MODELING AND SIMULATIONS OF WORMS AND MITIGATION TECHNIQUES

A Dissertation
Presented to
The Academic Faculty

By

Mohamed Abdelhafez

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
in
Electrical and Computer Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology
December 2007

MODELING AND SIMULATIONS OF WORMS AND MITIGATION TECHNIQUES

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Internet worm attacks have become increasingly more frequent and have had a major impact on the economy, making the detection and prevention of these attacks a top security concern. Several counter–measures have been proposed and evaluated in recent literature. However, the effect of these proposed defensive mechanisms on legitimate competing traffic has not been analyzed.

The first contribution of this thesis is a comparative analysis of the effectiveness of several of these proposed mechanisms, including a measure of their effect on normal web browsing activities. In addition, we introduce a new defensive approach that can easily be implemented on existing hosts, and which significantly reduces the rate of spread of worms using TCP connections to perform the infiltration. Our approach has no measurable effect on legitimate traffic.

The second contribution is presenting a variant of the flash worm that we term Compact Flash or CFlash that is capable of spreading even faster than its predecessor. We perform a comparative study between the flash worm and the CFlash worm using a full-detail packet-level simulator, and the results show the increase in propagation rate of the new worm given the same set of parameters.

The third contribution is the study of the behavior of TCP based worms in MANETs. We develop an analytical model for the worm spread of TCP worms in the MANETs environment that accounts for payload–size, bandwidth–sharing, radio range, nodal density and several other parameters specific for MANET topologies. We also present numerical solutions for the model and verify the results using packet–level simulations. The results show that the analytical model developed here matches the results of the packet–level simulation in most cases.

CHAPTER 1
INTRODUCTION

A computer worm is a piece of software that replicates itself on the network. A worm uses vulnerabilities in popular applications to attack victim hosts, which would become infected and start to infect others. The first known worm was the Morris worm in 1988 [1]. Since then, the security threats and damaging effects of modern worms have increased dramatically. The Code Red [2] and Nimda [3] worms infected hundreds of thousands computers around the world. In 2003 the SQL Slammer worm [4] infected more than 90% of the vulnerable hosts (75,000) in less than 10 minutes. It has become apparent that no human intervention can react in a timely enough manner to these types of attacks, and therefore automatic detection and prevention mechanisms are a necessity.

This research provides detailed simulation analysis of some worm outbreaks and mitigation techniques. The simulation models developed here offer researchers background to develop any new worm detection algorithm. These models are implemented in *Georgia Tech Network Simulator* (*GTNetS*). *GTNetS* features detailed and scalable network models that simplifies the development of large-scale complex simulation experiments needed for accurate studies of worm outbreaks.

The rest of this chapter is organized as follows. An overview of computer worms is discussed in section 1.1. Section 1.2 gives an overview of mitigation strategies. Section 1.3 motivates the need for analyzing worms and mitigation techniques. Thesis contributions are introduced in section 1.4. Finally, section 1.5 provides the outline of the thesis.

1.1 Computer Worm Overview

A computer worm is a malicious code that exploits software bugs in popular applications to infect vulnerable hosts. When a host is infected, it starts searching for other vulnerable hosts to infect. This spread pattern results in exponential growth of the infected population and this spread alone can cause major network failures because of the increasing traffic volume with the growth of the infected population.

There are three stages in the worm life-cycle:

1. Propagation: The worm is transfered to a certain host by exploiting some vulnerability.

2. Activation: The worm starts to execute a set of commands to gain higher access to the compromised system.

3. Infection: The worm starts looking for other hosts to infect, and replicates itself on those hosts.

There are many factors used to classify worms which include target selection (algorithm for finding vulnerable machines), worm carrier mechanisms, possible payloads, worm activation, and types of attackers using the worm [5]. As an example worms can be classified according to how they find their targets into:

1. Topological worms: These worms find information about new targets from data stored on an infected host. Many applications contain information about other hosts, and therefore give the worm a good source for finding other potential victims. The Morris worm was a topological worm.

2. Passive worms: These worms do not actively seek potential victims. Instead they rely on user actions to spread elsewhere. There have been been many passive worms like, Gnuman [6] and the CRClean [7]. Gnuman operates by acting as a gnutella node which replies to all queries with copies of itself. If this copy is run the worm starts on the victim machine and repeats the process. CRClean was intended to remove the Code Red II worm from the machine. The CRClean worm waits for a Code Red II probe. When it detects an infection attempt it launches a counterattack removing Code Red II and installs itself on the machine. These worms spread without any scanning.

3. Scanning worms: These worms search for new targets by probing IP addresses across the Internet. Scanning can be sequential where the worm works through an address block using an ordered set of addresses, or random where the worm selects addresses in a random fashion.

Our focus in this study is the scanning category of worms, such as Code Red, Nimda, and SQL-Slammer.

Many researchers studied worm behavior by collecting traces of worm spread and producing several statistics about infected hosts and infection rates. One such study is the work by Moore et al. [2], where they traced the Code-Red worm by collecting data in the form of a packet header trace of hosts sending unsolicited TCP SYN packets into their \8 network.

Researchers also developed different models to represent the worm spread. These models can be divided into three types: analytical, simulation based, and hybrid models.

Analytical models rely on equations that represent the dynamics of worms. The random constant spread (RCS) model [8] is an example of an analytical model. This model is also called the classic susceptible-infected (SI) model in [9] and is used to describe the worm spread through homogeneous random contacts between susceptible and infected hosts. The susceptible-infected-removed (SIR) and susceptible-infected-susceptible (SIS) models [10, 11] add the repair and the removal of infected hosts into the SI model. Zou et al. introduced an analytical model called the "two factor" worm model that includes the effect of human countermeasures [12].

Network simulations were also used to study the worm problem. Sharif [13] built worm models into GTNetS [14] and was able to simulate networks having hundreds of thousands of hosts and measure the effect of different network parameters on the worm spread rate. Wagner [15] discusses an efficient simulator for worm propagation implemented in the Perl scripting language, where he used models for large groups of nodes for the Internet rather than single hosts as well as simplified models of UDP and TCP behavior to reduce

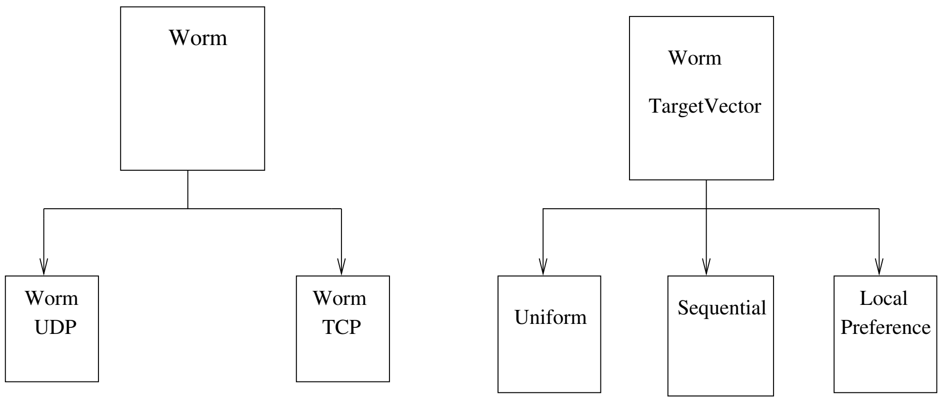


Figure 1. Class hierarchy for worms in GTNetS

complexity. This enabled large simulations but without packet-level detail.

Hybrid models incorporate both analytical models and packet-level simulations. Liljenstam et al. [16] used the SSFNet [17] simulator with packet-level details for a small section of the network and represented the rest of the Internet with an analytic model.

We are using the worm models in GTNetS, which model the behavior of scanning worms [13]. Figure 1 shows the class hierarchy for the worms. The Worm class has the common features of all scanning worms like: payload size, signature, target selection, scan range, and infection port. According to the transport protocol, the worm is represented by TCP or UDP class. The scanning algorithm in the worm can be set through the WormTargetVector member variable, which can be either sequential, uniform, or local preference. In sequential scanning the worm selects a random start address which is uniformly distributed in the scan range and then generates sequential addresses after it. In uniform scanning the worm chooses random addresses uniformly distributed in the scan range. In local preference scanning the worm generates addresses that can be in the whole scan range or in the local range based on a defined probability.

1.2 Mitigation Techniques Overview

In this section we discuss some of the most widely used worm detection and mitigation strategies. Detection strategies can be generally divided into two main categories: signature based and behavior based. The signature based approach looks for a common string or byte pattern in the payload and identifies packets that match that pattern as anomalous. It is clear that this strategy is only as good as the signature. The signature has to be specific enough so that it will not be present in normal traffic and also general enough to catch different forms of the same worm. Signature based detection therefore requires considerable analysis of the payload to be able to come out with the ideal signature. There has been a lot of work done in automating signature generation such as Autograph [18] and Vigilante [19]. This form of detection is very effective against known attacks. However, a polymorphic worm (a worm that can change its payload by compression, encryption or other methods) can render this detection method useless. Moreover, Chung and Mok [20] showed that the automatic signature generating (ASG) system can be used to attack the protected network by what they call “allergy” attack. The authors used crafted packets that would cause the ASG system to generate signatures that would match normal traffic and therefore cause a denial of service attack when the ASG starts dropping packets that match these signatures.

Behavior based detection on the other hand does not require extensive analysis of the payload, but it does require the identification of the common behavior between attacks or at least the range of acceptable behavior patterns that normal hosts do not deviate from. The rest of this section will look into some of the most widely used detection strategies of this category. The behavior based detection algorithms would be organized according to the type of network statistics they depend on to signal their detection.

1.2.1 Outbound Traffic Analysis

These detection strategies look at the outbound traffic from the host or the network and decide whether it is anomalous or not by comparing the traffic to the “normal profile”. The normal or anomalous behavior is decided based on different network traffic metrics, some of which are discussed below.

1.2.1.1 Connection Rate

This metric relies on the fact that normal hosts communicate with a small number of servers, and therefore would have a relatively small number of connection requests per unit time. On the other hand, an anomalous host would try to infect as many hosts as possible and therefore would have a much higher connection rate.

In [21] the authors present the Network Security Monitor (NSM) which was developed at the University of California, Davis. NSM works by maintining a four dimension matrix of which the axes are: Source, Destination, Service, and Connection ID, where each cell in that matrix holds the number of packets sent for that connection as well as the total size of data transfered. In order to detect an attack this matrix can be compared to a matrix containing known attack patterns, or some detection rules can be applied to it. One simple rule is that a host communicating with 15 or more hosts during a five minutes interval is flagged anomalous.

The idea of rate limiting was proposed by Williamson in [22], where each host is allowed to have a working set of n hosts to communicate with during t seconds. Any new connection attempt is delayed by placing it at the end of a delay queue. As the queue fills up more delay is introduced to new connection attempts. The size of the delay queue can also be used to detect worm attacks. In other words, if the queue size is more than a certain threshold then this host is infected. One problem with this setup is that this algorithm has to be implemented on all end hosts to get good results [23] and that proves to be expensive. However, the same idea can be implemented on a router or on a special hardware as proposed by Staniford [24].

The host-based connection rate detection is effective against greedy worms, which try to infect as many hosts as possible in the shortest time. However, a stealthy worm can easily avoid them by reducing its scan rate to be below the threshold. To address this problem, the algorithm can be applied on the LAN level and the network level. So even if the worm is slow to be detected on each individual host, after it reaches a certain population size, its compound connection rate would prove to be significantly more than that of normal traffic.

1.2.1.2 DNS Anomalies

David Whyte et al. [25] present a worm detection strategy based on DNS anomalies. They observed that users tend to remember alphanumeric strings and use the network service provided (i.e. DNS) for new connections, whereas a scanning worm would directly use numeric IP addresses to connect to. The authors also note that there are some protocols that would directly use the numeric IP addresses for connections and they propose to put those on a whitelist so as not to trigger false positives.

The basic methodology is to divide the network into segments called cells. Each cell contains a worm detection device that monitors the DNS queries and the connection requests from that cell. Any connection attempt to an IP address that was not a result of a DNS query or not in the whitelist would generate an alarm. In the discussion of the false positives resulting from that algorithm the authors suggest using two anomalous connections from the same host to generate an alarm. One drawback of that approach is that it does not detect scanning behavior within the same cell.

1.2.1.3 ARP Anomalies

In [26] the authors address the shortcoming of the previous approach in detecting intra-cell scans by monitoring the ARP requests from the hosts within each cell. They observe that anomalous hosts would exhibit an increase in their ARP request activities.

During a training period an ARP chain of each host is built. This chain represents the other hosts in the cell that this host communicates with. An anomaly score is calculated for each host based on three metrics:

1. Number of ARP requests to hosts outside the ARP chain.
2. Total number of ARP requests.
3. Number of ARP requests to unused IP addresses within the cell.

Each of those metrics is calculated for a sample interval t (60 seconds), weighted and together they present an anomaly score for a certain detection window w. The anomaly score is compared to a certain threshold to decide wether a certain host is infected.

The authors note that this approach can cause a high number of false positives to occur if all the hosts have the same threshold of detection, as they note that some servers might exhibit bursts in their ARP requests at certain intervals which would trigger an alarm. As a solution to this problem they propose function-specific thresholds for servers and end hosts. Also because only ARP requests are monitored and not replies, the system has a serious limitation for cells that have dynamic IP assigning. In order to address that limitation the ARP replies would have to be monitored as well to record the MAC addresses of the different hosts in the cell.

1.2.1.4 Failed Connection Rate

Because a greedy worm wants to infect as many hosts as it can and it does not know ahead of time which hosts are vulnerable, it would send an infection attempt to any target. This target might not be offering the vulnerable service that the worm is attacking, or such target may be turned off or simply non existing. In such situations the connection would fail. A failed connection can be detected by the receipt of TCP RST message, or ICMP unreachable message, or simply non receipt of a SYN ACK message (in case of TCP worms).

Several approaches have considered that metric as essential for worm detection [24], [27], and [28]. In [28] the authors present the idea of Credit based connection rate limiting (CBCRL) that works by giving each host a certain number of credits and for each connection attempt that credit is reduced by one, if the connection succeeds then the credit is increased by two. If a host reaches zero credit then it is blocked. They also introduce some techniques to prevent a host from being starved or from getting too large a number of credits by successive inflation.

1.2.2 Inbound Traffic Analysis

The analysis of inbound traffic received attention in the literature, some of the most promising are listed in this section

1.2.2.1 Honeynets

Honeynets or honeypots form an isolated network of vulnerable machines that are meant to be attacked. These hosts do not offer any real services, therefore any traffic towards them is considered anomalous. These machines would have monitoring processes running on them or on their network. The machines are running old versions of most common applications so as to attract attackers to them. By monitoring the attacks on them, the attack trends and the vulnerable ports can be identified. Also by analyzing the attack traffic, dynamic signatures could be generated to help protect the real network. It is important that the honeynet be isolated from the rest of the network so that it will not be a source of threat when it is infected.

1.2.2.2 Gateway Sensors

While the honeynet can monitor only a small portion of addresses, a monitor placed on the gateway or at the egress router can monitor the traffic going to the whole address space. Some interesting statistics can be gathered from that traffic and common attack signatures would be identified. Using a collection of these monitors that report alerts to a common database like DShield, one can detect common sets of source IPs executing scanning and probing activities across a wide area of victims indicating worm propagations are underway.

1.2.3 Ratio Based Detection

In [29] the authors present an entropy based approach for worm detection. They chose entropy because it is a measure of how random a dataset is, and scanning activity is more uniform than normal traffic in some respects and more random in others. Entropy contents of a finite sequence of values can be measured by representing the sequence in binary form and then using data compression on that sequence. The size of the compressed object corresponds to the entropy contents of the sequence. The authors store the source/destination addresses and source/destination ports for observed traffic in four separate data streams and then perform compression on each of them. They then calculate an inverse compression ratio defined as :

$$Inverse\ compression\ ratio = \frac{size\ compressed}{size\ uncompressed}$$

This ratio is monitored over time and any significant change in it would generate an alarm. This approach is geared towards fast scanning worms and as the authors note might not be effective for detecting slow scanning worms.

Another approach presented in [28] used sequential hypothesis testing where H_1 represents an infected host and H_0 represents a normal host. The idea is to monitor the connection attempts from each user and find the number of successful and failed connection attempts (a failed connection is detected by having no reply for the connection attempt after a certain timeout). After having these numbers one can calculate the likelihood of a host being infected using the following formula

$$L(Y_n) \equiv \prod_{i=1}^{n} \frac{Pr[Y_i|H_1]}{Pr[Y_i|H_0]}$$

where Y_i is the result of connection i (1 for success and 0 for failure), so as the ratio of failed connections to successful ones increase that likelihood function would increase and if it passes a certain threshold then that host is declared infected. In the other case if the successful attempts to failed attempts ratio increases then the likelihood function would decrease and if it gets lower than another threshold then that host is declared normal.

1.2.4 Fixed Set Approach

Sellke et al. [30] introduce the idea of a fixed set approach for worm containment and detection. They base their argument on modeling the early stages of worm spread using a branching process. Using that model they observe that the total number of scans that any infected host attempts is what determines wether the worm can spread regardless of the scan rate. The authors define a maximum number of allowed scans for each host to be M during a *containment cycle* (in the order of weeks). If the number of scans from a certain host exceeds M before the end of the containment cycle, that host is removed from the network and is subjected to a heavy duty checking process. The rest of the hosts are thoroughly checked for infection at the end of the containment cycle.

The authors showed that by using $M = 1000$ they were able to contain the CodeRed worm with 10 initially infected hosts to less than 360 infected hosts which is less than 0.1% of the total vulnerable population. To consider the effect on normal traffic the authors looked at a 30 day trace of wide-area TCP connections originating from 1645 hosts in the Lawrence Berkeley laboratory to analyze the growth of the number of unique IP addresses contacted per host. They showed the maximum number of unique IP addresses contacted by a single host was 4000. So setting M to be 5000 in a one-month containment cycle would not interfere with normal traffic. This approach assumes that a worm is doing uniform random scanning of the address space, and that all the hosts are willing to contribute to this worm containment approach.

1.3 Motivation

The worm spread problem is complex and very hard to study. This is generaly because of the large number of hosts that contribute to the spread of the worm and the worm interactions with many dynamic network parameters. Reliable evaluation tools are needed for studying the worm spread and containment strategies. Current evaluation tools are of three types; theoretical analysis, laboratory testbeds, and simulation models.

Theoretical analysis relies on equations that represent the dynamics of the worms. These models have the benefit of computational efficiency, meaning that they can be easily scaled to predict the behavior of networks of millions of hosts and therefore provide an efficient way to study the problem. The main drawback of analytical models is that they often do not take into account many important worm characteristics, such as payload size, transport protocol used, and the different probabilities of infection resulting from different topologies and network conditions.

Labortatory testbeds are usually of a small scale because of the high cost of resources. This small scale makes them incapable of addressing such a problem of worm spread that affects hundereds of thousands of hosts.

Simulation has become the method of choice for many networking security research problems. As new protocols are designed and tested, computer based simulations are used to validate the correctness of the new protocol, and are used to measure the performance of the new protocol under a variety of experimental conditions. Current network simulators such as ns-2 [31], OPNet [32], SSFNet [17], and GTNetS are common platforms for network security research. However, the development of simulation models to represent worms and mitigation techniques that are accurate and can scale well, is a difficult problem to solve.

This research provides researchers with enough simulation models that represent most kinds of worm and mitigation techniques. These models can be easily extended to include more specific aspects of worms or mitigation algorithms. A full packet-level network simulator is used to represent most kinds of networks and which also collects the required measurements about worm spread and background traffic. Simulating and analyzing worms/mitigation techniques is a central part of this dissertation.

1.4 Thesis Contributions

In this Section, we summarize the contributions of this dissertation.

1.4.1 Evaluation of Contemporary Worm Defense Strategies

Several counter–measures for worm attacks have been proposed and evaluated in recent literature. However, the effect of these proposed defensive mechanisms on legitimate competing traffic has not been analyzed. Clearly, a defensive approach that slows down or stops worm propagation at the expense of completely restricting any legitimate traffic is of little value.

In this research we develop simulation models capable of representing most worm–mitigation strategies. We use these models to perform a comparative analysis of the effectiveness of several of the proposed mechanisms, including a measure of their effect on normal web browsing activities. In addition, we introduce a new defensive approach that can easily be implemented on existing hosts. This approach significantly reduces the rate of spread of worms using TCP connections to perform the infiltration. Our approach has no measurable effect on legitimate traffic.

1.4.2 Design of Worst Case Scenarios

The study of worst-case scenarios has received increased interest from researchers. Of the traits that make worms dangerous, the speed of spread is the most alarming. The speed of the spread of Internet worms increased dramatically in recent years. Moreover, it is almost certain that it will continue to increase in the near future with the increase in available bandwidth and network resources. The fastest known worm (only in literature) is the flash worm, which can infect over a million hosts in less than one second. This thesis presents a variant of the flash worm that we term Compact Flash or CFlash that is capable of spreading even faster than its predecessor. We perform a comparative study between the flash worm and the CFlash worm using a full packet-level simulator, and the results show the increase in propagation rate of the new worm given the same set of parameters.

1.4.3 Modeling and Simulations of MANET Worms

Mobile Ad-hoc Networks (MANETs) are used for emergency situations like disaster–relief, military applications, and emergency medical situations. These applications make MANETs attractive targets for cyber–attacks and make the development of counter–measures paramount.

The study of worm behavior is critical to the design of effective counter measures in MANET environments. This research studies the behavior of TCP based worms in MANETs. We develop analytical models for the spread of TCP worms in the MANET environment that account for payload–size, bandwidth–sharing, radio range, nodal density, packet discards and several other parameters specific to MANETs. We present numerical solutions for the models and verify the results using high fidelity packet–level simulations.

The results show that the analytical model developed here matches the results of the packet–level simulation in all cases except when topologies result in a high probability of disconnected clusters. Our simulation studies show that under many cases, due to the resource constrained nature of the MANET and its underlying wireless layers, the TCP-based worms rapidly become self-throttling. This may benefit the design of effective mitigation technologies in these critical networking environments.

1.5 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 describes simulation models for worm spread and mitigation techniques and how they are used to evaluate a number of mitigation techniques. Chapter 3 presents a worst case scenario for worm outbreak. The development of an analytical model for TCP worm in the MANET environment is discussed in chapter 4. Finally, Chapter 5 outlines the conclusions drawn from the research throughout this doctoral thesis.

CHAPTER 2
EVALUATION OF CONTEMPORARY WORM DEFENSE STRATEGIES

This chapter presents simulation models capable of representing most worm-mitigation strategies. In addition to a comparative analysis of the effectiveness of several of these mitigation mechanisms, we introduce a new defense mechanism that can be easily implemented on existing hosts, and which significantly reduces the rate of spread of worms.

