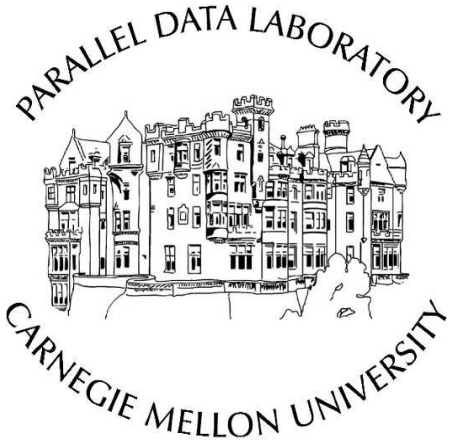


On the Effectiveness of Rate Limiting Mechanisms

Cynthia Wong, Stan Bielski, Ahren Studer, Chenxi Wang

CMU-PDL-05-103
March 2005

Parallel Data Laboratory
Carnegie Mellon University
Pittsburgh, PA 15213-3890

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 empirical analysis additionally brings to light a number of issues with respect to rate limiting in practice. We explore the impact of these issues in the context of general worm containment.

Acknowledgments: We thank the members and companies of the PDL Consortium for their support. This work was partially supported by the National Science Foundation under Grant No. CNS-0433540 and ANI-0326472.

Keywords: Internet Worms, Network Security, Traffic Analysis, Worm Containment

1 Introduction

Recent fast-spreading worms such as Slammer [3], Blaster [20], and SoBig [1] 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, MyDoom, and Blaster [1, 2, 20]. The need to mitigate worm spread is apparent and pressing.

Researchers have proposed various techniques for worm defense, both in detection [11, 25, 13, 17] and response [26, 24, 4, 16, 7]. 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 [26, 4, 16] 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[1]. A recent analytical study showed that when deployed at appropriate points in the network, rate limiting can substantially reduce the spread of infection [28].

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 edge router of a network of more than 1200 hosts. The trace data includes real attack traffic of Blaster and Welchia worms. The outbreak of Blaster and Welchia was substantial in our network, over 100 hosts were infected. 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 [26], Chen's failed-connection-based scheme [4] and Schechter's credit-based rate limiting [16]. Williamson's throttling scheme limits the rate of distinct IP connections from an end host [26]. Chen et al. [4] and Schechter et al. [16] 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 [7]. Recently Whyte et al. showed that DNS-based worm detection can be extended to a network setting [25]. The DNS-based rate limiting mechanism we study is a modified version of Ganger's scheme [7]. 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, 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. Since most of the rate limiting mechanisms analyzed here target specifically enterprise networks similar to the one where the trace data is collected, we believe that our analysis provides reasonable insights into how well these schemes might perform in practice.

The remainder of this paper is structured as follows. Section 2 gives an overview of the existing rate limiting technology. Section 3 describes the trace data used in the experiments. Section 4 describes our

[1]This is especially true for network operators such as ISPs whose business depends on offering continued network access to customers.

analysis methodology. Sections 5, 6, 7, and 8 analyze the performance of Williamson's, Chen's, Schechter's and the DNS-based rate limiting mechanism respectively. Section 9 offers a discussion of the results and insights. We conclude in Section 10.

2 Related Work

The individual rate limiting schemes by Williamson et al. [26], Chen et al. [4], and Schechter et al. [16] are the target of our analysis. We thus 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 [24, 11, 13, 25, 21, 17]. Some of the schemes are complimentary to rate limiting at large, which can be combined in practice. For instance, the scan detection work by Weaver et al. [24] and Jung et al. [11] can be used to protect enterprise networks from incoming infections while rate limiting seeks to contain outbound propagations. Also of interest is worm detection work built on similar principles [25, 17, 13]. But in this work, 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.

