ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Roman Hiestand

Simulation-Based Analysis of Internet Worm Characteristics

Student Thesis SA-2003.26
28th May 2003 / Summer Term 2003

Tutors: Arno Wagner, Thomas Dübendorfer
Supervisor: Prof. Dr. Bernhard Plattner

Abstract

Distributed Denial of Service (DDoS) is an important issue since fast spreading worms are a real possibility today. Worms, which spread through the Internet by searching and infecting hosts have been a serious problem since the first worm, called Morris Internet worm, appeared. On the 25th January 2003 a worm called Sapphire/SQL Slammer infected at least 75'000 hosts within 10 minutes and has shown that fast worms are no longer only a theoretical issue.

This semester thesis provides a careful analysis of worm parameters and explores the question what enables worm spreading of these speeds. This analysis has been done using a simulation tool, which has been implemented during this thesis specifically to explore the various parameters. In this report the implementation is explained in detail, too.

Furthermore an Internet model has been developed, to represent the Internet in the simulations.

The simulation can simulate the spreading of Code Red quite well and furthermore demonstrates the difficulties to find an appropriate Internet model for UDP worms, which have a fast spreading speed like Sapphire/SQL Slammer.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Problem description

1.1.1 Subject

Distributed Denial of Service (DDoS) attacks are a threat to Internet services ever since the widely published attacks on e-bay.com and amazon.com in 2000. ETH itself was the target of such an attack 6 months before these commercial sites where hit. ETH suffered repeated complete loss of Internet connectivity ranging from minutes to hours in duration. Massively distributed DDoS attacks have the potential to cause major disruption of Internet functionality up to and including severely decreasing backbone availability.

1.1.2 Attack Model

Most DDoS attacks share a common pattern: An *infection phase* where the initiator acquires the attack resources by compromising a large number of weakly protected hosts, ideally causing little or no visible change in host behavior, in order to make the compromise hard to notice. An infection phase can last from less then 10 minutes to several months. Attacks within the order of 100.000 and more compromised hosts have already been observed in practice (Code Red, Sapphire).
In a second phase, the *attack phase*, the attacker uses the compromised hosts to initiate actual attacks on a target computer or network. These attacks can be done autonomously or under direct or indirect control of the attacker.

1.1.3 Semester Thesis Task

The focus of this semester thesis is to use simulations to explore the parameter space for worms. Typical parameters are *time to infect a host*, *number of parallel scans*, *scanning strategy*, *amount of data needed for infection* and others. While some worms may scan and infect at the same time with a single UDP packet, others need an explicit host and vulnerability scan first via TCP and then a second TCP connection to actually infect the target.
Different choices for these parameters lead to different spreading speeds, different patterns in the observed scanning and infection traffic and different side-effects like network overload.
The tasks was to identify ideal choices for these parameters, together with the resulting observable traffic characteristics, as well as a partial exploration of sub-optimal parameter choices.

1.2 Approach

The semester thesis has been split into several subtasks. The subtasks have been solved in the following sequence.

1.2.1 Understanding Worm Characteristics

Worms can use different protocols and mechanisms to propagate themselves. They are multifaceted and the parameter choice determines the spreading characteristics. An in-depth look at past worms enables an understanding of common and future worm behaviour. The goal is to understand the possible methods a worm can use to propagate and identify the associated parameters and traffic patterns resulting from them. A lot of reading and Internet research has been done in this phase of the thesis. A discussion of the worm characteristics can be found in section 3.

1.2.2 Selection of the Internet Model

An effort has been done to find information about an appropriate model in the literature. The description of the Internet model is given in section 2. This simulation is quantitative and abstracts from individual hosts and instead regards groups of hosts that have a certain type of interconnectivity and a certain type of connectivity to the rest of the Internet.

1.2.3 Design and Implementation of a Simulator for Worm Spreading

A simulation architecture has been designed that is appropriate for the Internet model(s) chosen in the previous step. The main result of a simulation are a time vs. infection percentage statistic and a statistic of the generated scanning and infection traffic (amount vs. time).
Care has been taken that the architecture is modular enough to allow its partial or full re-use in future work. A big effort has been done during this design phase to ensure modularity.
A prototypical simulator based on this architecture, using the scripting language Perl, has been implemented.
The design and implementation of the simulator required a lot of time. Especially implementing the differences between the two transport protocols TCP and UDP asked for an additional effort.
The architecture and description of the program can be found in section 4.

1.2.4 Exploration of Typical Worm Parameters

The created simulator allows simulation of worm spreading patterns for worms with different characteristics. This parameters have been explored in the test and simulation phase of the thesis. A lot of simulations have been done to prove the correctness of the results, which the simulator outputs.
Simulations, with parameters of worms observed in the past, have been executed, too.
The results and comparisons can be found in section 5.

1.2.5 Documentation

The documentation has been written with LaTeX and contains all results. The conclusions and an outlook of further work and research can be found in section 6, the summary in section 7 and the bibliography and acknowledgments in the appendix.

The main effort has been made for researching the parameters and worm strategies and for the architecture and implementation of the simulator.

1.3 Related Work

A lot of information about worms can be found in the Internet, especially about past worms. Several studies have been done at CAIDA.org to observe past worms and to develop ideas of further worm characteristics and scanning methods. Few work has been done on simulating worm spreading. The following subsections give an overview of related work.

1.3.1 DDoS Attacks

The tutorial *Defending against flooding-based distributed denial-of-Service Attacks* from Rocky K. C. Chang, [1], gives a good overview about DDoS attacks. It gives answers to the question of how an attack can be launched with several methods and what sort of defending methods could be implemented against DDoS.

1.3.2 Worm Characteristics

The two articles *How to 0wn the Internet in Your Spare Time*, [2], and *Potential Strategies for High Speed Active Worms: A Worst Case Analysis*, [3], gives a lot of information about possible worm scanning methods and the effects to the Internet. The scanning methods of past worms and ideas, how more effective worms could look are described in detail. A worst case analysis shows a worm which might infect 9 million web-servers within 30 seconds.

1.3.3 Worms Observed in the Past

Most information can be found about worms observed in the past. Every anti-virus program web-page yields information about this worms. Most of them give mainly information about preventing an infection caused by a worm or a detailed analysis of a worm. E.g. [4] explains the

disassembled sapphire worm. But a lot of them do not give information, which can be used in this thesis.
Anyway a few of them yield very interesting information. Caida.org provides a detailed analysis of CodeRed in [5]. The overview page has links to several subtopics. Among other things an animation of the spreading of Code Red Iv2 can be found. Another subtopic gives detailed infection statistics of Code Red Iv2, [6].
The paper *Code Red Worm Propagation Modeling and Analysis*, [7], yields up with a two-factor worm model to calculate the number of infected worms theoretically. Simulations based on this calculations are made. Moreover, these papers give some information about the Code Red behaviour.
Furthermore, the paper *Code-Red: a case study on the spread and victims of an Internet worm*, [8], shows measurements of the spreading characteristics and the victims of an Internet worm. Further information about Code Red can be found in [9] and [10].
The report *Global Routing Instabilities during Code Red II and Nimda Worm Propagation*, [11] investigates into the research of BGP messages during the worm spreading.
There are several pages, which publish information about the Sapphire/Slammer Worm. An interesting paper is published by caida.org and is called *The Spread of the Sapphire/Slammer Worm*, [12]. Further information can also be found for example on the web-page of Robert Graham [13].

1.3.4 Internet Model

An extensive measurement study of Peer-to-Peer file sharing systems was made at the University of Washington [14]. This study surveyed the traffic in Peer-to-Peer file sharing system. Some information about the bandwidth and latency of the participating hosts can be found.
The Atlas of Cyberspaces [15] show network topology maps created by Internet Service Providers and Internet backbone operators.
The Internet Protocol is defined in RFC 791, [16].

1.3.5 IPv6

The address architecture of IPv6 is defined in RFC 3513, [17].
In [18] among other things a spreading calculation of Sapphire with IPv6 is presented.

2 Internet Model

The simulation abstracts from individual hosts. Therefore a model, which represents the Internet, is defined. This representation should fit the real Internet so well, that the simulated results correspond to observed traffic and infection of worms in the past but should also be as simple and basic to enable fast simulations.
An abstraction from individual hosts drives the model to divide the hosts into groups. Each group identifies hosts with similar connectivity. The main parameters to express connectivity are bandwidth and latency time. They are discussed in the next subsections in more detail.

2.1 Groups Regarding Bandwidth and Latency

2.1.1 Groups Divided by Bandwidth

Stefan Saroiu, P. Krishna Gummadi and Steven D. Gribble wrote a *Measurement study of Peer-to-Peer File Sharing Systems* [14]. They made measurements to gain more experience how Peer-to-Peer file sharing systems can be improved. For the Internet model mainly the information about bandwidth and latency of the participating hosts is interesting.
The reported bandwidths for Napster users can be found in the Table 1 divided into groups of similar bandwidths[1]. The first two columns come from the measurement study, the last two were added to further define the groups.[2] In [14] no definition of the bandwidth of Cable and DSL is given. For defining the Internet groups the assumption has been done, that Cable hosts have a bandwidth of 256 Kbps and DSL 512 Kbps.[3]

Group	Percentage		Bandwidth	Latency
14.4 Kbps	4 %		14.4 Kbps	1000 ms
28.8 Kbps	1 %		28.8 Kbps	1000 ms
33.6 Kbps	1 %		33.6 Kbps	1000 ms
56 Kbps	23 %		56 Kbps	1000 ms
64 Kbps	3 %		64 Kbps	1000 ms
128 Kbps	3 %		128 Kbps	300 ms
Cable	44 %		256 Kbps	300 ms
DSL	14 %		512 Kbps	100 ms
T1	5 %		1.544 Mbps	60 ms
T3	2 %		44.736 Mbps	60 ms

Table 1: Reported Groups for Napster [14] and the corresponding bandwidth and latency time

The measurement study observed traffic from Napster and Gnutella. Unfortunately the study does not provide a table with the exact distribution of the Gnutella hosts. But anyway it provides some information about the Gnutella hosts in the text. Table 2 shows a summary of the information, which can be found in [14].[4]

Both measurements showed approximately the same results. The percentage of Gnutella users connected with a modem is much smaller than the corresponding percentage of Napster users. Whereas Gnutella is stronger represented in the very high bandwidth connection field.

For the implementation of the simulator the simplification has been made, that only one value for the bandwidth has been chosen, representing as well the upstream as the downstream bandwidth for a certain Internet group. Asymmetric links are not simulated.[5] The

[1] Excluding the hosts reported as unknown in the measurement study.
[2] The last column "Latency" is discussed in the subsection below.
[3] There exists no explicit definition or usage of these items. Cable and DSL connection can be found from 64 Kbps to over 1 Mbps. The bandwidth for these two groups are chosen between these limits.
[4] The column "Latency" is discussed in the subsection below.
[5] Most home users, especially in countries outside of Europe, have symmetric connections. Using a bandwidth between the up- and downstream bandwidth gives a good approximation of asymmetric links, too.

Description	Bandwidth	% Napster	% Gnutella	Latency
Modems (of 64 Kbps or less)	64 Kbps	32 %	10%	1'000 ms
Rest	128 Kbps	5 %	14 %	300 ms
Broadband connections	1 Mbps	38 %	38 %	100 ms
Very high bandwidth connections (at least 3 Mbps)	3 Mbps	25 %	38 %	60 ms

Table 2: Group of hosts split regarding bandwidth and latency time into 4 groups (observed nets: Gnutella and Napster)

reported bandwidth of the measurement study is the downstream bandwidth.

2.1.2 Groups Divided by Latency

The measurements in [14] show the latencies in Gnutella Peers. Two major groups of latency can be identified, see Figure 1. An important fact is the correspondence between latency time and bandwidth. A smaller bandwidth corresponds to a higher latency time.

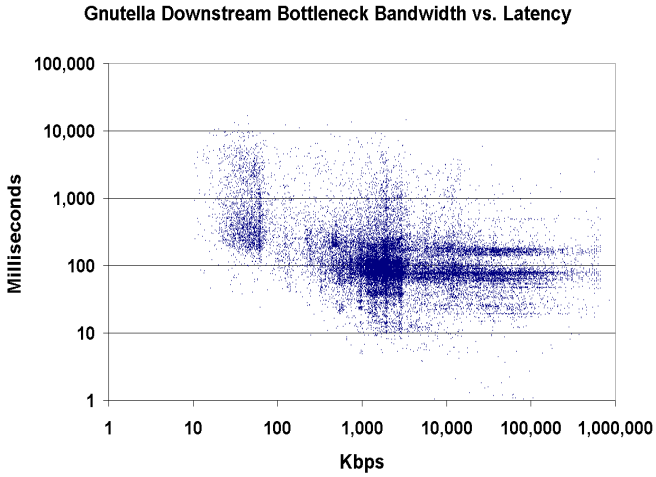

Figure 1: Gnutella Downstream Bottleneck vs. Latency [14]

The two mentioned groups are situated at:

- 20 - 60 Kbps, latency: 100 - 1'000 ms
- $>$ 1'000 Kbps, latency: 60 - 300 ms

In the second group two areas can be identified in the same bandwidth, which represent the location (geographically). One area is situated at 90 ms, the other at 120 ms. These geographical differences in the latency will not be implemented in the first version of the simulator. This is an area for further research.
For the Internet model, the latency is chosen from these measurements. In Table 1 and Table 2 in the column "Latency" an approximated value from [14] can be found.[6]

The latency completes the characteristics of the groups described in the paragraph above. The measurements summarized in table 1 give a classification of the Internet groups using 10 groups, this model will be called *10 groups Internet model*. The table 2 divides the Internet into 4 groups and gives 2 classifications differing in the percentage, they will be called *4 groups Napster Internet model* and *4 groups Gnutella Internet model*.

[6] Latency is defined as the time difference when the first bit is sent until the first bit is received at the receiving host.

2.2 Connection Between Two Hosts

The bottleneck in a connection between two hosts is the smallest bandwidth of the network path between the two hosts. All the traffic has to pass this bottleneck and is therefore limited by it.
Therefore in the simulation only the lower bandwidth of two communicating hosts is used to model the connection behaviour. (E.g. a host with a 1 Mbps connection to the Internet is trying to infect a host with a 56 Kbps connection. The connection will be modeled as full 56 Kbps connection.)
Correspondingly, the bigger latency gives the limiting time between two hosts.

TCP uses a so called *slow start* mechanism to prevent network overload. When a infected host starts to send scans out, the maximal possible bandwidth is only reached if the packages do not collide with other packages during a certain time. Small TCP worms only use a small bandwidth and therefore the maximal bandwidth will never been reached and this mechanism has no infect on the spreading behaviour. This slow start mechanism is not implemented in this thesis. The assumption made in this thesis is that the maximal bandwidth can be used from the beginning on.[7]

2.3 Backbone Routers vs. the Last Mile During an Attack

During a worm propagation the smallest bandwidth of the network path between the two hosts is the determining factor and therefore the last mile decides about the worm propagation speed. During the spreading of Sapphire/SQL Slammer there were several reports of Internet backbone disruption, but most of the backbone providers appear to have remained stable throughout the epidemic and therefore did not limit the traffic amount. During the spreading of Sapphire the bandwidth of the individual hosts was overloaded with copies of the worm and consequently many individual sites lost connectivity. [12]
The Internet models chosen for this simulation do not deal with the Internet backbones, because the reported disruption during past worms were small compared to the effect of the overload of the last mile.