The motivation behind the need for worm defense simulations is laid out in Section 2.1. Section 2.2 describes the simulation models used in our experiments. Section 2.3 gives an overview of worm detection algorithms and presents the five chosen algorithms in detail. Section 2.4 talks about our experiments and discusses the results. The extension of our studies to large-scale simulations is discussed in section 2.5 Finaly, section 2.6 gives the conclusion of this chapter

2.1 Motivation

A number of methods have been proposed to detect, react to, or prevent worm attacks. Since almost all worms exploit some software coding error or design flaw to infect the hosts, the most effective method would be to eliminate these software errors. However, it is fairly clear that some large fraction of existing or new Internet hosts will always have some exploitable vulnerability, since not all users or system administrators are willing or able to install security patches as they become available. Further, new software releases almost always introduce a new set of exploits.

Another approach is through the use of so-called Intrusion Detection Systems (*IDS*) [33], which are either signature-based or anomaly-based. In signature-based systems, a firewall checks incoming packets against a database of known worm signatures and drops the packet if a match is found. In anomaly-based systems the normal behavior of hosts is monitored, and if a significant deviation in the hosts activities is detected then some defensive action is taken. The signature-based approach is only effective against known worms, and has no effect against a new worm until that worm is analyzed, and its signature extracted and added to the database. Since this is a time-consuming activity, it is clear that such approaches have little hope in containing new worms. Thus, the anomaly detection method is the method of choice when faced with the detection and prevention of unknown worm attacks.

In this chapter, we study different algorithms for worm containment,and evaluate their effectiveness. There are several requirements for a successful worm containment algorithm.

1. Quick response: The algorithm should be able to quickly detect worm activity and stop it before it infects other hosts.

2. Low false negatives: The algorithm should be sensitive enough not to miss any attack.

3. Low false detections: The algorithm should not designate a healthy host infected (false positive), since such detections typically trigger some sort of filtering or reduction in capacity of that host.

4. Simplicity: The algorithm must be simple to implement and deploy, and not take excessive resources on routers or end-systems.

2.2 Simulation Models

We are using the worm models in GTNetS, which model the behavior of scanning worms [13]. Figure 2 shows the additional classes that we have introduced into GTNetS to model worms and mitigation algorithms.

We have added the models for the flash and cflash worms. These types of worms do not scan while propagating. The flash worm collects a list of all vulnerable IP addresses before infection starts and then infects a few nodes and distributes the list of vulnerable IPs among them to continue the infection process. The full details of the flash and cflash worms will be explained in chapter 3.

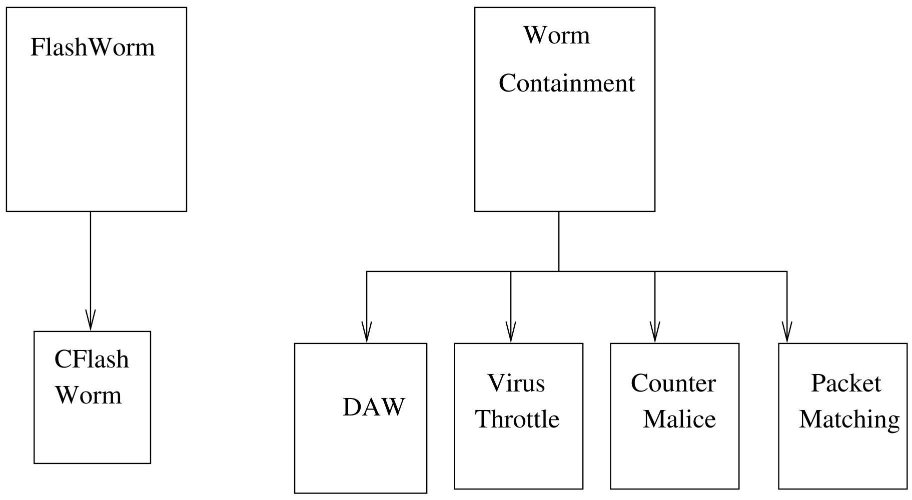


Figure 2. Class hierarchy for worms and mitigation algorithms

The mitigation algorithms all inherit from the parent class WormContainment. The WormContainment class only defines two methods that intercepts incoming and outgoing packets. The implementation of these methods is left for the child classes. The different mitigation algorithms will be discussed in detail in section 2.3.

For the network topology model we use a ring of Random Tree topologies as discussed in [13]. Figure 3 shows an example of a random tree object. Each random tree is characterized by depth and fan-out as in typical tree topologies. However, the random tree provides an additional random probability factor that decides whether a child node will be created or not. This leads to more realistic topologies with holes in the assigned IP address space. In our models we also have variation in the bandwidth of links in the topology.

The web traffic in our simulator is modeled after the empirical model developed by Mah [34], who studied web traffic through a campus network and generated distributions for a number of parameters representing this kind of traffic. The model has several parameters to characterize traffic, such as:

- Request length: HTTP request size.
- Reply length: HTTP response size.
- Consecutive pages: Number of consecutive documents retrieved from any given

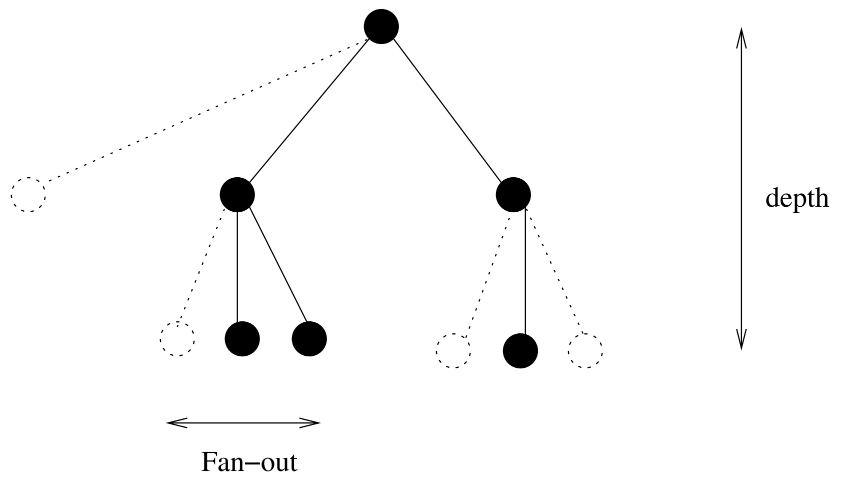


Figure 3. Random tree topology with depth 3 and fan-out 3

server.

- Think time: Time between retrieval of any two successive documents.

Each of the previous parameters has a statistical distribution that is sampled to provide actual values in the *GTNetS* simulations.

2.3 Worm Detection Algorithms

In this section, we give a short overview of worm detection and prevention algorithms, and then discuss in detail the five different proposed algorithms that are compared here. Moore et al. [9] have studied the effectiveness of worm containment systems and divided them into 2 types: Address blacklisting and content filtering. The address blacklisting approach detects the misbehavior of certain network addresses and blocks any connection attempts from them. The content filtering approach identifies common features of worm network connections and then filters all connections that share these features. Of the methods we study in detail: the rate limiting approaches, which can be implemented at the host-level as the virus throttle (section 2.3.1) and network-level as counter malice (section 2.3.2) are examples of the first type; Packet Matching (section 2.3.3), DAW (section 2.3.4) and TCP-ACK (section 2.3.5) are examples of the second type. Each of the proposed algorithms is discussed in detail below.

2.3.1 Virus Throttle

The virus throttle approach, proposed by Williamson [22] relies on the fact that worm scanning involves communicating with a large number of hosts simultaneously (or nearly so) in order to find a vulnerable host to infect. This behavior is assumed to be atypical of normal application activity, which tends to communicate with a limited number of hosts. The goal of the algorithm is to delay connection attempts that appear to be more than what the host normally makes in a certain period of time. The more aggressive the infection action is, the more delay its connection requests would experience.

2.3.1.1 Implementation Details

The virus throttle approach has the following parameters:

- *WorkingSet*: The set of the IP Addresses of the machines that this host has connected with recently. This list has a limited size; our implementation uses 5. Each entry in the working set has a time flag.
- *DelayQueue*: A queue used to store packets that are to be delayed by the algorithm.

The virus throttle approach inspects all outgoing packets from a host, searching for TCP *SYN* packets. When a SYN packet is detected, the following algorithm is run.

```
• If this host is in blocked state
    – Drop the packet.
• else
    – Compare destination address with addresses in the working set
    – If destination address is in the working set
        * Allow the connection immediately
    – Else if working set is not full
        * Add destination address to the working set.
        * Allow the connection to proceed immediately.
    – else
        * Add the packet to the delay queue.
        * If delay queue size is more than 100
            · Set the state of this host to blocked state.
```

The following method is called once every second.

```
Process-Queue()
• If working set is full
    – Remove oldest member.
• If delay queue is not empty
    – Pop the SYN packet from its head and any other packets addressed
      to the same destination.
    – Send the packet(s).
    – Add the destination address of the packet to the working set.
```

2.3.2 CounterMalice

The *CounterMalice* [24] approach was developed by Silicon Defense and is conceptually similar to the virus throttle approach, except that it is intended to operate on a network device, such as a router, rather than on an end-host. Counter malice works by monitoring the packets sent by a given host and building a composite score of misbehavior based both on the number of unique destinations and the number of those destinations that have not responded.

2.3.2.1 Implementation Details

We based our implementation of the counter malice approach on the information published in [24]. The published work lacks complete details on the workings of the algorithm, but does provide sufficient information to make an approximation of the approach. In our implementation, we have an entry for each host in the subnetwork containing all the parameters mentioned in the virus throttle approach. When a host within a subnetwork sends a SYN packet to a host outside the subnetwork the following method is called.

```
Output-packet-received()

• Does the source address represent a new entry ?
    – Create new entry
• Is the connection blocked ?
    – Drop packet.
• else
    – If destination address is in the working set
        * Allow the connection immediately.
    – Else if working set is not full
        * Add destination address to the working set.
        * Allow the connection immediately.
    – else
        * Add the packet to the delay queue.
        * If delay queue size is more than 100.
            · Set the state of that host to blocked.
```

The following method is called once every second.

```
Process-Queues()

• Loop through the list of host entries
    – If working set is full
```

```
      * Remove oldest member.
  - If delay queue is not empty
      * Pop the SYN packet from its head and any other packets addressed
        to the same destination.
      * Send the packet(s).
      * Add the destination address of the packet to the working set.
```

2.3.3 Packet Matching

The Packet Matching algorithm was proposed by Xuan Chen and John Heidemann [35]. This algorithm relies on the fact that a worm usually exploits some particular security vulnerability corresponding to a specific port number. Further, the nature of worms is such that an infected host will probe other vulnerable hosts with the same vulnerability. Therefore, routers seeing unusually high levels of bi-directional probing traffic with the same destination port number can infer a new worm attack is underway.

2.3.3.1 Implementation Details

The algorithm operates on 2 steps; port matching and address checking. In the port matching step the algorithm compares the list of destination ports observed for inbound traffic to the list of destination ports observed for outbound traffic. If a match is found, then the port is flagged as suspicious. In the address checking step, suspicious ports are monitored to detect how many unique IP addresses are being contacted, and an exponentially weighted moving average is computed for the number of unique destination IP addresses seen. When the instantaneous number of unique destinations is much larger than the moving average, the port is flagged as infected.

The authors also suggest using collaboration between routers to disseminate suspicious and infected port information. We did not model this extension in the algorithm in our simulations. Table 1 shows the parameters for the packet matching algorithm.

When a local host within a subnetwork sends connection request packets to a remote

Table 1. Parameter definition for packet matching algorithm

Parameter	Description
Out port list	Ports on remote hosts that local hosts sends packets to
In port list	Ports on the local hosts that are the destination port for received packets
Suspicious port list	Ports that are suspicious or infected
β	Average number of unique IPs contacted for a given port
N	Instantaneous number of unique IPs contacted for a given port
δ	Sensitivity parameter set to 3 in our implementation
α	Weight for the moving average set to 0.125 in our implementation

host outside the subnetwork, the following method is called.

```
Out-Syn-packet()
• If the destination port is infected ?
    – Drop packet.
• Else if the destination port is suspicious ?
    – Add the destination IP to the list of Unique IPs associated with
      this port
• Else
    – Add the destination port to the outgoing port list.
    – Forward the packet.
    – If the destination port is in the incoming port list
        * Add the port to the suspicious ports list
```

When a remote host outside the local subnetwork sends connection request packets to a local host, the following method is called:

```
In-Syn-packet()
• If the destination port is infected ?
    – Drop packet.
```

```
• Else if the destination port is suspicious ?
    – Add the destination IP to the list of Unique IPs associated with
      this port
• Else
    – Add the destination port to the incoming port list.
    – Forward the packet.
    – If the destination port is in the outgoing port list
        * Add the port to the suspicious ports list
```

The following method is called periodically.

```
Check-Infection()

• Loop through the list of suspicious ports
• If N $>\beta\times\delta$
    – Mark this port as infected
• Else
    – Update the moving average $\beta = \alpha \times \beta + (1 - \alpha) \times N$
    – N = 0
```

2.3.4 DAW

The Distributed Anti-Worm architecture (DAW) [36], has been proposed as a distributed solution with ISPs deploying the algorithm on edge routers. This algorithm relies on the fact that the failure rate for a random scanning worm is much higher than that of a normal well-behaved host.

A connection fails if the destination host does not exist (an ICMP Host Unreachable or Network Unreachable packet is sent) or if the destination host does exist, but has no layer 4

Table 2. Parameter definition for the DAW algorithm

Parameter	Description
size	Size of the token bucket
tokens	Number of tokens, initialized to the size
c	Failure counter
f	failure rate
β	Weight for the moving average for the failure rate; set to 0.2 in our implementation
t	Timestamp
λ	Failure rate threshold

protocol accepting connections on the destination port (an ICMP Port Unreachable packet is sent). In addition, a TCP reset packet will be sent if the destination host and port are valid but the receiving application detects malformed data and closes the connection. In the DAW algorithm, the failure rate is measured as the number of ICMP host, network, or port unreachable messages and TCP resets per unit time. Clearly this algorithm assumes that there is no filtering of ICMP or TCP reset packets by a firewall or gateway between the source and destination. The algorithm has 2 components, the DAW agent that is deployed on the edge routers and a management station that collects data from multiple agents. In our simulations, we only consider the actions of individual agents and do not take collaboration between agents into account. The basic principle is that if the connection failure rate of a host exceeds a pre-configured threshold, the DAW agent will begin dropping some of the connection requests from that host in order to keep its failure rate under the threshold.

2.3.4.1 Implementation Details

Table 2 shows the parameters used in the DAW algorithm. The following method is called every time an indication of a failed connection is received.

```
Update-Failure-Rate-Record()

• tokens = tokens - 1

• c = c + 1

• If ( c is a multiple of 10 )
```

```
    – f' = 10 / (the current system clock - t)
    – If (c == 10)
        * f = f'
    – Else
        * $f = \beta \times f + (1 - \beta) \times f'$
• t = the current system clock
```

Upon observation of a connection request from a host in the local subnetwork, the following method is called.

```
Basic-Rate-Limit()

• $\delta$ = the current system clock - time
• $tokens$ = min(tokens + $\delta \times \lambda$, size)
• time = the current system clock
• If (tokens $\geq$ 1)
    – Forward the request
• Else
    – Drop the request
```

2.3.5 TCP–ACK

In all the previously mentioned algorithms, we notice that the deployment cost is significant. Another problem is the amount of state the detection devices need to keep, which grows exponentially with the number of hosts in the network. This motivates the design of a stateless, low-cost and 0day ready worm containment algorithm. We introduce a new method called TCP–ACK. The approach is simple, easy to deploy on a large scale, and takes practically no resources. Our approach requires modifications to the protocol stack for existing hosts connected to the Internet (i.e. Windows, Linux, MAC-OSX, etc.), which can easily be accomplished using the security update mechanisms already in place for existing operating systems. With our modified protocol stack, any host receiving a TCP SYN packet for a non-existent port will unconditionally send a SYN–ACK to the originator, indicating that the connection has been accepted. The originator will then begin sending to the same destination port data packets which are silently dropped.

To see the rationale behind our TCP–ACK approach, consider the actions by a normal TCP worm without our approach in place. A TCP worm creates multiple threads (up to some fixed limit) that attempt connection requests to random hosts. Without TCP–ACK, a host that has no corresponding layer 4 protocol at the specified port will create an *ICMP port unreachable* message, and the connecting thread receives an indication that the connection has failed. Thus, in only one round–trip–time the worm has determined that the target host and port is not vulnerable, and is free to try another one.

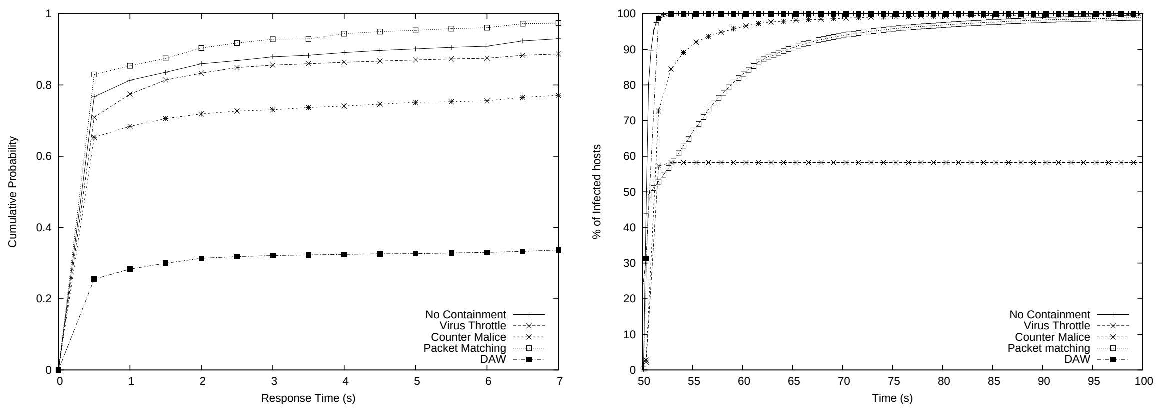
With TCP–ACK, the connection request to hosts that are not vulnerable (ie. those with no protocol bound to the destination port) will act as if they are. At that point, the worm will begin sending the payload packets which are silently dropped. From the worms perspective, this appears as normal lost packets with the corresponding timeouts and retransmissions. Instead of one round–trip–time per failed connection, the worm is tied up for several re–transmission timeout periods which is substantially longer, potentially several minutes. The net result is a decrease in the effective probing rate of the worm and a resulting decrease in the rate of spread. If a large fraction of hosts on the Internet implement the TCP–ACK mechanism, it will be nearly impossible for a TCP–style worm to effectively probe for vulnerabilities.

We point out that LaBrea is an approach that is conceptually similar to ours. The *LaBrea* [37] method uses un-allocated IP address space to create the same trap. In this method, if the worm sends a SYN packet to an un-allocated address the LaBrea program would reply with a SYN ACK with a window size of zero trapping that thread. However, this approach requires substantial infrastructure enhancements at subnetworks in order to forward the un–mapped IP addresses to some host to create and send the *SYN–ACK*. Furthermore, worms can easily detect the window size of zero and simply ignore any *SYN–ACK* with this signature. Our results also show that for any similar approach to work, the number of addresses with the trap installed must be more than the number of the vulnerable hosts. This means that in the case of *LaBrea* the ratio of unallocated addresses to real hosts in a subnetwork has to be greater than one, which can not be used on a wide scale. In contrast, our approach can be easily implemented on a wide scale simply by including it as part of an operating system update.

2.4 Experimental Results

In this section, we describe the simulation experiments we used to measure the effectiveness of each of the previously discussed detection algorithms and defenses. In addition to measuring their effect on the overall worm spread rate, we also monitored the effect on normal web browsing activity.

For the network topology model we created a topology consisting of about 9000 nodes using 11 Random Tree topologies connected with a ring. There are a mix of parameters for the random trees, as follows. We have 2 trees with fanout 8 and depth 5 allocating 4096 addresses each; 4 trees with fanout 4 and depth 5 allocating 256 addresses each; 4 trees with fanout 8 and depth 4 allocating 512 addresses each; and one tree with fanout 16 and depth 4 allocating 4096 addresses, for a total of 15,360 possible IP addresses. The tree is populated with child probability such that we have only about 60% (on average) of the possible 15,360 leaf nodes, or about an average of 9,000 leaf nodes in each simulation. Since the random scanning of IP addresses is an essential feature of all worms, and since the probability of a "correct guess" is a fundamental parameter in determining the worm's spreading rate, we have reduced the entire Internet IP address space down to the 15,360



(a) cdf of response times (b) Worm spread

Figure 4. Effect of algorithms on the network for a UDP worm attack

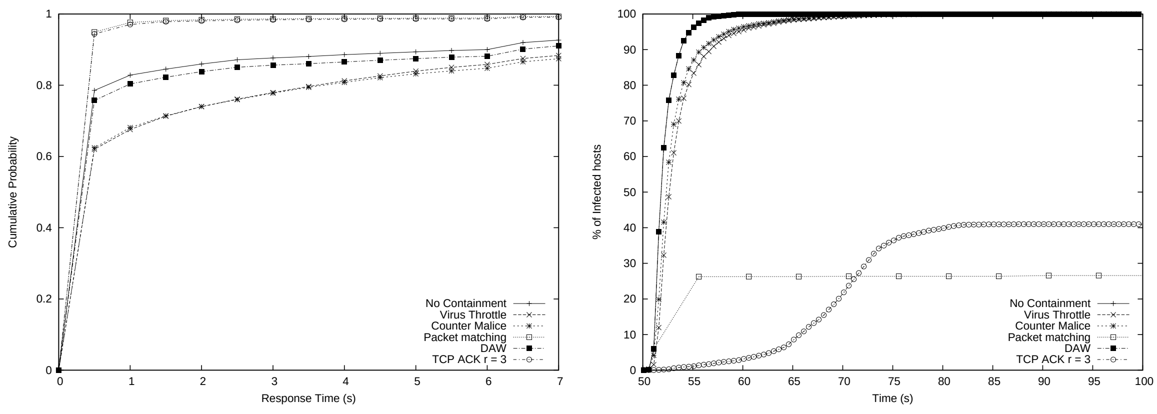
possible addresses in our simulation. Thus the probability of guessing a "good" IP address is approximately 60%.

For the worm parameters, we set the infection length to 500, the infection port to 1040, the target vector to a uniform random generator spanning the defined address space. We have set the UDP worm to have a scan rate of 100 probes per second, while the TCP worm is set to have 3 simultaneous connections.

