Appears in 8^{th} International Symposium on Recent Advances in Intrusion Detection (RAID 2005)
September 7-9, 2005, Seattle, Washington

Empirical Analysis of Rate Limiting Mechanisms

Cynthia Wong, Stan Bielski, Ahren Studer, Chenxi Wang
Carnegie Mellon University

Abstract

One class of worm defense techniques that received attention of late is to "rate limit" outbound traffic to contain fast spreading worms. Several proposals of rate limiting techniques have appeared in the literature, each with a different take on the impetus behind rate limiting. This paper presents an empirical analysis on different rate limiting schemes using real traffic and attack traces from a sizable network. In the analysis we isolate and investigate the impact of the critical parameters for each scheme and seek to understand how these parameters might be set in realistic network settings. Analysis shows that using DNS-based rate limiting has substantially lower error rates than schemes based on other traffic statistics. The analysis additionally brings to light a number of issues with respect to rate limiting at large. We explore the impact of these issues in the context of general worm containment.

Keywords: Rate Limiting, Internet Worms, Worm Containment

1 Introduction

Fast-spreading worms such as Blaster [16], and SoBig [11] wreaked havoc on the Internet and caused millions of dollars in downtime and IT expenses. In addition to consuming valuable network and computing resources, worms provide potential vehicles for DDoS attacks, as seen in the case of SoBig and Blaster [11, 16]. The need to mitigate worm spread is apparent and pressing.

Researchers have proposed various techniques for worm defense, both in detection [7, 22, 9, 13] and response [23, 21, 1, 12, 4]. Automatic response techniques are of particular interest because methods that require human intervention simply cannot match the speed and voracity of modern day worms. One class of automated response techniques seeks to *rate limit* the outbound spread of worm traffic [23, 1, 12] while allowing the continued operation of legitimate applications. These rate limiting schemes offer a gentler alternative to the simple detect-and-block-the-host approach, and therefore are more palatable to actual deployment. A recent analytical study also showed that when deployed at appropriate points in the network, rate limiting can substantially reduce the spread of infection [25].

In this work, we undertake an empirical analysis of existing rate limiting mechanisms, with the goal of understanding the relative performance of the various schemes. Our study is based on real traffic traces collected from the border of a network with 1200 hosts. The trace data includes real attack traffic of Blaster and Welchia, which infected over 100 hosts. We implement each scheme against the trace data and analyze their performance in terms of false positive and false negative rates. In the case of worm defense, it is particularly important that false positives are kept at a minimum without greatly impacting false negatives.

We analyze the efficacy of the various schemes on both worm traces and normal traffic. The inclusion of real worm data allows us to draw insights without having to consider the limitations of simulated attacks. We study three rate limiting schemes, Williamson's IP throttling [23], Chen's failed-connection-based scheme [1] and Schechter's credit-based rate limiting [12]. Williamson's throttling scheme limits the rate of distinct IP connections from an end host [23]. Chen et al. [1] and Schechter et al. [12] both apply rate limiting to hosts that exhibit an abnormally high number of failed connections. In addition, we study an alternative rate limiting strategy based on DNS statistics—namely limiting outgoing connections without prior DNS translations, thereby restricting the contact rate of scanning worms. Ganger et al. made the first observation that DNS-based statistics can be used to detect and contain malicious worms [4]. Recently Whyte et al. showed that DNS-based worm detection can be extended to a network setting [22]. The DNS-based rate limiting mechanism we study is a modified version of [4]. One goal of this study is to investigate using DNS behavior as a basis for rate limiting and its relative performance with respect to other schemes.

In addition to studying DNS-based rate limiting, the other components of our analysis seek to understand the fundamentals of rate limiting technology. For instance, we evaluate the impact of dynamic vs. static rates. We study the effect of host vs. edge-based deployment. Some of these issues were not explored adequately in the studies of the individual schemes.

Our analysis is the first that we are aware of that offers

evaluation of the different rate limiting schemes on an equal footing—running against the same traffic traces. The trace data we use in this study is from an open network without strict traffic policies. Since most of the rate limiting mechanisms target enterprise networks with stricter traffic settings, we believe that our analysis provides reasonable insights into how well these schemes might perform in practice.

2 Related Work

The rate limiting schemes by Williamson et al. [23], Chen et al. [1], and Schechter et al. [12] are the target of our analysis. We defer discussions of these schemes to later sections of the paper.

Our work aims to provide a study of rate-limiting techniques as a defense against Internet worm propagation. Worm defense is a richly studied field; there exist many schemes outside rate limiting [21, 7, 9, 22, 18, 13, 3]. Some are complimentary to rate limiting at large, which can be combined in practice. For instance, the scan detection work by Weaver et al. [21] and Jung et al. [7] can be used to protect enterprise networks from incoming infections while rate limiting seeks to contain outbound propagations. Also of interest are the various forms of worm detection work [22, 13, 9, 3]. In this paper we choose to focus on analysis of automated response techniques. We find it beneficial to limit our discussion to a set of similar technologies so as to permit meaningful comparisons.

We note that there exists a rich body of worm modeling and analysis work [15, 26, 8, 19, 20, 10, 14] that offers theoretical understanding of and technical insights into worm defense. Our goal is not to study worm propagation in a broad sense, but rather we seek to evaluate and understand the impact and limitations of a particular defense strategy, rate limiting. We believe that rate limiting is a lightweight technique that can be readily deployed and administered, and therefore represents a promising defense strategy.

Our study is the first that offers a direct comparison of different rate limiting technologies, using real traffic and attack traces. The analysis part of our study is similar in spirit to the DDoS filter analysis by Collins et. al. [2], though the target of our analysis is different and therefore offers different insights and conclusions.

3 Trace Data

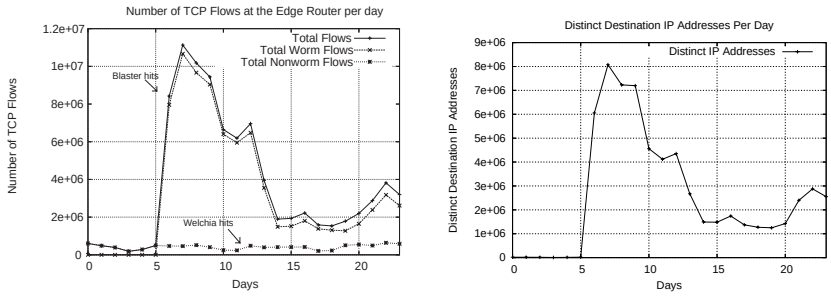
The study in this paper is conducted using traffic traces collected from the border of an academic department. The network has 1200 externally routable hosts and serves approximately 1500 users. Hosts are used for research, administration, and general computing (web browsing, mail, etc). There is a diverse mix of operating systems on the network. Since May 2003 we recorded in an anonymized form all IP and common second layer headers of packets (e.g., TCP or UDP) leaving and entering the network. We also recorded DNS traffic payloads for use in the experiment in Section 8.