2.4 Number of Host and Vulnerable Hosts in the Internet

There are 2^{32} possible IP adresses in the address space. In the simulator the number of addresses can also be chosen different.
The number of vulnerable hosts in the Internet says how many hosts in the Internet have the security hole and could therefore get infected during the worm spreading. A host is either infectable or not and it cannot change its properties during the spreading. Furthermore a already infected host cannot be reinfected. Therefore with TCP every vulnerable host receives the worm at most once. For UDP, where the worm is sent anyway this assumption has no effect. With TCP this assumption fits the reality if the worm is smart enough to find enough information in the ACK+SYN reply package to know when the scanned host is vulnerable or when it is not.

[7] In a simulation a smaller bandwidth can be chosen to simulate the effect of the slow start mechanism.

3 Worm Characteristics

Different kind of worms have already been observed in the past. They use various transport protocols, differ in size and choose any scanning methods and other parameters. This section discusses the worm parameters and finally points out, which of them are used for the simulation and which are not observed in this thesis. The first subsection restricts the term worms to the worms observed in this thesis. The next subsections discuss the different parameters.

3.1 Worm Definition

There exist various definitions of worms, a good one is given in [2]:
"Programs that self-propagate across the Internet by exploiting security flaws in widely-used services."
This semester thesis will focus on so-called active worms which propagate without human interaction; viruses and e-mail worms are not part of this thesis.[8]

3.2 Possible Worm Transmission

Observed worms used a variety of different possibilities to spread themselves. The multi-vector worm Nimda for example spread using at least five different ways [2]:

- By infecting web servers from infected client machines via active probing for a Microsoft IIS vulnerability.
- By bulk emailing itself as an attachment to email addresses determined from the infected machine.
- By copying itself across open network shares.
- By adding exploit code to Web pages on compromised servers in order to infect clients which browse the page.
- By scanning for the backdoors left behind by CodeRed II and also the "sadmin" worm.

Spreading, by using several ways, results in a much higher scanning rate and consequently in a higher infection rate.[9]

3.3 Scanning Algorithms

Past worms have used various scanning algorithms. A fast and successful infection of the vulnerable hosts is strongly dependent from the scanning strategy. Below can be found a list of observed and theoretical possible methods.

3.3.1 Single Stage Scanning - Code Red I

The idea of the single stage scanning strategy is to choose IP-addresses randomly without any constraints. The first two versions of Code Red (CRv1 and CRv2 respectively Code Red I) and Sapphire/SQL Slammer Worm implied this strategy. Code Red II implemented an upgrade of this simple scanning algorithm. It used a localized scanning strategy, where it scans 3/8 of the next scanned hosts in the same class B net, 1/2 in the same class A net and only 1/8 from the whole Internet.
This strategy was quite successful. It allowed to infect hosts which are geographically close to each other, which results in shorter latency time. Secondly vulnerable hosts stay often close to each other and finally after passing the external firewall, it could propagate within the internal net with little effort.

[8] There are several restricting reasons, if one observes only active worms. The most important reason is that most worms with human interaction spread much slower than comparable automatically spreading worms. Moreover a lot of anti-virus programs help to avoid spreading e-mail worms or viruses which can only be simulated as a constant rate.

[9] Nevertheless the probably fastest worm in the past - Sapphire/SQL Slammer - spreads with only one transmission possibility.

3.3.2 Multi-Vector Worms - Nimda

Multi-vector worms exploit several different methods to spread themselves. Nimda e.g. used at least five different methods. [2] Observing the traffic resulted by the propagation from Nimda, scan rates of 100 scans per seconds were recognized.
A big advantage of using mails as propagation possibility is, that mail traffic can pass firewalls mostly untouched. Therefore Nimda could invade effectively into internal networks and use other scanning methods to spread itself within the internal network.

The following strategies have not been observed yet, but could be implemented within a short time.

3.3.3 Hit-List Scanning

Achieving a very rapid rate of infection, one of the major problem is to infect the first few thousand hosts. (E.g. It took 9 hours until Code Red I infected the first 10'000 hosts, 210'000 hosts have already been infected 9 hours later.) There is a simple way to solve this problem. It is called *hit-list scanning*.
The overall idea is to collect several thousand potentially vulnerable hosts in a list, before the worm spreading starts, to hit them first. This results, from the spreading perspective, in a higher success rate of vulnerable hosts and consequently in a faster spreading behaviour.
To gain a list of potential victim hosts there exist several possibilities. In [2] a detailed description of the possibilities can be found.
The more hosts the hit-list contains the more effective the strategy is but a long hitlist is paid by a higher data amount.[10]

3.3.4 Permutation Scanning

Another big problem which a worm faces, while achieving a very rapid rate of infection, is that with the random scan of the IP range, a lot of hosts are scanned several times. This effect results not only in a lower scanning rate, it can even results in a breakdown of little systems due to the congestion of the net by scanning traffic.
A possible solution of solving this limitation to very fast infection is the so-called *Permutation scanning* [2]. The worm has implemented a pseudo random permutation of the IP address space. A host, infected during the hitlist phase, starts scanning by using the permutation algorithm starting just after its own IP address. When the worm sees an already infected machine, it starts the scanning from a new randomly chosen IP address and starts the permutation list again. After having encountered several infected hosts the worm stops the scanning process to avoid congestion by scanning traffic.
Permutation scanning is only possible with a protocol, where the sending host gets some information about the transmitted packages, therefore UDP as transport protocol (as Sapphire implemented) is no possibility.
There exist also ideas of so called *partitioned permutation scanning* [2]. Partitioned permutation scanning implements a fragmentation of the permutation range after each infection and finally changes to permutation scanning. This method is more complex to implement than normal permutation scanning but could effect a very effective distribution of the worm with lower congestion traffic.

3.3.5 Warhol Worm

Worms, which implement a combination of hit-list scanning and permutation scanning are known under the name *warhol worm* [2]. There have been simulated infection behaviour of less than 15 minutes to infect all vulnerable hosts in the Internet.[11]

[10] A hitlist with about 9 million server addresses is about 48 MB and compressed at least 7.5 MB depending on the compression characteristics. [2]

[11] The most important parameter chosen for this simulation are: a vulnerable population of 300'000 hosts, a hitlist with 10'000 entries and permutation scanning which gives up when two infected machines are discovered without finding a new target. [2] provides a detailed description of the simulation parameters and results.

3.3.6 Topological Scanning

Topological scanning [2] uses information about the topological situation from the target victims. This scanning technic could be implemented in an early propagation phase.

3.3.7 Flash Worms

[2] shows how effective a variant of the hit-list scanning from worm perspective can be. Starting with a hit-list with 12.6 million Web servers and a worm, which can handle multiple threads, calculations show an infection of all vulnerable hosts in less than 30 seconds.
Although it is difficult to write such a complex worm and to collect a hit-list of this size might not be done without being noticed, this calculation shows how big the risk for the worldwide Internet is and how important it is to put more effort into the research of worms to avoid such nightmares.

3.4 Latency Limited vs Bandwidth Limited

Even though Code Red I and Sapphire/SQL Slammer used the same scanning strategy, namely random scanning without any location constraints, the propagation speed differed about several orders of magnitude. Sapphire doubled in size every 8.5 seconds, the Code Red Iv2 worm population had a doubling time of about 37 minutes [12].
The reason of this enormous difference can be found in the different way of propagating itself. Sapphire uses a single UDP packet with a size of 404 bytes (worm size of 376 bytes plus the header). Due to the chosen transport protocol Sapphire was independent of the latency time and only dependent on the bandwidth and consequently could send as many scans as the host could produce.[12]
Whereas Code Red used the transport protocol TCP and therefore needs to complete the three way handshake and consequently is latency limited.[13]
Sapphire did not implement a highly complex scanning algorithm with a hit-list or another scanning strategy feature as explained in section 3.3. Only the choice of the transport protocol resulted in such an effective spread within a quarter hour. Additionally, the relatively small amount of data compared to CodeRed facilitated the worm spreading.
Within this short time of 20 minutes no human interaction is possible, thus automatic filtering systems and traffic observation become more and more important.

3.5 Number of Parallel Scans

The previous subsection has shown the difference resulting by the choice of the transport protocol. One limiting factor in TCP is the number of parallel scans.
Worms, observed in the past, have already used the possibility of parallel scans using TCP, as transport protocol. For each scan a thread has to be launched and therefore the maximal number of parallel scans is not limited by the number the creator of a worm defines, but by the number of threads a system can handle. The context switch overhead is significant and there are insufficient resources to create enough threads to counteract the network delays. [12]
Therefore a worm spreading with TCP is always limited by the number of parallel scans at least in the first phase of the infection.
UDP worms do not scan parallel. Due to the fact that UDP worms do not have to wait for an answer from the receiving hosts, a UDP worm sends as many scans as possible and is therefore limited by - as mentioned in section 3.4 - the bandwidth. Consequently the number of scans can be measured as ”scans per second”.[14]

[12] When attached to a gigabit link, a single Pentium 4 2 GHz machine can generate several hundred megabits-per-second, sending out over 100’000 UDP packets per second. [13]

[13] Code Red became also bandwidth-limited after 15 hours. A lot of scanning traffic resulted into a congestion in various networks. This effect cannot be observed with the current implementation of the simulator.

[14] The implementation of the simulator takes the minimum of (a) the number of possible scans limited by the bandwidth and (b) the maximal possible scans a host can produce defined by an input parameter. That means if a host could produce 100’000 scans per second, but the bandwidth limits the output to 30’000 scans per second, the number of sent scans per second is limited by the bandwidth.

3.6 Relevant Ideas and Parameters for the Simulation

This section describes which of the above mentioned strategies and concepts are relevant for the simulation and which are implemented.

3.6.1 Scanning Strategies

The worms differ most in the scanning strategy. In the simulator, created during this thesis, random scanning, hitlist scanning and a special form of geographical scanning are implemented.

Random Scanning: For *Random scanning* it is expected that the random generator of the worm is perfect and therefore randomly chooses one address from the whole address range of hosts[15]. A bad implementation or even a wrong use of a random generator is not implemented.

Hitlist: Using a Hitlist enables a fast and efficient start of a worm spreading from a worm perspective. Therefore this scanning strategy is implemented in the simulator. A changing worm size is not implemented in this first version of the simulator and consequently only an approximated spreading speed can be simulated.[16]

Local forced Scanning: So called *Local forced scanning strategy* is a first step into the direction of geographical or topological scanning. With a certain rate the scans are executed within the same group of hosts. Only with the inverse rate other groups are scanned.

The other scanning strategies, which are mainly permutation scanning and some combinations of permutation scanning with implemented strategies, are not implemented now.

3.6.2 Transport Protocol

The transport protocols UDP and TCP are implemented and cause a different program run. See section 4.1.1 for more details.
UDP is implemented without any restrictions. The sender sends out as many scans per second it can produce and the bandwidth of the sending host can deliver. The receiving host can receive as many scans as possible within the total bandwidth of all receiving hosts from the receiving group.
TCP is simplified. Instead of sending two packages to establish a connection, only one package of the double size is sent at a certain time. The implementation of the simulator does not allow to react on a received package and therefore the whole data amount of an infection has to be sent at once as one theoretical package. Packages from the receiving host as SYN and ACK are not observed. They do not infect the sending host and the traffic in the first part of the simulation. During the first phase of an infection all hosts in the Internet are receiving packages from a very small group of hosts and can handle them easily. The download stream of the sending hosts does not receive as much traffic as the upload stream sends and also here the receiving packages do not have to be observed, because they can be handled, too. Furthermore, by using TCP the restriction is mostly the number of parallel scans and not the bandwidth.

3.6.3 TCP Timeout

The timeout is an important factor in TCP. If the sending host does not receive an ACK on a sent package, by default TCP waits 120 second and then resends the package. CodeRedIv2 set the timeout to 21 seconds and did not resend the scan after the timeout [7]. The two parameters, timeout duration and enable resp. disable resending, are completely implemented in the simulation.

[15] The number of hosts correspond in general to the full address space, which are 2^{32} possible IP addresses.

[16] A worm using a hitlist strategy contains the normal worm, the hitlist and the code for the hitlist spreading. A good implementation of a hitlist changes the worm size in every spreading step. The hitlist is split at every new infected host into two parts and the additional data amount caused by the hitlist decreases therefore inverse exponential. After the hitlist phase the worm get his normal size, without any hitlist data.

3.6.4 Worm Size

The worm size is an important worm parameter and is therefore implemented. The additional header length of a package depends on the transport protocol and on the size of an IP package and is implemented, too.
A single IP package can have a size of 65536 Bytes[17]. And therefore the maximal length of a worm to be sent in one IP package is 65508 Bytes. This is a theoretical value, because of IP fragmentation by the underlying network protocol, the IP package is split into several packages and the amount of data is consequently bigger than with only one IP package. Regarding that worms observed in the near past, as CodeRed and Sapphire/SQL Slammer were much smaller[18] than this theoretical maximum, the whole worm can be sent at once.

3.6.5 Number of Parallel Scans

The number of parallel scans are implemented as described in section 3.5 and consequently the meaning of this constant differs between the two possible transport protocols.

3.6.6 Time to Infect a Host

The time to infect a host slows down the worm propagation speed. Additional time to infect a target host can be used to execute a program on the target host, to reboot a machine or to sleep for a certain time. If an infecting host does stay in a sleep modus it is more difficult to detect an infection than if the infected hosts starts sending out scans - this effect of being less noticeable could be desired by the worm creator.
All this additional time can be chosen as one constant time and will effect a retarded infection of the host. A retarded infection means, the host is infected after the whole additional time, even if the real infection was some time before. Indeed this restriction does not infect the scans to new hosts but block new scans under TCP. Therefore a very long additional time causes very long waiting times of the sending hosts and blocks them herewith.

3.6.7 Further Protocols

Further protocols, as ARP or ICMP, or e.g. BGP[19] messages caused by the worm propagation are not observed in this thesis.

3.6.8 Overview of All Parameters

Table 3 provides an overview of all worm and Internet parameters observed in this thesis and gives their value range. Most of them can be chosen within a wide range. A vulnerable population of 1 host, or a worm size of 0 bytes does not make any sense for a simulation of a worm spreading.

3.7 The Effect of IPv6 on the Spreading Behaviour of Worms

IPv6 offers a 128 bit address space [17]. These are $3.4 * 10^{38}$ possible addresses. A random scan to an address will not effect into a successful infection of a host. Assuming a population of 1 million vulnerable hosts, the probability to choose randomly a vulnerable host is only $2.9 * 10^{-31}\%$. A worm does not want to scan the whole address space to infect for example 1'000'000 computers. If Sapphire ran under IPv6 it would take approximately four billion times the estimated life of the universe before the next host would be infected [18].
With IPv6 worm creators have to develop other scanning algorithms to find host addresses of possible victims. How worms will look running under IPv6 is not known so far.

[17] Per definition of the Internet Protocol [16] the maximal length of the IP package is 2^{16} Bytes.
[18] CodeRedIv2 had a size of 4 KB and Sapphire SQL Slammer was only 376 Bytes small. [12]
[19] BGP message storms have been observed during the worm propagation period of CodeRed and Nimda. [11]

Worm parameter	Units	Lower limit	Upper limit
Number of hosts in the Internet	hosts	1	4'294'967'296
Number of vulnerable hosts	hosts	1	Number of hosts in the Internet
Start population	hosts	1	Number of vulnerable hosts
Simulation time	sec.	0	no limit
Transport protocol	TCP or UDP	–	–
TCP resending	enable/disable	–	–
TCP timeout	ms	0	no limit
Worm size (without header)	Bytes	0	65508
Number of parallel scans resp. Number of scans per seconds	–	0	no limit
Time (additional) to infect a host	ms	0	no limit
Hitlist	enable/disable	–	–
Hitlist length	hosts	0	Number of hosts in the Internet
Probability a host in the hitlist is vulnerable	%	0	100

Table 3: Overview of worm parameters observed in this thesis

4 Program Architecture

The program implemented during this semester thesis offers the possibility to simulate the worm propagation and the volume of traffic caused by the worm. Choosing parameters concerning the worm behaviour and the Internet model lets the user observe propagation with various behaviours.
In this section a detailed description of the program architecture is given. The first subsection describes the general concept and program flow. In the next subsections the implementation with Perl is given in more detail.