We conducted experiments by simulating a worm outbreak on the "ring–of–trees" network described above and measured the overall rate of spread of the worm. In addition, we monitored the web response time for normal web browsing actions. The web browser model is that defined by Mah [34], and is the default web browser model in *GTNetS* . The worm attack was not started in the simulations until time $t = 50$ seconds, to allow the web browsing traffic to get started and reach steady state.

Figure 4a shows the effect of the different implementations of the discussed algorithms on the web browsing response times when a UDP worm attacks the network. Figure 4b shows their effect on the worm spread rate. Figures 5a,b shows the same results but for a TCP worm.

Since some of the defenses look specifically for the TCP-SYN packet as an indication,



(a) cdf of response times (b) Worm spread

Figure 5. Effect of algorithms on the network for a TCP worm attack

we modified those algorithms to treat a UDP packet as an infection attempt (since we had no "normal" UDP traffic in these scenarios, this is a reasonable approach). We discuss the implications of these results for each algorithm in the following sections.

2.4.1 Virus Throttle

The figures show that the throttle is capable of stopping the UDP worm spread in less than three seconds for this small network. However, in this environment, nearly 60% of vulnerable hosts are infected in that same time period. Moreover, this approach has a significant impact on the normal web browsing activity, considerably increasing the average web response time.

In the case of the TCP worm, the virus throttle approach fails to detect it or to cause any significant reduction in its infection spread. Furthermore, we still notice a considerable reduction in the performance of browsers in the network. This is due to the fact that the throttle is slowing down infected hosts (all hosts in this case) both for the infection packets and normal web browser connections, but the slow down is not significant enough to affect the worm spread .

Some of the shortcomings in the practical implementation of this approach are:

1. It is host based, so there is a risk that the worm actually attacks the algorithm and

stops it from executing.

2. This approach is ineffective against slow spreading worms with spreading rates below the threshold of detection

3. Deployment must be complete in order to attain good results, which means that deployment cost is going to be high.

4. Complete blocking of the infected host would result in blocking non worm traffic from that host as well.

2.4.2 CounterMalice

In our experiments for this approach, we placed the counter malice algorithm on each of the first level routers in the random trees (the children of the root of each tree). Thus, each counter malice process has a variable number of existing and non-existing hosts in the tree below it.

From the performance figures, we can see that the performance is worse than the virus throttle approach both for TCP and UDP worms. This is primarily due to the fact that the counter malice algorithm can not detect infections within a subnet, since the detection is on the gateway to other networks. Furthermore, once the counter malice algorithm begins blocking actions it has a significant degradation on normal web browsing.

However, this approach does address some of the shortcomings of the throttle approach. Since it is not host based we do not need to deploy it on every host but rather just on the routers. Also it can detect slow spreading worms as it does take into account the number of hosts that do not respond to a connection request. But it has the additional shortcoming of being unable to detect worm spread within a subnetwork. Unfortunately, it still has the same detrimental effect on the normal web traffic as the virus throttle approach has.

2.4.3 Packet Matching

In our experiments for this approach, we placed the packet matching algorithm on each of the first level routers in the random trees (the children of the root of each tree).

This approach is better than the previous two in that it blocks only certain port access rather than all traffic from suspected host. This means that the infected host can still have its normal traffic go through without any delays while the worm traffic is blocked or delayed.

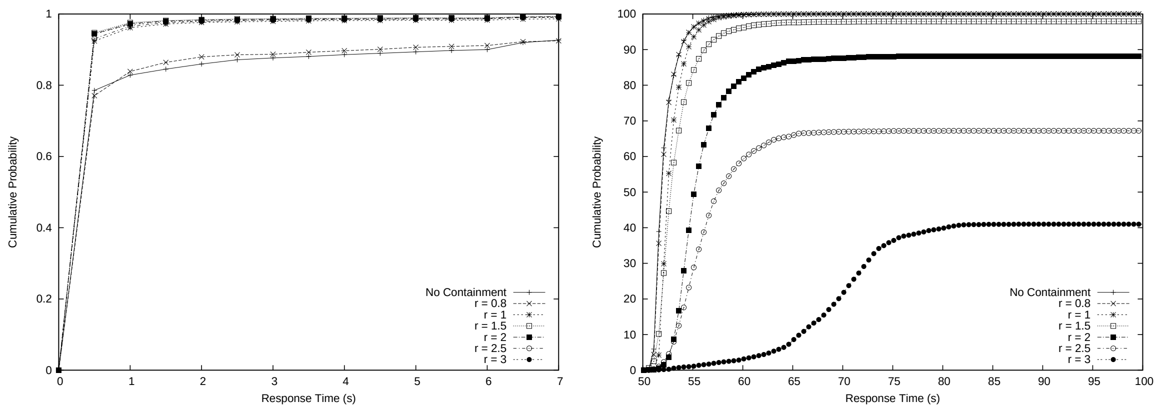
Figure 5a shows that, for the TCP worm, the packet matching algorithm was able to stop the worm infections very quickly and Figure 5b shows that the performance of the web browsing activities improved by 20% because of the suppression of worm traffic.

Figure 4a shows that, for a UDP worm, the packet matching algorithm was not able to stop the worm completely and the infections continued to spread until it reached 100%, This is due to the dynamic threshold used in this approach in which a moving average is calculated. This means that if the worm traffic was increasing slowly for a subnetwork such that the average would also increase slowly, the worm would go undetected and would be considered as normal traffic. We would expect a fixed threshold to perform better, but this would require a customized threshold for each port depending on the application running on that port and the expected level of activity on that port. Another issue is that if the infected port had been the same as the web browser port (port 80) the detrimental effects would be extremely severe, since this approach not only blocks the infected hosts, but the entire subnetwork containing the infected host. This approach also has a similar drawback to the counter malice approach in that it cannot detect or affect infections within a subnetwork.

2.4.4 DAW

In our experiments for this approach, we placed the DAW algorithm on each of the first level routers in the random trees (the children of the root of each tree).

Figures 4 and 5 show that the DAW approach, with just the *Basic-Rate-Limit* method applied, has almost no effect on worm spread in either the TCP or UDP worm cases. However, it has a significant impact on web response time resulting in more than 50% of all



(a) cdf of response times (b) Worm spread

Figure 6. Effect of TCP–ACK on the network for TCP worm

requests failing to complete in seven seconds or less in the case of the UDP worm, and 20% failing in the TCP worm case. This approach also has the shortcoming of not being able to detect or react to infections within a subnetwork.

The authors also provided two additional methods to be used for limiting the connection requests, called the *Temporal Rate-Limit Algorithm* and the *Spatial Rate-Limit Algorithm*. In the first algorithm they take into account the number of failed connections during an entire day, as they state that a normal user may generate high failure rate in a short period of time but that should not continue for 24 hours. However, an infected host would have a high failure rate all the time. Thus they define another parameter Γ that represents the threshold for failed connections in a day.

The second method takes into account the number of failed connections of the network as a whole using collaborative methods between the *DAW* processes.

We modeled the first method but instead of a period of a day we defined for a period of one minute and we set Γ to be equal to 30, meaning that we allow 30 failed connections in one minute. The results are not shown here, but did not show any major improvement to the *BasicRateLimit* method presented.

We did not model the second method as we are not taking into account collaborative efforts between agents and the central station.

2.4.5 TCP–ACK

In our new TCP–ACK approach, we define *r* as the ratio between the number of nodes that do not have an application associated with the worm infection port to the number of nodes that have the vulnerable application. We further assume that all systems have the required kernel patch to send the SYN-ACK in response to connection requests to non-existent ports.

Figure 5a shows that for the value of $r = 3$, the TCP worm is blocked completely. Further, Figure 5b shows that the performance of the web browsers was improved, since our approach does nothing to packets addressed to legitimate hosts and ports, and our approach caused suppression of the worm traffic.

We also ran experiments on the network by varying the ratio *r*. Figure 6 shows that for small values of *r* (0.8, 1) this method has no noticeable effect but by increasing r the effectiveness of the TCP ACK algorithm increases until it is able to stop the worm completely for $r = 3$ without any negative effect on the normal web traffic.

We point out that in our model the worm does not provide its own timeout period for the hung connections to fail. We expect that experienced worm developers will become aware of this defensive method and will provide some timeout period to terminate the connection. Regardless, the timeout period must be much longer than the single round-trip-time connection failure in present worms, and thus will still reduce the overall rate of spread for TCP-style worms.

2.5 Large-scale Simulations

In this section we discuss the effect of large-scale topologies on worm spread and measure the simulator performance. We are using *GTNetS* running on a large Linux cluster consisting of 16 machines running Red Hat Enterprise Linux AS release 4 (Nahant Update 5). Eight of the machines have four Intel(R) Xeon(TM) CPU 3.06GHz CPUs and 2 GB of RAM each, the rest of the machines each have two Pentium III (Coppermine) 847.402MHz

Table 3. Parameter definition for large-scale simulation experiments

Parameter Description	*Base Case*
Time of infection (sec)	10
Ratio of web servers	0.4
Ratio of web browsers	0.6
Maximum think time (sec)	10
Transport layer protocol	UDP
Worm scan rate (per sec)	50
Worm target vector	local scanning
Payload size (bytes)	500
Simulation time (seconds)	50

CPUs and 2 GB of RAM. The 16 machines are connected to each other via a Gigabit Ethernet network.

We performed several simulation experiments. In each experiment we run the simulator for 50 simulated seconds and measured the different parameters for worm spread and background traffic as well as the time and memory required for the simulation. We have 40% of the active nodes working as web servers, whereas 60% working as web browsers. In the first set of experiments we created a netwrok of 11000 nodes on one machine and then distribute the same topology on a number of machines ranging from one to fifteen. We measure the time taken and memory overhead for each experiment and calculate the speedup for having parallel simulations. In the next set of experiments we fix the number of machines to fifteen and increase the size of the topology. Table 3 represents some of the parameters used in our simulation experiments.

Figure 7 shows the effect of the different containment strategies on the worm spread, whereas Figure 8 shows the effect of them on background traffic. From the figures we see similar results as before. We see that rate limiting is effective in stopping the worm, but it also has a significant delaying effect on background traffic. From the figures we see that with rate limiting only 70% of the web browsers are able to complete their sessions in less than three seconds, as opposed to 100% when there is no containment. We can aslo see that

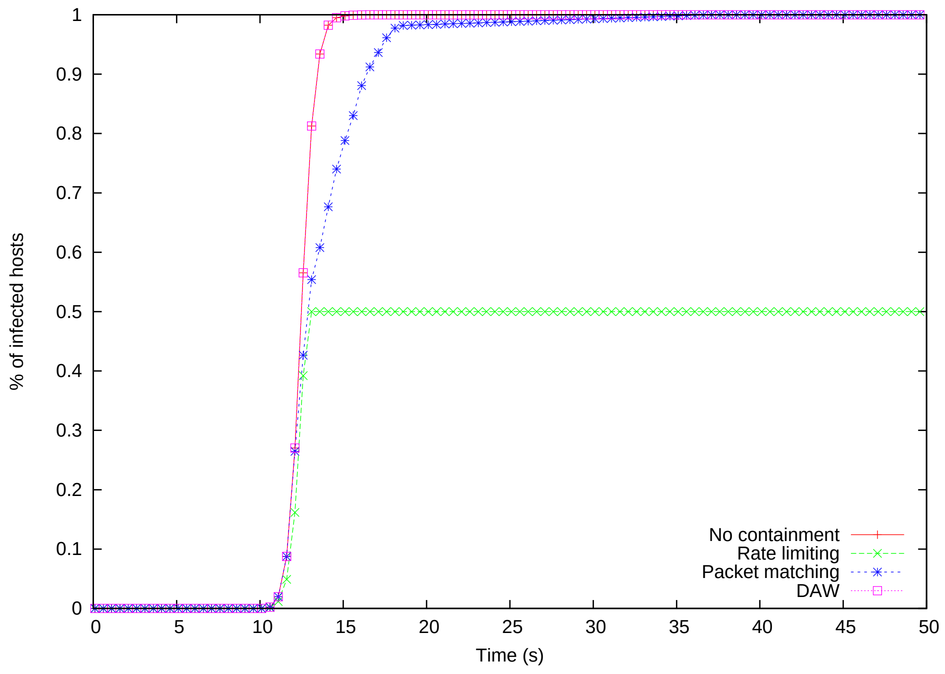


Figure 7. Effect of containment strategies on worm spread

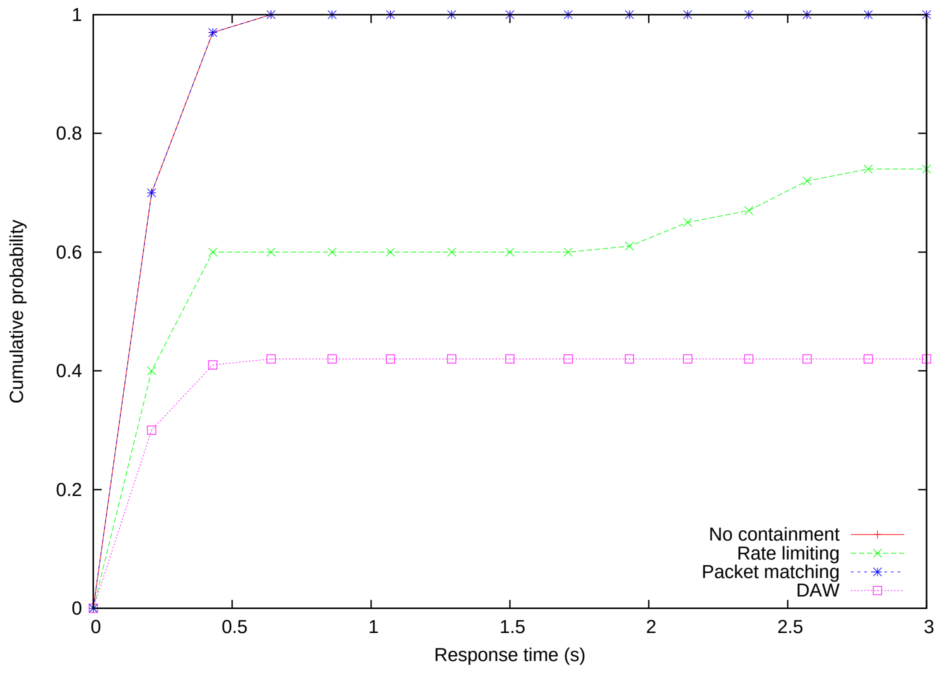
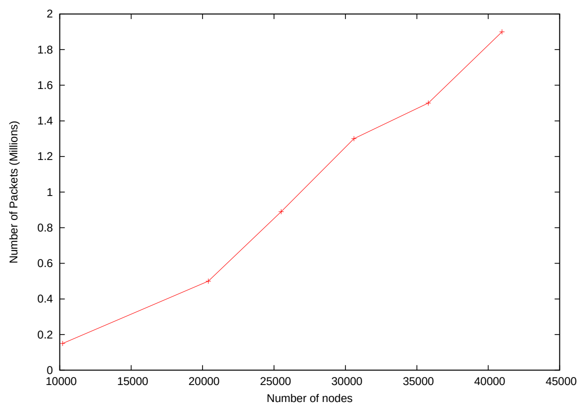
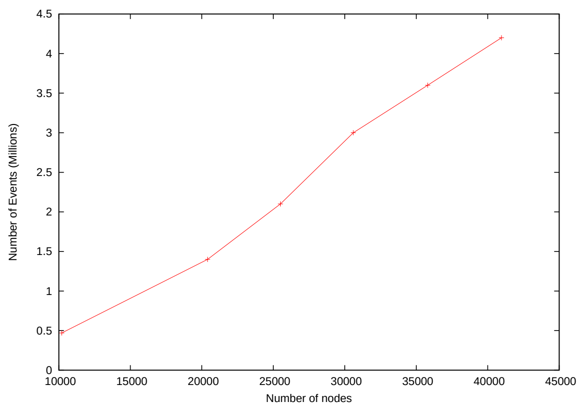


Figure 8. Effect of containment strategies on background traffic



(a) Number of packets transmitted increase



(b) Events increase

Figure 9. Effect of increasing topology size on simulator overhead

DAW has no measurable effect on stopping the worm, yet its effect on background traffic is very large. This is because DAW relies on failed connections to detect the worm while slowing down new connections, and failed connections are slower to detect. Finaly, we see that packet matching has no harmful effect on background traffic, but it can not stop a fast worm in time to prevent full infection.

Figure 9 gives a picture of the increased load for the simulator as the size of the topology increases. We can see that the number of packets transferred and events are increasing linearly with increasing topology size which is a good indication that we can simulate larger topologies and can calculate the amount of processing power we need. Figure 10 shows the increase in wallclock time to simulate 50 seconds of worm and background traffic as we increase the topology size.

Figure 11(a) shows the cdf of web response times for a variable size of networks. In this experiment we fixed the number of federates to ten and started increasing the topology size from ten to forty thousand active nodes in the simulation. We can see from the figure that as the size of the network increases, while keeping the ratio of web browsers and servers constant, the number of active flows increases which in turn increases the contention on the link. This increased contention is the cause of the drop in the web response times in the

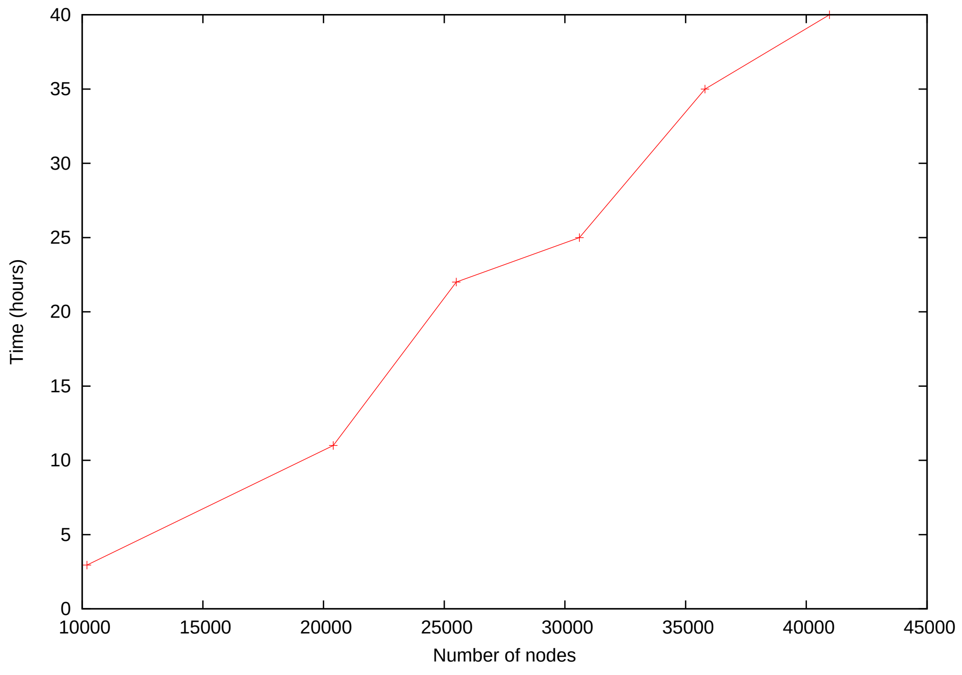
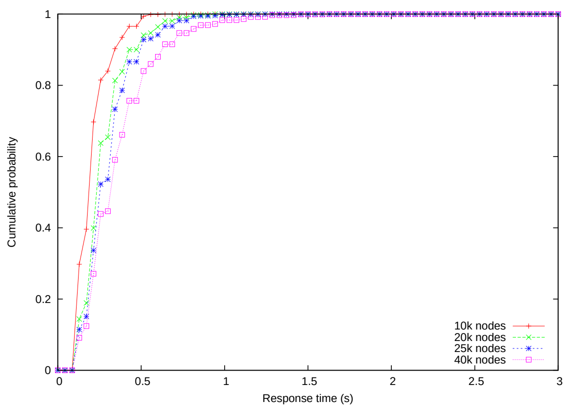
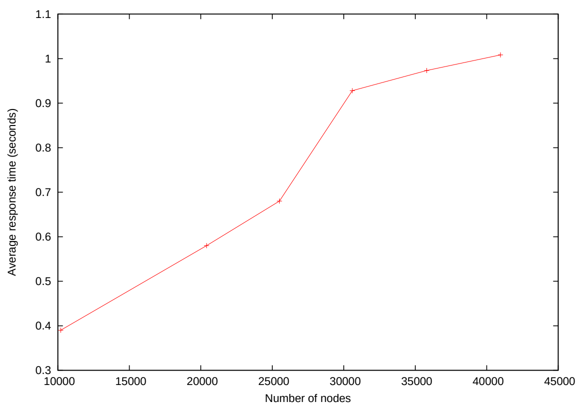


Figure 10. Effect of increasing topology size on time required to run the simulator



(a) cdf of web response times



(b) Average web response time

Figure 11. Effect of increasing topology size on web traffic performance

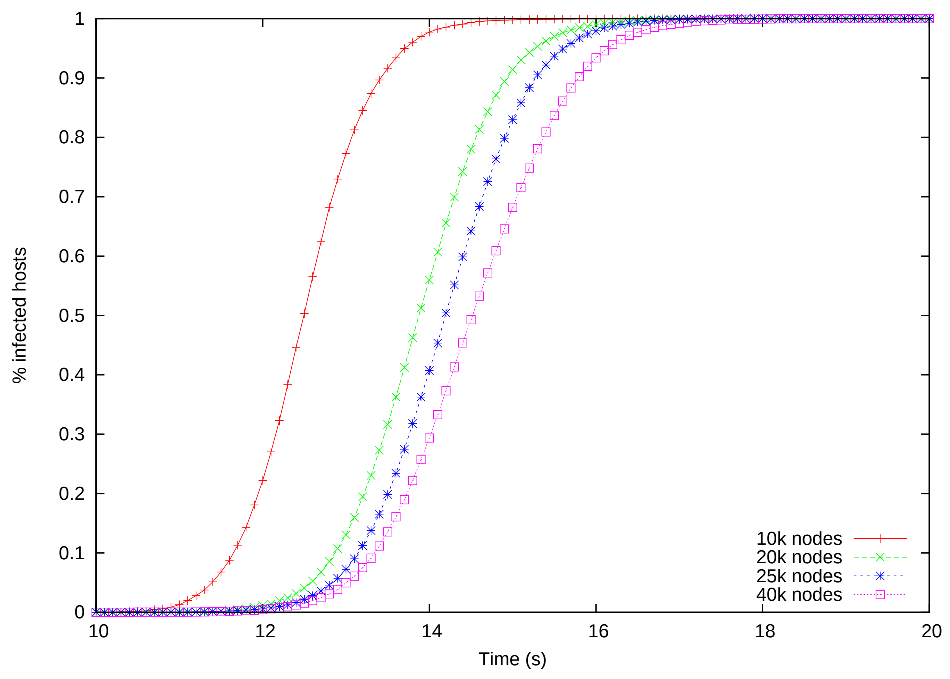
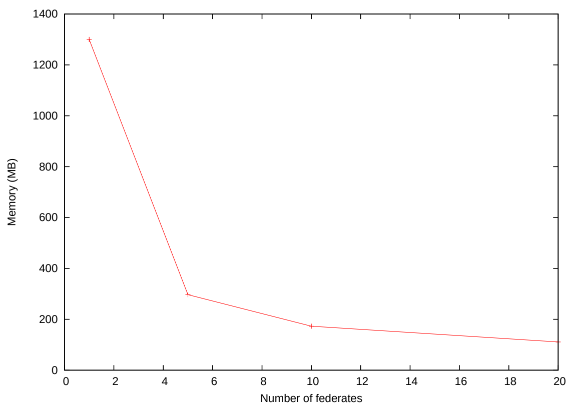


Figure 12. Worm spread as we increase the toplogy size

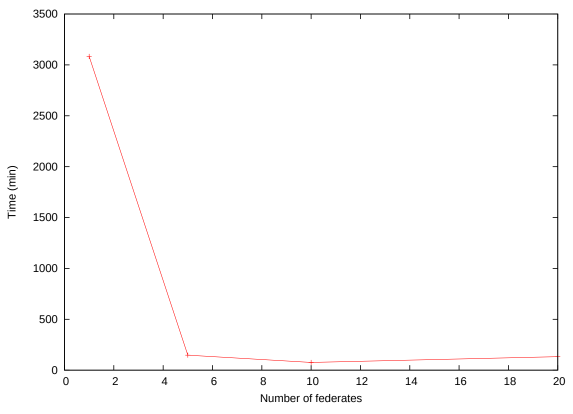
figure. Figure 11(b) gives the same conclusion by looking at the increase in the average response time for web objects as the topology is increased.