During the course of tracing, we recorded two worm attacks: *Blaster* and *Welchia* [16, 17]. Both are scanning worms that exploited the Windows DCOM RPC vulnerability. For each attack recorded, we conducted post-mortem analysis to identify the set of infected hosts within the network. We further identified outbound worm traffic as those from infected hosts with a particular destination port (e.g., port 135 for Blaster). Whenever possible, a payload size identical or similar to those publicized in Symantec's worm advisories is used as additional evidence to identify worm traffic. It is important to note that infected hosts in our network were exclusively Windows clients that, under normal circumstances, rarely (if ever) made any outbound port 135 connections to external addresses. Once infected, these hosts initiated tens of thousands of outbound connections to port 135. As such, the task of identifying worm traffic is made relatively easy.

For the purpose of this analysis, we use a period of 24-day outbound trace, from August 6th to August 30th 2003. This period contains the first documented infection of Blaster in our network, which occurred on August 11th. Welchia hit the network on the 18th. Collectively, Blaster and Welchia infected 100 hosts in the network. Since hosts infected by Blaster and Welchia exhibited similar traffic patterns during the overlapping time period, we do not attempt to separate the two attacks. Our data suggests that residual effects of the worms lingered on for months but the effects of the infection are most prominent during the first two weeks of the attack.

Figure 1(a) shows the daily volume of outgoing traffic as seen by the edge router for the trace period. Figure 1(b) shows the number of distinct IP addresses daily. As shown, the aggregate outgoing traffic experienced a large spike as Blaster hits the network on day 6. At its peak, the edge router saw 11 million outbound flows in a day. This is in contrast to the normal 500,000 flows/day. The increase in traffic is predominantly due to worm activities.

Unless otherwise noted, the trace data refers to aggregate traffic as seen by the edge router. In some of the later analysis (e.g., Williamson's host-based throttling), we use host-level traffic from the aggregate trace. In



(a) Daily outgoing flows (b) Daily volume of distinct IPs

Figure 1: Traffic Statistics for the Blaster/Welchia Trace

those cases we will differentiate between infected host traffic and normal host traffic.

4 Analysis Methodology

As previously mentioned, we use a period of 24-day outbound traces collected at the border of a 1200-host network with documented Blaster and Welchia activities. Our goal is to evaluate the performance of various proposed rate limiting schemes. The performance criteria we use in the analysis is error rates (e.g., false positives and false negatives) of the different schemes. We define the false positive rate as the percentage of normal traffic misidentified as worm traffic and subsequently rate limited. False negative rate is the percentage of worm traffic that is not affected by the rate limiting mechanism and permitted through without delay. Rate limited traffic can be either blocked or delayed. In the analysis that follow, we differentiate between these two cases and present error rates accordingly. Note the false negative rate is only meaningful during infection, while false positives are considered throughout the entire trace period. Whenever appropriate, we present Receiver Operator Curves (ROC) to contrast false negatives with false positives.

For each scheme analyzed, there exists a set of parameters that impact the performance of the mechanism. We identify these parameters and evaluate the sensitivity of the error rates with respect to each parameter. In some cases, the impact of the parameters has not been studied previously. A contribution of our study is to understand precisely how these parameters might be implemented in practice.

One factor that we were unable to evaluate fully in our work was the placement of RL mechanisms within the network. Our trace does not include internal traffic and due to the anonymized nature of our trace data, we were unable to reconstruct the internal network topology.

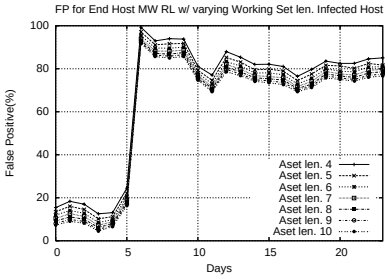
5 Williamson's IP Throttling

Williamson's IP throttling scheme operates on the assumption that normal applications typically exhibit a stable contact rate to a limited number of external hosts (e.g., web servers, file servers) [23]. Restricting host-level contact rates to unique IPs can limit rapid connections to random addresses (e.g., worm traffic). Williamson accomplishes this by keeping a *working set* of addresses for each host, which models the normal contact behavior of the host. The throttling mechanism permits outgoing connections for addresses in the working set, but delays other packets by placing them in a delay queue. If the delay queue is full, further packets are simply dropped. The packets in the delay queue are dequeued and processed at a constant rate (one per second, as suggested by [23]). At the same rate, the least recently used address in the working set is evicted to make room for the new connection. As a result, connections to frequently contacted addresses are allowed through with a high probability while connections to random addresses (as those initiated by scanning worms) are likely delayed and possibly dropped.

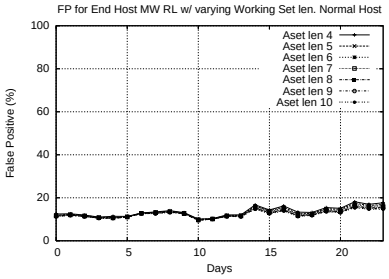
For this scheme, the size of the working set and the delay queue are important. A larger working set permits a higher contact rate while the delay queue length determines how liberal (or restrictive) the scheme is. Williamson recommends a five-address working set and a delay queue length of 100 for host-based implementations. Our analysis reports on the impact of these parameter settings. We also analyze a version of Williamson's throttling on the edge router.

End Host Throttling To analyze Williamson's end host IP throttling, we reconstructed end-host traffic from our trace and simulated Williamson's rate limiting scheme using these traces.