4.1 Concept

4.1.1 General Concept

The program run is split into various parts, which are executed once or several times. These parts are listed below:

- Initialization, input reading, arrays and log-file creation and start plotting:
  Is executed once at the beginning.
- Availability:
  Checks the availability of the sending hosts. How many scans can be sent from each group. Is executed in every time step.
- Scanning:
  Calculates the number of scans from each group to each group. The output is a matrix. Is executed in every time step.
- Sending and Infecting, Time calculation:
  Sends the scans from the sending hosts, receives them at the receiving hosts and infects the receiving hosts. Calculates the time needed and the amount of bandwidth and controls the allowed number of parallel scans. Is executed in every time step.
- Write log-files, update plots and clean the receiving and sending arrays:
  Is executed in every writing interval step resp. plotting or cleaning interval step.
- Final log-file writing and ps-file creation:
  Is executed once at the end of the program run.

A detailed overview is given in Figure 2. It shows the program run and all parts mentioned above can be recognized in this figure.

Figure 3 shows the data flow of the program. In the figure five columns can be recognized. The left most column shows the input data, right most column the generated output data. The three middle columns mainly show the program parts (main routine or subroutines) which read the input resp. which produce the output.

There are three core arrays which are used during the whole simulation time. The next two paragraphs describe these three arrays and the concept behind them.

4.1.2 Group Array

There is one array which saves the behaviour of the Internet model.
As discussed in section 2 the Internet is modeled as several groups of hosts and all hosts in one group have the same behaviour. The file `file/internetmodel.txt` contains as many lines as groups in the model exist and in each line the following information can be found (separated by ”;”):
”group-number; bandwidth (in bits per seconds); latency time (in milliseconds); percentage in the Internet (the total of all lines is 100%); number of infected hosts (start population)”
This information are stored in `$group` and can be accessed by using `$group->[group number][element number]`.

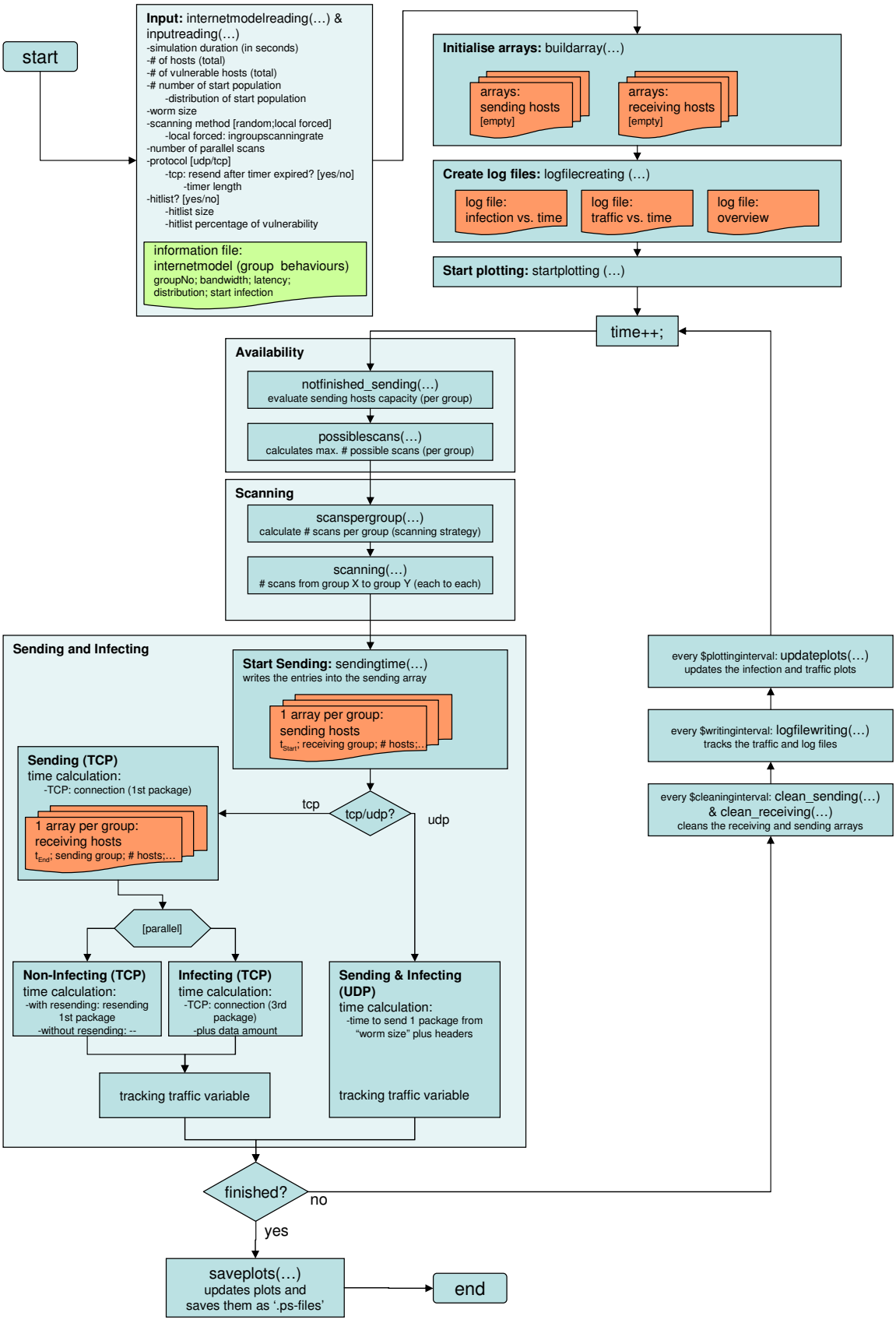

Figure 2: Architecture and Control Flow of the Simulator

The last entry of each line "number of infected hosts" saves always the actual infected hosts. Starts with the start population and adds the new infected hosts. The total over all groups gives

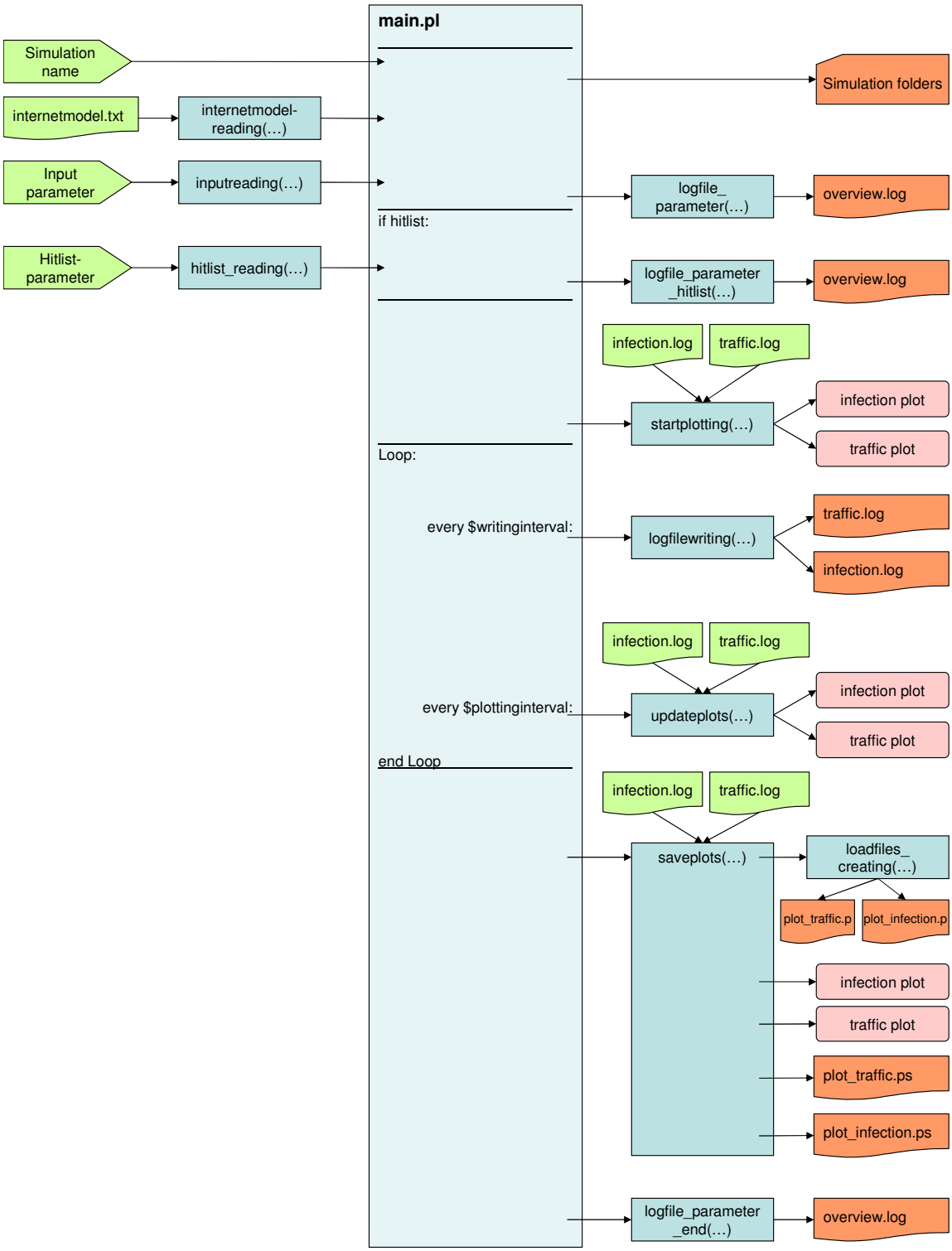

Figure 3: Data Flow of the Simulator

the total infected hosts in the Internet in a certain moment.

4.1.3 Sending and Receiving Arrays

There are two further important arrays during the whole simulation time. They are initialized by calling the subroutine `buildarray($groupno)`. The idea is to save all scans on both sites - sender and receiver site. And then go through the list and check for every entry if it can be sent respectively if it can be received.

This means, on the sender site, we can handle each entry and check if we can send it in a certain time step. If it is possible we write this entry into the receiving array to the group the scans go to. After going through all entries of the sending array we go through the whole receiving array and check if we can receive this scans and consequently change new infected hosts and raise the traffic amount.
To be exactly this are array of references to arrays. Therefore we have three dimensions, which are:

1. This dimension is the group number. For the sending array it is the sending group and for the receiving the receiving group.
2. The length of the second dimension changes in every simulation step several time. This dimension has for every time step several entries. An entry can be understood as a line.
3. The last dimension are the elements of such a line. For every new entry in the second dimension a whole line is added. The element of such a line are:
   - 0: Time, when the entry has been written into the array.
   - 1: Only for the receiving array: Earliest time, when the receiving host could finish the infection from his side. (included is latency time and time to final infect a host (e.g. time to reboot a host))
   - 2: Indicates to respectively from which group go/comes the scan. This is important for the bandwidth and latency time and to write an entry from the sending array into the correct group of the receiving array.
   - 3: Amount of Scans in real numbers.
   - 4: Percentage of not finished scans. If this is 1 it means that no scans of the total number of scans given in the element 3 are executed. 0 means all scans are executed and finished.
   - 5: This indicates if the scan is during the hitlist phase (value 1) or not (0). For worms without a hitlist this has always the value 0.
   - 6: The 6th element is only needed for the TCP transport protocol and in the receiving array and says if the scans will infect the scanned hosts (1) or not (0). UDP checks this at the end and therefore this entry is not needed with UDP. TCP does not send the same data amount to not-vulnerable hosts as it sends to vulnerable ones and therefore this difference has to be treated in an earlier step than with UDP.

To access a certain entry, the following command can be written `$array->[group number (from 0 to the total number of groups in the Internet model)][line number][element number (from 0 to 6)]`.

The next subsections cover the various parts of the simulator in more detail. The simulator is written with perl. There is one main routine and several module packages. These module packages contain the subroutines. To plot the infection and traffic figures the tool gnuplot is used.

4.1.4 When is a Host Infected?

In this simulation hosts are not observed separately and therefore only the behaviour of a whole group can be observed. Furthermore the program repeats several steps and simulates fixed time steps. Executed scans infect other hosts with a certain probability. All these mentioned steps result into infected hosts as fraction numbers. For example 100 parallel scans with TCP from 1 host with an infection probability of 1 to 10'000 results in 0.01 new infected hosts. This number is added to the already infected hosts.
If the new scans are calculated from the infected hosts, than the number of infected hosts is rounded down from the actual infected hosts. If the not totally infected hosts already can infect other hosts, the new scans are higher than they can be, with this round down this can be limited to the infected hosts.

Thus a host is infected when the rounded down number gives an infection. The logfiles still show fraction to enable a detailed view of the partly infected hosts, which is mainly interesting in the first quarter of a simulation.

4.2 Main Routine

The main routine leads through the simulation. It handles the program flow. The following steps are executed:

- The location of gnuplot is saved into the variable `$location_gnuplot`.
- The cleaning plotting and writing interval are initialized.
- The user is asked to enter a name for the simulation and the output folders are generated - these are a simulation output folder into the `output` folder, which has the same name as the simulation and two folders into this simulation output folder, one for the `logfiles` and one for the `plotfiles`.
- The variables `$group` and `$groupno` are initialized and the subroutine `internetmodelreading()` is called to read the Internet model file (files/internetmodel.txt) and to save the information into an array - the reference is save in `$group`.
- All variables which are used during the program run time are initialized and all counters (writing, cleaning and plotting) are set to 0.
- Calls the subroutine `buildarray($goupno)` twice to build the sending and receiving arrays.
- Calls the subroutine `inputreading(...)` to asks the user to input all information about the simulation, as simulation time, Internet model and worm parameters.
- Calls the subroutine `logfile_parameter(...)` to write the input parameter into a logfile, called `overview.log`.
- Calculates the protocol dependend parameter as wormsize including header, amount of bits to establish a connection and for TCP the time to give up scanning a host, which does not give answer.
- Calls the subroutine `logfilecreating($startpopulation,$location_logfiles)` to create a traffic and an infection log file.
- If a hitlist is chosen, calls the subroutines `logfile_parameter_hitlist` and `hitlist_reading` to ask to enter the hitlist parameter and to write them into the log file `overview.log`.
- Initializes two filehandler and calls the subroutine `startplotting(...)` to open two instances of gnuplot and start plotting.
- Enter a loop and repeat it as often as the simulation takes, chosen by the `$totalsimulationtime` or until all vulnerable hosts are infected:
  - Checks availability of the sending hosts using the subroutines `notfinished_sending(...)` and `possible_scans(...)`.
  - Calls the subroutines `scanspergroup(...)` and `scanning(...)` to calculate the number of scans from each group to each group.
  - Calls the subroutine `sending(...)` to send and receive scans and to calculate the new infected hosts.
  - Every `$cleaninterval`: calls the subroutines `clean_sending($sending, $groupno)` and `clean_receiving($receiving, $groupno)` to clean the sending and receiving arrays.