Figure 12 shows the worm spread as we increase the topology size distributed on ten federates. We can still see the effect of increased contention on the worm traffic. There is a two seconds delay between the 10k and 40k node worm spread curves, which is a considerable amount of time when we note that four seconds is all the time needed for saturation for the 10k node simulation.

Figure 13 shows the effect of splitting the topology on a number of federates. We started out by simulating 10k node topology on one machine, and then distributed that topology on a number of machines up to 20. We can see that as we move from the sequential simulation to parallel simulation we get a significant reduction in wallclock time spent for the simulation. This is because of the memory overhead for the simulation which is about 1.3 GB, meaning that most of the time was spent doing memory operations. When we remove some of that overhead by distributing the topology on five and ten machines we can see an improvement on the time spent running the simulation. However when we split



(a) memory overhead per federate



(b) time required to run the simulation

Figure 13. Effect of parallelism

the simulation to 20 machines, we see that the time rises again. This is because the overhead of running parallel simulation and message passing between federates is now more than the memory and processing overhead.

2.6 Conclusion

We have performed a detailed simulation-based study of the effectiveness of several proposed worm detection and defensive methods, and have quantified the effect of these methods on normal web-browsing activities. Further, we introduced a new defensive mechanism we call TCP–ACK, which is shown to be effective against worms using TCP connections for payload propagation. We also presented large-scale simulations for worm spread and background traffic with full-detail. We have shown the benefit of using parallel simulations to speed up the experiments and to have larger topologies.

CHAPTER 3
DESIGN OF WORST CASE SCENARIOS

The study of worst case scenarios has received high interest from researchers. In [8] the authors introduced some improvements to random scanning worms such as permutation scanning, hit-list scanning, and flash worm.

In the hit-list scanning approach the worm author collects a list of potentially vulnerable machines before the worm is released. The worm, when released onto an initial machine on this hit-list, begins scanning down the list. When it infects a machine, it divides the hit-list in half communicating half to the recipient worm and keeping the other half.

A variation on the hit-list scanning approach is the flash worm, which employs hit-list scanning with a large scale list containing all the vulnerable hosts on the Internet. In [38] the authors showed that such a worm can infect 95% of a population of one million hosts in 510 milliseconds.

This chapter introduces the Compact Flash (CFlash) worm that is a variant of the flash worm. In this approach the worm does not send the actual list of addresses to its children, rather it sends a smaller version of the address list represented by relative offsets of the addresses to each other. Thus the worm is capable of spreading even faster than its predecessor.

The remainder of this chapter is organized as follows. Section 3.1 discusses worst case scenarios and motivates the rest of this chapter. Section 3.2 provides some related work and a description of the flash worm. Section 3.3 describes the CFlash worm and its implementation in GTNetS. Section 3.4 gives a basis for the evaluation of the flash and CFlash worms as implemented in GTNetS. In Section 3.5 the results are provided. Section 3.6 gives the conclusions of this chapter.

3.1 Motivation

Internet worms have demonstrated that they are serious security risks in recent years. The Code Red worm attack in 2001 infected 360,000 hosts in 14 hours [2]. The direct costs of recovering from this epidemic (including subsequent strains of Code Red) have been estimated to be in excess of \$2.6 billion [9]. The Slammer worm outbreak in January 2002 infected 90% of its vulnerable hosts (75,000) in less than 10 minutes [4], the estimated loss is about \$1 billion [39]. In August 2003 the Blaster worm was estimated to have infected more than 500,000 systems worldwide and the cost to North American companies was \$1.3 billion [39]. However, a more recent report showed that the number of infections was between 8 million and 16 million systems [40], marking the Blaster worm as the most widely spread worm to date. The Witty worm in 2004 had a malicious payload that targets firewalls. Not only did it spread to additional hosts, but it also formatted a portion of the hard drive of the infected host [41]. A worst-case worm could cause an excess of \$50 billion in direct economic damage by attacking widely spread applications [42]. Moreover, the worm can infect critical systems as when the Slammer worm crashed the Ohio nuke plant network [43]. The damage caused by attacks on such critical systems is unmeasurable.

The design and development of automated systems to counter the worm threat is a major investment in terms of research and cost. This makes the study of worst case scenarios crucial in determining the cost/benefit of such an investment. One of the most dangerous types of worm attacks is one that is able to infect all the vulnerable population before any detection algorithm is able to detect it. In this chapter we present a simulation study of the fastest known worm (flash worm) and show some improvements that cause it to become even faster and hence more dangerous.

3.2 Related Work

There have been several studies on how to increase the speed of worm attacks and most of them focused on better target selection strategies. There are some optimizations to the random scanning method, such as those discussed in [44] and [8]. These include localized scanning (Code Red II [45]), where the worm chooses to infect a random address from within the same class B or class A address space with higher probability than other non-local addresses. This uses the intuition that if a vulnerable host is found, then there is a high probability that there are other vulnerable hosts in the same local network.

Another optimization is hit-list scanning, where the attacker collects a list of known vulnerable hosts on the Internet before releasing the worm. The worm chooses victims from this list and assigns the newly infected host a subset of the list to continue the spread.

A third optimization is permutation scanning where all the worm instances share a common pseudo random permutation of the IP address space. With this approach, there is less chance that different worm instances will choose the same victim, thus leading to faster infection spread. This way worms will not spend a lot of time in scanning the same host multiple times. In a permutation scan the already infected host responds differently than a potential target as a way of telling the worm that it is already infected. When the worm detects that it scanned an already infected machine it realizes that another worm already scanned this portion of the address space so it chooses a new random starting point and proceeds from there. This way coordination is imposed on the worm and needless reinfections are removed. A combination of hit-list and permutation scanning can create what is termed a Warhol worm [8], which is capable of attacking most vulnerable targets in less than 15 minutes.

The flash worm was first introduced in [8]. In that work the authors described hit-list scanning in the following manner: "Before the worm is released, the worm author collects a list of say 10,000 to 50,000 potentially vulnerable machines, ideally ones with good network connections. The worm, when released onto an initial machine on this hit-list, begins scanning down the list. When it infects a machine, it divides the hit-list in half, communicating half to the recipient worm, keeping the other half." This approach clearly aims at separating the scanning phase from the infection phase of the worm. The worm typically spends most of the time in searching for vulnerable hosts so by doing this separation the infection phase can be very fast. Also this separation keeps the worm undetected till infection time because usually a worm outbreak is detected when large number of hosts start to exhibit the same scanning behavior suddenly. So by doing the scanning from one or a small number of hosts, the scanning will probably go undetected.

According to the authors of the flash worm, the hit-list can be generated using one or several of the following techniques:

- *Stealthy scans*. A fast scan of the whole Internet would be unlikely to attract attention. However, for attackers wishing to be very careful, a randomized stealthy scan taking several months would not be detected.

- *Distributed scans*. An attacker can scan the Internet using a few already-compromised "zombies."

- *DNS searches*. A list of domains can be assembled and the DNS can then be searched for the IP addresses of mail servers or web servers.

- *Spiders*. Web-crawling techniques are similar to search engines and they are used in order to produce a list of most Internet-connected web sites.

- *Public surveys*. For many potential targets, there might be surveys available listing them, such as the Netcraft surveys [46].

- *Just listening*. Some applications, such as peer-to-peer networks, wind up advertising many of their servers. Similarly, many previous worms effectively broadcast that the infected machine is vulnerable to further attacks.

The flash worm works by forming a hit-list that contains all the vulnerable hosts in the Internet. The mechanics of splitting the list with every child can be represented by a logical tree (Figure 14), where a node at each level infects its descendants and divides the list of vulnerable IP addresses among them. The logical tree can be k-way (meaning each instance infects k other hosts). The number of generations (layers in the tree) to infect N vulnerable hosts is $O(\log_k N)$. The total infection time is bound by this number multiplied by the time required to infect a generation. The authors observe that a Flash worm that has a hit-list of most servers with the relevant service open to the Internet in advance of the release of the worm appears able to infect almost all vulnerable servers on the Internet in less than 30 seconds.

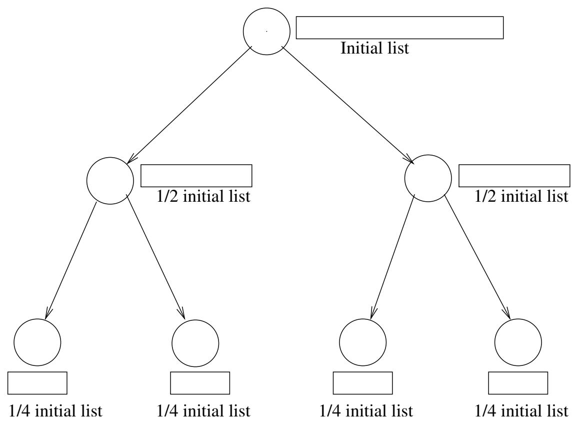


Figure 14. The logical tree for the flash worm

In a later work [38] the authors revisited their calculations and concluded that to infect 95% of a one million host topology it only takes 510 milliseconds. The parameters used to achieve that speed were:

- The logical tree has 3 layers with 9260 secondary nodes infecting 107 addresses each for a total of 1000080.
- The root node is chosen to have a high bandwidth (750 Mbps.)
- The distribution of link delays is calculated from the round-trip (RTT) measurements in CAIDA's skitter datasets [47].

The authors also note that one of the main drawbacks of the flash worm is the lack of robustness when the list of vulnerable addresses is imperfect. Since the list is assembled in advance and networks constantly change, the list is likely to be out of date by the time the worm is released. This has two effects, some vulnerable hosts may not be on the list which means that the worm will not reach full saturation. More seriously, some addresses may turn out to be invulnerable to the worm and if such hosts are on the head of the tree they would prevent all the addresses under them from getting infected and this would be more serious in deep trees. The authors in [38] studied this problem and suggested solutions to deal with it as will be shown in section 3.3.1.

3.3 Compact Flash

The list of vulnerable IP addresses across the Internet can be huge. For instance, according to the Netcraft survey for May 2005 there are about 43 Million Apache web servers and 12 Million Microsoft web servers [46]. This means that in the case of a 0-day vulnerability assuming that over 80% will be vulnerable, there will be about 35 Million vulnerable hosts running the Apache server application. The size of the list will therefore be 140 MBytes (4 bytes per address). Even if the root node had the assumed high bandwidth of 750 Mbps, it would require at least 1.5 seconds just to transfer that list to the first layer children. With that large amount of traffic, the probability of being noticed is increased therefore for an attacker who wishes to achieve optimum speed it is very important to reduce the size of that list as much as possible. By using the Compact Flash (CFlash) approach it is possible to reduce that size by a factor of 1/4 to 1/2, meaning that the size would then be 35 to 70 MBytes.

The CFlash worm has the same structure as the flash worm except that the list of IP addresses is represented by the relative offsets rather than the actual IP addresses. It can be argued that if the offsets are small enough (less than 256), then the list can be reduced to a quarter of its original size as each IP address takes four bytes of storage, whereas the small offset can take one byte. However, this is not a very practical assumption as it is almost certain that there will be two consecutive IP addresses that are separated by more than one byte offset. To deal with this problem two special characters are defined: “two-byte-offset” and “four-byte-offset” and assigned the values 0xff and 0xfe, respectively.

When the CFlash worm receives the list representing the vulnerable hosts, it calculates the next victim by adding the first offset to its own IP address and takes a chunk of the offset list and sends it to that new victim. If it encounters 0xff or 0xfe, it skips that character and reads the next two or four bytes to determine the correct offset. The size of the chunk to send to the new victim depends on the number of children for that worm (i.e. the fan out for the logical tree). The size of the vulnerable hosts list is now reduced by a factor of $1/2$ to $1/4$, however, this does not necessarily increase the worm speed by an order of two or four. The reason being that the worm speed is not only dependent on the packet size but on link-delays as well. The worm speed (for one generation) is proportional to the following:

$Speed \propto \frac{1}{packetsize/BW+linkdelays}$

This imposes an upper bound on the increase of the speed that might occur by packet size reduction of $1/linkdelays$. Moreover, if the $linkdelays$ are much more than the $packetsize/BW$, then the $linkdelays$ term becomes the dominant term and improvement from packet size reduction is minimized as will be shown in section 3.5

The algorithm for the CFlash processing is as follows:

- Initialize *current_IP* = our own IP
- Initialize *victim_IP* = our own IP
- The worm receives the list of offsets representing the vulnerable IP addresses
    - Read the size of the main list
    - *child_list_size* = main list size / number of children
    - for i = 0:number of children
        - copy the i^{th}*child_list_size* from the main list into the i^{th} child list
        - *victim_IP* = *current_IP* + i^{th} offset

∗ update the *current_IP* by adding up all the offsets in the child list

∗ Send the child list along with its size to the *victim_IP*

3.3.1 Resilience to Imperfect Maps

As we discussed before, one of the main drawbacks of the Flash worm technique is that the initial list of the vulnerable hosts' IP addresses has a good chance of being inaccurate, because the formation of that list is done before the worm starts spreading and the state of the vulnerable population could have changed by the time the worm starts its spread. This can happen in one of the following ways:

- Removal or patching of vulnerable hosts from the network.
- False positives resulting from the use of some worm defenses like TCP-ACK [48].
- The possibility of some hosts having dynamic IP addresses.

The authors of the flash worm [38] studied this problem and presented two solutions. The first solution is to have a form of an acknowledgment from the child to its parent by adding the address of the parent to the child list. If that acknowledgment is not received after a certain period of time another child is chosen from the child list and another packet is sent to the new host with the same child list. This clearly would complicate the code for the worm and would require too many timers and data structures to handle a high number of children cases and to keep state information. The second approach is to add redundancy in infecting hosts, meaning that the child list is sent to more than one host in the list. Figure 15(a) shows the case of a binary tree, where at each level a node infects its own two descendants and then sends worm copies to the two descendants of its sibling, just in case the sibling turned out to be invulnerable. The effect of this is to make it less likely that a portion of the tree will fail.

This approach has the problem that now the list has to be carried on for two layers rather than just one in the original design. This is because now each node has to know

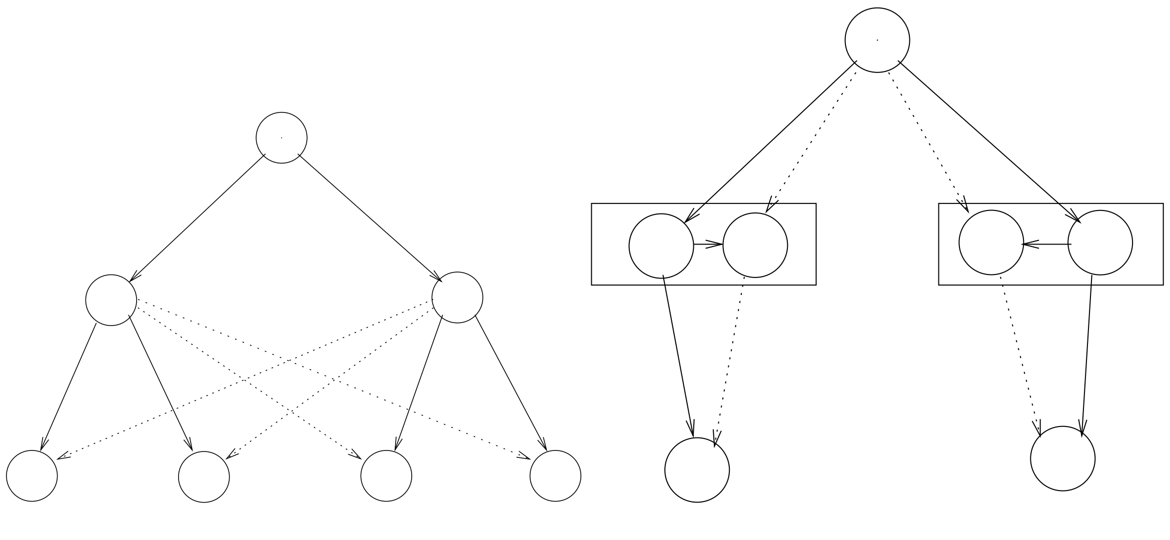

(a) Scheme I (b) Scheme II

Figure 15. Two schemes for doubling up worm delivery resilience

the addresses of the descendants of its siblings and also has to send the proper address list to those descendants. However, the same idea can be applied in a more efficient manner. Figure 15(b) shows the same case of the binary tree but now, at each level, a node infects its own two descendants as well as the next address in each of its descendant's list. The effect of this would still make it less likely that a portion of the tree will fail, as now both chosen hosts (the original descendant and its first descendant) have to be invulnerable for the nodes under them not to get infected.

If we assume that the vulnerability ratio is p then in our simple 3 layer tree model for a leaf node to get infected, the leaf node and its parent should be vulnerable. The probability for a leaf node to get infected is thus p^2 (p raised to the power of the number of layers - 1). This shows that for a list that is 50% accurate only 25% of it will get infected, meaning that 50% of the vulnerable hosts will not get infected (the remaining 25%).

This added redundancy causes the worm to be more resilient to imperfect maps because of the fact that now for a certain node in the list to not get infected the two chosen parents have to be invulnerable, which has less probability of occurring. In this case for a leaf node to get infected either one of its parents could be vulnerable ($2p(1-p)$) or both of them are vulnerable (p^2) and the leaf node has to be vulnerable as well, thus now the probability of a leaf node to get infected is $2p^2(1 - p) + p^3$ after expansion it would be $2p^2 - p^3$. In that case if the list was 50% accurate then the infection would be 37.5%, meaning that only 12.5% vulnerable hosts from the original list will not get infected (25% of the vulnerable hosts will not get infected.)

3.3.2 Simulation Model

The flash and CFlash worms are modeled using GTNetS as application layer objects with the following parameters:

- *Layer4proto*, the layer 4 protocol object to be associated with the worm (either UDP or TCP.)
- *Fanout*, the maximum number of children for each parent node in the logical tree.
- *Vulhosts*, the list of vulnerable IP addresses known to this worm.
- *Infected*, a flag indicating if this node is infected or not.
- *Vulnerable*, a flag indicating if this node is vulnerable to the worm attack or not.
- *Node*, the attached host node.
- *InfectionPort*, the port which has the vulnerable application running on it.
- *TotalInfected*, the number of infected hosts.
- *TotalVulnerable*, the number of vulnerable hosts.
- *PayloadSize*, the size of the infectious payload.

The processing of the flash worm is in the following manner:

```
• The worm receives the list of vulnerable hosts' IP addresses
    – Read the size of the main list
    – $child\_list\_size$ = main list size / number of children
    – for i = 0:number of children
        * copy the $i^{th}$ $child\_list\_size$ from the main list into the list of
          the $i^{th}$ child
        * $victim\_IP$ = first IP in the child list
        * Send the child list along with its size to the $victim\_IP$
```

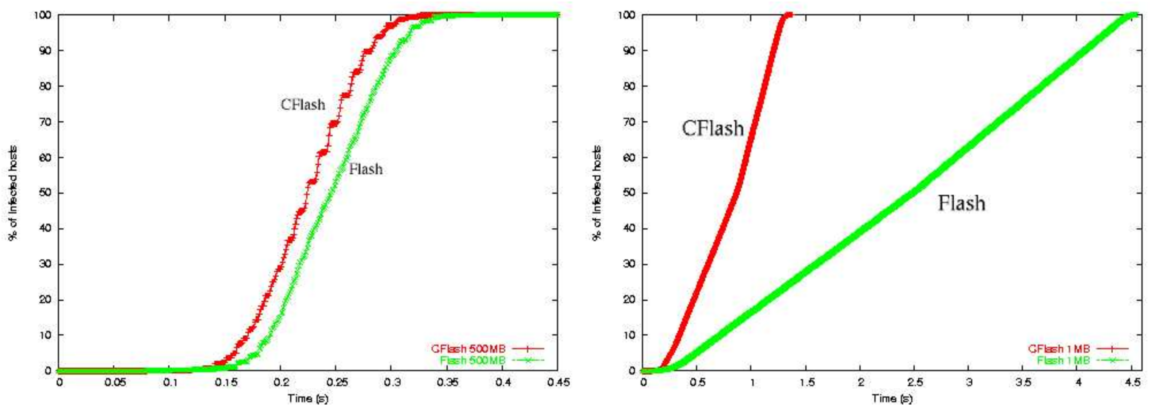
3.4 Experimental Setup

The experiments were conducted using the random tree topology [13]. The random tree is a tree topology with the addition of a random probability factor that determines if a child node should be created or not, which leads to the presence of holes in the IP address space, leading to a more realistic representation of the network.

In this work there is a difference between the logical tree and the physical tree. The logical tree is the tree representing the hierarchy of infection from parent to child nodes regardless of how they are actually connected in the network, whereas the physical tree is the chosen topology structure that connects the hosts with actual links.