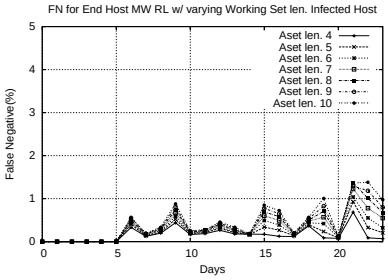
Figure 2(a) shows the daily false positive rate for infected hosts with the size of the working set ranging



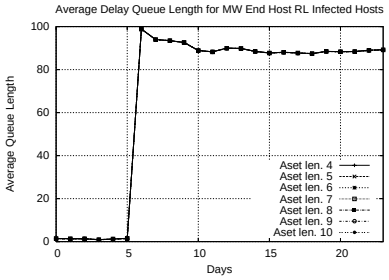
(a) FP per day for Infected Hosts



(b) FP per day for Normal Hosts



(c) FN per day for Infected Hosts



(d) Avg. Delay Queue Length(Infected Hosts)

Figure 2: Results for Williamson's End Host RL mechanism

from 4 to 10. Again, false positive rates are calculated as the percentage of benign traffic subjected to rate limiting. The data points in Figure 2(a) show daily false positive statistics as averages across *infected* hosts while the host stayed infected. For comparison reasons, we tested Williamson's scheme on normal hosts, the result of which are shown in Figure 2(b).

A few high-level insights are important here: First, Figure 2 suggests that false positives are low during normal operation (about 15%). Once infection occurs, however, Williamson's scheme yields false positive rates nearly 90%. This is undesirable as during the worm outbreak, essentially all benign traffic is subjected to delay incurred by the throttling scheme. Figure 2(d) shows the average queue length for infected hosts. As shown, when infection hit on day 6, the average queue length quickly reached the maximum (100 in this case) and remained in the neighborhood of 90%. This means that during infection, delay for each fresh IP connection was approximately 90 seconds or greater if the queue was filled with distinct hosts, which is likely to be the case due to the random scanning nature of the worms.

We note, however, the way we define false positives is slightly unfair; we label every delayed non-worm SYN packet a false positive. In reality, many applications can tolerate a slight delay. Table 1 shows the delay statistics for a normal host during a 3-hour period. As shown, all delays were less than 10 seconds, which may be entirely acceptable for certain applications. In contrast, Table 2 shows the worst case delay statistics for an infected host for the same time period. As shown, once a host is infected, the delay queue becomes saturated with worm packets and legitimate applications on the host are subjected to excessive delays and blockage.

Another observation is that the size of the working set (at least for the values experimented here) has very little effect on the error rates of the scheme. This is at least partially due to the fact that we averaged statistics across hosts. However, our experiments suggest that Williamson's throttling scheme exhibits a bimodal behavior with respect to legitimate traffic: minimal impact during normal operation and greatly restrictive if infected. This behavior, we conjecture, is inherent to the scheme regardless of the size of the working set, provided that the working set permits at least the host's normal contact rate. In practice, one can observe the connection pattern of a host for some period of time before determining the normal contact rate.

Figure 2(c) shows the false negative rates, which are predominantly below 1%. This means that Williamson's scheme is effective against worm spread, though it also incurs large delays for legitimate applications running on the same host. The strength of Williamson's scheme lies in its logical simplicity and ease of management. One can imagine a more complex data structure than a simple queue to deal with delayed connections. Alternatively, one can employ a dynamic rate scheme that changes the dequeuing rate accordingly with the length of the delay

Delay Amount.	Number of Flows
No delay	1759
1 - 10 sec.	385
11 - 20 sec.	0
Total number of benign flows	2144

Table 1: Delay statistics for a normal host during a 3-hour period

queue. Schemes such as these can potentially reduce the false positive rates, but at the price of increased complexity.

Throttling at the Edge Router: Previous studies [10, 25] showed that end-host rate limiting is ineffective unless deployment is universal. As part of this study, we investigate the effect of applying Williamson's throttling to the aggregate traffic at the edge of the network. Aggregate, edge-based throttling is attractive because it requires the instrumentation of only the ingress/egress point of the subnet. Furthermore, aggregate throttling dose not require per-host state to be kept. We note that the logic of aggregate throttling can be extended to the border point of a network cell within an enterprise, as shown in [14], which can provide a finer protection granularity.

In a previous traffic study, we identified a candidate rate of 16 addresses per five seconds for edge throttling for a similar network [25]. In the analysis that follow, we present results obtained with five aggregate rate limits: 10, 16, 20, 25 and 50 IPs per every five-second window.

Figure 3(a) shows the false positive rates for edge-router rate limiting using various rate limits. The corresponding false negative rates are shown in Figure 3(b). Compared with the end-host case, edge-based rate limiting exhibits significantly higher false positive rates during normal operation. This is primarily due to the fact that aggregate throttling penalizes hosts with atypical traffic patterns, thereby contributing to a higher false positive rate. We can increase the working set size at the edge to reduce the false positives, but false positives will increase accordingly. As such, Williamson's throttling is best suited for end-host rate limiting where behavior of a host is somewhat predictable.

6 Failed Connection Rate Limiting (FC)

Chen et al. proposed another rate limiting scheme based on the assumption that a host infected by a scanning worm will generate a large number of failed TCP requests [1]. Their scheme attempts to rate limit hosts that exhibit such behavior. In the discussions that follow, we refer to this scheme as FC (for Failed Connection).

FC is an edge-router based scheme that consists of two phases. The first phase identifies the potential "infected" hosts. During this phase a highly contended hash table is used to store failure statistics for hosts. The hash table is used to limit the amount of per-host state kept at the router. Once the failure rate for a hash entry exceeds a certain threshold, the algorithm enters the second phase, which attempts to rate limit the hosts in the entry. Chen proposed a "basic" and "temporal" rate limiting algorithm. We analyze both in this study.

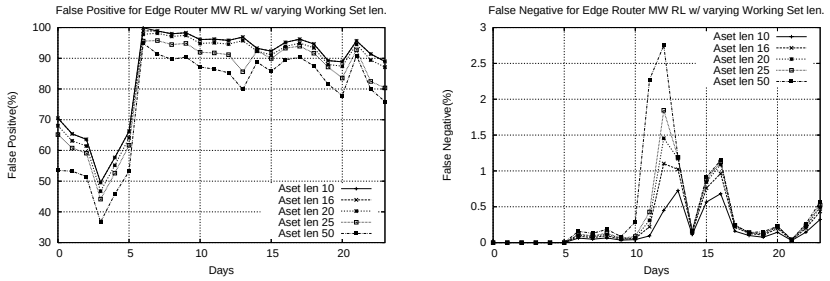
The basic FC algorithm focuses on a short-term failure rate, λ. Chen recommends a λ value of one failure per second. Once a hash entry exceeds λ, the rate-limiting engine attempts to limit the failure rate of each host in the entry to at most λ, using a leaky bucket token algorithm—a token is removed from the bucket for each failed connection and every λ seconds a new token is added to the bucket. Once the bucket for a particular host is empty, further connections from that host are dropped.