- Every `$writinginterval`: calls the subroutines `logfilewriting(...)` to write the logfiles.
- Every `$plottinginterval`: calls the subroutines `updateplots(...)` to update the running gnuplots.
- Increments all counters with `$timestep`.

- Calls the subroutine `saveplots(...)` to save the plots as ".ps"-files.

4.3 Package sub::input

Contains the subroutines `internetmodelreading` and `inputreading`.

sub::input - internetmodelreading

to call: `internetmodelreading()`
Reads the file `file/internetmodel.txt` line after line into the `$group` array and returns the reference to this array and the number of groups of the Internet model. See section 4.1.2 for more details about the group array.
The bandwidth is saved in bits per millisecond and the latency remains in millisecond.

sub::input - inputreading

to call: `inputreading($group,$groupno)`
Asks to enter the input data. The following inputs are needed and asked to enter one after one.

- Duration of the simulation [in seconds]: This is the time the simulation should simulate the propagation. The simulation will be stopped earlier if already all hosts are infected.
- Number of hosts in the Internet: The number of all connected hosts in the network.
- Number of vulnerable hosts: The number of vulnerable hosts in the network, has to be smaller than the number of hosts.
- Number of start population: This is the initial number of the start population. This input will overwrite the input from the `internetmodel.txt` file.
- Distribution of start population according to internetgroups? [Yes/no]: "Yes" calculates the number of the start population according to the percentage of the Internet groups in the model. "No" starts to request the percentage of the start population for each group.
- Worm size (in Bytes): The input will be multiplied by 8 to get the worm size in bits.
- Scanning method: The scanning method the worm uses is either random scanning or local forced scanning. Random scanning chooses the scans randomly from all host without any constraints. Local forced scanning scans with a certain rate `$ingroupscanningrate` to the same group and with `(1-$ingroupscanningrate)` to the other groups.
- Additional time to infect a host (e.g. 300 seconds to reboot): This is the additional time until a new infected host can send out scans himself, it includes all sorts of additional time.
- Number of parallel scans: The number of parallel scans has two meanings, for UDP it means the number of scans per second, for TCP, which waits for an answer it is the allowed number of parallel scans. If there are no constraints from the worm site with UDP the number is set to the maximum the average hosts can sent without any bandwidth limitations; the bandwidth limitations are treated later.
- Protocol [udp/tcp]: transport protocol, the worm uses to propagate himself.
  - TCP: resend after timer expired? [Yes/no]: TCP set a timer sending the first package to establish a connection. When the timer is up, the package will be sent again. You can disable this resending choosing "no".

  - TCP: TCP timeout (timer to resend TCP request and/or give up in seconds) [120 sec]: the duration of the timer is chosen here. The host waits to give up at least the duration of the timer. With resending the sending hosts waits the double TCP timeout time.

- Hitlist [No/yes]: To choose a hitlist.

Returns all the new input parameter.

4.4 Package sub::logfiles

Contains the subroutines `logfilecreating`, `logfilewriting`, `logfile_parameter`, `logfile_parameter_hitlist`, `logfile_parameter_end` and `printout`.

sub::logfiles - logfilecreating

to call: `logfilecreating($startpopulation, $location_logfiles)`
Creates the log file to save the traffic vs. time and the infection vs. time behaviour. They are stored into the `logfile` folder of the current simulation folder.

sub::logfiles - logfilewriting

to call: `logfilewriting($time, $group, $timestep, $groupno, $totaltraffic, $writinginterval, $location_logfiles)`
Writes the total infected hosts at this time to the `infection.log` file and the traffic in bits per second to the `traffic.log` file.

sub::logfiles - logfile_parameter

to call: `logfile_parameter($location_simulation, $simulation_name, $group, $groupno, $totalhosts, $totalvulnerable, $startpopulation, $totalsimulationtime, $wormsize, $scanningmethod, $ingroupscanningrate, $timetofinalinfect, $parallelscans, $protocol, $usetcptimer, $tcptimer, $hitlist)`
Creates a new log file, called `overview.log` to store all input parameter, the name of the simulation and the execution date and time. The file is stored into the simulation folder.

sub::logfiles - logfile_parameter_hitlist

to call: `logfile_parameter_hitlist($location_simulation, $hitlistlength, $hitlist_ptoinfect)`
Stores the hitlist parameter into the `overview.log` file.

sub::logfiles - logfile_parameter_end

to call: `logfile_parameter_end($location_simulation, $time, $group, $groupno)`
Stores the duration of the simulation and the number of infected hosts into the overview log file.

sub::logfiles - printout

to call, either: `printout($sending, $time, $groupno, $location_logfiles, "sending.log")`
or `printout($receiving, $time, $groupno, $location_logfiles, "receiving.log")`
Is not use during a simulation. It was useful during the implementation phase to check the entries of the receiving and sending arrays.
Writes the content of this arrays into a logfile. To use this printout subroutine it is recomended to remove the # signs in the subroutine `logfilecreating(...)` to initialize the log files.

4.5 Package sub::hitlist

Contains the subroutine `hitlist_reading`.

sub::hitlist - hitlist_reading

to call: `hitlist_reading($group, $groupno, $totalhosts)`
Asks to enter the inputs for the hitlist:

- Number of hosts in hitlist: This is the length of the hitlist, how many known IP-Adresses are stored in the hitlist.
- Probability that a host in the hitlist is vulnerable [%]: The common probability of all hosts in the hitlist, that such a host is vulnerable. This infection percentage will be used during the hitlist phase. It is stored as decimal number.

Returns the hitlist length and the infection probability.

4.6 Package sub::array

Contains the subroutines `endarray`, `buildarray`, `clean_receiving` and `clean_sending`.

sub::array - endarray

to call: `endarray($array, $groupid)`
Calculates the number of lines in the receiving or sending array for a certain group given with `$groupid` and returns this number.

sub::array - buildarray

to call: `buildarray($groupno)`
Initialize an array as described in section 4.1.3 and writes for each group one line filled with `0`'s.
Returns the new array.

sub::array - clean_receiving

to call: `clean_receiving($receiving, $groupno)`
Cleans the receiving array and therefore starts at the beginning of the array groupwise, checks if there are open scans in this line (checks the 5th element of the line, 0 means there are no open scans), stops as soon he finds any open scans and deletes all entry before this open one.
Returns the cleaned receiving array.

sub::array - clean_sending

to call: `clean_sending($sending, $groupno)`
Cleans the sending array groupwise. This subroutine is more complex than the subroutine above to clean the receiving array, because it checks all entries and replaces the whole array. This is necessary to avoid a lot of very small entries which get smaller and smaller and slow down the program run speed.
Again groupwise the lines are checked and scans to the same groups are collect and finally for each group only one scan to each other group is stored.
Returns the cleaned sending array.

4.7 Package sub::plotting

Contains the subroutines `startplotting`, `updateplots`, `loadfiles_creating` and `saveplots`.

sub::plotting - startplotting

to call: `startplotting($location_gnuplot,$location_plots_settings, $location_logfiles)`
Starts gnuplot in the pipe twice and start plotting in one plot the `traffic.log` and into the other the `infection.log`. Gnuplot is started with the argument `-persist`, that after the simulation program end, the plots stay open. The plots are plotted without a key, axis name and title are defined and the data points are connected with a (red) line.
Returns the two filehandlers to write to this two gnuplots later.

sub::plotting - updateplots

to call: `updateplots($location_logfiles, $gnu_infection, $gnu_traffic)`
Updates the traffic and infection plot with the new datas from the log files.

sub::plotting - loadfiles_creating

to call: `loadfiles_creating($location_plots, $location_logfiles)`
Creates load files used by gnuplot to save the traffic and infection plot as postscript files. They are written during run time, because the directory, where the plots will be saved, depends on the simulation name. The load files are stored into the `plotfiles` folder.

sub::plotting - saveplots

to call: `saveplots($location_gnuplot,$location_plots_settings,$location_plots, $location_logfiles, $gnu_infection, $gnu_traffic)`
Calls the subroutine `loadfiles_creating(...)` to create the load files. Executes the two load files to save the plots as postscript files and closes the gnuplot. The two plot windows still stay open due to the `-persist` argument, but are no longer connected to gnuplot.

4.8 Package sub::availability

Contains the subroutines `notfinished_sending` and `possiblescans`.

sub::availability - notfinished_sending

to call: `notfinished_sending($group, $sending, $groupno)`
Goes through the sending array and adds up all not finished entries. The multiplication of the number of the scans and the percentage, which is not finished gives the open scans for each line. The sum over all line of one group gives the total for this group.
Returns the number of open scans per group as reference to an array.

sub::availability - possiblescans

to call: `possiblescans($group, $notfinished_sending, $possiblescanspermillisecond, $timestep, $wormsize, $connection_etablishment, $protocol, $totalhosts, $totalvulnerable, $parallelcounter, $groupno, $totaltraffic, $hitlistphase, $hitlist_ptoinfect)`
Calculates the number of possible scans within this time step for each group in the Internet model. This number depends on the possible number of parallel scans for TCP, resp. on the number of scans per second for UDP, on the total bandwidth of the sending host and on the not finished scans. Therefore the subroutine executes the following steps groupwise:

- Calculates the possible parallel scans for TCP or the possible scans per second for UDP.
- Calculates the sending bandwidth and than divides it through the amount of bits sent within one scan. The amount of sent bits per scans is different in TCP and UDP. With UDP the worm is sent to every scanned host, whereas with TCP the worm is only sent to the

hosts which are infected. Therefore the subroutine `ptoinfect` is called to calculate the probability to infect a host if the transport protocol is TCP and the number of bits the worm is, is than multiplied with the probability to infect a host.

- The smaller of this two number (possible parallel scans and possible scans within given bandwidth) limits the number of scans and defines therefore the maximal possible scans. From the maximal possible scans the not finished scans are subtracted.
- The final check of the result ensure that scans never are negative.

Returns the possible scans as reference to an array.

4.9 Package sub::scanning

Contains the subroutines `scandistribution`, `scanspergroup` and `scanning`.

sub::scanning - scandistribution

to call: `scandistribution($group, $scanningmethod, $groupno, $ingroupscanningrate)`
Calculates the distribution in percentage of the scans from each group to each group as a matrix. The distribution of the scans depends on the scanning strategy.
For scanning method 1, which is random scanning without any constraints, the percentage of scans to a certain group correspond to the percentage of the group in the Internet model. Therefore no calculation or random generator is needed here, the values are directly stored into the matrix.
For scanning method 2, which is local forced scanning, the percentage of scans to the same group are given by the `$ingroupscanningrate`. Scans to other groups are calculated as followed:

```
$scandistribution->[$fromgroup][$togroup] = $group->[$togroup][3] /
(1 - $group->[$fromgroup][3]) * (1 - $ingroupscanningrate)
```

Every position of the matrix is calculated and stored.
Finally the subroutine returns the percentage of scans from each to each group as reference to the matrix.

sub::scanning - scanspergroup

to call: `scanspergroup($group, $scanningmethod, $possiblescans, $scanspermillisecond, $timestep, $groupno)`
Limits the possible scans per group to the real scans per group and returns these numbers as the reference to the array.

sub::scanning - scanning

to call: `scanning($group, $scanningmethod, $scanspergroup, $groupno, $ingroupscanningrate)`
Multiplies the number of scans in percentage (`scandistribution`) with the number of scans per group and returns the reference to the matrix with the effective scans.

4.10 Package sub::sending

Contains the subroutines `ptoinfect`, `sendingtime`, `time_udpsending` and `time_tcpsending`. They are splitted into the three files `sending.pm`, `udp.pm` and `tcp.pm` to avoid one very long file.

sub::sending - ptoinfect

to call: `ptoinfect($group, $totalhosts, $totalvulnerable, $groupno, $hitlistphase, $hitlist_ptoinfect)`
Calculates the probability for each group to infect a host.
During the hitlist phase the probability is given by the variable `$hitlist_ptoinfect`. If there is no hitlist implemented or after the hitlist phase, the probability is calculated regarding the total hosts in the Internet, the vulnerable hosts and the already infected host. To be exact, the calculated probability is not the probability to infect a host, it is the probability to infect a new (!) host. Herewith the problem of infecting a infected host, so called reinfection, is solved.
Furthermore there is a check made to avoid divisions by zero[20].
Returns the probability as a reference to an array.

sub::sending - sendingtime

to call: `sendingtime($group, $scanmatrix, $wormsize, $totalhosts, $totalvulnerable, $time, $sending, $receiving, $timestep, $timetofinalinfect, $protocol, $connection_etablishment, $parallelcounter, $possiblescanspermillisecond, $groupno, $totaltraffic, $hitlistphase, $hitlist_ptoinfect, $hitlistsum, $hitlistlength, $tcptimer_time)`
First of all it saves the number of new scans into the sending array.
If the simulation is in the hitlist phase probably not all scans are executed during the hitlist phase. It might happen that only one more scan could be executed in the hitlist phase and therefore infect another host with the hight probability to infect a host during hitlist phase and all other thousands of scans infect only with the normal low probability. If in this time step the change-over from hitlist phase to normal phase happen, two entries are written into the sending array, otherwise only one entry.
The first check is, if at the start of the time step the program is in the hitlist phase. If it is not, it goes directly to write all new scans into the sending array. If it is in the hitlist phase, it adds up all new scans and tests than if after this step the program is still in hitlist phase. In this case one entry is written into the sending array and this part is finished. In the last case, hitlist and no hitlist phase, the program checks how many of the scans are executed in the hitlist phase and writes them to the sending array with the element to indicate hitlist phase as 1 and writes all other scans to the sending array with this hitlist indication element as 0.
Finally it starts the subroutine `time_XXpsending` (TCP: `time_udpsending` or UDP: `time_tcpsending`) to go through the sending and receiving array to calculate the new infected hosts.
Returns the reference to the sending array.

sub::sending - time_udpsending

to call: `time_udpsending($group, $timestep, $sending, $receiving, $wormsize, $time, $totalhosts, $timetofinalinfect, $groupno, $totaltraffic, $hitlist_ptoinfect, $totalvulnerable)`
Calculates the number of new infected hosts, going through the sending and receiving array and sending and receiving as many scans as possible. It is split into three main parts. The first part sends as many scans as possible and store them into the sending array, the second part receives as many scans as possible and saves the number of scans and the third part stores the sum of new infected hosts into the `$group` array. The first to parts are described in the next two paragraphs in more details; 1st part:

1. Goes through all entries and therefore has two ”for-loops”, one for the number of groups and one for the number of lines per group in the sending array.
2. Checks if the actual line has open scans and if there is free bandwidth to send at least a part of these new scans. If one of this conditions not are satisfied it returns to step 1 and takes the next line in the sending array.

[20] Does normally not happen, only in the case the Internet model has a group with no hosts in this group. This can be if the file `internetmodel.txt` has an Internet group with a percentage of 0.

3. Checks if all scans from this line can be executed under the bandwidth constraint. If this is possible the 5th element is set to 0 to indicate this. Otherwise the number of possible scans is calculated and subtracted from the whole number and the percentage of open scans is stored into the 5th element of this line in the sending array.
4. The next step is to save these entries into the receiving array. They are now sent and at will at any time arrive at the receiving hosts, when that happens is not longer interesting for the sending host using UDP.