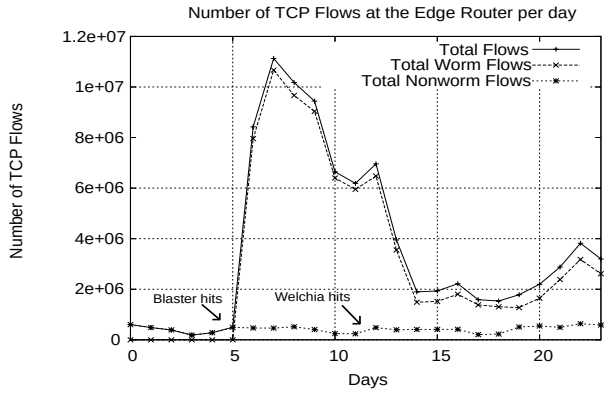
We note that there exists a rich body of worm modeling and analysis work [14, 19, 30, 5, 12, 22, 23, 15, 18] that offers theoretical understanding of and technical insights into worm defense. Our goal is not to study worm propagation in a broad sense, but rather we seek to evaluate and understand the 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. [6], though the target of our analysis is different and therefore offers different insights and conclusions.

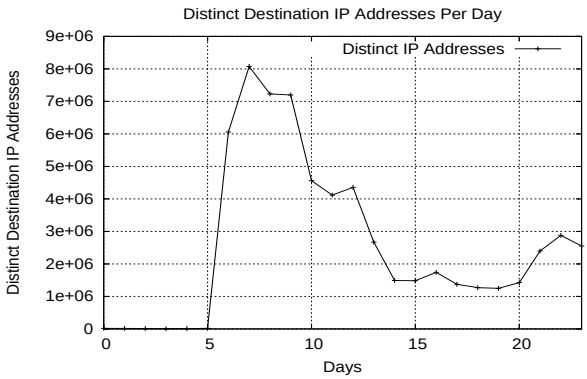
3 Trace Data

The study in this paper is conducted using traffic traces collected from the edge router 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, including Windows, Linux, Solaris, and Mac OS. The network spans multiple domains and supports a variety of servers, including Windows, AFS, SMTP, and DNS. 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. Included among the header information was the source address, destination address, port number, and size of the payload. 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* [20]. 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 port 135 connections to destinations outside of the network. 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.



(a) Daily outgoing flows



(b) Daily volume of distinct IPs

Figure 1: Traffic Statistics for the Blaster/Welchia Trace

For the purpose of this analysis, we use a period of 24-day outbound trace, from August 6th, 2003 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 a week later on the 18th. 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 seen daily for the same time period. 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 11.5 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 activity. Note that there is a small increase in worm traffic at the end of the trace period. We conjecture that this is because a small number of worms were reintroduced into the network due to the beginning of the semester 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 traffic trace. In those cases we will differentiate between infected host traffic and normal host traffic.

4 Analysis Methodology

In this section we describe the high-level analysis methodology used in this study.

As previously mentioned, we use traffic traces collected at the border router of an academic department network. For this particular study, we use a period of 24-day outbound trace, which includes documented Blaster and Welchia activities. We traced malicious flows to infected machines and benign flows to legitimate applications as described in Section 3. Since the traces are for outbound traffic and we have nearly comprehensive knowledge of the network, we believe that our separation of malicious vs. benign traffic is fairly accurate.

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 here 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 will attempt to differentiate between these two

Anon IP	# Benign Flows Dropped	Total # of Benign Flows	Cause
188.139.182.84	17050	43451	HTTP/IM client
188.139.160.173	11774	30819	HTTP client
188.139.222.217	113512	454699	HTTP/IMAP client

Table 1: Per Host False Positives and Cause for Day 6 with Active Set = 5

cases and present error rates accordingly. Also note that we do not calculate false negative rates for the pre-infection period (for which false negatives are clearly zero), but false positives are considered throughout the entire 24-day trace period. Whenever appropriate, we present Receiver Operator Curves (ROC) to contrast false negatives with false positives.