Temporal FC attempts to limit both the short term failure rate λ and a longer term rate Ω. Chen suggested Ω be a daily rate and λ a per second rate. The value of Ω is intended to be significantly smaller than λ * (seconds in a day). Hosts in a hash table entry are subjected to rate limiting if the failure rate of the entry exceeds λ per second or Ω per day. The objective of temporal FC is to catch prolonged but somewhat less aggressive scanning behavior—worms that spread under the short-term rate of λ.

To evaluate these two algorithms we conducted experiments with the border trace, with varying values of λ and Ω. Figure 4(a) and (b) show the error rates for basic and temporal FC, with λ equaling 1 and Ω equaling 300, as recommended by Chen. Figure 4(a) shows an increase in the false positive rates during the first week of infection. This increase is due to the fact that a worm generates rapid failed connections and quickly depletes the available tokens. Until more tokens become available, legitimate traffic is stopped altogether, as seen in

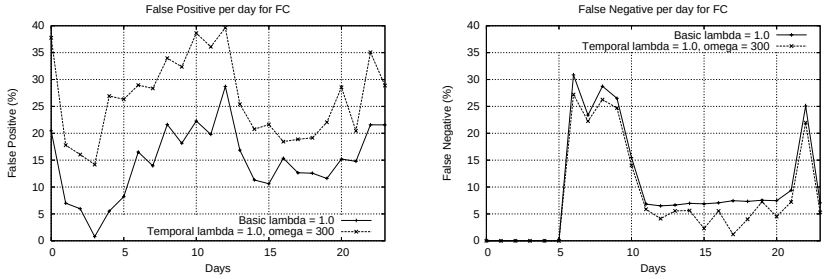
Delay Amount.	Benign	Malicious
No delay	1	12
1 - 30 sec.	1	36
31 - 60 sec.	1	36
61 - 90 sec.	0	50
91 - 100 sec.	141	10115
Dropped	866	107080
Total	1010	117314

Table 2: Delay Statistics for an infected host during a 3-hour period



(a) FP per day at Edge Router (b) FN per day at Edge Router

Figure 3: Results for Williamson's RL mechanism at Edge Router



(a) FP per day (b) FN per day

Figure 4: Error rates per day for Basic and Temporal FC with λ = 1.0 & Ω = 300.

the third and forth row of Table 3.

In Figure 4(b) there is a pronounced initial jump in the false negative rates as Blaster hits on day 6, and in a few days the false negatives decrease significantly. The bulk of false negatives can be attributed to the fact that Chen's scheme uses only `TCP_RST` as an indication of a failed connection. Since many firewalls simply drop packets instead of responding with `TCP_RST`s, using `TCP_RST`s exclusively underestimates the number of failed connections. Figure 5(b) shows the error rates including `TCP_TIMEOUT`s. As shown, false negative rates of FC are reduced significantly when Timeouts are considered. The drop in false negative rate on day 10 in Figure 4(b) is correlated with the onset of the Welchia outbreak. Blaster scanning generates a substantial number of `TCP_TIMEOUT`s while Welchia tends to generate `TCP_RST`s (Welchia scans via `ICMP_ECHO`). As more and more Blaster hosts are patched and Welchia makes up a greater portion of the worm traffic, the false negatives are reduced.

Figure 5 plots the false positive rates against the false negative rates with varying values for λ and Ω. The data points in this graph are averaged daily statistics over the entire trace period. In temporal FC, when failures reach $\Omega/2$, the rate limiting algorithm proceeds to rate limit hosts in a much more aggressive fashion than the basic

IP	# Good Flows Dropped		Total # Good Flows	Cause
	Basic	Temporal		
188.139.199.15	32896	56979	57336	eDonkey Client
188.139.202.79	25990	32945	33961	BearShare Client
188.139.173.123	5386	13457	15108	HTTP Client
188.139.173.104	4852	6175	6254	Good Flows(Inf. Client)

Table 3: False Positives and Cause for Day 6 λ = 1.0 and Ω = 300

scheme. This strategy results in a significant amount of non-worm traffic from "infected" hosts being dropped. In the third row of Table 3, temporal FC dropped approximately 2.5 times more benign traffic compared to basic FC. Since a typical worm outbreak will quickly reach $\Omega/2$ failures, temporal FC is more restrictive and thus renders higher false positives.

Comparing FC results to host-based Williamson's, we can see that FC renders significantly lower false positives during infection but yields slightly higher false negatives. In fact, with FC's drop-only approach and Williamson's tendency to saturate the delay queue, both closely approximate a detect-and-block approach, which is less interesting from the standpoint of rate limiting.

7 Credit-based Rate Limiting (CB)

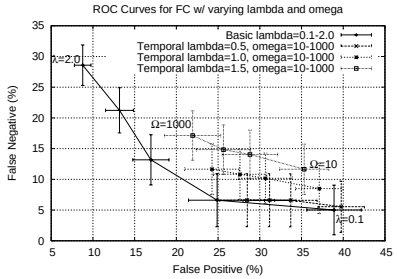
Another rate limiting scheme based on failed connection statistics is the credit-based scheme by Schechter et. al. [12]. We refer to it as CB (for Credit Based). CB differs from Chen's in two significant ways. First, it performs rate limiting exclusively on *first contact* connections—outgoing connections for destination IPs that have not been visited previously. The underlying rationale is that scanning worms produce a large volume of failed connections, but more specifically they produce failed first-contact connections, therefore anomalous first-contact statistics are indicative of scanning behavior. The notion of first contact is fundamental to CB and as we show later is instrumental to its success. Second, CB considers both failed and successful connection statistics. Simply described, CB allocates a certain number of connection credits per host; each failed first-contact connection depletes one credit while a successful one adds a credit. A host is only allowed to make first-contact connections if its credit balance is positive.

It is straightforward to see that CB limits the first-contact failure rate at each host, but does not restrict the number of successful connections if the credit balance remains positive. Further, non-first-contact connections (typically legitimate traffic) are permitted through irrespective of the credit balance. Consequently, a scanning worm producing a large number of failed first contacts will quickly exhaust its credit balance and be contained. Legitimate applications typically contact previously seen addresses, thereby are largely unaffected by the rate limiting mechanism.