2nd part:

1. Goes through all entries and therefore has two ”for-loops”, one for the number of groups and one for the number of lines per group in the receiving array.
2. The maximal bandwidth of the receiving host within a certain groups is calculated.
3. Checks if the actual line has open receiving scans, if the ”end time” is before the actual time and if there is free bandwidth to receive at least a part of this new scans. If one of this conditions not are satisfied it returns to step 1 and takes the next line in the receiving array.
   The second condition with the ”end time” is to ensure that the time between sending and infecting is at least the latency time to receive a scans plus the time to final infect a host to finish the infection.
4. Checks if all scans from this line can be received under the bandwidth constraint. If this is possible the 5th element is set to 0 to indicate this. Otherwise the number of possible scans is calculated and subtracted from the whole number and the percentage of open scans is stored into the 5th element of this line in the sending array.
5. Also the number of scans is multiplied with the probability to infect a host (calls the subroutine `ptoinfect(...)`) and stored and the amount of received scans is stored.
6. The amount of received scans is multiplied with the bit amount from one scan to receive the total traffic amount caused by these scans.

sub::sending - time_tcpsending

to call: `time_tcpsending($group, $timestep, $sending, $receiving, $wormsize, $time, $totalhosts, $timetofinalinfect, $connection_etablishment, $totalvulnerable, $parallelcounter, $possiblescanspermillisecond, $groupno, $totaltraffic, $hitlist_ptoinfect, $tcptimer_time)`
Calculates the number of new infected hosts going through the sending and receiving array and sending and receiving as many scans as possible. This subroutine is quite similar to `time_udpsending` described above. The main difference is the constraint of waiting to receive the ACK and SYN from the receiving site and the consequential new handling of the number of parallel scans, which makes this subroutine more complex. The principal concept did not change and the three parts remain.
All if conditions are enlarged by this condition of maximal parallel scans. An array, which is initialized in the main routine adds up all open scans and subtract the finished once and thus has the control over all open scans, not only within a certain time step.
An other big difference in this routine compared to UDP is the different way of calculating the new infected hosts. Already on the sender site the decision is made which scans infect a host, which do not. They are special marked in the receiving array in the 6th element (0 for not infecting, 1 for infecting). The ”end time” is now different between this two entries, for the not infecting once it depends on the duration of the resend timeout and for the infecting once as in UDP latency time plus the time to final infect a host.

4.11 Package sub::auxiliaries

Contains the subroutines `min`, `max`, `totalinfected`, `connection_bandwidth` and `connection_latency`.

sub::auxiliaries - min

to call: `min($number1, $number2)`
Returns the smaller of two numbers.

sub::auxiliaries - max

to call: `max($number1, $number2)`
Returns the bigger of two numbers.

sub::auxiliaries - totalinfected

to call: `totalinfected($group, $groupno)`
Adds up the infected hosts from each group and returns the sum.

sub::auxiliaries - connection_bandwidth

to call: `connection_bandwidth($group, $group1, $group2)`
Returns the smaller bandwidth of two groups.

sub::auxiliaries - connection_latency

to call: `connection_latency($group, $group1, $group2)`
Returns the bigger latency of two groups.

4.12 Usage of the Simulator

This subsection gives a brief description of the use of the simulator. The simulator is written with Perl and contains several sub routines in the sub folder `sub`.

4.12.1 First Steps

First, change the directory to the program root folder. To give the execution permission, the command `chmod +x main.pl` has to be executed in the program folder.
At the beginning of the main routine, `main.pl`, the path to start gnuplot is stored into the variable `$location_gnuplot`. By default the location path is set to `/usr/bin/gnuplot`. If gnuplot is started from another place, this has to be changed here.[21]
The size of the time step is saved in the variable `$timestep` and can also be changed at the beginning of the main routine directly in the code. All intervals, as to write the log files `$writinginterval`, to clean the arrays `$cleaninginterval` and to update the plots `$plottinginterval` are also initialized at the beginning of the `main.pl` file. Changes can be done directly in the file. All other parameters concerning the worm, the Internet model or the simulation itself can be chosen during the execution time of the simulation and are read from the `internetmodel.txt` file[22].
The simulation program is started with `perl main.pl` or with `./main.pl` if the perl path is set correctly.

4.12.2 Input Parameter During the Simulation

The first step is to choose a name for the simulation. Any name can be chosen, which contains only letters, numbers and the "_" sign.
The program creates a new folder with this simulation name into the output folder, which is created during the first program execution. The output folder can be found in the program folder. If there exist already a folder with the simulation name, the new files will overwrite the old ones in the simulation folder without prompting. Therefore the user should ensure to choose each time a different name, to avoid loosing his old simulation data.

[21] On the student tardis machine the gnuplot path is `/usr/sepp/bin/gnuplot`.
[22] See section 4.1.2 for further information about the format of the Internet model file.

Then the program asks to input first the simulation time and than the Internet model parameters followed by the worm parameters. After inputting all data the program writes a log file, called `overview.log` which contains all input data.
Than the program starts two instances of gnuplot to plot the infected hosts and the traffic in bps and executes the spreading simulation.
When the simulation finishes after the simulation time or if all vulnerable hosts are infected, the plots are replotted and stored as postscript files into the `plotfiles` folder.
Than the end time of the simulation and the total number of infected hosts is written and the program stops. The two plots remain on the screen.[23]

4.12.3 Output and Plots

During the program run a simulation folder is created with the name of the simulation. In the simulation folder two sub folder, called `logfiles` and `plotfiles`, and an overview log file (`overview.log`) are created.
The folder `logfiles` contains the infection log file and the traffic log file. The first column of each file is the time, the second the number of infected hosts respectively the traffic amount in bits per second at this time.
In the folder `plotfiles` all plot files are stored. This are the two load files `plot_infection.p` and `plot_traffic.p`, they are used to create the postscript file and are not used anymore after finishing the simulation. The two files `plot_infection.ps` and `plot_traffic.ps` are the postscript files which contain the plots from the simulation.

4.13 Simulator Performance

The Simulator is written with the scripting language Perl. Perl is slow compared to C. Therefore the performance question is an important issue and will be discussed in this subsection.
The duration of a simulation varies from seconds to days and depends on the input parameters and the Internet model. The next paragraph gives an overview of the parameters, which impact the duration of a simulation. The section 4.13.2 gives example of good parameter choices in the perspective of a fast, but exactly result.

4.13.1 Input Parameters

In general we can say, that the duration of the program run can be operated by the choice of the simulation time, the time step, the Internet model and the various intervals. The simulation time, the time step and the number of groups in the Internet model are the most important parameters and are therefore mainly responsible for the program run duration:

Simulation time: The simulation time, `$totalsimulationtime`, is the number of seconds, which should be simulated. Consequently, the duration is linear dependent on this time. Since a faster worm needs a shorter simulated time the program run time is much shorter, than for a slower worm. The simulation of the spreading of CodeRedIv2 within approximately 24 hours takes 24 times the simulation of Sapphire within approx one hour takes.

Time step: The time step, `$timestep`, gives the steps to execute a program run. One program pass corresponds to one time step. Thus the program run duration is linear dependent on the simulation time divided by a time step. These two time parameters are the most important ones to effect the total program run duration.

Number of groups in the Internet model: The number of groups in the Internet model, `$groupno`, has an important effect on the program run time. For each group a separate array is generated. Within each time step the arrays of each groups are gone trough twice and consequently, the program run duration dependence is linear with factor 2. A simpler Internet model results in a much faster simulation.

[23] To change, that the plots remain on the screen after finishing the simulation, the parameter `-persist` which is sent to gnuplot, when the gnuplot instances are initialized through the pipe has to be deleted. It can be found in the subroutine `startplotting` in line 20 and 29.

As mentioned above the various interval, which are writing interval, plotting interval and cleaning interval, have also an effect on the program run time. All interval are multiples of the constant `$timestep` and have therefore the unit milliseconds.

Writing interval: The writing interval, `$writinginterval`, says how often the subroutine `logfilewriting(...)` is called to write the infection and traffic log files. Calling this subroutine takes time and therefore this call should not be done too often. But this interval has a small effect on the program run duration, as long it is not called in every time step.

Plotting interval: The plotting interval, `$plottinginterval`, says how often the subroutine `updateplots(...)` is called to update the infection and traffic plots. It is a multiple of the writing interval, because it does not make sense to call it more often, than values are written into the log files. Calling this subroutine takes time, furthermore updating the gnuplot requires a lot CPU time. The perl program continues to simulate, while gnuplot executes the replot command parallel. Updating the plot in every time step, results in a long gnuplot queue of replot commands, which cannot be handled anymore within a short time. Such a short plotting interval consequently turns the simulation to a very slow one.

Cleaning interval: The cleaning interval, `$cleaninginterval`, says how often the subroutine `clean_sending(...)` and `clean_receiving(...)` are called to clean the receiving and sending arrays. Without executing this subroutine the sending and receiving array grow in every time step and also a short simulation time (e.g. one hour) with a big time step (e.g. 10 seconds) results in a program run duration of several days using 100% CPU and several hundred Mega Bytes of Memory. But the execution of this subroutines takes time, too.

4.13.2 Good Parameter Choice

This section discusses the critical parameter from the program run duration and the accuracy of the results perspective.

Simulation time: The simulation time can normally not be chosen because this time has to be as long as the whole worm propagation can be observed. The program stops simulating when all vulnerable hosts are infected. Herewith the simulation does not simulate for ever, if all hosts have already been infected and nothing new can be observed.

Time step: Choosing the time step shorter than the common greatest divisor of the latencies of all groups, results in inexactness. In each time step the receiving hosts will receive the scans from the sending hosts, which have sent them the latency time before. Executing the simulation with time steps, which are bigger than the latency time, produces an inexact result. But it has only a effect on the scans, which will successfully infect a new host. In UDP this means that the new infected hosts will scan for new hosts a time step too late. In TCP both, the sending and the new infected hosts, are blocked a time step to send out new scans.
The mentioned inexactness faces the performance. Simulating Code Red with a time step of 10 ms takes, depending on the other input parameter, several days. A good value for simulating a whole day is about 500 ms. For simulating only an hour 50 ms can be chosen.

Number of groups in the Internet model: The number of groups should be chosen as little as possible. 4 groups represent the Internet model well.
By using a time step of seconds the differences of the latency which is mostly in the area of milliseconds can not be observed and accordingly, using a lot of groups does not result in a higher accuracy.
To compare the effect of bigger and littler worm parameters, frequently the number of group does not have an impact.

Writing interval: The writing interval should be chosen so that at least 1'000 entries are written to the log files until the end of the simulation. Values which produce more entry are also possible and have a minor effect on the program run time.

Plotting interval: A useful value is in the area of 500 times the writing interval.

Cleaning interval: The cleaning interval should be small. A value around 10 times the time step is useful.

The Table 4 gives an overview of a good parameter set to observe the spreading of Sapphire/SQL Slammer and CodeRedIv2 by achieving a short program run time and a good approximated result.

Parameter	Units	Sapphire	CodeRedIv2
Simulation time	sec.	3600	86400
Time step	ms	50	500
Number of groups	-	4	4
Writing interval	time step	20	40
Plotting interval	writing interval	500	500
Cleaning interval	time step	10	10

Table 4: A good parameter choice to achieve a fast and precise simulation

5 Results

This section gives a discussion of all worm parameters, which have an impact on the infection or traffic behaviour and which also can be varied in the simulator. The first subsection discusses general parameters, as the Internet model, the second subsection shows the results of the discussion about worm parameters and the last subsection gives comparisons of worms observed in the past to simulations.

5.1 General Parameters

This subsection gives information about the different parameters, which have an impact on the infection speed and the traffic amount. The parameter were explored using simulation results of the ”Worm Spreading Simulator”.

5.1.1 Internet Model: Number of Groups

As already discussed in section 4.13, the number of groups in the Internet models have a major impact on the program run time. The run time for the simulation of a TCP worm with an Internet model of 10 groups, a time step of 1 second and a simulated time of 24 hours, is around 12 hours (running on a Pentium 4, 1.8 GHz machine). Whereas simulating the same worm with a Internet model of 4 groups results in a run time of a 5th part of the one before, which is around 2.5 hours.
The question if it is necessary to use a more complex Internet model, which causes longer run time and if the chosen Internet model represents the Internet good enough will be answered in this paragraph. UDP worms and TCP worms will be discussed separately.

The Internet model represents groups of hosts with a certain connectivity. In section 2 three divisions of the hosts into groups have be done. The different models are:

- **10 groups:** 10 groups as shown in Table 1
- **4 groups Napster:** 4 groups from the measurements of Napster users, as shown in Table 2
- **4 groups Gnutella:** 4 groups from the measurements of Gnutella users, as shown in Table 2

There is another model added to this group discussion, with the following connectivity:

- **1 group:** 1 group with a bandwidth of 3 Mbps and a latency of 60 ms

Hence the Internet models differ only in the number of groups and in their characteristics.

5.1.2 UDP-Worms

A first simulation is done with a UDP worm with each Internet model[24]. Figure 4 shows the simulation with the 1 group Internet model, Figure 5 the simulation with the 10 groups model, Figure 6 with the 4 groups Napster model and Figure 7 with the 4 groups Gnutella model.

The differences of the various models are enormous. Whereas the simulation with the 1 group Internet model (Figure 4) shows an infection of all vulnerable hosts within 800 seconds. The same worm has about 2000 seconds to infect all vulnerable hosts with the 4 groups Napster model (Figure 6). A UDP worm is bandwidth limited and therefore the worm spreading speed depends a lot on the bandwidth of the single hosts.

The 1 group Internet model (Figure 4) shows the fastest infection of all vulnerable hosts. This model is an oversimplified representation of the Internet. The 4th group with the 3 Mbps bandwidth from the 4 groups Internet models has been chosen arbitrarily to characterize the

[24]The further parameters used in these simulations are: 1 initially infected host in the group of the highest bandwidth, 100’000 vulnerable hosts, a worm size of 376 Bytes and 100’000 parallel scans using UDP.

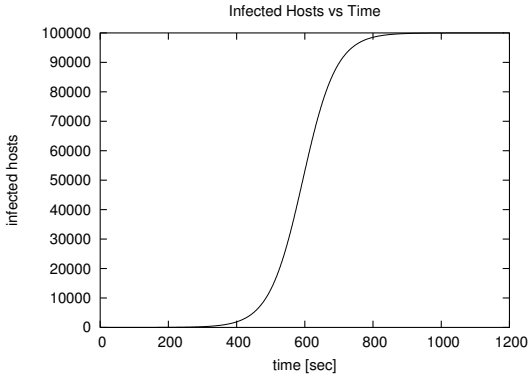

Figure 4: Number of infected hosts vs time simulated with the 1 group Internet model

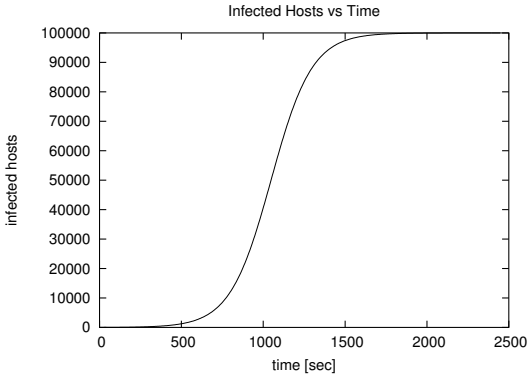

Figure 5: Number of infected hosts vs time simulated with the 10 groups Internet model

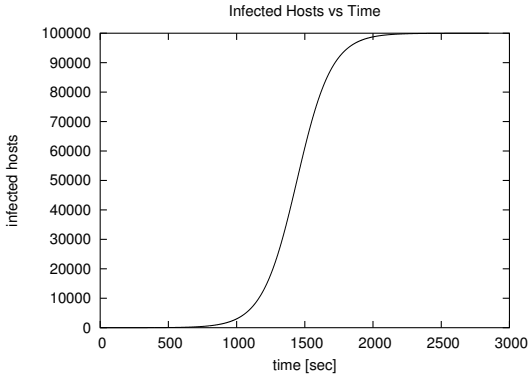

Figure 6: Number of infected hosts vs time simulated with the 4 groups Napster Internet model

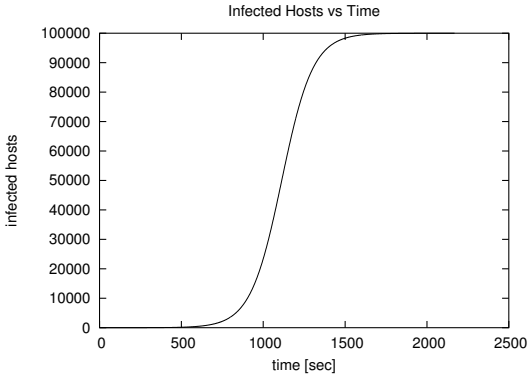

Figure 7: Number of infected hosts vs time simulated with the 4 groups Gnutella Internet model

1 group Internet model. It represents a faster Internet than measurements have shown. But also using the Internet models, which are based on measurements, gives completely varying results. The 10 groups model (Figure 5), which is based on the measurements of the Napster users, compared to the 4 groups Napster model (Figure 6) shows differences in the spreading speed of 33%.