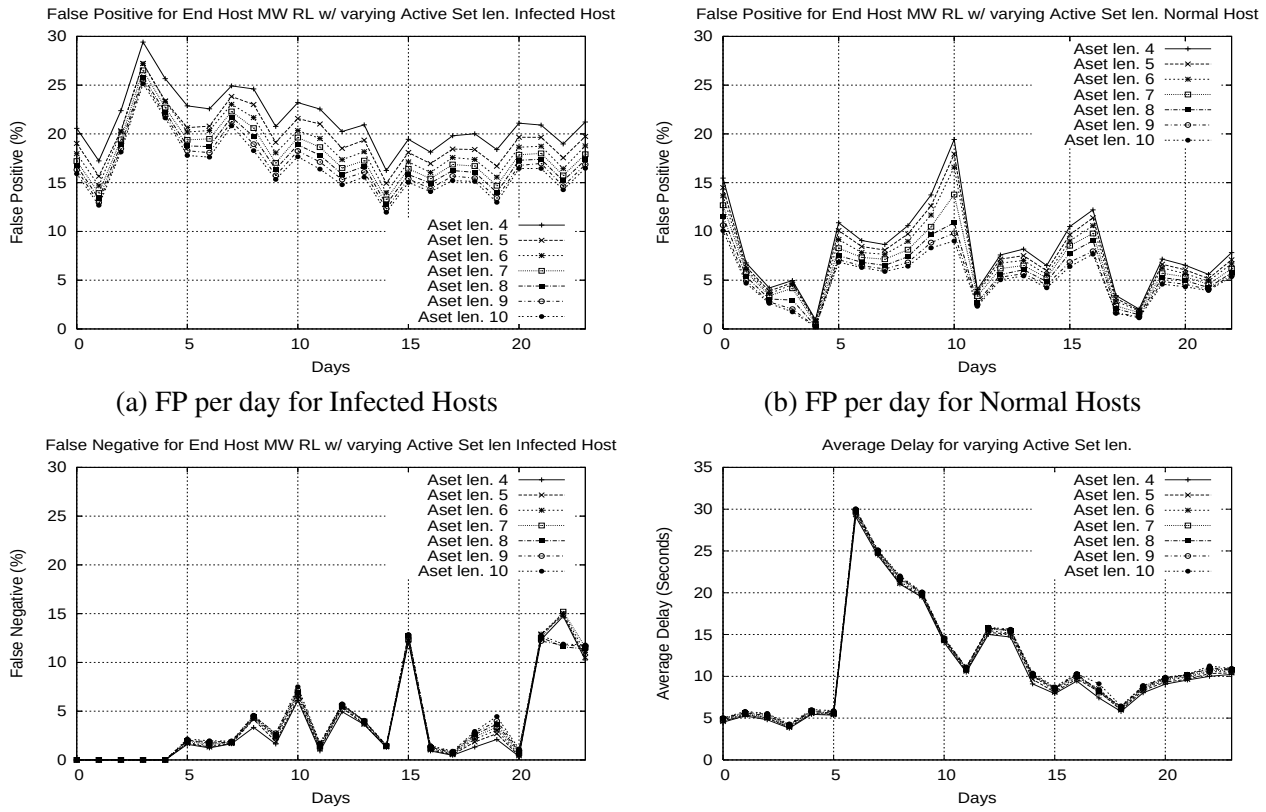
For each scheme analyzed here, there exists a set of parameters that critically impact the performance of the mechanism. We identify these parameters and evaluate the sensitivity of the error rates with respect to each parameter. For instance, Chen's failed-connection-based scheme rate limits those hosts whose rate of failed connections goes beyond a pre-defined threshold. Clearly, the value of the threshold is important here—a threshold set too high will permit malicious traffic through thereby resulting in a high false negative rate while an overly low threshold will produce a high false positive rate. 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 to minimize worm propagation while limiting the interference to normal traffic.

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) [26]. Restricting host-level contact rates to unique IPs can limit rapid connections to random addresses. Williamson accomplishes this by keeping an *active 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 active 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 Williamson). At the same rate, the least recently used address in the active 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 active set and the delay queue are important. A larger active set permits a higher contact rate while the length of the delay queue determines how liberal (or restrictive) the scheme is. Williamson recommended a five-address active set and a delay queue length of 100 for the host-based scheme. 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 the aggregate Blaster/Welchia trace and simulated Williamson's rate limiting scheme on these traces. Blaster and Welchia had a significant impact on the network we studied; over 100 hosts were found infected with Blaster and/or Welchia. In our discussion of Williamson's mechanism, rate limited traffic is any traffic delayed or dropped.