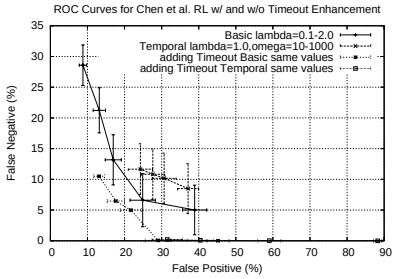
In order to determine whether an outgoing TCP request is a first contact, CB maintains a PCH (Previously Contacted Host) list for each host. Additionally, a failure-credit balance is maintained for each host. We implemented the CB algorithm and experimented with the per-host trace data. Schechter suggested a 64-address PCH and a 10-credit initial balance. We conducted experiments with PCH ranging from 8 to 128 entries with Least Recently Used (LRU) replacement. Our experience suggests that the level of the initial credit balance has minimal impact on the performance of the scheme, as that only approximates the number of failures that can occur within a time period; in reality a host can accrue more credits by initiating successful first contacts. For the experiments, we use an initial credit balance of 10 per host.

Figure 6(a) shows CB's daily false positive and false negative rates with a 64-address PCH. The data points in this graph are averages across all hosts. As shown, the average false positive and false negative rates are between 5% and 15% during the infection period. The false positive results significantly outperform both FC and Williamson's. CB's false negative results are comparable to those of Williamson's. These results speak strongly of CB's insight of rate limiting first contacts rather than distinct IPs or straightforward failed connections. Since worm scanning consists primarily of first-contact connections, CB's strategy gives rise to a more precise means of rate limiting.

Table 4 shows the false positive data for the top two false-positive-generating hosts. Both clients that incurred high false positives are P2P clients. The data show that the worst case false positive rate is rather high—nearly 40% for the host in row one. For comparison reasons, here we also include the HTTP client discussed previously (row 3 from Table 3). As shown, CB is able to accommodate this bursty web client while FC dropped a significant por-

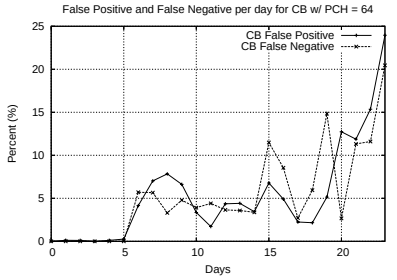


(a) ROC w/ diff. values of λ and Ω

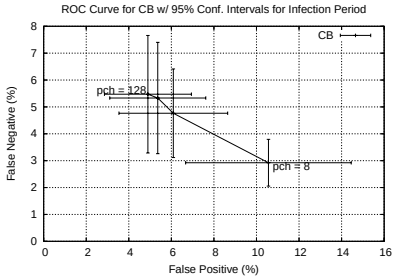


(b) ROC and timeout enhancement

Figure 5: ROC for different λ and Ω values for Basic and Temporal RL algorithms



(a) Error rates (PCH = 64)



(b) ROC w/ varying PCH sizes

Figure 6: Results of Error Rates for CB RL

tion of the client's traffic.

Figure 6(b) plots the average false positive rates against the corresponding false negative rates for PCH of 8, 16, 32, 64, and 128. The data points in this graph are obtained by averaging per-host statistics over the entire 24-day trace period (sans the pre-infection days). As shown, CB's error rates are not particularly sensitive to the length of the PCH's. A 6% increase in the false positive value is observed when PCH is reduced from 128 entries to 8. As the PCH size increased so did the false negative rate, which is a peculiar phenomenon. We are unable to find a satisfactory explanation for this. We conjecture that a possible error in the Blaster mutex code allowed multiple instances of Blaster to execute on the same machine, thereby generating repeated scanning to the same addresses.

Note that CB is essentially a host-based scheme since states are kept for each host. Aggregating and correlating connection statistics across the network can reduce the amount of state kept. For example, if host A makes a successful first-contact connection to an external address, further connections for that address could be permitted through regardless of the identity of the originating host. This optimizes for the scenario that legitimate applications (e.g., web browsing) on different hosts may visit identical external addresses (e.g., cnn.com). A more detailed investigation of aggregate CB can be found later in Section 9.

8 DNS-based Rate Limiting

In this section we analyze a rate limiting scheme based on DNS statistics. The underlying principle is that worm programs induce visibly different DNS statistics from those of legitimate applications [24, 22, 4]. For instance, the non-existence of DNS lookups is a telltale sign for scanning activity. This observation was first made by Ganger et al. [4]. The scheme we analyze here is a modification of Ganger's NIC-based DNS detection scheme.

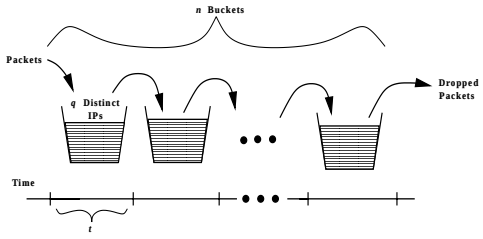


Figure 7: Cascading Bucket RL Scheme

IP	# Good Flows Dropped	Total # of Good Flows	Cause
188.139.199.15	22907	57336	eDonkey Client
188.139.202.79	13269	33961	BearShare Client
188.139.173.123	0	15108	HTTP Client

Table 4: Per Host False Positives and Cause for Day 6 for PCH = 64

The high-level strategy of the DNS rate limiting scheme (hereafter refer to as DNS RL) is simple: for every outgoing TCP SYN, the rate limiting scheme permits it through if there exists a prior DNS translation for the destination IP, otherwise the SYN packet is rate limited. The algorithm uses a *cascading bucket* scheme to contain untranslated IP connections. A graphical illustration of the algorithm is shown in Figure 7. In this scheme, there exists a set of n buckets, each capable of holding q distinct IPs. The buckets are placed contiguously along the time axis and each spans a time interval t.

The algorithm works as follows: When a TCP SYN is sent to an address that does not have a prior DNS translation, the destination IP is added into the bucket for the current time interval and the packet is delayed. When a bucket is filled with q distinct IPs, new connection requests are placed into the subsequent bucket, thus each bucket *cascades* into the next one. Requests in the i-th bucket are delayed until the beginning of the i+1 time interval. The n-th bucket, the last in line, has no overflow bucket and once it is full, new TCP SYN packets without DNS translations are simply dropped. At the end of the n*t time periods, we reinstate another n buckets for the next $n*t$ time period. This algorithm permits a maximum of q distinct IPs (without DNS translations) per time interval t and packets (if not dropped) are delayed at most $n * t$.