The logical topology for the Flash and CFlash worms is a three-layer tree with a maximum of 1000 children under each parent node. The *layer4proto* is set to UDP, and the *PayloadSize* is set to 400 bytes. The basic setup for all the experiments is that the topology is generated with different parameters settings (bandwidth of links, time delay, and vulnerability) and we start the worm infection and then measure the spread rate.

The physical topology for the experiments consists of 24 random trees connected together using a ring topology. Each tree has an address space of 65 k and an average of 45 k real nodes. The total topology size is thus about one million leaf nodes. The size of the list of IP addresses for the Flash worm is 4.119 MBytes; by using the offsets technique the size becomes 1.03 MBytes. The bandwidth of the connecting ring links is fixed at 1 Gbps. In the next section we discuss the results of our simulation experiments.



(a) High bandwidth links (b) low bandwidth links

Figure 16. Worm spread for the flash and Compact flash (CFlash) worms

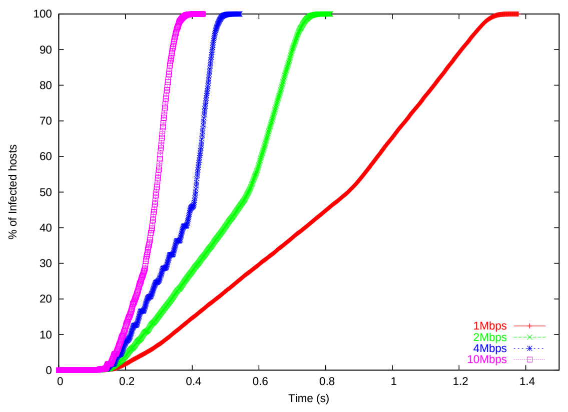
3.5 Results

3.5.1 Effect of Varying Bandwidth

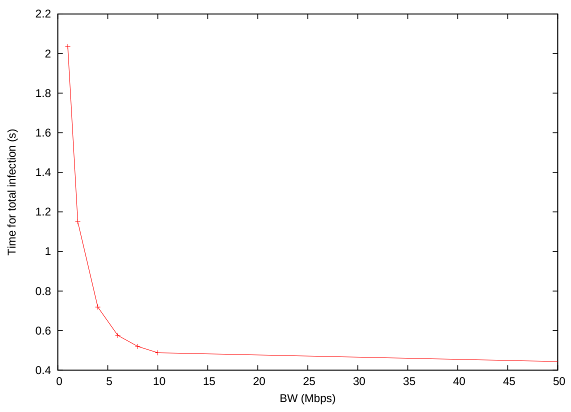
In this section the effect of varying bandwidth of the links in the trees is examined. For the high bandwidth case (Figure 16(a)) the experiments show that for a one million node topology the flash worm took 340 milliseconds to infect 95% and 413 milliseconds for full infection, while the CFlash worm took 290 milliseconds to infect 95% and 380 milliseconds for full infection. That is, a difference of 30-50 milliseconds is observed because of the size reduction of the list of vulnerable IP addresses by using the CFlash worm. In that case small improvement is observed because the high bandwidth caused the $packet - size/BW$ term to be negligible compared to the $link - delays$ term.

The same experiment was carried out on low bandwidth links as shown in Figure 16(b) and it can be observed in that case that the speed difference between the flash and the CFlash worms is much bigger (about 3 seconds difference for total infection). In this case the CFlash worm is more than 3 times faster than the flash worm.

Figure 17 shows the effect of having different bandwidth settings for the connecting links in the trees for the CFlash worm, with a fixed time-delay of 10ms. It is clear that the slope of the infection curve is reduced with reduction of available bandwidth, this is



(a) Worm spread



(b) Time for total infection

Figure 17. Effect of varying bandwidth of links on the worm spread and infection time

probably due to saturation of the links which is clearer for low bandwidth as the worm spread is not exponential anymore and it tends to be linear.

3.5.2 Effect of Varying Link-delays

Another set of experiments shows the effect of varying the link delays on the speedup for the CFlash worm relative to the flash worm. Figure 18 shows the worm spread for the Flash and CFlash worms for different time-delay links with 10 Mbps bandwidth of connecting links. It can be observed that the increase in speed is higher in case of the low time delay of links compared to when the time delay of links is high.

Figure 19 shows the effect of having different time-delay settings for the connecting links in the trees on the worm spread and infection time for the CFlash worm with fixed bandwidth of 10 Mbps. It is clear that the slope of the infection curve remains the same but the curve shifts to the right with increase in the time delay of the links.

3.5.3 Resilience to Imperfect Maps

In this section we study the effect of having inaccurate maps to represent the vulnerable IP addresses.

Figure 20 shows the simulation results for 20 experiments using different values for the

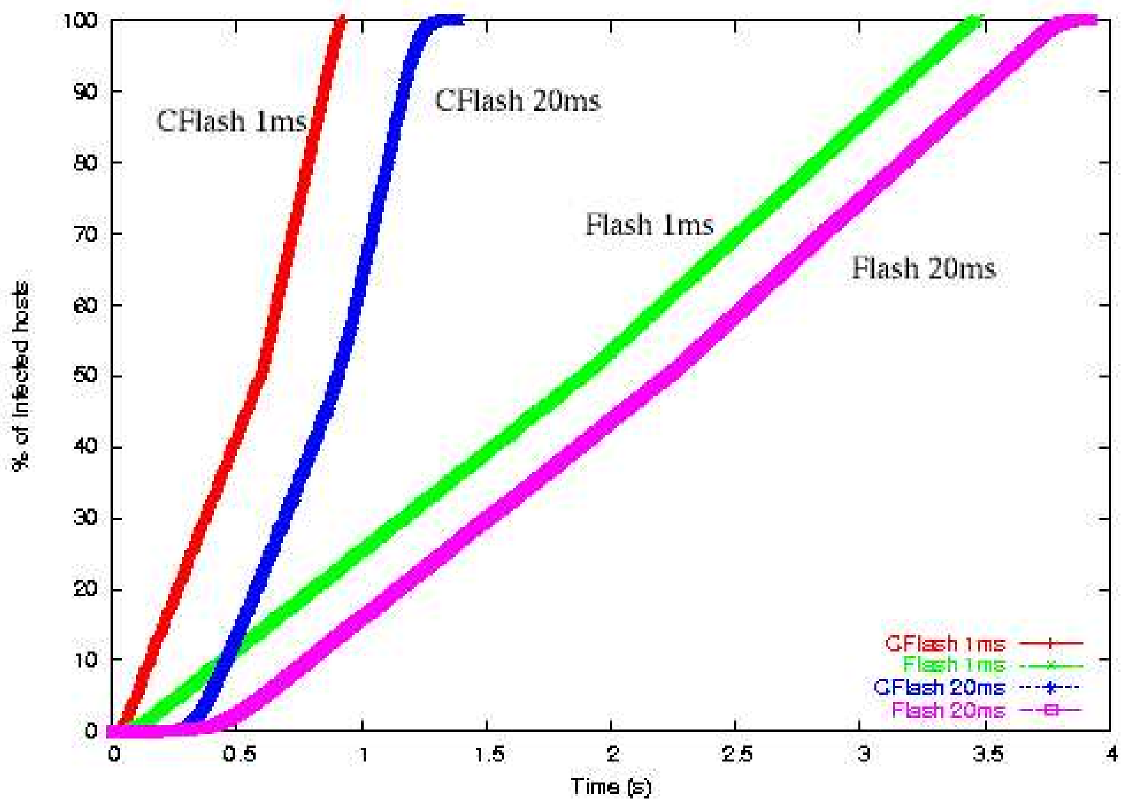


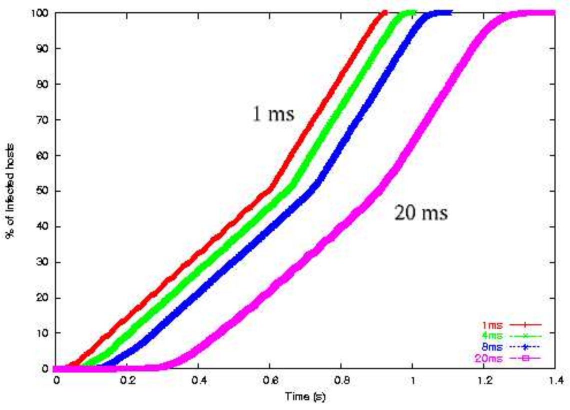
Figure 18. Worm spread for the flash and Compact flash (CFlash) worms for different time-delay links

vulnerability ratio and measuring the infection percentage for the original case with single infection. The same figure also shows the effect of the added redundancy by using double infections.

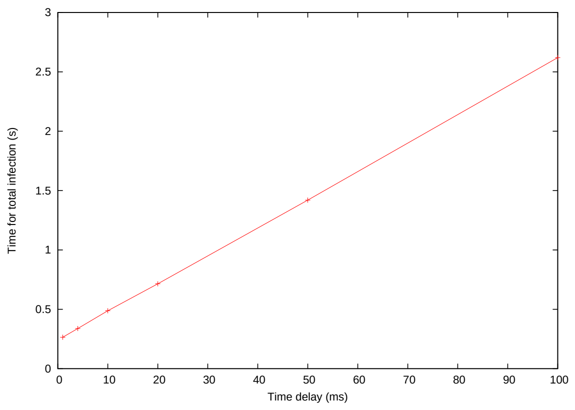
It is clear that when the vulnerability ratio is low, the worm spread fails to infect all the vulnerable population. Even with 50% vulnerability, we see that only 30% of the vulnerable hosts (30% of 50% of one million = 150 k out of 500 k vulnerable hosts) are infected. In these tests if the tree was a deep tree (less number of children and more layers), then the effect of the map being inaccurate would be more severe meaning that even with slight inaccuracy the infection would fail to spread to most of the vulnerable hosts.

3.6 Conclusions

It has been shown that by efficiently decreasing the size of the packets sent by the flash worm, the worm can be sped up further, leading to the possibility of infecting a one million



(a) Worm spread



(b) Time for total infection

Figure 19. Effect of varying time-delay of links

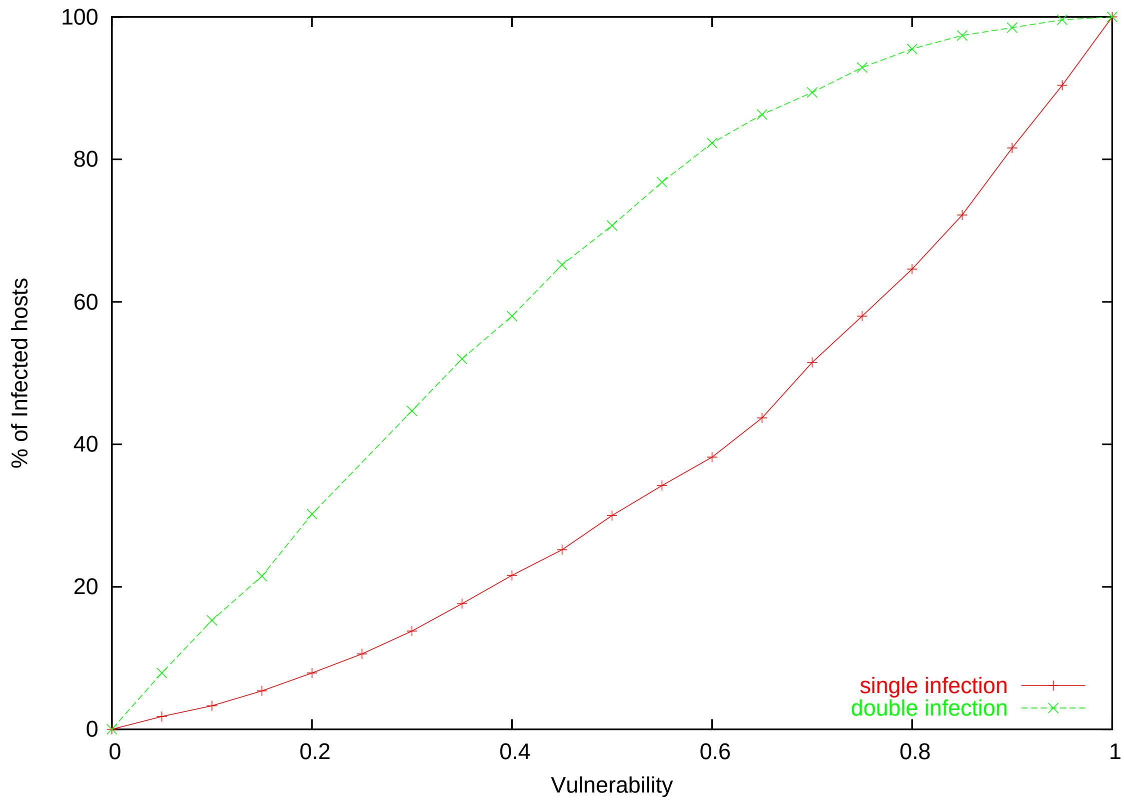


Figure 20. Varying the vulnerability of hosts in the initial list using single and double infections

host population in less than 400 milliseconds for high bandwidth links. The improvement is clearer in the low bandwidth case where the CFlash worm becomes three times faster than the flash worm. This main idea can be extended by using compression and decompression of the address list at the infected hosts. A detailed study of the benefit of reduced size against the cost of increased complexity needs to be carried out to determine the optimum size of the infection packets.

This high speed of infection shows that no human counter-measure can stop the worm in time, and even automatic mitigation techniques with time delays over 400 milliseconds are not efficient in stopping that worm once it starts spreading. The only hope of countering such an attack is to stop this kind of worm in the scanning phase before it starts to spread.

CHAPTER 4
MODELING AND SIMULATIONS OF MANET WORMS

The effect of worms on a Mobile Ad-hoc Network (MANET) topology is more serious than on a computer network because of the resource constrained nature of such a network. According to the recent DARPA BAA Defense Against Cyber Attacks on MANETS [49], "One of the most severe cyber threats is expected to be worms with arbitrary payload that can infect and saturate MANET based networks on the order of seconds."

A MANET is a self-configuring network of mobile nodes that act as routers and hosts, and are connected by wireless links. The topology of MANETs is arbitrary and dynamic. MANETs can operate in a standalone fashion, or may be connected to the Internet. Because of the fact that MANETs require minimal configuration and are quickly and easily deployed, they are suitable for emergency situations like disaster–relief, military applications, and emergency medical situations. These applications make MANETs attractive targets for cyber–attacks and make the development of counter–measures paramount.

We developed an analytical model for the TCP worm spread in the MANETs environment. The results show that this model matches the results of the packet- level simulation in most cases except when the probability of buffer overflow is high.

The rest of the chapter is organized as follows. Section 4.1 discusses some challenges for worm modeling in MANETS and motivates the rest of the chapter. Section 4.2 gives a brief overview of related work. Section 4.3 describes the TCP worm model that we developed to describe salient aspects of the TCP worm propagation with the wireless MANET environment. Section 4.4 describes the simulation experiments that were conducted to validate our model. Section 4.5 provides the results and their discussion. Finally, the conclusions of the chapter are provided in the end.

4.1 Motivation

The modeling of TCP behavior in the MANET topology is very challenging. This is because the wireless medium is very different from wired networks. Fu et al. [50] studied how TCP behaves in a multihop wireless network that uses the IEEE 802.11 protocol for medium access. They noted that multihop wireless networks have several charachteristics different from wired networks. 1) Packets may be dropped because of buffer-overflow or link-layer contention. 2) The wireless medium is a scarce shared resource. They showed that there is a special window size at which TCP delivers best throughput. However, TCP does not operate arround that window size, and typically grows its average window size much larger; this leads to increased packet loss and decreased throughput. They also noted that network overload is specified by wireless link contention and not because of buffer overflow. They also showed that as the offered load increases, the link contention drop probability also increases but saturates eventually.

Morever, there are yet even more characteristics about MANET that makes it more different from the Internet.

- Small scale: The MANET network is usualy in the range of hundereds to thousands of nodes.
- Application distribution: The Internet has large variations in the applications and operating systems running on end hosts. On the other hand, all hosts in the MANET usually run the same operating system and applications. This means that worm infections spread at a faster rate in this kind of environment.

All this suggests that the models developed for worm propagation on the Internet are not suitable for the representation of worm outbreaks in the MANET environment. Therefore, more studies about this specific problem are needed to develop accurate mitigation techniques for this critical environment. This chapter studies the problem of TCP worm spread in the MANET environment and builds a simple analytical model that compares well to simulation results.

4.2 Related Work

Several studies were carried out to analyze and model the propagation of computer worms in digital communication networks. In [5] the authors present the different kinds of worms depending on their scanning strategies, worm carrier mechanism, possible payload and plausible attackers who would employ such a worm. In [8] the authors provide an extensive investigation into the mechanisms of worm propagation and their performance. They also provide some improvements for the worms and the effort needed to mitigate worm propagation throughout the Internet.

Zou et. al. [51], studied the Code Red worm outbreak and provided an analysis of its propagation by accounting for two factors: i) the dynamic countermeasures taken by ISPs and users, and ii) the slowed infection rate due to the rampant propagation of the worm causing congestion and troubles to some routers. They derived a general Internet worm model called the *two-factor worm model.*

The most relevant work to this chapter is [52] and [53]. In [52] the authors investigated the impact of communications and mobility effects on worm propagation mechanisms in MANETs, where they found that network delays and channel congestion had a large impact on the UDP-based worm propagation behavior. They also provided a set of relatively simple analytical models that reproduced these communications. In [53] the authors discussed the effect of mitigation techniques on the UDP-based worm spread in MANETs and provided analytical models and simulation experiments to validate their findings. The authors represented the mitigation technologies as having a constant detection time represented by a lifetime parameter of the worm, after which the worm dies and stops infection of other hosts. These studies were limited to UDP worms. In this chapter we extend their simulation models to be used in GTNetS and to include TCP style worms in MANETs as well.

4.3 The TCP Worm Propagation Model

Here we extend that prior analysis to the investigation of TCP-based worms; the previous work focused on UDP-based worms. Our investigations involve extensive, high fidelity simulations of TCP-based worms using the *Georgia Tech Network Simulator* (GTNetS) [14]. Further, we develop and investigate a model of the TCP worm propagation in MANETs. The model results are compared against the simulation experiments. The model is an extension to the Standard Epidemic Model, which is

$$\frac{di(t)}{dt} = \beta i(t)[1 - i(t)] \tag{1}$$

where $i(t)$ is the probability of nodal infection at time t, and β is the rate at which a given infected node is successful in infecting other susceptible nodes.

For a simple, UDP-based flash worm, β is generally set to the product of: i) the inverse of the UDP packet transmission time onto the communication interface of the infected host, times ii) the probability that the packet is addressed to a susceptible host, times iii) the probability that the UDP packet is not lost in transit due to network congestion. The first term is simply proportional to the communications line speed divided by the size of the single UDP packet. The second term is usually taken as the ratio of the susceptible host population divided by the entire address space, assuming a worm which implements a random address search strategy. Zou et. al. [51], have suggested that the last term be approximated by $[1 - i(t)/N]^{\eta}$ where η is a fitting parameter. This term estimates the probability of packet receipt success at the susceptible host under conditions of buffer overflows within the network.

For a TCP-based worm, β must assume a somewhat different form for two reasons: i) the rate at which an infected node can transmit the worm to other nodes is related to the number of simultaneous TCP connections divided by the mean time for a TCP connection to transmit the worm payload, and ii) network congestion does not decrease the probability of receipt of the payload at the susceptible host, but instead congestion slows the time to transmit the payload due to bandwidth competition. Bandwidth competition can slow the TCP transmit times due to increased bandwidth sharing and increased packet losses. Therefore, we investigate a TCP model which accounts for bandwidth sharing and includes mechanisms within the underlying 802.11 Link and Physical layers which cause packet discards at higher loads.

We first investigate the performance of TCP worm propagation through analytic models. Our analytic modeling describes the 802.11 wireless MANET as residing in one of two states; a low load state where the number of TCP flows in the network is relatively small and a higher load state where the number of TCP flows is moderate to high. In the state with a small number of flows we develop a bandwidth sharing expression for β which accounts for a slowing of the worm propagation due to competition for the radio channel bandwidth, without however causing significant packet losses. In the state with moderate number of TCP flows we develop a model which accounts for high probabilities of packet discards due to collisions and 802.11 retry limits, but not due to buffer overflows. This follows the work of Fu, et al., [50] who found that 802.11 networks settle into a state where the nodal probability of packet losses becomes flat as more and more transmitters (or flows) are added to the network. In fact, we see this same effect in our simulation modeling and find that the transition between the low number of flows state and the moderate to high number of flows state occurs at a very small value for the probability of nodal infection, i.e., around 0.1.

4.3.1 Low Number of Flows

This model applies to small values for the probability of nodal infections.

Define:

$i(t)$	=	probability of infection
b	=	bandwidth of the radio channel (Bps)
d	=	nodal density (nodes/sq.meter)
r	=	radio range of the channel (meters)
c	=	radio interference range factor
$n(t)$	=	mean number of infected neighbors
α	=	(zero load) TCP throughput in proportion to the channel bandwidth

Assuming perfect sharing of channel bandwidth during the transmission of a TCP worm payload, the channel bandwidth is partitioned equally to each of the neighboring TCP transmissions. Thus, the portion of the channel bandwidth available to each TCP transmission is

$$\bar{b} = \frac{b}{(n(t) + 1)} \tag{2}$$

Assuming uniform placement of nodes in the MANET, we write

$$\bar{b} = \frac{b}{[\pi d(r(1 + c))^2 i(t) + 1]} \tag{3}$$

So, we consider each infected node seeing a time varying available bandwidth due to increasing probability of infection over time. The Standard Epidemic Model becomes

$$\frac{di(t)}{dt} = \beta(t)i(t)[1 - i(t)] \tag{4}$$

where β is now time dependent.

For a single threaded TCP operation, as modeled in our simulation studies discussed below, let τ_{tcp} be the average TCP throughput for the transmission of the worm payload. Then, the time to transmit the TCP worm payload is

$$t_{tcp} = \frac{P_w}{\tau_{tcp}} \tag{5}$$

where P_w is the payload size of the worm. From the above, modified Standard Epidemic Model, $\beta(t)$ is

$$\beta(t) = \frac{1}{t_{tcp}(t)} = \frac{\tau_{tcp}(t)}{P_w} \tag{6}$$

and

$$\tau_{tcp}(t) = \alpha\bar{b}(t) \tag{7}$$

where α is the proportion of the available channel bandwidth that a TCP connection is able to obtain under zero load situations, see, e.g., [54] [55] and [56]. Hence, α is a function of the mean number of hops across the MANET, the packet size, etc. We assume that α is a constant with respect to time. Inserting these expressions into the above Epidemic Model yields

$$\frac{di(t)}{dt} = \left[\frac{\alpha b}{P_w(1 + \pi d(r(1 + c))^2 i(t))}\right] i(t)[1 - i(t)] \tag{8}$$

This expression is comparable to the expression for the UDP-based Flash worm proposed by Zou et. al., [51]

$$\frac{di(t)}{dt} = \beta[1 - i(t)]^n i(t)[1 - i(t)] \tag{9}$$

These equations are different because they consider the different effects of network congestion. Our TCP worm model is predicting a diminishing TCP throughput due to channel sharing at higher infection probabilities. While Zou et al.'s model is predicting a decreasing probability of end-to-end packet delivery success due to buffer overflow under increased network competition at higher infection probabilities.