The question comes up, which of these models represents the Internet best. The answer can only be given using measurements and comparing them to worms, observed in the past. In the subsection 5.3 the comparison to past worms will be done and also an answer will be given to the question of the best model.

5.1.3 TCP-Worms

The comparison of TCP-worms simulated with different Internet models do not show the differences which have been observed for UDP worms. Figure 8 shows the simulated infection behaviour of a TCP-worm with the 4 groups Napster Internet model and Figure 9 the spreading of the same worm with the 10 groups Internet model.

A very small difference of less than 5% can be observed at the infection speed of the two simulations. We can say, that the results have shown, that the Internet model has a small effect on the spreading speed. In TCP mainly the number of infected hosts are the determining factor, not the total bandwidth of the infected hosts.

In the following simulation mainly the 1 group Internet model is used, because compar-

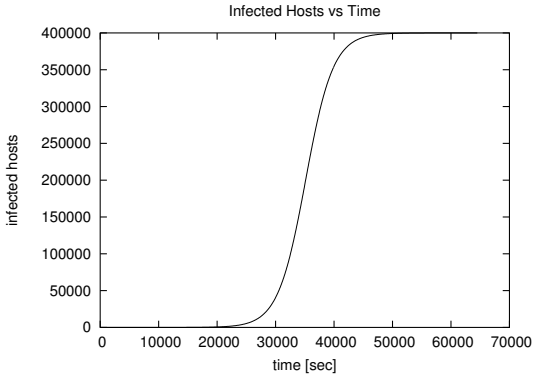

Figure 8: Number of infected hosts vs time simulated with the 4 groups Napster Internet model

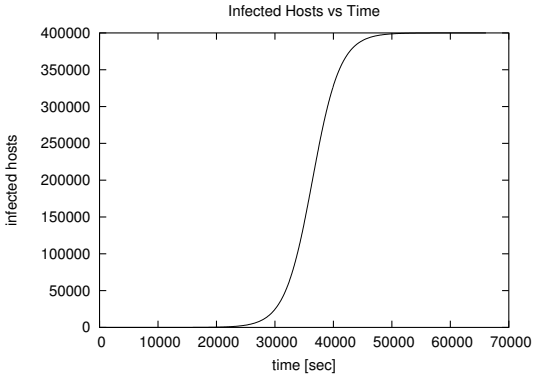

Figure 9: Number of infected hosts vs time simulated with the 10 groups Internet model

ing one parameter with various values, does not need a perfect modeling of the Internet and it delivers a model which is good enough for TCP worms. However, in some simulations the 4 groups Napster Internet model has been used as well.

5.1.4 Number of Vulnerable Hosts

A worm which scans for a rare security hole uses so much scans that it takes a very long time to find and infect the next vulnerable host. Whereas a worm, which uses a common security hole, infects within a short time all vulnerable hosts.

The Figure 10 and the Figure 11 show the differences that various numbers of vulnerable hosts cause. Figure 10 shows the infection with 400'000 vulnerable hosts, Figure 11 the infection with 1'000'000 vulnerable hosts.[25]

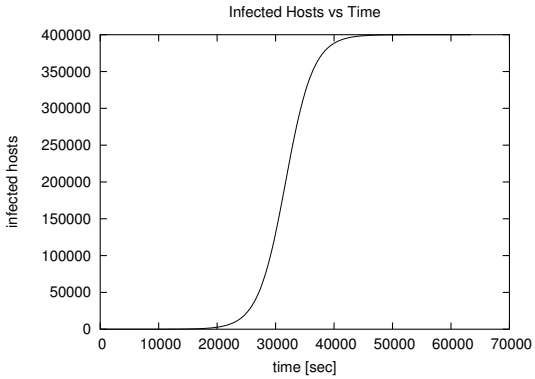

Figure 10: Number of infected hosts vs time simulated with 400'000 vulnerable hosts

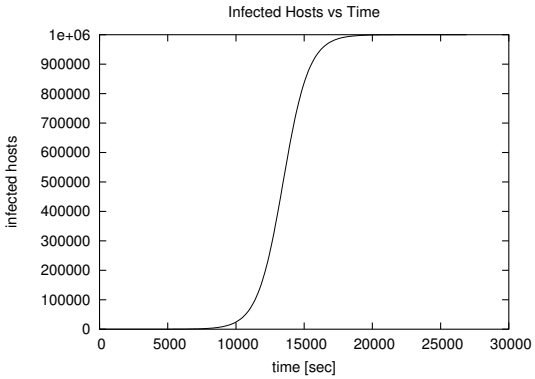

Figure 11: Number of infected hosts vs time simulated with 1'000'000 vulnerable hosts

The number of vulnerable hosts is an important value and greatly determines the speed of a worm. The resulting traffic from scanning and infecting a host depends on the number of scanning hosts. A bigger group of vulnerable hosts results therefore in more scans and herewith in a higher traffic per second. Whereas infecting a smaller group is much harder and therefore needs much more scans and the total traffic amount during the whole infection time is higher than with more vulnerable hosts. The exact difference depends on the worm size and

[25] The further parameters used in these simulations are: 1 Internet group with a 3 Mbps bandwidth and a 60 ms latency, 1 initially infected host, a worm size of 3569 Bytes, 100 parallel scans using TCP without resending and a TCP timer of 21 seconds.

the transport protocol[26].
This parameter explicit shows the importance to keep the duration of infecting the first 10%[27] of the vulnerable hosts small, in order to achieve a fast spreading behaviour. If this barrier (of 10%) is hard to reach because of the small amount of vulnerable hosts, the whole infection takes much longer.

5.1.5 Start Infection

The number of hosts, which are initially infected, has a determining effect onto the spreading speed and the total traffic amount. In the beginning phase of a worm spreading the number of sending hosts limit the spreading speed and therefore the start infection influences the first 10% infected hosts extensive.
Starting a worm from several hundred hosts, is a difficult issue and can be solved for example with a so called hitlist. The infection and traffic behaviours of hitlists are discussed in section 5.2.6.

5.2 Worm Parameter

This subsection discusses typical worm parameters and their effect on the infection and traffic behaviour. The parameters were explored using simulation results of the ”Worm Spreading Simulator”.

5.2.1 Transport Protocol: UDP vs. TCP

The critical difference between the two transport protocols UDP and TCP is the connection, which has to be established using TCP. Waiting for the answer of the receiving hosts costs a lot of time and prevents the scanning host to send out new scans. Even if the worm can treat several threads, the maximal number of threads is limited by the host. Frequently content switches cost a lot of resources and consequently, the number of threads is limited. The maximal number of possible threads is much smaller than parallel scans would be needed to send out as many scans the bandwidth allows within the waiting time. TCP is therefore always bandwidth limited as already discussed in section 3.4. UDP does not have this constraint of parallel scans and is, if the worm does not limit the number of possible scans per second, only limited by the bandwidth; see section 3.4 for more details.

Comparing the two protocols is therefore difficult and has to be done very carefully. TCP does send the worm only to vulnerable hosts[28] but needs two packages more than UDP to establish a connection. To simulate two worms which differ only in the protocol, for a UDP worm the number of scans per second would correspond to the number of parallel scans a TCP worm can produce. Due to the fact this comparison is very theoretical and a UDP-worm, restricting the number of scans per second, is not very senseful, no simulation results are shown here.

Other scanning strategies bring a new interesting topic into this discussion about UDP and TCP. Since a host, scanning with permutation scanning, needs to know if the scans have found a vulnerable host or not and further if the vulnerable host is already infected or not, a transport protocol has to be used, which gives an answer to a sent package. Comparing TCP-worms with complex scanning algorithms such as permutation scanning with simple UDP-worms using random scanning might give interesting results and should be a topic of further research.

[26] UDP sends the data with the scanning package and consequently scans and infects a new host with one package. What means, that the worm is sent to every host. In TCP the worm is sent after establishing a connection and therefore the worm is only sent to vulnerable hosts according to the assumption described in section 2.4.

[27] The infection plot shows that the gradient increases exponential starting at the beginning of the infection. After infecting a certain amount of vulnerable host the increasing speed slows down and after infecting the half population of vulnerable host, the gradient starts decreasing towards zero. The gradient reaches the maximal value after approximately 10%. We will use this value (10%) even the exact value stays somewhere between 5% and 15%.

[28] According to the assumption in section 2.4.

5.2.2 Worm Size

The worm size has a varying effect on the infection behaviour, which depends on the transport protocol. Using UDP the worm size is a major criteria to enable a fast spreading. Within TCP the latency and not the bandwidth is the limiting factor and therefore changing the worm size has a small effect on the spreading speed.
To visualize the effect the worm size has within UDP, the following simulations have been done. Figure 12 shows a small worm with a size of 376 Bytes, as the Sapphire / SQL Slammer worm has. On the right side in Figure 13 an infection plot can be seen from a worm with a size of 40'000 Bytes[29].

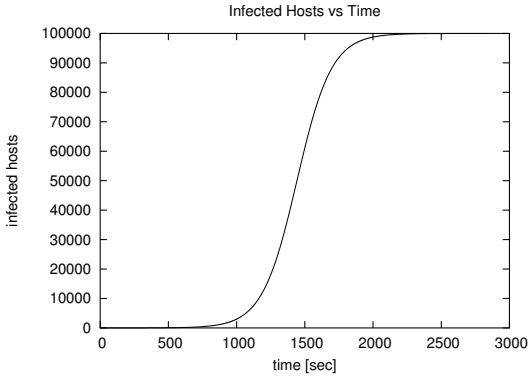

Figure 12: Number of infected hosts vs time simulated with a worm size of 376 Bytes

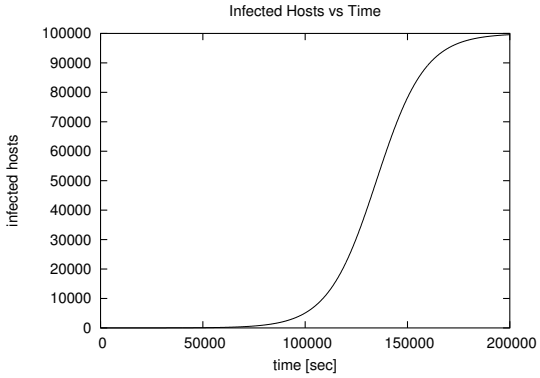

Figure 13: Number of infected hosts vs time simulated with a worm size of 40'000 Bytes

The difference of the two infection behaviours is enormous. A 376 Bytes worm infects 100'000 hosts within 10 minutes, whereas a worm with the completely same characteristics but with a size of 40'000 Bytes uses approximately three days to infect the same amount of hosts. This comparison explicitly shows what bandwidth limited means.
Worms with the same characteristics and a size of 376 Bytes and resp. 40'000 Bytes with TCP do not show this difference. Simulations, which have been produced, show exactly the same results for the two TCP worms, varying in the worm size.[30] Finally we can say, the smaller a worm is, the faster it spreads.

A surprising behaviour can be observed in the traffic which is caused by these worms. Off the top of one's head we would say, that a bigger worm also causes more traffic. A comparison of two traffic plot, one with a 376 Bytes worm and one with a 40'000 Bytes worm, both using UDP, does not show any difference in the traffic amount per second. The same traffic amount in bits per second can be observed in both simulations (spread over the varying time).
UDP sends as many scans as the sending host can produce. The limiting factor is the bandwidth. If a sending host has to send a bigger worm, than it takes longer to send one scan. The total amount of bits which are sent in a second does not change. The traffic caused by UDP worms is therefore determined by the bandwidth, with the simplification made in this simulator as discussed in the sections before.

[29] The further parameters used in these simulations are: 4 Internet groups, 1 initially infected host from the 3 Mbps group, 100'000 possible scans per second using UDP and 100'000 vulnerable hosts.

[30] The simulations for TCP have been done with an Internet model of only one group with a bandwidth of 3 Mbps. The following calculation shows that within a Internet model with more groups, which has also groups with bandwidths below 1 Mbps, the worm size has also an effect and consequently a bigger worm needs more time to be spread. Calculation: the number of parallel scans multiplied by the worm size divided by the timeout time gives an approximated value for the minimal bandwidth within TCP. With the numbers for these simulations:

$$100 * 40'000 * 8bits/21second = 1.5Mbps; \quad (1)$$

That means, hosts with a smaller bandwidth than 1.5 Mbps will be limited by the bandwidth and not by the number of parallel scans. But the effect of the bigger worm size is still much smaller than within UDP.

At first view the traffic of TCP worms from various sizes neither show a difference on the data amount. Due to the fact that many scans are sent without finding a host and therefore without sending the worm, the worm is only sent a few times. A few times means, as many times a host gets infected, which is in the simulation the number of vulnerable hosts. The additional data amount from a smaller worm to a bigger worm is therefore compared to the whole scanning traffic very small and therefore the additional data amount is so small, it cannot be observed in the plots, though it exists.

5.2.3 Scanning Method

All the simulations presented in this result section 5 show worms, which use random scanning, with a sole exception, which is this subsection about scanning methods. Therefore no simulation plots from worms using random scanning will be shown in this subsection. Hitlists are discussed separately in section 5.2.6.

The simulator can simulate worms with a special type of geographical scanning, which is called *local forced scanning*. This method sends scans to the same group with a certain rate, so called *in group scanning rate*, and the remaining scans to the other groups.
An interesting behaviour can be simulated, using this scanning strategy, if the rate of scans to the same group is chosen as 95%. Figure 14 shows the infection plot and Figure 15 the traffic plot. The simulation has been done with a four group Internet model,with one initially infected host from the group in the highest bandwidth, which is 3 Mbps[31].