The notion of the buckets provides an abstraction, with which an administrator could define rules such as "Permit 10 new flows every 30 seconds dropping anything over 120 seconds." This example rule, then, would translate to 4 buckets (30 seconds * 4 = 2 minutes) with q = 10 and t = 30. Expressing rate limiting rules in this manner is more intuitive and easier than attempting to characterize network traffic in terms of working sets or the failure rate of connections.

This scheme can be implemented at the host level or at the edge router of a network. A host-level implementation requires keeping DNS-related statistics on each host. Edge-router-based implementation would require the border router to keep a shadow DNS cache for the entire network.

In our study, we tested DNS RL both at the host level and at the edge, using DNS server cache information and all DNS traffic recorded at the network border. More specifically, we mirrored the DNS cache (and all TTLs) at the edge and updated the cache as new DNS queries/replies are recorded. Traffic to destination addresses with an unexpired DNS record is permitted through, while all others are delayed.

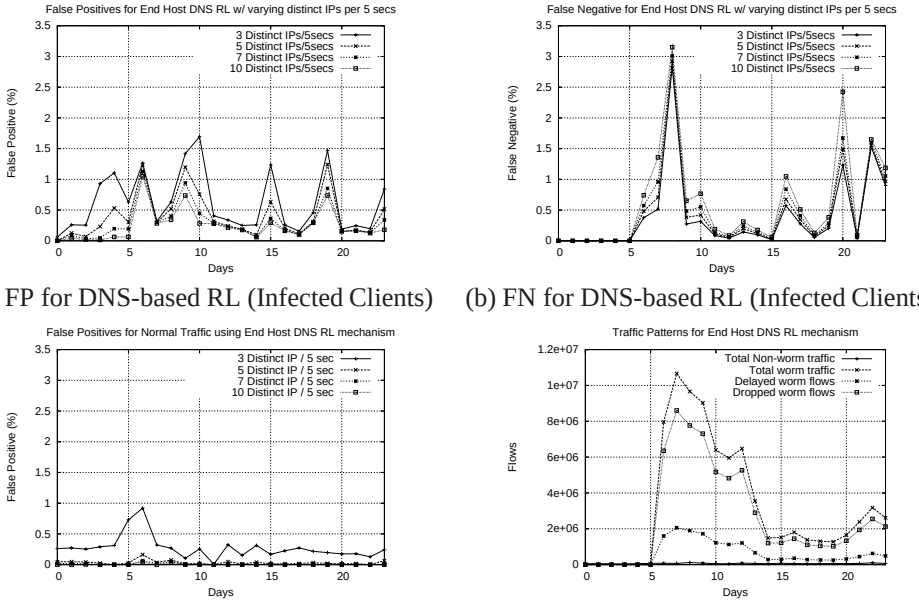
8.1 Analysis

The critical parameter for the cascading-bucket scheme is the rate limit, which manifests in the values of q (the size of each bucket), t (the time interval), and n (number of buckets). To simplify our analysis, we varied the value of q and kept n and t constant[1]. Additionally, the value of $n * t$ was set to 120 seconds to model the TCP timeout period. This scheme allows a certain number of untranslated IP connections to exit the network, which intends to accommodate legitimate direct-IP connections. In our data set, we observed some direct server-server communication and direct-IP connections due to peer-to-peer, streaming audio and passive FTP traffic. These were the main cause of false positives observed. One can attempt to maintain a white list to allow legitimate direct-IP connections and thus further reduce false positives. However, as observed in [22], a comprehensive white list for an open network may not be feasible.

We first analyze the host-level DNS throttling scheme. For this, we maintain a set of cascading buckets for each host. Figure 8(a) and (b) show the false positive and false negative rates for infected hosts. The data in these graphs are daily error rates averaged over all infected hosts. Figure 8(c) plots the analogous false positive rates for normal hosts. In addition, Table 5 presents the delay statistics for a normal host and Table 6 shows the worst case delay statistics for an infected host.

These results yield a number of observations: First, host-level DNS throttling significantly outperforms the other mechanisms analyzed previously. As seen in Figure 8, the average false positive rates fall in the range of 0.1% to 1.7% with corresponding false negative rates between 0.1% to 3.2%, both significantly lower than the error statistics of the others. We also observed that ap-

[1]By varying q and leaving n and t constant, we can achieve the goal of regulating the rate limits



(a) FP for DNS-based RL (Infected Clients) (b) FN for DNS-based RL (Infected Clients)

(c) FP for DNS-based RL (Normal Clients) (d) Flows Dropped / Delayed

Figure 8: Results for DNS-based End Host RL

Delay Amount.	# of Flows
No delay	2136
1 - 10 sec.	8
$>$ 10 sec.	0
Total number of benign flows	2144

Table 5: DNS RL delay statistics for a normal host (3-hour period).

plications that do experience false positives here tend to be those that fall outside of the security policies of an enterprise network (e.g., peer-to-peer applications)— disruption of such applications are generally considered not critical to the network operation.

Table 5 shows the delay statistics for a normal host. As shown, DNS RL delayed 8 total flows for this host, as opposed to the 385 flows using Williamson's (Table 2 in Section 5). Also note that all the delays in Table 5 are less than 10 seconds, which are not significant. Table 6 shows the worst case delay statistics for an infected host during the peak of its infection period. The statistics show that DNS RL dropped approximately 17% of the host's benign traffic, compared to over 90% when using Williamson's. In addition, DNS RL delays less flows for normal hosts than Williamson's. Also note in Table 6, nearly delayed malicious flows are subjected to the maximum allowed delay and over 95% of the malicious flows are dropped.

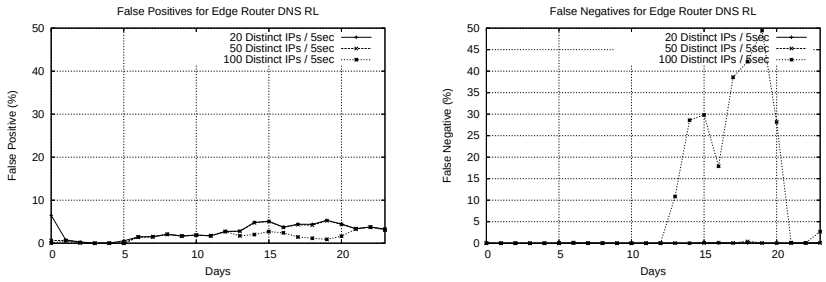
During the outbreak period, the false positives for infected hosts included both dropped and delayed traffic flows. A q value of 5 would drop approximately 0.075% and delay 0.375% of the legitimate traffic. Figure 8(d) shows summarized statistics from our analysis for a liberal value of $q = 10$. During Blaster's outbreak, on average 97% of the worm traffic was rate limited— approximately 82% dropped and the other 18% delayed with an average delay of one minute each.