(a) FP per day for Infected Hosts

(b) FP per day for Normal Hosts

(c) FN per day for Infected Hosts

(d) Avg. delay per packet (Infected Hosts)

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

Figure 2(a) shows the daily false positive rate of the throttling scheme with the size of the active set ranging from 4 to 10. Again, false positive rates are calculated as the ratio between benign traffic rate limited and all benign traffic within the same time period. Recall that Blaster hit on the 5th day of the trace period. The data points in Figure 2(a) are daily averages across all infected hosts. For comparison reasons, we also tested Williamson's scheme on normal host traffic, the result of which are shown in Figure 2(b).

A few high-level insights are important here: First, our results suggest that the false positive rates for infected hosts hover in the range of 20% to 30%, which is a non-trivial percentage. As an example, Table 1 shows the false positives for day 6 of our trace period (the day of infection). Analysis of false positives for client machines indicate that Williamson's scheme performs poorly with extremely active web clients. For instance, row 1 of Table 1 shows data for a bursty desktop client that runs a web browser and instant messaging simultaneously. As shown, 39% of the legitimate traffic for this client was subjected to rate limiting. A closer inspection of the false positives reveals another interesting factor; Williamson's scheme attemps to rate limit every packet, as opposed to only the SYN's. As a result, approximately 99% of false positives occurred on packets of an ongoing session being rate limited because the destination IP was evicted prematurely out of the active set. The second observation from the false positive results is also interesting. Note that the infection occurred on day 5, but we do not observe a visible increase in the false positive rate; that is, the pre-infection false positive rate is comparable to what is observed during infection. This result speaks positively of Williamson's scheme as infections do not seem to cause more normal traffic to be delayed. Finally, the results in Figure 2 show that the size of the active set (at least for the values experimented here) has minimum effect on the error rates of the scheme. This is partially due to the fact that

we averaged statistics across hosts. In practice, one can observe the connection pattern of a particular host for some period of time before determining the optimal active set size.

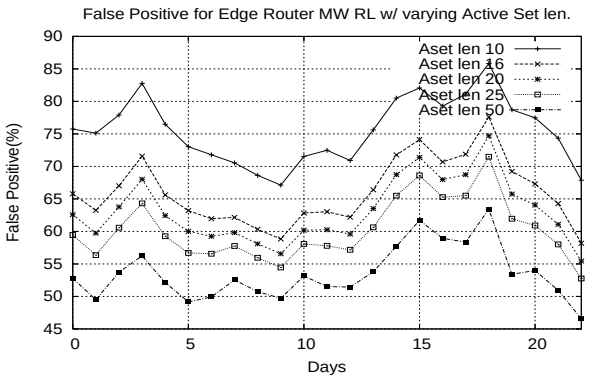
Figure 2(c) shows the false negative rates (averaged over infected hosts) for the same time period. Again, false negative rates are calculated as the ratio between worm traffic that is not rate limited and all worm traffic within the same time period. As shown, the false negatives are predominantly below 10% with a few days reaching into the neighborhood of 20%. We also recorded the average delay per packet (calculated on a daily basis) and the number of dropped flows per day. The average delay statistics are shown in Figure 2(d). During infection, our record shows that 5% of benign traffic and 10.9% of malicious traffic were dropped every day. These statistics indicate that during the Blaster outbreak, 89% of worm traffic was allowed to exit our network, with delays of approximately 18 seconds each. Our traffic data shows that Blaster scanned at a rate of 10 - 20 scans per second and Welchia at 70 scans per second, the delay and drop statistics of Williamson's scheme suggests that it is insufficient to contain the worm.