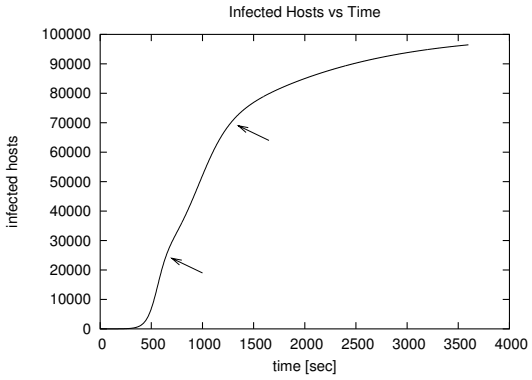

Figure 14: Simulated number of infected hosts vs time using local forced scanning with a in group scanning rate of 95%

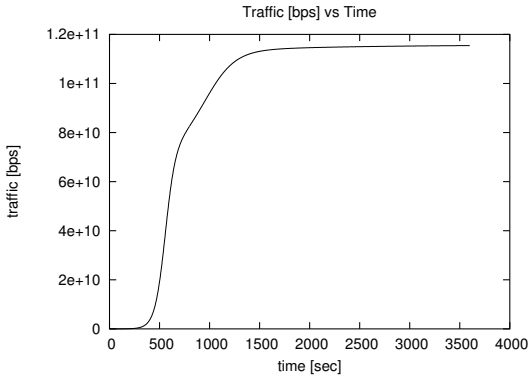

Figure 15: Simulated traffic in bps vs time using local forced scanning with a in group scanning rate of 95%

The start population has a big impact onto the spreading speed. The worm used in the simulation is bandwidth limited and therefore the choice of the first infected host does mainly determine the spreading speed. In this case a worm from the highest group has been chosen and a in group scanning rate of 95%. Other simulations, which are not explicit shown here, demonstrated that with a low scanning rate to the same group, the initial infection, does not have such strong implications.
In the graph the different groups can be recognized. 25% of the hosts are in the highest bandwidth field and exactly at 25'000 hosts a break can be seen. At this point nearly all hosts from the group with the highest bandwidth are infected. The next group, which has a bandwidth of 1 Mbps (38% of all hosts), is completely infected after approximately 1300 seconds. Here another break can be recognized.

Also the traffic plot shows this groupwise infection quite well. The breaks are even more conspicuous in this plot. The reason is that the traffic per second contains the scans from the

[31] The further parameters used in these simulations are: A worm size of 376 Bytes, 100'000 possible scans per second using UDP and 100'000 vulnerable hosts.

group in the highest bandwidth area as well - 95% of this scans produce only traffic but do not infect further hosts from this group, because all hosts are already infected. Due to the choice of the initial infection, the traffic amount rises so fast, as can be seen in Figure 15.
These simulation results are a good verification check for the simulator, further they show the different groups and the effect of such a theoretical scanning algorithm clearly. But the Internet is not topologically or geographically divided into these groups chosen in the Internet model. However enlarging the Internet model to a model which represents groups which contain hosts which stay geographically near to each other, gives the possibility to simulate, with the local forced scanning strategy, without changing the simulator, a simplified topological scanning.

5.2.4 TCP Timeout and Resending

The effect of the TCP timeout and if resending after the timeout is done, will be shown in the following example. Figure 16 shows the spreading off a TCP worm which does not resend the first package to establish a connection. If the first package cannot successfully start establishing a connection, the worm starts connecting to another host after the timeout. In this first simulation the timeout is set to 10 seconds.[32]
In Figure 17 the same TCP worm is simulated, but this worm does resend the first SYN package after the timeout, and the timeout is chosen to be 120 seconds[33], 12 times longer than in the first simulation.

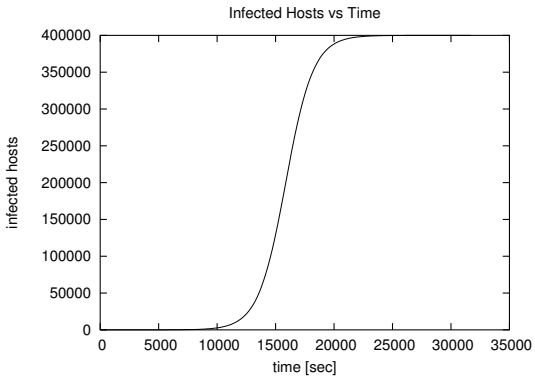

Figure 16: Number of infected hosts vs time, timeout=10 sec and no resending after the timeout

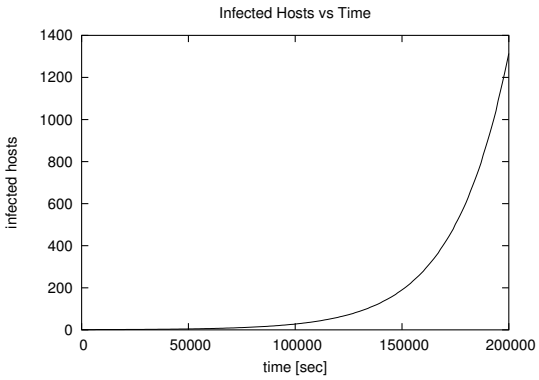

Figure 17: Number of infected hosts vs time, timeout=120 sec and resending after the timeout

These simulations show the shorter the timeout is, the faster the infection takes place. But this conclusion is only true until a certain value. The value depends on the network congestion and the latency and cannot be simulated with this simulator. A too short value, especially in a later spreading phase, causes to many give-ups. The package takes then longer than the timeout because it is buffered at some routers due to congestion, and than the sending host gives the waiting on this package up due to a too short timeout. A worm which uses a value, which is not as short that little network congestion causes a too fast give-up but which is as short as possible, enables an efficient spreading behaviour.

5.2.5 Additional Time to Infect a Host

The following simulations show the difference in the infection speed if a worm needs additional time to infect a host (to e.g. reboot a new infected machine) and if he does not. The exact implementation of this parameter is discussed in section 3.6.6.

[32] The further parameters used in these simulations are: a Internet model with one group with a bandwidth of 3 Mbps and a 60 ms latency, 1 initially infected host, a worm size of 3569 Bytes, 100 parallel scans using TCP and 400'000 vulnerable hosts.

[33] By default the timeout in TCP is 120 seconds.

The Figure 18 shows the simulation with a worm which directly can infect new hosts[34]. The spreading of the same worm, with an additional waiting time of 300 second, is shown in Figure 19.

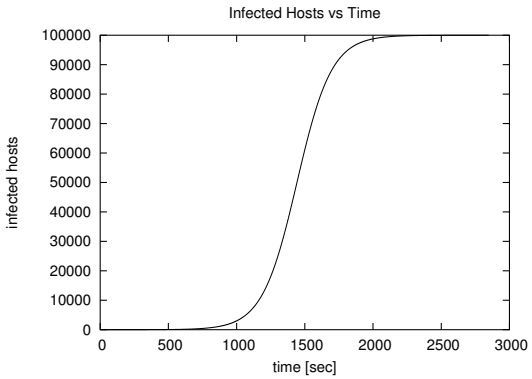

Figure 18: Number of infected hosts vs time without a wait time

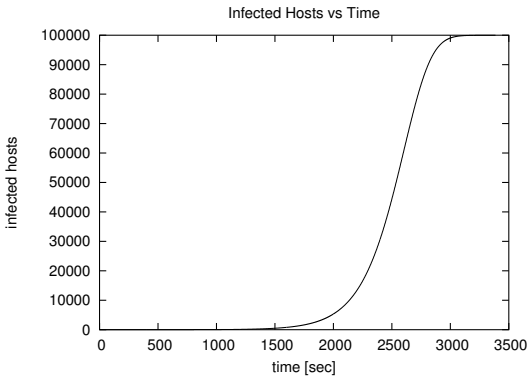

Figure 19: Number of infected hosts vs time with a wait time of 300 seconds

There can be drawn the simple conclusion, that a shorter wait time or even no wait time results in a faster infecting behaviour. Further it is obvious, that a long wait time has a strong effect in the first phase of the worm spreading. Comparing the infection after the first 10% does not show big differences between the two simulations.

5.2.6 Hitlist

Hitlist scanning is always a big advantage, achieving a faster spreading behaviour. To infect the first 10% of the vulnerable hosts take the most time and therefore if the scan to this host can be done within a short time, this results in a very fast infection.
Figure 20 shows the first 1000 second of a worm spreading without a hitlist and Figure 21 shows the same worm with a hitlist[35].[36] The simulated time is 1'000 seconds.

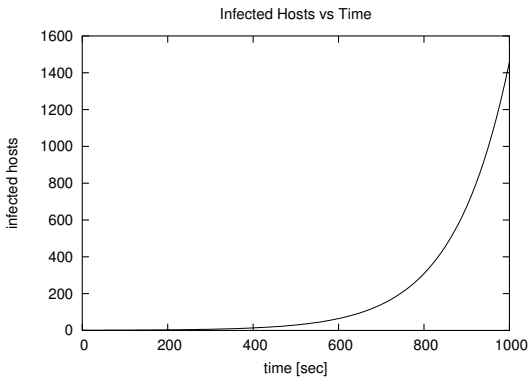

Figure 20: Number of infected hosts vs time simulated without a hitlist

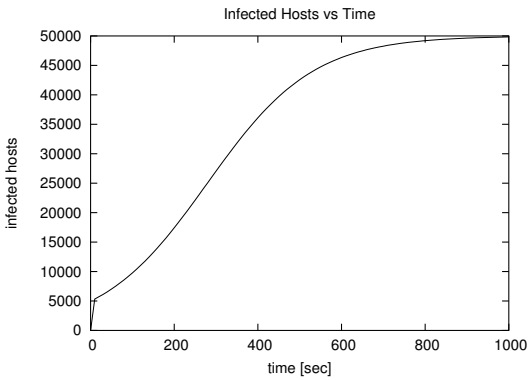

Figure 21: Number of infected hosts vs time simulated with a hitlist

[34]The further parameters used in these simulations are: a Internet model with one group with a bandwidth of 3 Mbps and a 60 ms latency, 1 initially infected host, a worm size of 376 Bytes, 100'000 scans per second using UDP and 100'000 vulnerable hosts.

[35]The simulated hitlist contains 20'000 IP-Addresses of hosts. Each of these hosts are vulnerable with a probability of 25%.

[36]The further parameters used in these simulations are: 1 Internet group with a 3 Mbps bandwidth and a 60 ms latency, 1 initially infected host, 50'000 vulnerable hosts, a worm size of 3569 Bytes, 100'000 scans per second using UDP.

The spreading with the hitlist shows an infection of all vulnerable hosts within 1'000 seconds. Whereas the same worm with a hitlist within the same time only infects 3% of the vulnerable hosts.
The simulator makes a simplification with hitlist scanning which has to be mentioned here. The simulator does not include the additional amount of Bytes, which are needed to store the hitlist data. The amount of additional data for this simulation (with a hitlist of 20'000 IP-addresses) is about 80 KB[37] without any compression plus the additional hitlist program code. A good worm algorithm divides the hitlist in every infection step and therefore the additional data amount can be kept small.[38] Nevertheless, the additional data amount is a fact which has a decelerating effect on the infection speed during the hitlist phase. Furthermore, creating a large hitlist bears the danger of being detected during scanning for vulnerabilities.

This effect guides the discussion to a new issue. To the issue, if a longer hitlist with a worse vulnerability probability or a shorter hitlist with a higher vulnerability probability is more effective concerning the spreading speed.
This question depends on three parameter, the additional data amount of the hitlist, the hitlist length and the probability the entries are vulnerable. Since the simulator has not implemented the additional data amount a hitlist causes, the answer to this question cannot be given completely. The complete answer would be a diagram with several lines. Each line represents a certain vulnerability probability and shows the infection speed (as infection in e.g. 100 seconds) versus the hitlist length.
Anyhow the conclusion can be given, that a longer hitlist always speeds up the infection rate as calculated in [2].

5.3 Comparisons to Past Worms

This subsection compares the spreading behaviour of past worms to simulations.

5.3.1 Code Red Iv2

On the 19th July, 2001 more than 359'000 were infected during the spreading of the Code Red Iv2 worm [7]. The first version of this worm emerged on the 13th July 2001, but due to an error in its random number generator, it did not propagate well. 6 days later the second version started spreading, called Code Red Iv2.
Code Red Iv2 generates on a new infected host 100 threads (100 parallel scans). The threads starts to chose a random IP address and to try to set up a connection to the target machine using TCP. The TCP timeout has been set to 21 seconds [9] and no resending has been done. The worm has a size off 3569 Bytes [10]. The issue how many hosts have been vulnerable during the spreading can not be solved completely. 359'000 hosts were infected during the spreading, but the worm stopt spreading at midnight and there were probably more vulnerable hosts. Because not an official number of vulnerable hosts exist, for the simulation the number of 400'000 vulnerable hosts is chosen.

Figure 22 shows the simulation of a worm with the above mentioned parameter using the 4 groups Napster Internet model. Figure 23 shows the measurements from Caida.org during the spreading of Code Red Iv2.
At the first view the two graphs show a similar infection, mainly in the first half of the infection time. A second view shows the differences from the measurements to the simulation.
The first difference is the spreading speed. The time to infect the half number of vulnerable hosts, 200'000 hosts, takes in the measurement 16 hours, whereas in the simulation this amount of hosts is infected after 10 hours. The reason for this difference can be found in the implementation of the random algorithm. The random algorithm of Code Red starts always at the same IP Address and therefore the random algorithm is not really random [9]. Each new infected host starts at the same IP-address and go through the same addresses. This gives a much slower spreading behaviour. The reason why the creator of the worm used this slower scan type is not completely clear but there might be two possible answers (from [9]):

[37] 20'000 * 32 bit = 640'000 bits. These are approx. 80 KB.
[38] For more information about hitlists see [2].

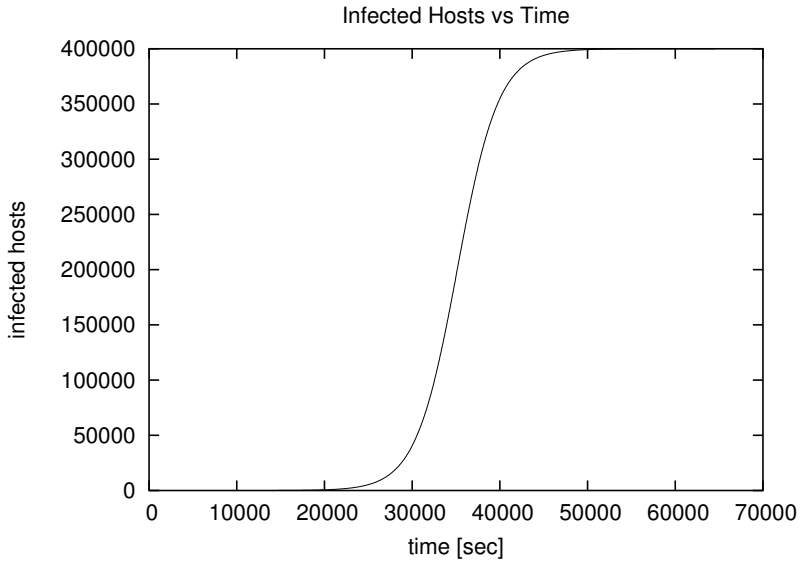

Figure 22: Simulated number of infected hosts vs time with Code Red Iv2 parameters

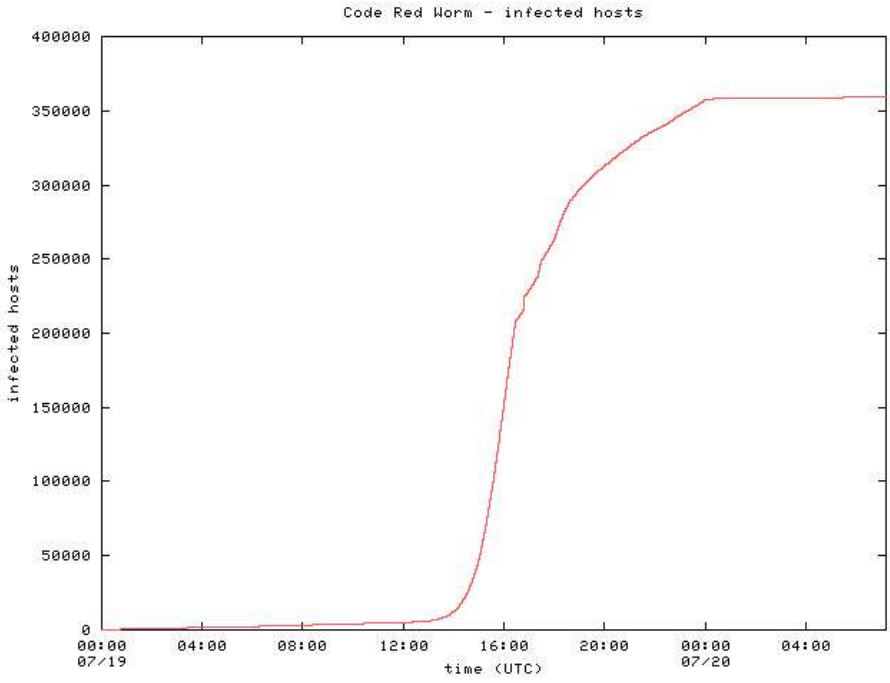

Figure 23: Number of infected hosts vs time measurements from Caida.org [6] during Code Red Iv2 spreading on 19th July 2001

1. The programmer of the worm might not have only targeted to deface web servers but also might have targeted to demolish any other service that can be offered by the victim hosts that are in the list of arranged IP addresses to be scanned. By launching a DoS stated as above, the services available by the host before the infection can be disabled. And this causes to disable the services of certain hosts which makes it DoS smaller and less networks but in more devastating effects.