Our results also show that DNS rate limiting is capable of containing slow spreading worms. As a comparison, Weaver's Approximate TRW containment mechanism can block worms that scan faster than 1 scan per second [21]. Using the DNS scheme, with value of $q = 3$ and $t = 5$ for instance (3 direct-IP connections in 5-second window), we can contain worms that scan at the rate of 0.6 scans per second (or more) with 99% accuracy.

To test the effect of aggregate throttling, we implemented a single set of cascading buckets for the entire network. For this set of experiments, the value of q was set to

Delay Amount.	Benign	Malicious
No delay	806	1
1 - 30 sec.	4	34
31 - 60 sec.	2	35
61 - 100 sec.	12	40
$>$ 100 sec.	11	4903
Dropped	172	112862
Total	1007	117875

Table 6: DNS RL Delay Statistics for an infected host during a 3-hour period



(a) FP for DNS-based RL (b)FN for DNS-based RL

Figure 9: Results for DNS-based RL at the Edge Router

20, 50, and 100 IPs per five second window. Figure 9 shows the error rates for the aggregate implementation. As shown, a q value of 20 or 50 IPs yielded few false negatives and a false positive rate of approximately three to five percent. Note that when q is set to 20 or 50, the false negative rates of edge-based rate limiting are lower than the host-level scheme. This is because the aggregate traffic limit is more restrictive overall than the collective limit in the host-based case. Although the false positive rates for the aggregate case are slightly higher than the host-level case, overall the error rates are fairly low—5% false positive and $<$ 1% false positive.

9 Discussions

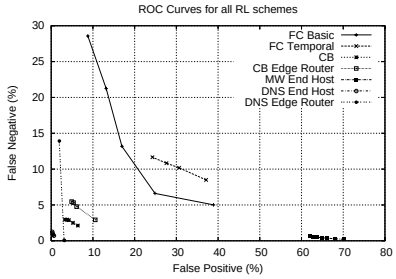
Analysis in the previous sections brought to light a number of issues with respect to rate limiting technology. In this section we attempt to extrapolate from these results and discuss some general insights.

DNS-based RL vs. others: A summary comparison of the DNS-based scheme with the others is in Figure 10. The parameters here are consistent with the values used in the previous sections. As shown, DNS-based rate limiting has the best performing false positive and false negative rates. Host-based DNS throttling renders an average false positive and false negative rate below 1%. These results present a strong case for DNS-based rate limiting.

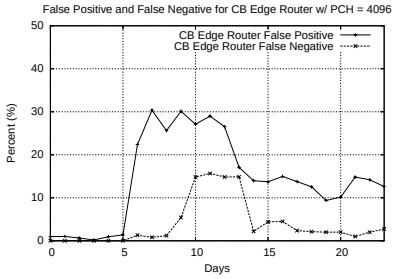
Recall that the q value in DNS throttling allows for q untranslated IP connections per host to exit the network every t seconds. To put things in perspective, for the first day of infection, the network had a total of 468,300 outbound legitimate flows. When $q = 7$ a total of 463 legitimate flows are dropped, which yields a false positive rate of 0.099%. This is less than 1 dropped flow per host per day. As a comparison, CB dropped 3767 legitimate flows for the same day, a false positive rate of 7.8%.

The success of DNS RL can be attributed primarily to the fact that DNS traffic patterns (or the lack thereof), compared to other statistics, more precisely delineate worm traffic from normal behavior. DNS-based RL can thus impose severe limitations on worm traffic without visibly impacting normal traffic.

One of the reasons that scanning worms are successful is because they are able to probe the numeric IP space extremely rapidly in their search for potential victims. Navigating the DNS name space is a far more difficult process to automate, since the name space is less populated and has poorer locality properties. DNS-based throttling forces scanning worms to probe the DNS name space, thereby reducing the scan hit rate and substantially raising the level of difficulties for scanning worms



(a) Avg error rates



(b) Edge CB Results for PCH = 4096

Figure 10: Avg. error rates for all RL schemes and Edge CB Results

to propagate.

We note that although our trace data reflects a simple worm that does not attempt to mask itself, extending the DNS RL scheme to more sophisticated worms is straightforward. We plan to address this in future work.

Issues with DNS-based rate limiting: An attacker can attempt to circumvent the DNS rate limiting mechanism in a number of ways:

First, a worm could use reverse DNS-lookups (PTR lookups) to "pretend" that it has received a DNS translation for a destination IP. Jung et. al. [6] characterizes that PTR lookups are primarily for incoming TCP connections or lookups related to reverse blacklist services. These types of lookups can be easily filtered and not considered as valid entries in the DNS cache. In addition, a PTR lookup prior to an infection attempt will significantly reduce the infection speed.

Second, an attacker could setup a fake external DNS server and issue a DNS query for each IP. We can alleviate this threat by establishing a "white-list" of legitimate external DNS servers. Also, the attacker needs a server with a substantial bandwidth to accommodate the scan speed, which is not trivial. A case of interest here is SOHO (Small office Home office) users who may set up their own routers and use legitimate external DNS servers. To accommodate such usage, we can use a packet scrubber such as Hogwash [5] to help correlate DNS queries to responses.

Another attack against DNS throttling is to equip each worm with a dictionary of host names and domains. This effectively turns a scanning worm into a worm with a hit-list. Hit-list worms are significantly more difficult to engineer. If the only viable means to bypass DNS-based throttling is for the worm to carry a hit-list, that in itself is a positive testimony for DNS-based throttling.

Dynamic vs static rates: Rate limiting schemes impact the rate of both legitimate and malicious connections. Williamson's imposes a strictly static rate, e.g., five distinct IPs per second, irrespective of the traffic demand. FC is predominately static while CB allows for a dynamic traffic rate by rewarding successful connection and penalizing failed connections. Results in Figure 10 show that CB outperforms FC. This is partially due to CB's dynamic rates which render a more graceful filtering scheme that permits both bursty application behavior and temporarily abnormal-but-benign traffic patterns. As we briefly discussed in Section 5, mechanisms that impose a static rate can benefit from incorporating dynamic rate limits. Dynamic rate limiting is an interesting topic worth further study.

