. *Physical Review E*, 63(6):66117, 2001.

[38] Romualdo Pastor-Satorras, Alexei Vazquez, and Alessandro Vespignani. Dynamical and correlation properties of the internet. *Physical Review Letters*, 87:258701, 2001. URL: `http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0105161`.

[39] Romualdo Pastor-Satorras and Alessandro Vespignani. Epidemic dynamics and endemic states in complex networks. *Physical Review E*, 63:066117, 2001. URL: `http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0102028`.

[40] Romualdo Pastor-Satorras and Alessandro Vespignani. Epidemic dynamics in finite size scale-free networks. *Physical Review E*, 65:035108, 2002. URL: `http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0202298`.

[41] Roland Regoes. Stochastic simulation of epidemics. *notes of Modelling Course in Population and Evolutionary Biology, Institute of Integrative Biology, ETH Zurich*, 2007.

[42] Konstantin Rozinov. Reverse code engineering: An in-dept analysis of the bagle virus. *Systems and Software Group, Bell Labs - Government Communication Laboratory - Internet Research*, 2004.

[43] Dominik Schatzmann. Analyzing network traffic from the SWITCH network from 2003 until today. *Master Thesis MA-2007-12, Communication System Group, ETH Zurich*, 2007.

[44] C. Shannon and D. Moore. The spread of the Witty worm. *Security & Privacy Magazine, IEEE*, 2(4):46–50, 2004.

[45] Loris Siegenthaler and David Benninger. Software Package for Netflow V9 Data Analysis. *Diploma Thesis, Hochschule Rapperswil*, 2007.

[46] Reginald Smith. Instant messaging as a scale-free network. *University of Virginia, PACS No. 89.75.Hc, 89.65.20.Hh*, 2006.

[47] D. Sornette. *Critical phenomena in natural sciences*. Springer New York, 2000.

[48] D. Sornette. Endogenous versus Exogenous Origins of Crises. *Arxiv preprint physics/0412026*, 2004.

[49] D. Sornette and A. Helmstetter. Endogenous versus exogenous shocks in systems with memory. *Physica A*, 318(3):577–591(15), 2003.

[50] Didier Sornette, Thomas Gilbert, Agnes Helmstetter, and Yann Ageon. Endogenous versus exogenous shocks in complex networks: An empirical test using book sale rankings. *ETH Zurich, Department of Management, Technology and Economics*, 2003.

[51] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to 0wn the internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium (Security '02)*, 2002. URL: `http://www.icir.org/vern/papers/cdc-usenix-sec02/`.

[52] SWITCH - The Swiss Education and Reserach Network, 2008. URL: `http://www.switch.ch/`.

[53] The Symantec Security Responce Database, 2008. URL: `http://www.symantec.com/business/security_response/index.jsp`.

[54] Troy Tassier. Sir model of epidemics. *Epidemics and Development Policy, Fordham University NY*, 2005.

[55] Theus Hossmann Thomas Dübendorfer, Arno Wagner and Bernhard Plattner. Flow-level Traffic Analysis of the Blaster and Sobig Worm Outbreaks in an Internet Backbone. *Proceedings of DIMVA 2005, Springer's Lecture notes in Computer Science (LNCS), GI SIG SIDAR Conference 2005*, 2005.

[56] Kaspersky lab virus list, 2008. URL: `http://www.viruslist.com/en/viruses/`.

[57] Steve R. White. Open problems in computer virus research. *Virus Bulletin Conference, Munich Germany*, 1998.

[58] SWITCH Network Traffic Weather Map, 2008. URL: `http://www.switch.ch/network/operation/weathermap/`.

[59] Cliff Zhou, Don Towsley, and Weibo Gong. Email virus propagation modeling and analysis. *Technical Report TR-CSE-03-04, University Massachusetts, Amherst*, 2003.

[60] C. Zou, W. Gong, and D. Towsley. Code red worm propagation modeling and analysis, 2002. URL: `citeseer.ist.psu.edu/article/zou02code.html`.