2. The creators of the worm might have tried to track how many and which systems had been infected with the worm. If they are in the list of first hundred or thousand IP addresses to be scanned, they can set up a sniffer to listen connections from port 80 in order to track

which and how many systems had been infected by the worm.

The infection behaviour of the measurements does not show a such nice graph in the second phase of the infection as the simulation does. The measurements show several breaks and buckling and the infection speed slows faster down than the comparable simulation. There are several reason for this behaviour:

- Smaller networks suffer under congestion and cannot forward all messages to the their destinations. Packages will be lost and this results in a lower infection speed.
- People start manually switching off their computers or even disconnecting whole company networks.
- Routing instability increases the amount of BGP messages, which has again an effect on the congestion of smaller networks and backbone routers.
- Finally, the worm stopped propagating at midnight by design. [9]

The above discussed differences show that in reality there are much more factors, which have an influence on the spreading behaviour of a worm, than in the simulation are included. Nevertheless, the simulations give a good approximation of the measured spreading behaviour.

5.3.2 Sapphire/SQL Slammer

On the 25th January, 2003 the fastest computer worm in history infected within 10 minutes 90% of all vulnerable hosts. The worm, called Sapphire/SQL Slammer worm spread, exploiting a buffer overflow vulnerability in computers on the Internet running Microsoft's SQL Server or MSDE 2000 (Microsoft SQL Server Desktop Engine) [12].In the first minute, the infected population doubled in size approximately every 8.5 seconds and the worm achieved its full scanning rate of 55 million scans per second after approximately three minutes.
Sapphire/SQL Slammer uses the transport protocol UDP and has a size of 376 Bytes. Further the number of scans are limited by the bandwidth and the scans a host can produce. The scanning strategy of Sapphire is random scanning. The information about how many hosts were vulnerable differ a lot in the various papers about Sapphire spreading, for the simulation, we use 100'000 vulnerable hosts. [12] shows that at least 75'000 were infected and therefore we use an amount which is closed to this number.
Figure 24 and Figure 25 show the infection vs. time plots of simulations with the parameters of Sapphire as explained above. Figure 24 uses the 4 groups Napster Internet model and Figure 25 the 10 groups Internet model. (Figure 7 shows the simulation with the same parameters for the 4 groups Gnutella Internet model and Figure 4 for the 1 group Internet model.)

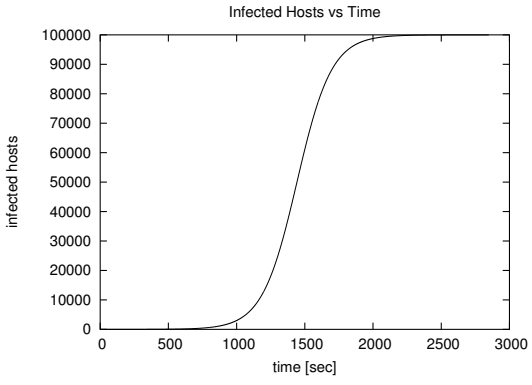

Figure 24: Number of infected hosts vs time simulated with Sapphire/SQL Slammer parameters (4 groups Napster Internet model)

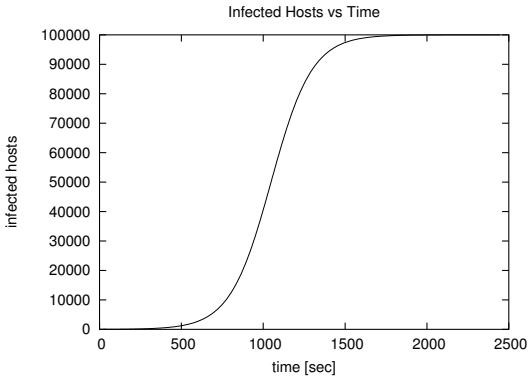

Figure 25: Number of infected hosts vs time simulated with Sapphire/SQL Slammer parameters (10 groups Internet model)

As already mentioned in section 5.1.1, the number of groups and the percentage of the different

group have a major impact on the spreading speed of UDP worms. The differences simulating Sapphire are enormous. But unfortunately none of this Internet model does reach the same propagation speed as Sapphire used. The simulations show a spreading which is from 100% to 200% slower, than the reality has shown. Only the simulation with the Internet model, containing one group of 3 Mbps (see Figure 4), shows the same spreading speed.
But having another look at the infection behaviour in the 10 groups Internet model (Figure 25) shows a much higher convergence to the real propagation speed than with the 4 group Napster Internet model (Figure 24). Though the two Internet model contain the same group of hosts, namely Napster user, with the single difference that in the 10 groups Internet model the hosts are split into smaller groups, the 10 groups model gives a much better approximation.
Observing what exactly happened during the spreading of the Sapphire/SQL Slammer worm, shows that a few hosts attached to gigabit links had an important impact on the infection speed. A single Pentium 4 2 GHz machine, attached to a gigabit link, can generate several hundred megabits per second, sending out over 100'000 UDP packets per second. Alone such a machine would infect the entire Internet (4 billion addresses) in 12 hours. A hundred such machines would infect the entire Internet in less than 10 minutes [13]. In the spreading of Sapphire/SQL Slammer more than 100 such hosts, attached to gigabit links, were involved and the slower machine helped spread the worm too. Therefore this Slammer worm was such efficient.

In the 10 group Internet model the highest bandwidth is 44.736 Mbps, which is much lower than the hosts, which were mentioned before. Still this Internet model delivers the best Internet representation as it can be seen in comparison with the spreading of Sapphire.

As already discussed for TCP worms above, also the simulations of UDP worms make a lot of simplifications and do not involve many parameters as additional traffic caused by ARP messages or others. Principally the conclusions done for TCP worms can be adopted for UDP worms.

6 Conclusion and Outlook

This section contains the conclusion and gives an outlook on further research topics.

6.1 Conclusion

As shown in the results section, the simulation tool allows to explore various worm parameters and make meaningful statements about good and bad chosen parameters, from the view of achieving a fast infection behaviour.

The Internet model, described in section 2, is mainly for latency limited worms a good representation of the real Internet as shown in section 5.1.1. For bandwidth limited worms the Internet model delivers only a rough approximation as the comparison to measurements during the spreading of Sapphire have shown in section 5.3.

It is difficult to make a quantitative conclusion of the traffic behaviour in the simulations. The simulator includes only the sent scanning and infecting bits and does not concern about other protocols, which had a major effect during the spreading of past worms. To draw conclusions on the worm characteristics from the traffic behaviour it is necessary to get more information about the protocols and the resulting traffic of the worm spreading and to implement them in the simulator. To gain this goal, the simulator provides a good bases and enables extensions and improvements with a small effort.

6.2 Outlook

The outlook section provides an outlook by describing what are topics of further research concerning worm characteristics.

6.2.1 Further Protocols

During the spreading of past worms a lot of additional network traffic has been noticed. Mainly an increase of ARP, ICMP and BGP messages had an effect on the spreading behaviours of past worms. What the effect of these protocols on the infection speed are and how they influence the whole network traffic and what the consequences on smaller networks are, are open research topics. Also the implementation of the effect from these protocols into the simulation tool could be done in a further work.

6.2.2 Worms in IPv6

The enormous address space of IPv6 does not offer the possibility of using random scanning strategy. Many discussed strategies can not be executed within IPv6. Worms have therefore to look different. What strategies worms could choose and what else will have an impact on the appearance of worms in IPv6 is an interesting topic of further research.

6.2.3 Internet Model

The Internet model which has been chosen during this thesis, gives a good approximation for latency limited worms. But for bandwidth limited ones it does not fit so well. An Internet model with more groups could enable better simulation results for bandwidth limited worms.
Furthermore the Internet model could be divided into geographical groups which represent hosts from the same area of the continents with similar connectivity.

6.2.4 Simplifications in the Worm Model

A lot of simplifications have been done during the implementation of this version of the simulator, some of them do not effect the spreading behaviour of worms, others could have a bigger impact on the infection speed and should therefore be improved in a next version of the simulator. The

following discussion shows a list of the parameters, which could be better implemented in the simulator.

Scanning strategy: A bad implementation or even wrong implementation of a random generator is not implemented in the simulator and could be done in future work.
An important issue in hitlist scanning is the additional data amount the implementation of a hitlist causes. This data amount is not implemented in this first version but should be done in a further work to make meaningful statements about hitlist scanning.
The implementation of further scanning strategies, as geographical scanning, permutation scanning and other scanning strategies, is also a potential topic of further works.
Furthermore the comparison of worm spreading behaviour using new scanning strategies is certainly an interesting issue.

Transport protocol TCP makes a lot of simplifications, some of them do not infect the infection behaviour, others (e.g. that the receiving hosts do not send SYN or ACK packets) should be improved.

Packet size The packet size depends on the route a packages chooses through the Internet. Some network split a package in smaller packages and therefore add additional headers. What the exact effect of splitting up packages and how often it happen, could also be researched in a further work.

Time to infect a host Waiting time during the infection of a host, can increase in various situations. All these times are summarized and added to the latency in the simulation tool. Within TCP this slows down the spreading behaviour, which does not fit the reality. The possible modification of this could be implemented in a next work.

7 Summary

Worms can be multifaceted and vary among other parameters in size and scanning strategy. The strongest implications on the infection behaviour can be recognized at worms using different transport protocol. Worms which use TCP as transport protocol are limited through the number of parallel scans. Whereas UDP worms do not have to wait for the answer of the receiving hosts and therefore are only limited by the bandwidth. Consequently a UDP worm can generate several thousands to hundred thousands of scans per second and is therefore much faster than a TCP worm.

The implemented simulator gives the possibility to simulate the spreading of worms and to explore worm parameters, achieving to make meaningful statements about good or bad chosen worm parameters.

The simulations have shown that the chosen Internet model mainly for latency limited worm represents a good approximation of the Internet and for bandwidth limited worm makes a rough approximation.

Furthermore the simulations have shown, that a worm, achieving a fast infection speed, should exploit a frequent vulnerability, be small, use UDP and probably even implement a hitlist scanning in the first phase of the worm spreading.

A Acknowledgements

Various helpful comments and ideas were received from Thomas Dübendorfer and Arno Wagner.

Furthermore, mainly lively discussions with colleagues created input to the work.

B Bibliography / References

References

[1] Rpcky K.C. Chang,
Defending against Flooding-Based Distributed Denial-of-Service Attacks: A Tutorial;
IEEE Communications Magazine, October 2002.

[2] Stuart Staniford, Vern Paxson, Nicholas Weaver,
How to 0wn the Internet in Your Spare Time;
http://www.icir.org/vern/papers/cdc-usenix-sec02/, In *Proc. USENIX Security Symposium*, 2002.

[3] Nicholas Weaver,
Potential Strategies for High Speed Active Worms: A Worst Case Analysis;
http://www.cs.berkeley.edu/~nweaver/worms.pdf, U.C. Berkeley BRASS group, March 2002.

[4] *Sapphire Worm Code Disassembled*;
http://www.eeye.com/html/Research/Flash/sapphire.txt, eEye Digital Security, January 2003.

[5] *CAIDA Analysis of Code-Red*;
http://www.caida.org/analysis/security/cod-red/, Caida.org, April 2003

[6] David Moore, Colleen Shannon,
The Spread of the Code-Red Worm (CRv2);
http://www.caida.org/analysis/security/code-red/coderedv2_analysis.xml, CAIDA.org, June 2002.

[7] Cliff Changchun Zou, Weibo Gong, Don Towsley,
Code Red Worm Propagation Modeling and Analysis;
http://tennis.ecs.umass.edu/~czou/research/codered.pdf, University Massachusetts, Amherst MA, November 2002.

[8] David Moore, Colleen Shannon, Jeffery Brown,
Code-Red: a case study on the spread and victims of an Internet worm;
http://www.caida.org/outreach/papers/2002/codered/codered.pdf, CAIDA.org, San Diego Supercomputer Center, University of California, November 2002.

[9] Silent Blade
Info and Analzsis of the 'Code Red' Worm;
http://www.infosecwriters.com/texts.php?op=display&id=38, September 2002.

[10] Szmantec, Eric Chien
Security Response: Code Red;
http://www.symantec.com/avcenter/venc/data/codered.worm.html, July 2001.

[11] J. Cowie, A. Ogielski, BJ Premore and Y. Yuan,
Global Routing Instabilities during CodeRed II and Nimda Worm Propagation;
http://www.renesys.com/projects/bgp_instability/, Renesys Corporation, September 2001.

[12] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, N. Weaver,
The Spread of the Sapphire/Slammer Worm;
http://www.cs.berkeley.edu/~nweaver/sapphire/, University of California, Berkeley, 14.04.2003.

[13] Robert Graham,
FAQ: Firewall Forensics (What am I seeing?), chapter 11.3;
http://www.robertgraham.com/pubs/firewall-seen.htm, Robert Graham, Version 1.2.0, January, 2003.

[14] Stefan Saroiu, P. Krishna Gummadi, Steven D. Gribble,
A Measurement Study of Peer-to-Peer File Sharing Systems;
http://www.cs.washington.edu/homes/gribble/papers/mmcn.pdf, Dept. of Computer Science and Engineering, Univ. of Washington, Seattle, WA, January 2002.

[15] *Maps of Internet Servic Provider and Internet Backbone Networks*;
http://www.geog.ucl.ac.uk/casa/martin/atlas/isp_maps.html and
http://www.geog.ucl.ac.uk/casa/martin/atlaas/more_isp_maps.html, April 2003.

[16] *RFC 791: Internet Protocol*;
http://rfc.net/rfc791.html, Septermber 1981.

[17] *RFC 3513: Internet Protocol Version 6 (IPv6) Addressing Architecture*;
http://rfc.net/rfc3513.html, April 2003.

[18] Steven Wallace,
Worms, DDoS presentation;
http://www.cs.indiana.edu/classes/b538/worms.ppt, Indiana University, April 2003.

C Presentation Slides