Host vs aggregate: An issue of significance is host versus aggregate rate limiting. The general wisdom is that host-level throttling is more precise but is at the same time more costly because per host state must be maintained. Indeed, Williamson's IP throttling, when applied at the edge, rendered visibly higher false positives than its host-based counterpart. This is because IP contact behavior at the host-level is more fine-grained and thus more likely to be stable. In contrast, aggregate traffic at the edge includes hosts whose behavior may vary significantly from each other, thereby contributing to a higher error rate. A similar case was observed with CB when applied to the aggregate traffic, the results of which are shown in Figure 10(b). As shown, the false positive rates reach approximately 30%, compared to the 10% with the host-based deployment. Edge-based DNS throttling, however, appears to be an exception. Figure 10(a) shows that a carefully chosen rate limit, e.g., 50 IPs per five seconds, yields excellent accuracy for edge-based DNS throttling. It has lower false positive and false negative rates than other host-based schemes. The fundamental reason behind this is that DNS statistics, in particular the presence (or the lack) of IP translations, remain largely

invariant from host to the aggregate level.

This result is extremely encouraging, as aggregate rate limiting has a lower storage overhead and is typically easier to deploy and maintain than host-based schemes. Note that our study did not include an analysis on processing overhead. Readers should be reminded that edge-based schemes in general imply processing a larger amount of data per connection, therefore a trade-off between storage and processing overhead exists. The aggregate DNS throttling result allude to the possibility of pushing rate limiting deeper into the core where a single instrumentation can cover many IP-to-IP paths and potentially achieve a greater impact.

We note that edge-based throttling in itself does not defend against internal infection. One way to protect against internal infection (and not pay the cost of host-level throttling) is to divide an enterprise network into various cells (as suggested by Staniford [14]) and apply the aggregate throttling at the border of each cell. We leave the analysis of more fine-grained, intra-network protection as future work.

10 Summary

A number of rate limiting schemes have been proposed recently to mitigate scanning worms. In this paper, we present the first empirical analysis of the different schemes, using real traffic and attack traces from an open network environment. We believe that the scheme that performs well in an open network and will perform equally well (if not better) in an environment with strict traffic policies (e.g., enterprise network).

We evaluate and contrast the false positive and false negative rates for each scheme. Our analysis reveals these insights. First, the subject of rate limiting is by far the most significant "parameter"—failed-connection behavior alone is too restrictive as evidenced by FC; rate limiting first-contacts renders better results and DNS behavior-based rate limiting is by far the most accurate strategy. Second, it is feasible to delineate worm behavior from normal traffic even at an aggregate level, as indicated by the DNS analysis. This is an interesting result because aggregate rate limiting alleviates the universal participation requirement thought necessary for worm containment [10, 25]. This result also suggests that it may be possible to apply rate limiting deeper into the core of the network, a subject that is of great interest to many. Third, preliminary investigation suggests that incorporating dynamic rates results in increased accuracy. As most of rate limiting schemes to-date focus on static rates, an immediate follow-up research is dynamic rate limiting and how that can be implemented in practice.

11 Acknowledgments

We thank the members and companies of the PDL Consortium (including APC, EMC, EqualLogic, Hewlett-Packard, Hitachi, IBM, Intel, Microsoft, Network Appliance, Oracle, Panasas, Seagate, Sun, and Veritas) for their interest, insights, feedback, and support. This material is based upon work supported by the National Science Foundation under Grant No. 0326472 and by the Air Force Research Laboratory via grant number FA8750-04-01-0238 and also by the Army Research Office via grant number DAAD19-02-1-0389. The authors thank Greg Ganger and Mike Reiter for providing insightful feedback on preliminary versions of this work. We also thank Matthew Williamson for technical discussions about this work.

References

[1] S. Chen and Y. Tang. Slowing Down Internet Worms.

[2] M. Collins and M. Reiter. An Empirical Analysis of Target-Resident DoS Filters.

[3] D. R. Ellis, J. G. Aiken, K. S. Attwood, and S. Tenaglia. A Behavioral Approach to Worm Detection. ACM Press.

[4] G. Ganger, G. Economou, and S. Bielski. Self-securing Network Interfaces: What, Why and How.

[5] Hogwash. Inline packet scrubber.

[6] H. B. J. Jung, E. Sit and R. Morris. DNS Performance and the Effectiveness of Caching.

[7] J. Jung, V. Paxon, A. W. Berger, and H. Balakrishman. Fast Portscan Detection Using Sequential Hypothesis Testing.

[8] J. Kephart and S. White. Directed-Graph Epidemiological Models of Computer Viruses. Pages 343-359.

[9] H. Kim and B. Karp. Autograph: Toward Automated, Distributed Worm Signature Detection.

[10] D. Moore, C. Shannon, G. Voelker, and S. Savage. Internet Quarantine: Requirements for Containing Self-Propagating Code.

[11] Network-Associates.

[12] S. Schechter, J. Jung, and A. W. Berger. Fast Detection of Scanning Worm Infections.

[13] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated Worm Fingerprinting. Proceedings of the 6th ACM/USENIX Symposium on Operating System Design and Implementation.

[14] S. Staniford. Containment of Scanning Worms in Enterprise Networks. *Journal of Computer Science*.

[15] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in Your Spare Time.

[16] Symantec. W32.Blaster.Worm, August 11, 2003.

[17] Symantec. W32.Welchia.Worm, August 18, 2003.

[18] H. J. Wang, C. Guo, D. R. Simon, and A. Zugenmaier. Shield: vulnerability-driven network filters for preventing known vulnerability exploits. Pages 193–204. ACM Press.

[19] Y. Wang, D. Chakrabarti, C. Wang, and C. Faloutsos. Epidemic Spreading in Real Networks: An Eigenvalue Viewpoint.

[20] Y. Wang and C. Wang. Modeling the effects of timing parameters on virus propagation. Pages 61–66. ACM Press.

[21] N. Weaver, S. Staniford, and V. Paxson. Very Fast Containment of Scanning Worms.

[22] D. Whyte, E. Kranakis, and P. van Oorschot. DNS-based Detection of Scanning Worms in an Enterprise Network.

[23] M. Williamson. Throttling Viruses: Restricting propagation to defeat malicious mobile code.

[24] C. Wong, S. Bielski, J. McCune, and C. Wang. A Study of Mass-mailing Worms. ACM Press.

[25] C. Wong, C. Wang, D. Song, S. Bielski, and G. Ganger. Dynamic Quarantine of Internet Worms.

[26] C. Zou, W. Gong, and D. Towsley. Code Red Worm Propagation Modeling and Analysis.