4.3.1.1 Analytic Results

In this section we investigate the analytic solution to our TCP worm model given in Eq.(8). Following the method of factoring used to solve the Standard Epidemic Model of Eq.(1), we rewrite Eq.(8) as

$$\frac{di(t)}{dt} = \gamma_{low} i(t)\left[\frac{1 - i(t)}{1 + \theta i(t)}\right] \tag{10}$$

where $\gamma_{low} = \alpha b/P_w$ and $\theta = \pi d(r(1+c))^2$. We factor this expression into

$$(1+\theta i(t))\left[\frac{1}{i(t)} + \frac{1}{1-i(t)}\right] di(t) = \gamma_{low} dt \tag{11}$$

Integrating both sides of this equation and rearranging terms yields

$$\frac{i(t)}{i(o)}\left[\frac{1-i(o)}{1-i(t)}\right]^{1+\theta} = e^{\gamma_{low} t} \tag{12}$$

or

$$\frac{i(t)}{[1-i(t)]^{1+\theta}} = g(i_0)e^{\gamma_{low} t} \tag{13}$$

where

$$g(i_0) = \frac{i_0}{[1-i_0]^{1+\theta}} \tag{14}$$

Here $i_0 = i(t=0)$. This is as far as we can get in writing the explicit solution to our TCP model. For general values of θ, we cannot solve this expression for $i(t)$. However, when $\theta = 1,\ 2\ or\ 3$ this expression represents a quadratic, cubic or quartic expression in $i(t)$, respectively. For other values of θ, i.e., θ real or $\theta > 3$, no known explicit solutions exist.

For $\theta = 0$, Eq.(13) reduces to

$$i(t) = \frac{g(i_0)e^{\gamma_{low} t}}{[1 + g(i_0)e^{\gamma_{low} t}]} \tag{15}$$

This is the well known solution to the Standard Epidemic Model. For $\theta = 1$, Eq.(13) reduces to

$$i(t) = \frac{1 + 2g(i_0)e^{\gamma_{low} t} \pm \sqrt{1 + 4g(i_0)e^{\gamma_{low} t}}}{[2g(i_0)e^{\gamma_{low} t}]} \tag{16}$$

The upper solution, obtained by choosing the upper plus sign, is a non-physical solution resulting in the probability of infection exceeding unity. The lower, i.e., minus sign, solution is the physical solution. In comparison with the previous $\theta = 0$ solution, this expression demonstrates the slowing effects of bandwidth competition in the network. Solutions are also possible for $\theta = 2$ and 3, but due to space limitations we do not present these here.

We can determine the general asymptotic behavior of $i(t)$ as $t \to \infty$ from Eq.(13). The right hand side of Eq.(13) clearly approaches infinity as $t \to \infty$. This implies that the left hand side of Eq.(13) also approaches infinity, which can only happen if $i(t \to \infty) = 1$. Also, by writing $i(t) \approx 1 - \epsilon$ where for large t, $\epsilon << 1$, we can perform an expansion of Eq.(13) in terms of ϵ. This yields the follow expression for $i(t \to \infty)$,

$$\lim_{t \to \infty} i(t) \approx 1 - g^{-1/(1+\theta)} e^{-\gamma_{low} t/(1+\theta)} + ... \tag{17}$$

Clearly, the larger θ is, the slower is the convergence of $i(t)$ toward unity, reflecting that fact that greater bandwidth competition is slowing the propagation of the TCP-based worm.

Finally, we can analyze Eq.(13) in the context of determining the time for the infection to reach a given percentage of infection, e.g., 50% of the population, $t_{1/2}$. Substituting $i(t_{1/2}) = 1/2$ into Eq.(13), we get

$$t_{1/2} = \gamma_{low}^{-1}(\theta ln2 - lng) \tag{18}$$

where $\gamma_{low} = \alpha b/P_w$ which is the mean time for a host to transmit the worm payload under conditions of no bandwidth competition. In time units of γ_{low}, we expand out the above expression to yield

$$t'_{1/2} = \pi d[r(1 + c)]^2 ln2 - ln\left(\frac{i_0}{1 - i_0}\right) \tag{19}$$

So, for a given initial condition, $t'_{1/2}$ increases linearly with the mean number of neighbors within radio range, e.g., $d\pi[r(1 + c)]^2$.

While it is interesting that results for the asymptotic behavior is available, we will find that in 802.11 wireless networks with TCP worms, the results of this low load model will apply only for very small probabilities of infection, i.e., $i(t) < 0.2$. We now turn our attention to the derivation of a TCP worm propagation model for moderate to high numbers of TCP flows.

4.3.2 Moderate to High Number of Flows

We rely on the fact, shown in [50], that the network saturates as the number of TCP flows increases. In this saturated state, many of the network performance characteristics become independent of further increases in the number of TCP flows. This allows us to derive a simple relationship between the TCP throughput versus the number of TCP flows in the network.

In this saturated state, the per node packet loss rate becomes a fixed constant independent of further increases in the number of TCP flows. Furthermore, the packet loss rate is a result of collisions due to hidden terminal issues and not buffer overflow. In fact, they show that the mean buffer occupancy in this saturated state is extremely low and they find that little, if any, packets are lost through buffer overflow as verified through simulation studies. In 802.11 networks, nodes will discard a data packet in the event that the node has attempted to send an RTS seven times without success. Here seven is a default parameter of the Medium Access Control (MAC) protocol. Hence, packets handled by the nodes are either transmitted to the next hop or are discarded due to a failure of the node to gain access to the channel after seven attempts. The explanation for the network saturation is that as the number of flows increases, the number of "backlogged nodes", i.e., those nodes with packets to transmit, quickly approaches N, the total number of nodes in the network. Once in this state, the network performance saturates in terms of the packet loss rate and the overall network throughput. This behavior does not change as further increases in the number of TCP flows occur. We observe this fact in our simulations.

Figure 4.3.2 shows the results of a typical simulation run where we plot the total number of packet discards in the network versus time. Here we see that at around 50 seconds into the simulation, the rate of packet discards becomes flat, independent of increasing the number of TCP flows into the network. The results for this plot are for a worm payload size of 400 Kbytes and are to be compared to the propagation results shown in Figure 23 below. There we see that the probability of infection in the network at a simulation time of 50 seconds is less than 0.2. This is an extremely low probability of infection for the network to enter into this relatively static, high packet loss state.

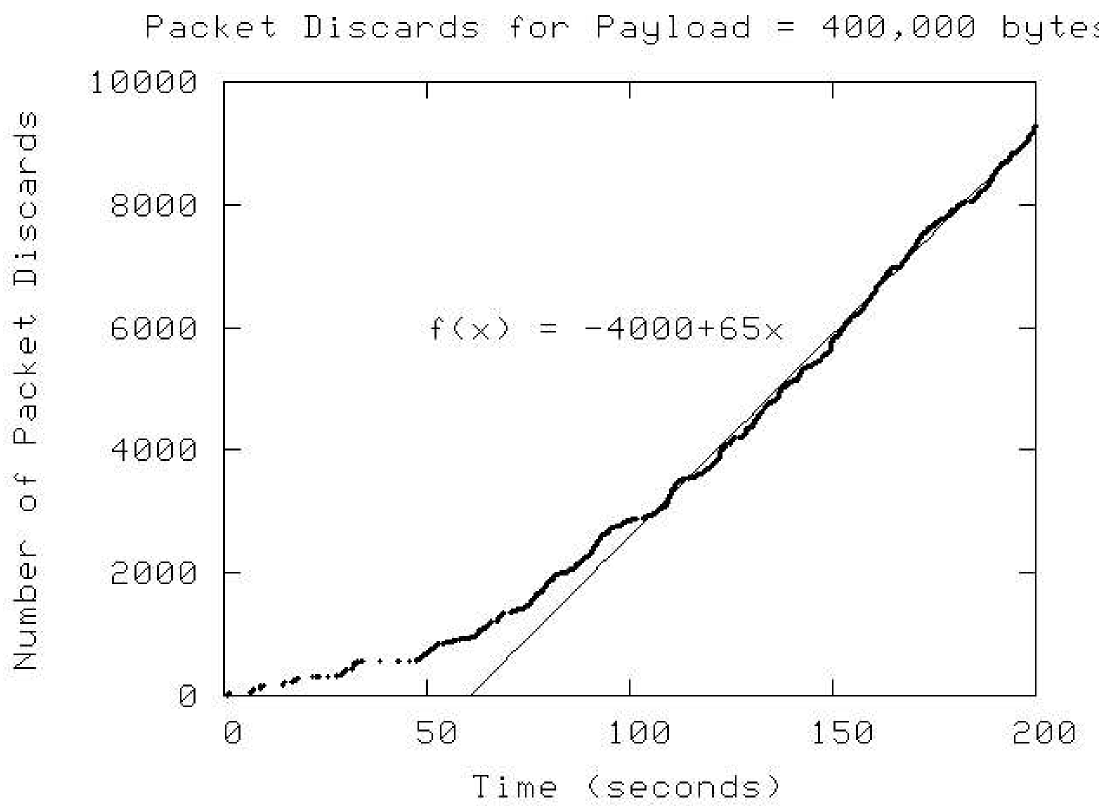


Figure 21. Example packet discard results from simulation traces for a TCP worm with payload of 400 Kbytes.

Define:

$$\begin{aligned} p &= \text{the nodal packet loss probability} \\ u &= \text{the nodal packet processing rate} \\ m &= \text{the number of “backlogged” nodes} \\ f &= \text{the number of flows in the MANET} \\ l &= \text{the mean number of hops per path} \end{aligned}$$

The TCP flows are generated by our TCP worm, and hence we have that $f = N \times i(t)$. In order to estimate the dependence of m on f let us assume the following model. At a minimum, $m \geq f$, because by definition each flow has a different source node in our worm model. Furthermore, because each flow on average makes l hops through the network, we know that m is greater than f. In fact, we can estimate the probability that a node is not backlogged given f as follows. As previously stated, at least f nodes are backlogged, one for each flow. Of the remaining $N - f$ nodes, imagine that each flow randomly travels over l nodes. Hence,

$$Pr\{node\ not\ backlogged|f\ flows\} = \left(\frac{N - f - l}{N - f}\right)^f \tag{20}$$

Given that f nodes are backlogged as sources of the f flows and the remaining nodes are backlogged according to one minus the probability in the above equation, we have that

$$m = f + (N - f)\left\{1 - \left(\frac{N - f - l}{N - f}\right)^f\right\} \tag{21}$$

This functional relationship between m and f shows that the network soon becomes saturated as the number of flows are randomly increased within the network. In fact, our simulation studies, indicate that this occurs when $i(t)$ is much less than 0.2.

Figure 22 shows a high level view of the network in this saturated state. We now analyze the relationship of the per flow throughput across the saturated network versus increased number of flows in terms of a simple flow model. The figure shows f flows entering the network on the left hand side and exiting the network on the right hand side. In the saturated

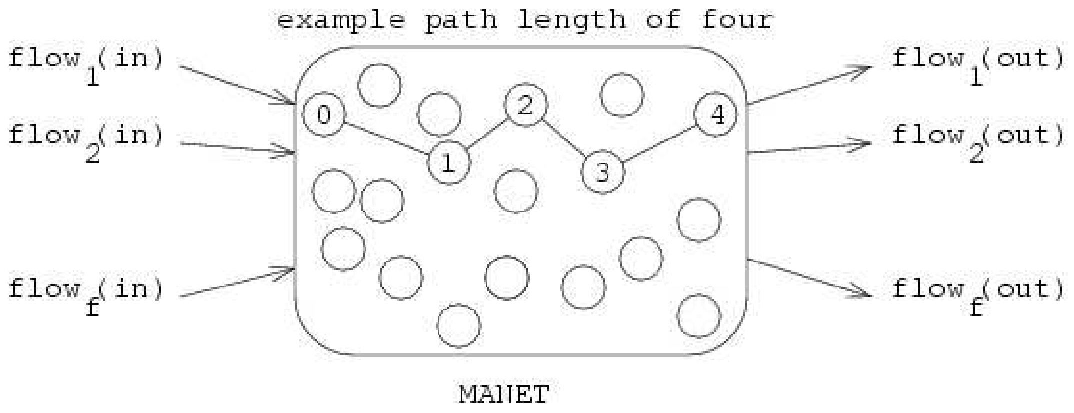


Figure 22. A stationary network flow model to derive the TCP throughput dependence upon number of flows.

state, the network is characterized by p, the per node packet loss rate which is independent of f in the saturated state. Furthermore, we define u as the per node packet processing rate, where packet processing includes both the time to successfully transmit a packet to the next node as well as the time to fail to transmit a data packet because of a failure to gain access to the channel and hence ending up discarding the packet. In the saturated network state, u is also independent of the number of active flows in the network.

Finally, because network routing is independent of network load, we know that the mean path length, l, is also independent of the number of flows. So the quantities p, u and l, characterizing aspects of the network performance are independent of the number of flows when the network reaches the saturated state.

Let $f_{r,in}$ be the average flow rate into the network for a single flow, and let $f_{r,out}$ be the average flow rate out of the network for a single flow. Given that each flow traverses (on average) l hops in the network and that each node has an average packet loss rate of p which is independent of the number of flow, we have that

$$f_{r,out} = f_{r,in}(1-p)^l \tag{22}$$

Therefore, the per flow loss rate is

$$p_{flow} = f_{r,in} - f_{r,out} = f_{r,out}\left(\frac{1}{(1-p)^l} - 1\right) \tag{23}$$

Equating the total network packet loss rate for N nodes and for f flows we get

$$N \times u \times p = f \times p_{flow} \tag{24}$$

or

$$f_{r,out} = \frac{up(1-p)^l}{i(t)(1-(1-p)^l)} \tag{25}$$

where we have used the fact that $f = N \times i(t)$.

4.3.2.1 Analytic Results

Previously we argued that β in the Standard Epidemic Model (SEM) was given by τ_{tcp}/P_w, where τ_{tcp} is the time dependent TCP throughput and P_w is the fixed worm payload size. When the 802.11 network is in the saturated state we have derived an expression for the flow rate per TCP flow in the network, Eq.(25) above. Assuming this is roughly equal to the TCP throughput, we get the following result for $\beta(t)$,

$$\beta(t) = \frac{\tau_{tcp}}{P_w} = \frac{up(1-p)^l}{P_w i(t)(1-(1-p)^l)} = \frac{\gamma_{sat}}{i(t)} \tag{26}$$

and the corresponding SEM becomes

$$\frac{di(t)}{dt} = \gamma_{sat}[1 - i(t)] \tag{27}$$

This expression has a simple solution, given by

$$i_s(t) = 1 - (1 - i^*)e^{-\gamma_{sat}t} \tag{28}$$

where we have defined (or labeled) $i_s(t)$ as the evolution of the probability of worm infection when the network is in the saturated state, and we have indicated the solution's initial condition as i^*, which we also define as the transition point between the low load network behavior and the network moderate load behavior.

4.3.3 Combined Results

We now combine our results for the low load and the moderate to high load regimes. Eq.(10) for the low load regime was

$$\frac{di(t)}{dt} = \gamma_{low} i(t)\left[\frac{1-i(t)}{1+\theta i(t)}\right] \qquad for\ i < i^* \tag{29}$$

Eq.(27) for the moderate to high load regime was

$$\frac{di(t)}{dt} = \gamma_{sat}[1 - i(t)] \qquad for\ i > i^* \tag{30}$$

Let us define i^* as the value of the infection probability where the transition from low load behavior to saturated network behavior occurs. Of course, in reality this is a smooth transition, but for our purposes we assume a specific transition value for the infection probability. We can pick the transition point, i.e., i^*, as a fitting parameter which is chosen to best fit the simulation data. Alternatively, we could specify that the transition point occurs where the respective β values from the two models are equal. That is, we define the β value such that

$$\beta \approx min\{\beta_{low}, \beta_{sat}\} \tag{31}$$

and the transition value occurs when $\beta_{low} = \beta_{sat}$ or

$$\frac{\gamma_{low}}{1 + \theta i^*} = \frac{\gamma_{sat}}{i^*} \tag{32}$$

This yields

$$i^* = \frac{\gamma_{sat}}{\gamma_{low} - \gamma_{sat}\theta} \tag{33}$$

We will consider the first approach in our comparisons to simulation results below.

Note, our combined TCP model makes the following assumptions:

- A single threaded TCP operation.
- A throughput model for the time to transmit the TCP worm payload, which assumes a large payload in relation to the TCP segment size in the network.
- Sufficient nodal density and radio transmission range to maintain connectivity across the MANET cluster.
- No background traffic.

It is relatively straightforward to incorporate the effects of multi-threaded TCP operation. Work to relax the assumption of large payloads may not be extremely useful because we

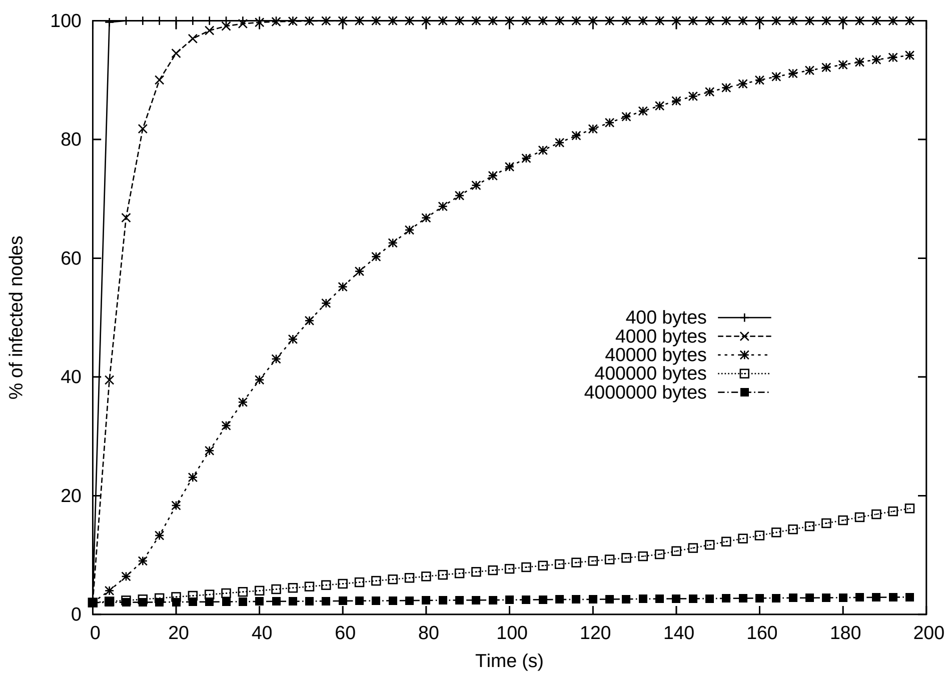
Table 4. Parameter definition for simulation experiments.

Parameter Description	*Range*	*Base Case*
Number of hosts	50-150	50
Initial population size	1-20	1
Transport layer protocol	TCP	TCP
Simultaneous connections	1	1
Range of vulnerable Addresses	50 - 500	50
Transmission Rate (Mbps)	0.1 - 2.0	2.0
Time delay(μ seconds)	2	2
Transmission range (m)	100 - 500	250
Area of topology (m^2)	1000	1000
Payload size (Kbytes)	0.4 - 4000	4
Simulation time (seconds)	200	200

would expect that most TCP worms have relatively large payloads. It may be possible to incorporate topological effects into our models to account for probabilities of island formation at low densities and or small radio range, although we have not investigated this to date. We plan to investigate improvements to our model relaxing these assumptions in future studies.

4.3.4 Numerical Results

Even though a general, explicit solution to Eqss (29) and (30) is not known, it is certainly easy to numerically integrate these expressions for $i(t)$. Therefore, we wrote a PERL script to numerically integrate Eqs. (29) and (30). We integrated the expression for $i(t)$ for a number of cases, based upon the parameter set given in Table 4.3.3. These correspond to a set of simulation studies discussed in the next section. For our purposes, Eqs. (29) and (30) contain several fitting parameters, i.e., α, γ_{high} and i^*. The parameter α is interpreted as the proportion of available channel bandwidth that a TCP connection is able to obtain under zero load situations. It is well known that TCP throughput decreases exponentially as a function of the number of hops in the wireless network due to self interference [56]. In fact in typical situations, it is not unusual to find that $\alpha < 0.1$. We choose α in order to fit the simulation results for the smaller load cases and found that a value of $\alpha = 0.068$ does a good job. The parameter i^* represents the probability of infection value where the



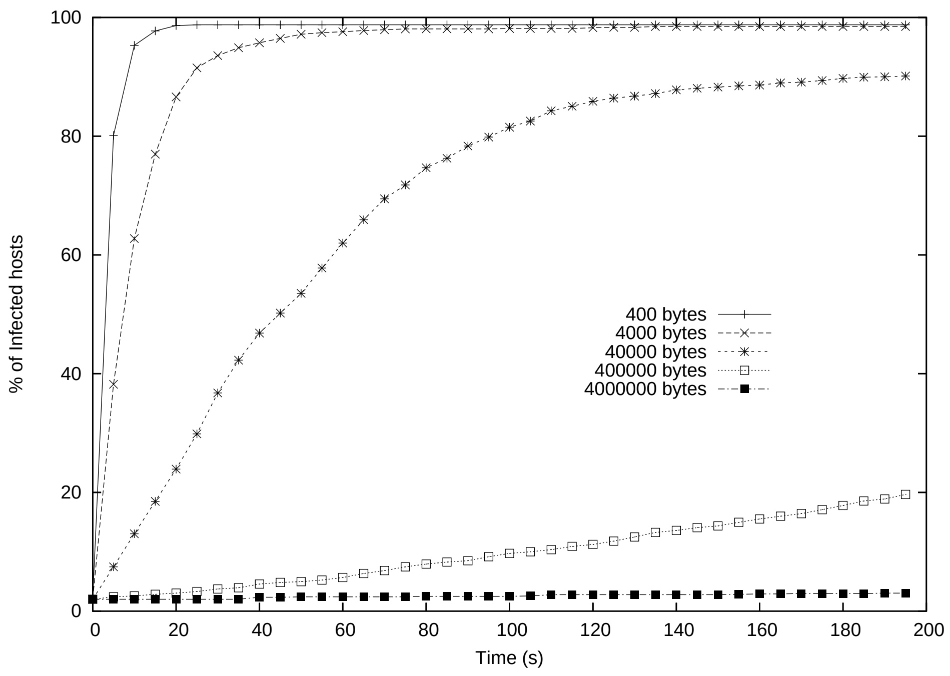


Figure 23. The TCP Model results (top) for various worm payload sizes compared with simulation results (bottom).

wireless network transitions from the low load to the moderate to high load state. From the simulation results on packet discards and propagation results, we find that $i^* = 0.1$ is a reasonable value for this parameter. Finally, the parameter γ_{high} is a fairly complex function of nodal packet processing and transmission rate, nodal packet discard probability and mean path length in the MANET, see, i.e., Eq.(26). Hence we varied this parameter until we found reasonable fit to the simulation results discussed below. We found that $\gamma_{high} = 600$ gave reasonable fitting results. We used these values below in our comparison of modeling to simulation results.

As an example, Figure 23 shows the results of the TCP worm model for various TCP worm payload sizes, ranging from a low of 400 bytes to a high of four million bytes. For each run, the other parameters for the model, as identified in Table 4.3.3 remained fixed. It can be seen that the model does a good job in qualitatively representing the simulation results. We discuss in detail below the simulation modeling and results.

4.4 Simulation

In this section, we present the simulation experiments conducted using the *Georgia Tech Network Simulator* (*GTNetS*).

GTNetS has an application that models the spread of a computer worm. The worm is designed as an application that exists on all susceptible nodes, which is listening on a specific port for incoming packets. When the worm application receives the infectious packet it is activated and starts choosing targets to send infectious packets to them.

There are different models for worms as discussed in [57]. The models include a number of parameters that specify the behavior of the worm :

- *Transport protocol*: The underlying transport protocol used by the worm, which can be either UDP or TCP. UDP worms do not wait for any acknowledgment from the target, while the TCP worms require a three–way handshake (*SYN/SYN–ACK/ACK*) before it can send its payload.

- *Infection length*: The size of the exploitation data that the worm needs to send to a host in order to infect it.
- *Infection port*: The transport layer port that exhibits the security vulnerability that is to be exploited.
- *Target vector*: The algorithm used by the worm to determine the IP address of a new victim. This can be either uniform, local preference or sequential scanning.
- *Scan rate*: The rate at which UDP worms send infection packets.
- *Scan range*: The range of addresses the worm chooses from.
- *Connections*: The number of simultaneous connections attempts used by TCP worms.

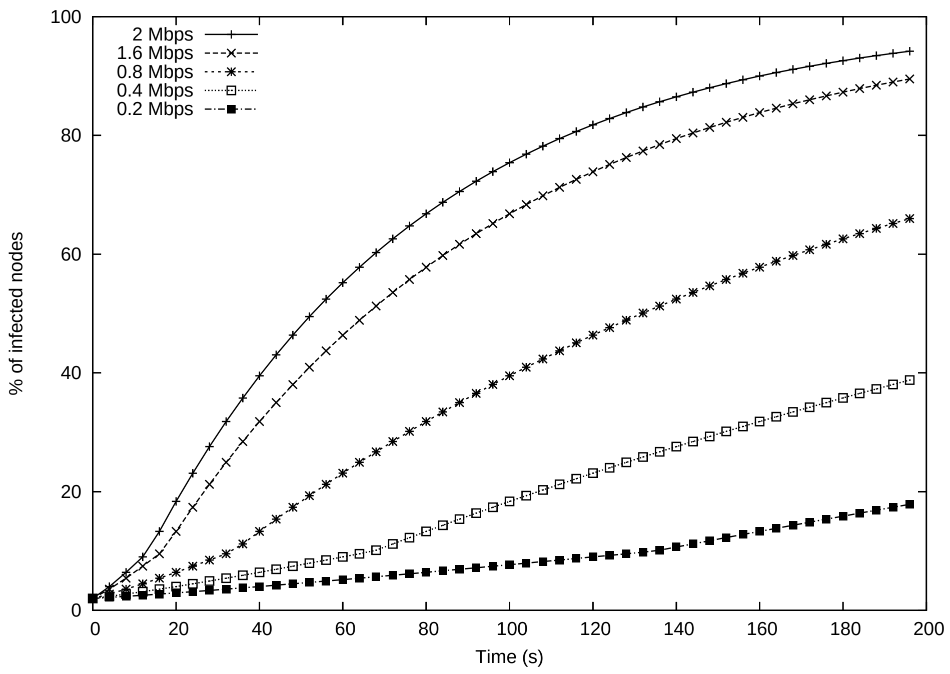
The MANET nodes are initially arranged in a rectangular grid, where they are uniformly placed across the grid. For our mobility studies, the Random Waypoint model was used to describe the motion of the nodes during the propagation of the TCP-based worm through the MANET.

Table 4.3.3 defines our baseline MANET TCP worm simulation model parameters.

4.5 Results

In our experiments we create the MANET topology and set the simulation parameters according to Table 4.3.3 baseline case. We then start the worm infection in the initial population and measure its spread against time for varying one parameter at a time and compare the average of the results of 30 runs with the output of the TCP worm model described in Section 4.3.

Figure 23 shows the results of varying the TCP payload size. Here it is clear that for payload sizes in excess of 4 Kbytes, the worm propagation soon congests the capacity of the radio links and the rate of spread decreases dramatically. Also apparent is the fact that for small payloads, the time for the worm to overrun the entire network is extremely



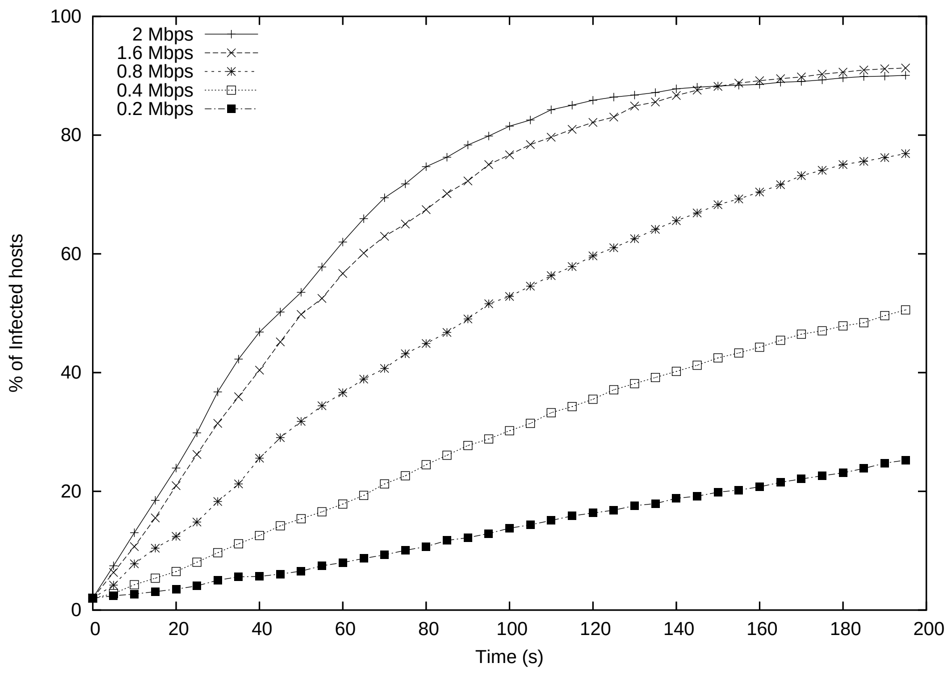


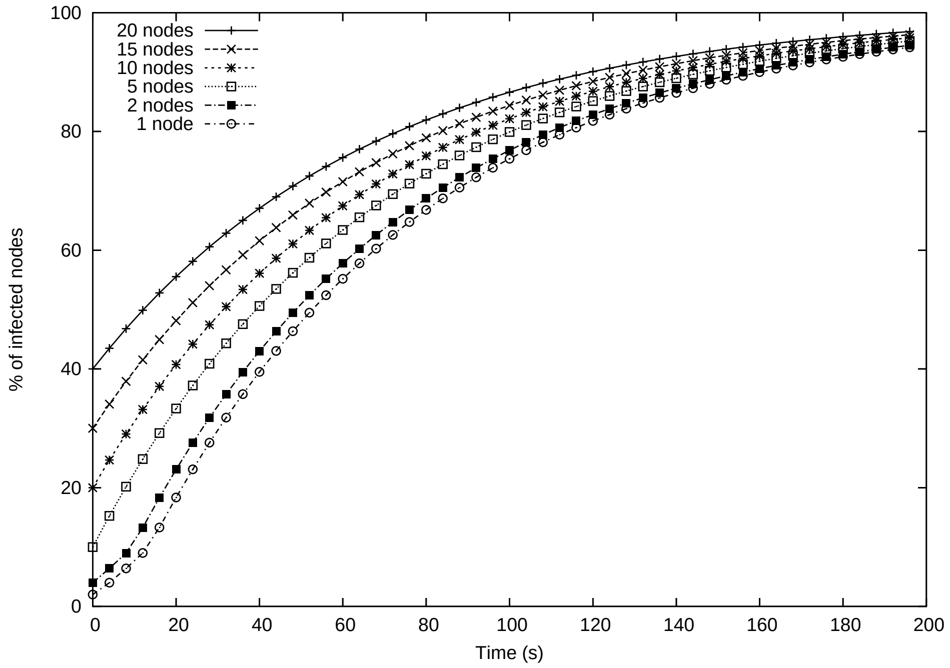
Figure 24. The TCP Model results (top) for various transmission rates with simulation results (bottom).

short. This is one of the reasons the DARPA program in [49] was so concerned about worm attacks against tactical MANETs. As mentioned before, the analytic modeling compares well with the simulation results shown in Figure 23.

Figure 24 shows the results of the TCP worm model and simulations after varying the transmission rate of the wireless channel, ranging from 0.1 Mbps to 2.0 Mbps. It is clear from the figures that as the transmission rate decreases the worm spread flattens due to saturation of the network and the infection growth tends to become linear. The model does a very good job in qualitatively representing the simulation results.

Figure 25 shows the results of the TCP worm model and simulations after varying the initial infected population size, ranging from 1 to 20 nodes. The model does a good job in qualitatively representing the simulation results. The results show as expected that with increasing the initial population, the worm spread rate is increased. The increase is not as significant as might be expected because this is a low scanning worm (only one simultaneous connection).

Figure 26 shows the results of the TCP worm model and simulations after varying the radio range of the nodes, ranging from 100 meters to 500 meters. It is clear from the simulation results that for low radio ranges some of the nodes can not communicate with each other and therefore the final infection probability is very low. To confirm this, in Figure 27 we show the topology plots of a set of random node placements in our 1000 by 1000 meter grid for various values of the radio range. In these plots, the circles around a node have a radius equal to one half the value of the radio range. Hence, two nodes have direct radio contact if their circles overlap. Here we clearly see that for radio ranges less than 250 m, there is a high probability that the network breaks up into smaller, disconnected clusters. As the radio range increases to 250 meters the final infection probability improves, afterwords the increase in radio range has a negative effect on the worm spread due to more bandwidth competition between nodes. The TCP worm model captures the effect of bandwidth competition and packet discards for topologies where the network remains



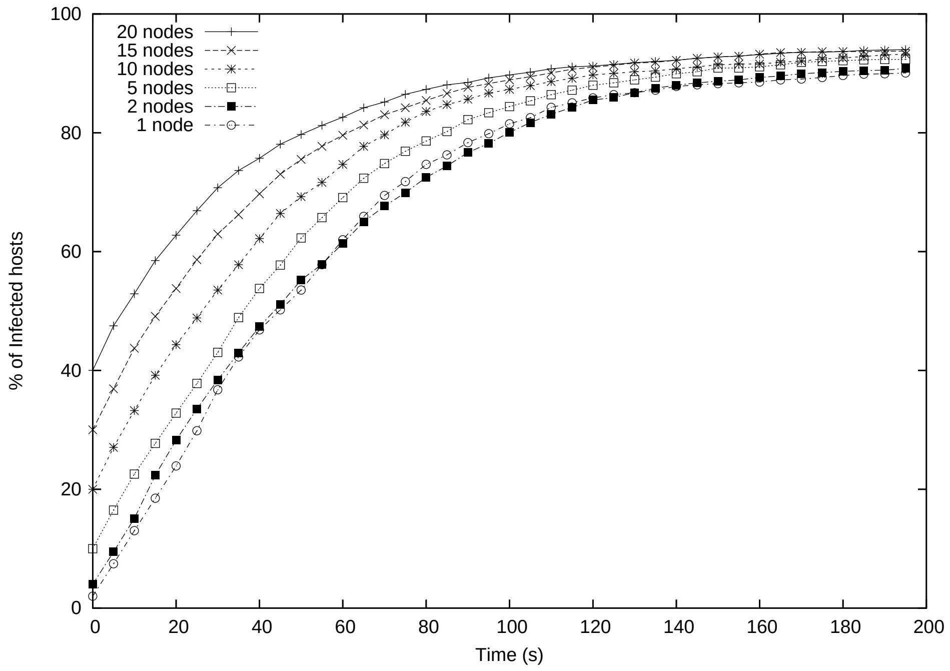
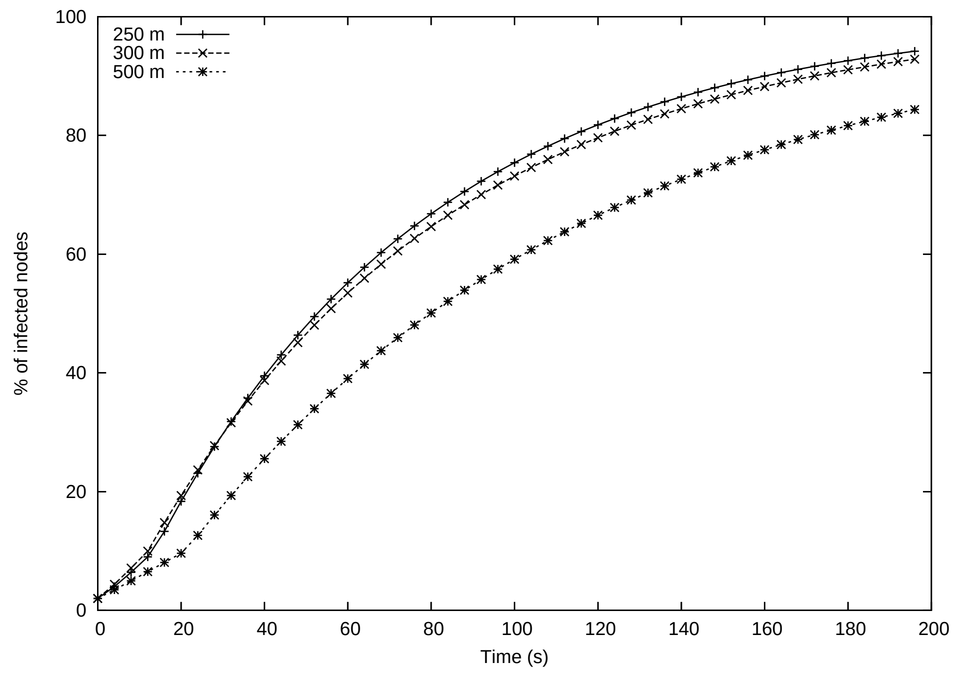


Figure 25. The TCP Model results (top) for various initial population size with simulation results (bottom).



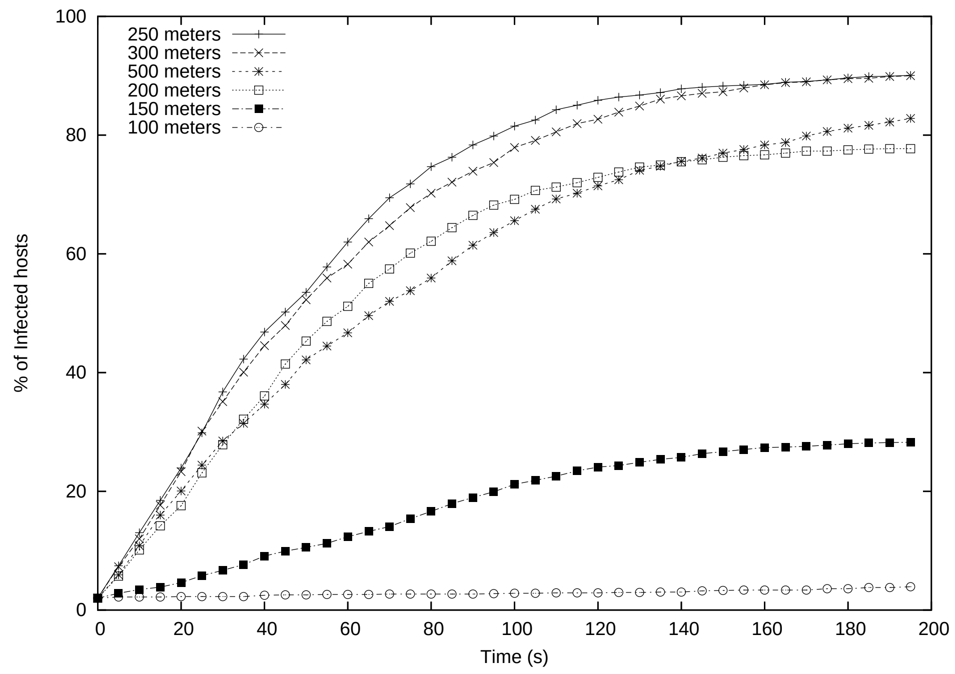


Figure 26. The TCP Model results (top) for various radio ranges with simulation results (bottom).

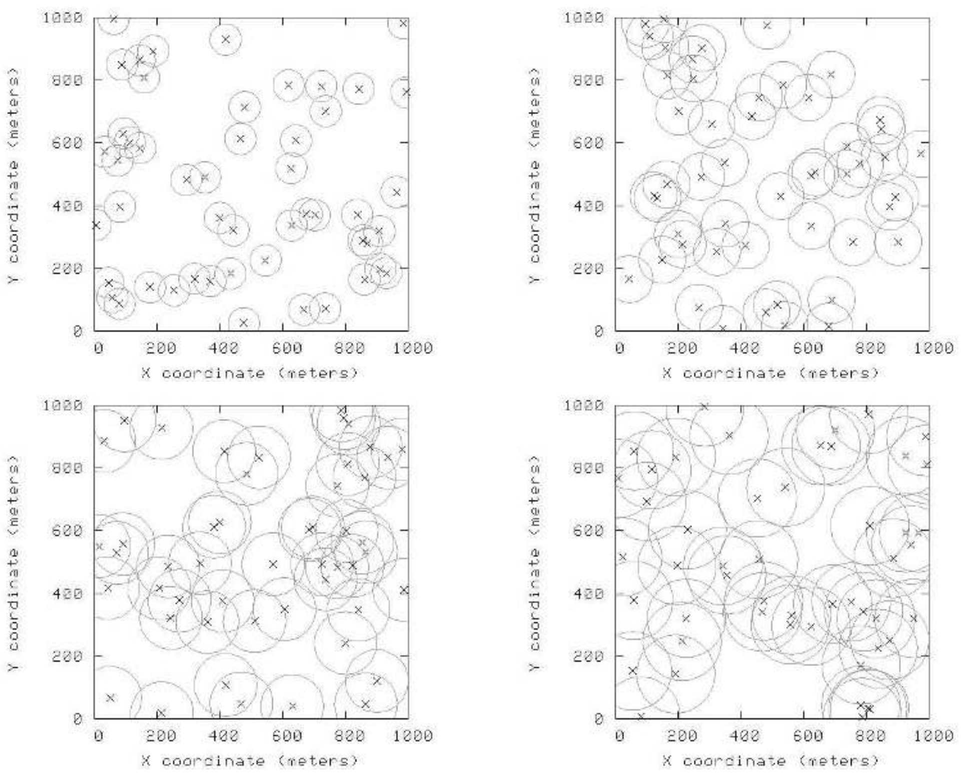
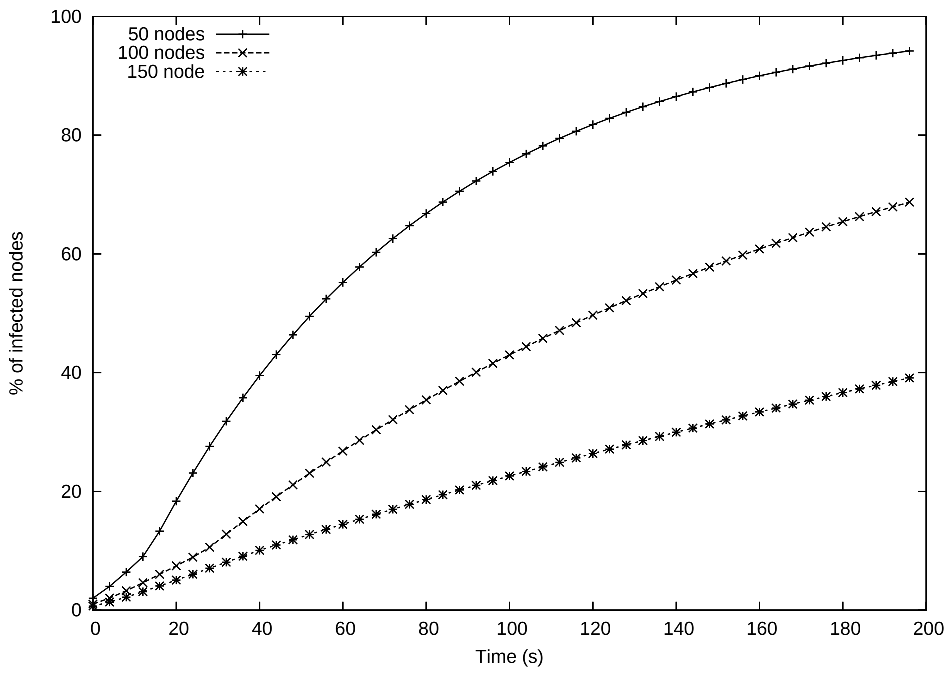


Figure 27. MANET topologies for various values of the radio range. The upper left plot results from $r = 100m$**, the upper right from** $r = 150m$**, the lower left from** $r = 200m$ **and the lower right from** $r = 250m$**.**

connected with a high probability. But the model breaks down at low radio ranges where connectivity begins to break down.

Figure 28 shows the results of the TCP worm model and simulations after varying the number of nodes (nodal density). The simulation results show the effect of increased contention with increasing nodal density, which results in an increase in the packet drop rate.

Figure 29 shows the results of the simulations after varying the routing protocol used. Three different routing protocols were used in these experiments; DNVR [58], DSR [59], and AODV [60]. It is clear from the figures that for our experiments there is no significant difference in the performance of these routing protocols. We did not provide a comparison to the analytic model for this case. This is because there is nothing in the analytic modeling that would suggest significant differences between the performance of the various routing protocols.



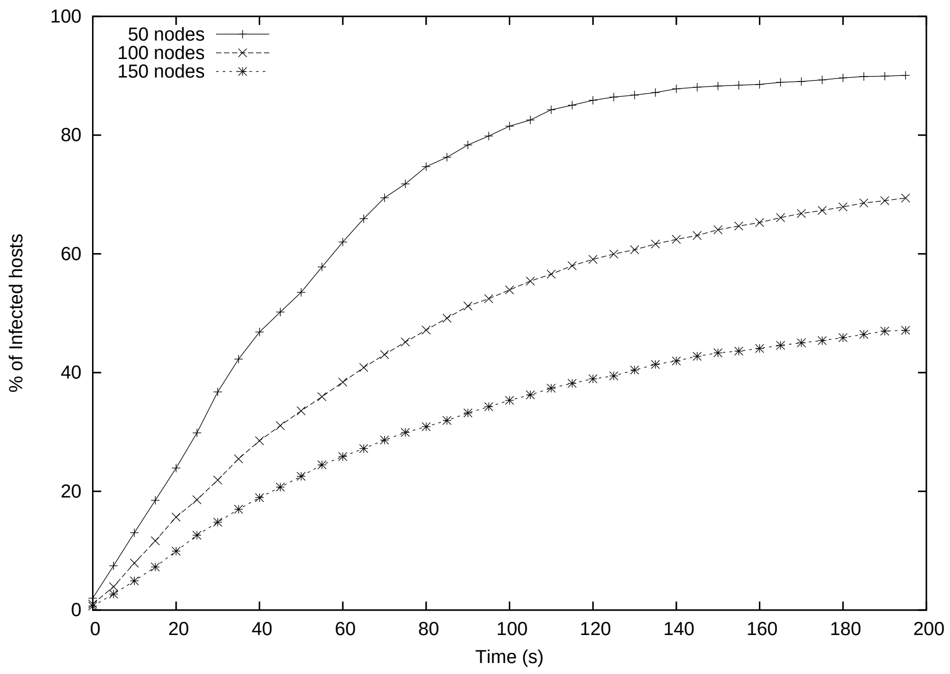


Figure 28. The TCP Model results (top) for various number of nodes with simulation results (bottom).

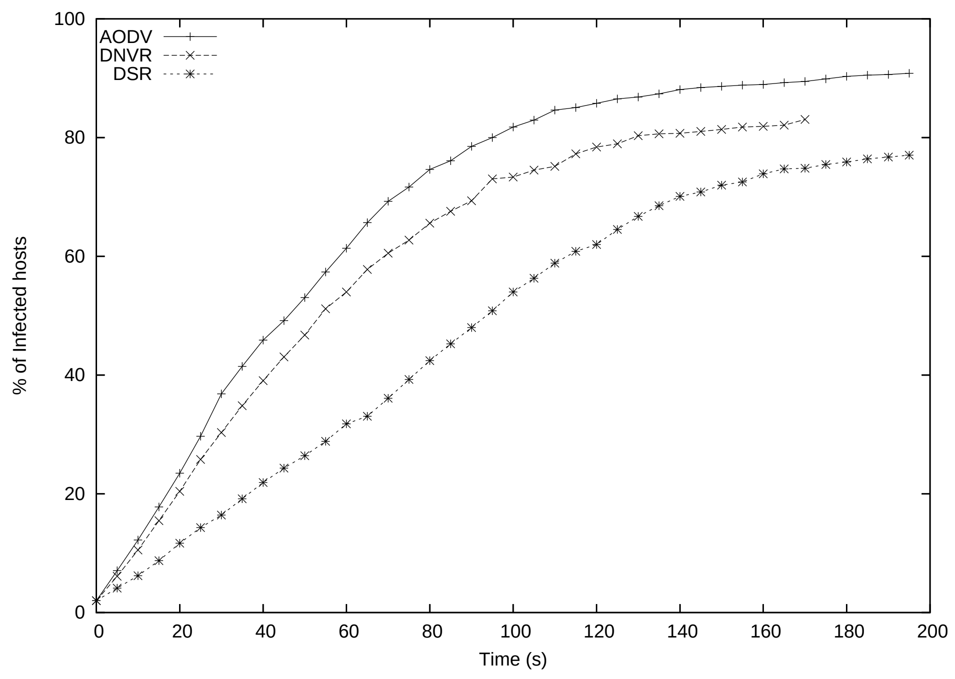


Figure 29. The simulation results for varying the routing protocol

4.6 Conclusion

We have presented a study of TCP worm propagation in MANETs. We investigated the impact of payload size, channel bandwidth, initial infection probabilities, packet discards due to collisions in the wireless channel, radio range and routing protocols on the effectiveness of the worm propagation. Previous studies have proposed analytical models of UDP-based worm propagation in MANETs. We developed an analytical model of TCP-based worm propagation in MANETs and showed that the model compares well to our simulation results. The model captures the effects of variable payload sizes (for large payloads), channel bandwidths, initial conditions, and radio range. The model does not include topological considerations and hence does not predict situations where the MANET becomes disconnected due to either low nodal densities or short radio ranges.

The model relies on the fact that the wireless network exists in one of two states; a low load state where packet discards are rare and the TCP flows share channel bandwidth and a moderate to high load state where the nodal packet discard probability is flat with respect to increases in the number of flows. This effect was discovered by Fu, et al., [50], and is confirmed by our simulation results.

CHAPTER 5
CONCLUSION AND FUTURE WORK

This thesis provides contributions in the field of modeling and simulations of worm traffic and mitigation algorithms. We first evaluated several proposed mitigation algorithms and introduced our own mitigation technique that addressed some of the limitations of these algorithms. The second contribution was the design of the Compact Flash worm that can saturate a one million node topology in less than 400 milliseconds. The third contribution was the development of an analytical model for representing TCP worm traffic in a MANET environment.

5.1 Evaluation of Contemporary Worm Defense Strategies

In chapter 2 we introduced simulation models that can represent several mitigation techniques. We used these models in *GTNetS* to study a number of the proposed mitigation algorithms. Our study is different from other studies in that we take into account the effect of the mitigation algorithms on normal background traffic. The idea is that a successful mitigation algorithm should not stop or delay in any way normal traffic.

We studied four different mitigation algorithms, namely: Virus Throttle, CounterMalice, Packet Matching and DAW. Our results showed that rate limiting algorithms (Virus Throttle and CounterMalice) are very effective in stopping worm traffic, however they also have a noticeable effect on normal traffic. The DAW algorithm proved to be ineffective in stopping our worm traffic and yet it caused significant delays on normal traffic. The Packet Matching algorithm performed well on slow scanning worms with no noticeable effect on normal traffic. One disadvantage that all these algorithms have is the high cost of deployment, whether this cost is represented by the number of machines running the algorithm or the amount of processing/memory the algorithm requires. As an example the Virus Throttle approach is a host based approach, which means it has to be implemented on every host on the network to have the desired results of stopping the worm. Implementing the same algorithm on the network level solves the problem of deployment, but it introduces higher complexity and exponential increase in memory requirement. Another disadvantage is that all of those algorithms are reactive, meaning that they have to detect the worm presence before they can start their limiting actions. This is in fact a necessity for those detection algorithm so as not to affect normal traffic. However, the time required for detection gives a fast worm ample opportunity to saturate the network.

To solve these problems a stateless algorithm is needed, which means its memory requirement is minimal. The algorithm must be 0day ready, i.e. a proactive solution that slows down worm scans from the first attempt. These requirements are met with our TCP-ACK algorithm. The TCP-ACK algorithm requires that any host receiving a TCP SYN packet for a non-existent port to unconditionally send a SYN–ACK to the originator, indicating that the connection has been accepted. The originator will then begin sending to the same destination port data packets, which are silently dropped. This means that the originator will consider the dropped packets as lost in transmission and will keep on retransmitting them for a long time before dropping this connection. Even if the originator is using multiple TCP threads for scanning, the TCP-ACK would significantly limit the number of threads, causing the worm spread to slow down considerably. The TCP-ACK algorithm also requires high deployment ratios to be effective, but this does not represent any significant cost because of its stateless nature. The TCP-ACK can be easily installed on all hosts on the network by updating the protocol stack using the security update mechanisms already in place for existing operating systems.

We believe that our studies will be very useful for the evaluation of proposed mitigation techniques under various network conditions. Such studies can not be done using analytical models as the complexity of the various network parameters makes it very hard to capture using simple equations.

Our research was addressing campus sized networks in the order of few thousands nodes. We did have several large-scale experiments but only upto 50 thousand nodes. The level of detail in our simulations prevented us from scaling to Internet size topology. To address the scaling issue, the study of fluid models could prove to be very useful for representing background traffic [61, 62, 63, 64]. Another possible extension to this research is the study of a more diverse representation of background traffic in the Internet, i.e. peer-to-peer, voice, and other forms of traffic that are emerging alongside web traffic. Another area of research is a better representation of the topology of the Internet.

5.2 Design of Worst Case Scenarios

In chapter 3 we studied one of the fastest known worms, namely the flash worm. We also proposed a modification to the address list representation that would make it even faster. We called the new version of the worm CFlash or Compact Flash worm. The CFlash worm has the same structure as the flash worm except that the list of IP addresses is represented by the relative offsets rather than the actual IP addresses. We have developed simulation experiments for a million node topology with full packet-level detail and tested the CFlash worm under various network conditions. Our studies showed that such a worm can infect a one million-node topology in under 400 millisecond. This alarming speed shows that no human response can stop this worm in time. This also presents a tight requirement for response time in automated defenses. We believe the only hope for stopping such a worm is in the list formation phase, which requires collaboration between various anomaly detectors on the Internet to signal an alarm whenever a strong correlation is found regarding suspicious source IP addresses.

Several areas need more careful consideration:

- The effect of having a more realistic topology representing the Internet.
- The effect of containment strategies attempting to delay or stop the worm spread.
- The effect of background traffic.

- Studying real worm outbreak cases and comparing the effect of having a CFlash worm instead in those cases.

5.3 Modeling and Simulations of MANET Worms

In chapter 4 we studied TCP worm propagation in a MANET environment. We have developed an analytical model which closely matches our detailed simulations results. The model captures the effects of variable payload sizes, channel bandwidths, initial conditions, and radio range.

The model relies on the fact that the wireless network exists in one of two states; a low load state where packet discards are rare and the TCP flows share channel bandwidth and a moderate to high load state where the nodal packet discard probability is flat with respect to increases in the number of flows. We have developed models for each of these two states and combined them to produce the final model. The model was compared to simulation results for a number of experiments with different payload sizes, radio ranges, and bandwidth settings. All the simulation results compares very well to the model results except for very low radio range or low nodal densities, which results in having disconnected regions in our topology. One extension to this research is to relax some of the assumptions of the analytical model to account for probability of island formation due to low radio range or low nodal density. Another possible extension is the study of the effect of normal background traffic on our model. Also, the simulation part can show the effect of mobility on the worm performance, for this purpose, the modified random waypoint model [65] for mobility needs to be implemented in *GTNetS* .

We believe that our studies will aid in the design of efficient counter measures for worm attacks in MANETs. Further studies of worm propagation and mitigation in the challenging networking environment afforded by MANETs is required. In future studies we hope to further quantify the behavior of TCP-based worms and to investigate the efficiency of specific worm mitigation technologies.

REFERENCES

[1] D. Seeley, “A tour of the worm,” in *Proceedings of the winter USENIX Conference*, (San Diego, CA, USA), pp. 287–304, January 1989.

[2] D. Moore, C. Shannon, and J. Brown, “Code-red: a case study on the spread and victims of an internet worm,” in *Proceedings Internet Measurement Workshop (IMW)*, (Marseille, France), November 2002.

[3] E. J. Aronne, “The nimda worm: An overview,” tech. rep., SANS, October 2001.

[4] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, “Inside the slammer worm,” *IEEE Magazine of Security and privacy*, vol. 1, pp. 33–39, July/August 2003.

[5] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham, “A taxonomy of computer worms,” in *Proceedings of the 2003 ACM CCS workshop on Rapid Malcode (WORM’03)*, (Washington, DC, USA), pp. 11–18, 2003.

[6] “W32/gnuman.worm.”

[7] M. Kern, “Re: Codegreen beta release.”

[8] V. P. Stuart Staniford and N. Weaver, “How to 0wn the internet in your spare time,” in *Proceedings of the 11th USENIX Security Symposium*, (San Francisco, CA, USA), August 2002.

[9] D. Moore, C. Shannon, G. M. Voelker, and S. Savage, “Internet quarantine: Requirements for containing self-propagating code,” in *Proceedings IEEE INFOCOM*, (San Francisco, CA, USA), March 2003.

[10] J. O. Kephart and S. R. White, “Measuring and modeling computer virus prevalence,” in *Proceedings of the 1993 IEEE Symposium on Security and Privacy*, (Washington, DC, USA), p. 2, 1993.

[11] H. W. Hethcote, “The mathematics of infectious diseases,” *SIAM Review*, vol. 42, pp. 599–653, October 2000.

[12] C. C. Zou, L. Gao, W. Gong, and D. Towsley, “Monitoring and early warning for internet worms,” in *CCS ’03: Proceedings of the 10th ACM conference on Computer and communications security*, pp. 190–199, ACM Press, 2003.

[13] G. F. Riley, M. I. Sharif, and W. Lee, “Simulating internet worms,” in *Proceedings of The IEEE Computer Society’s 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS’04)*, (Volendam, The Netherlands), pp. 268–274, October 2004.

[14] G. F. Riley, “The Georgia Tech Network Simulator,” in *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pp. 5–12, ACM Press, 2003.

[15] A. Wagner, T. D. bendorfer, B. Plattner, and R. Hiestand, “Experiences with worm propagation simulations,” in *WORM’03: Proceedings of the 2003 ACM workshop on Rapid Malcode*, pp. 34–41, ACM Press, 2003.

[16] M. Liljenstam, D. M. Nicol, V. H. Berk, and R. S. Gray, “Simulating realistic network worm traffic for worm warning system design and testing,” in *WORM’03: Proceedings of the 2003 ACM workshop on Rapid Malcode*, pp. 24–33, ACM Press, 2003.

[17] J. Cowie, A. Ogielski, and D. Nicol, “The SSFNet network simulator.” Software online: http://www.ssfnet.org/homePage.html, 2002. Renesys Corporation.

[18] H.-A. Kim and B. Karp, “Autograph: toward automated, distributed worm signature detection,” in *SSYM’04: Proceedings of the 13th conference on USENIX Security Symposium*, (Berkeley, CA, USA), pp. 19–19, USENIX Association, 2004.

[19] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham, “Vigilante: end-to-end containment of internet worms,” in *SOSP ’05: Proceedings of the twentieth ACM symposium on Operating systems principles*, (New York, NY, USA), pp. 133–147, ACM Press, 2005.

[20] S. P. Chung and A. K. Mok, “Allergy attack against automatic signature generation.,” in *RAID* (D. Zamboni and C. Krgel, eds.), vol. 4219 of *Lecture Notes in Computer Science*, pp. 61–80, Springer, 2006.

[21] B. M. J. W. L.Todd Heberlein, Gihan V. Dias. Karl N. Levitt and D. Wolber, “FA Network Security Monitor,” in *1990 IEEE Symp. Research in Security and Privacy*, (Oackland, California), pp. 296–304, May 1990.

[22] J. Twycross and M. M. Williamson, “Implementing and testing a virus throttle,” in *Proceedings of the 12th USENIX Security Symposium*, (Washington, D.C., USA), pp. 285–294, August 2003.

[23] C. Wong, C. Wang, D. Song, S. Bielski, and G. R. Ganger, “Dynamic quarantine of internet worms,” in *DSN ’04: Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN’04)*, (Washington, DC, USA), p. 73, IEEE Computer Society, 2004.

[24] S. Staniford, “Containment of scanning worms in enterprise networks,” *Journal of Computer Security*, to appear 2004.

[25] D. Whyte, E. Kranakis, and P. C. van Oorschot, “DNS-based detection of scanning worms in an enterprise network,” in *Proceedings of the 12th Network and Distributed System Security Symposium (NDSS)*, pp. 181–195, February 2005.

[26] P. C. v. O. David Whyte and E. Kranakis, “Detecting intra-enterprise scanning worms based on address resolution.”

[27] V. Paxson, “Bro: a system for detecting network intruders in real-time,” *Computer Networks (Amsterdam, Netherlands: 1999)*, vol. 31, no. 23–24, pp. 2435–2463, 1999.

[28] S. Schechter, J. Jung, and A. W. Berger, “Fast Detection of Scanning Worm Infections,” in *7th International Symposium on Recent Advances in Intrusion Detection (RAID)*, (French Riviera, France), September 2004.

[29] A. Wagner and B. Plattner, “Entropy based worm and anomaly detection in fast ip networks,” in *WETICE ’05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*, (Washington, DC, USA), pp. 172–177, IEEE Computer Society, 2005.

[30] S. Sellke, N. B. Shroff, and S. Bagchi, “Modeling and automated containment of worms.,” in *DSN*, pp. 528–537, 2005.

[31] S. McCanne and S. Floyd, “The LBNL network simulator.” Software on-line: http://www.isi.edu/nsnam, 1997. Lawrence Berkeley Laboratory.

[32] S. Bertolotti and L. Dunand, “Opnet 2.4: an environment for communication network modeling and simulation,” in *Proceedings of the European Simulation Symposium*, October 1993.

[33] “Intrusion detection and prevention: protecting your network from attacks.” http://www.ncasia.com/security/ NetScreen_IDP_WP.pdf.

[34] B. A. Mah, “An empirical model of http network traffic,” in *Proceedings of IEEE INFOCOMM*, pp. 592–600, 1997.

[35] X. Chen and J. Heidemann, “Detecting early worm propagation through packet matching,” Tech. Rep. ISI-TR-2004-585, USC/Information Sciences Institute, February 2004.

[36] S. Chen and Y. Tang, “Slowing down internet worms,” in *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS’04)*, (Tokyo, Japan), March 2004.

[37] T. Liston, “Labrea project,” 2001.

[38] S. Staniford, D. Moore, V. Paxson, and N. Weaver, “The top speed of flash worms,” in *WORM ’04: Proceedings of the 2004 ACM workshop on Rapid malcode*, pp. 33–42, ACM Press, 2004.

[39] J. Swartz, “Cops take a bite, or maybe a nibble, out of cybercrime,” *USA TODAY*, September 2003.

[40] R. Lemos, “Msblast epidemic far larger than believed,” *CNET News.com*, April 2004.

[41] C. Shannon and D. Moore, "The spread of the witty worm," tech. rep., CAIDA, April 2004.

[42] N. Weaver and V. Paxson, "A worst-case worm," in *Proceedings of the Third Annual Workshop on Economics and Information Security (WEIS'04)*, 2004.

[43] K. Poulsen, "Slammer worm crashed ohio nuke plant network, http://www.securityfocus.com/news/6767 (oct/2008)."

[44] N. Weaver, "Potential strategies for high speed active worms: A worst case analysis." http://www.cs.berkeley.edu/~nweaver/ worms.pdf, March 2002.

[45] R. Russell and A. Mackie, "Code red worm ii analysis report," tech. rep., SecurityFocus, August 2001.

[46] Netcraft, "Netcraft web server survey," *http://www.netcraft.com/survey, March/2005*.

[47] CAIDA, "Skitter datasets," *http://www.caida.org/tools/measurement/skitter, March/2005*.

[48] M. Abdelhafez and G. Riley, "Evaluation of worm containment algorithms and their effect on legitimate traffic," in *IWIA'05: Third IEEE International Workshop on Information Assurance*, pp. 33–42, March 2005.

[49] Ghosh and A.K., "How to 0wn the internet in your spare time," in *Proceedings of the 11th USENIX Security Symposium*, (San Francisco, CA, USA), August 2002.

[50] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla, "The impact of multihop wireless channel on tcp throughput and loss," in *IEEE Infocomm 2003*, August 2003.

[51] C.C.Zou, W.Gong, and D. Towsley, "Code red worm propagation modeling and analysis," in *In Proceedings of the 9th ACM Conference on Computer and Communications Security*, pp. 138–147, November 2002.

[52] R. G. Cole, "Initial studies of worm propagation in manets," (Orlando, FL, USA), 2004.

[53] R. G. Cole, N. Phamdo, M. A. Rajab, and A. Terzis, "Requirements on worm mitigation technologies in manets," in *PADS '05: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, (Washington, DC, USA), pp. 207–214, IEEE Computer Society, 2005.

[54] G. Holland and N. Vaidya, "Analysis of tcp performance over mobile ad hoc networks," in *Mobicom*, 1999.

[55] S. Papanastasiou, M. Ould-Khaoua, and L. Mackenzie, "On the evaluation of tcp in manets," in *Proceedings of the International Workshop on Wireless Ad Hoc Networks*, May 2005.

[56] D. D. perkins and H. D. Hughes, “Tcp performance in mobile ad hoc networks,” in *Proceedings of the Intl. Symp. on Perf. Eval. of Computer and telecommunications Systems*, July 2001.

[57] M. I. Sharif, G. Riley, and W. Lee, “Simulating internet worms,” in *Proceedings of the Twelvth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS’04)*, 2004.

[58] Y. Lee and G. Riley, “Dynamic nix-vector routing for mobile ad hoc networks.,” in *IEEE Wireless Communications and Networking Conference (WCNC 2005)*, Mar 2005.

[59] D. B. Johnson and D. A. Maltz, “Dynamic source routing in ad hoc wireless networks,” *Mobile Computing*, pp. 153–181, 1996.

[60] C. E. Perkins and E. M. Royer, “Ad hoc on-demand distance vector routing.,” in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pp. 90–100, Feb 1999.

[61] B. Liu, D. R. Figueiredo, Y. Guo, J. Kurose, and D. Towsley, “A study of networks simulation efficiency: Fluid simulation vs. packet-level simulation,” in *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM’01)*, April 2001.

[62] B. Liu, Y. Guo, J. Kurose, D. Towsley, and W. Gong, “Fluid simulation of large scale networks: issues and tradeoffs,” in *Proceedings of PDPTA’99*, June 1999.

[63] D. Nicol, M. Goldsby, and M. Johnson, “Fluid-based simulation of communication networks using ssf,” in *Proceedings of 1999 European Simulation Symposium*, June 1999.

[64] B. Melamed, S. Pan, and Y. Wardi, “Hybrid discrete-continuous fluid-flow simulation,” in *Proc. of the SPIE International Symposium on Information Technologies and Communications (ITCOM 01)*, Aug 2001.

[65] J. Yoon, M. Liu, and B. Noble, “Random waypoint considered harmful,” in *INFOCOM*, 2003.