A straightforward means to induce further delays is to reduce the rate at which connections are dequeued. However, this runs the risk of starving legitimate traffic. A plausible alternative is to associate a time-to-live (TTL) field with active set addresses. Each time a new connection is made for an address in the set, the TTL for that address is renewed. An address is evicted from the active set only when its lifetime has expired, and only then is a new connection dequeued and processed. This strategy gives rise to a dynamically changing dequeuing rate that is more aligned with the dynamic behavior of the host traffic. As a result, frequently contacted addresses would have a higher probability of remaining in the active set while worm traffic can be delayed for a longer period of time. The exact implications of this improvement will require further studies. A discussion on dynamic vs. static rates can be found in Section 9.

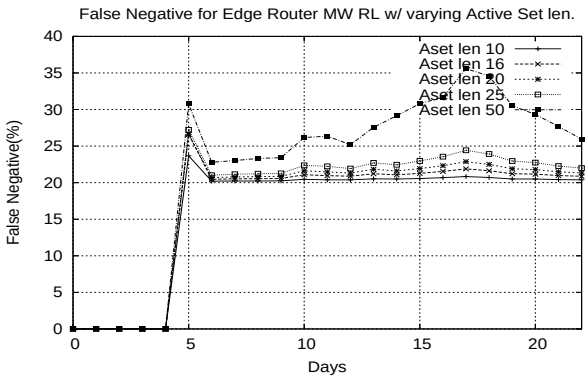
Throttling at the Edge Router Previous studies [15, 28] showed that end-host rate limiting is ineffective unless universal deployment is feasible. Hewlett Packard's recent announcement to withdraw support for the throttling product is indicative of the inherent difficulties that underscore universal deployment [29]. 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 an attractive alternative because it requires the instrumentation of only the ingress/egress point of the subnet. Furthermore, aggregate throttling requires a sublinear amount of states (sublinear to the number of hosts). We note that the logic can be extended to the ingress/egress point of a network cell within an enterprise, as shown in [18], which can provide additional protection granularity.

In this analysis we do not differentiate traces from different hosts—the rate limiting algorithm is applied to the aggregate traffic exactly the same as a host would apply it to host level traffic. In a previous traffic study, we identified a candidate rate of 16 addresses per five seconds for edge throttling for a similar network [28]. In the analysis that follow, we present results obtained with four rate limits: 10, 16, 20, and 25 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. 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 further increase the active set size at the edge to reduce the false positives. However, that will incur more false negatives as a 50-address active set already renders a 30% false negative rate (see Figure 3(b)). At the height of infection for this network, 30% of the worm traffic equate to 3.2 million scans per day. With that many scans permitted through without delay, an exponential spread of infection is highly likely; the rate limiting scheme has essentially failed.



(a) FP perday at Edge Router



(b) FN per day at Edge Router

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

6 Failed Connection Rate Limiting (FC)

Chen et al. proposed a failed connection-based rate limiting scheme based on the assumption that a host infected by a scanning worm will generate a large number of failed TCP requests [4]. Their scheme uses this phenomenon as an indication of infection and rate limits hosts with such behavior. In the discussions that follow, we refer to this scheme as FC (for Failed Connection).

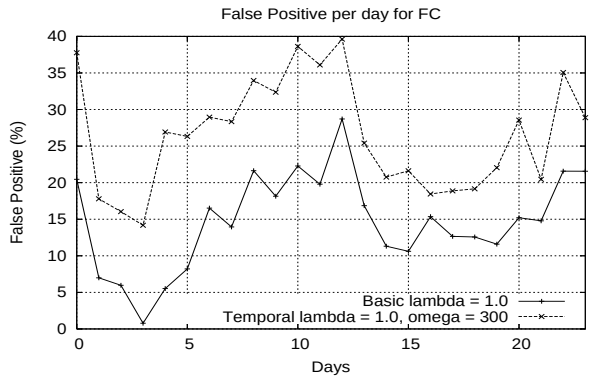
FC is edge router based and consists of two phases. The first phase identifies the potential "infected" hosts. During this phase a highly contended hash table is used to store failure statistics for hosts. The highly contended hash table is used to limit the amount of per-host state kept at the router. Once the failure rate for an entry in the hash table exceeds a certain threshold, the algorithm enters the second phase, which attempts to rate limit the hosts in the particular hash 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 failures per second, the rate-limiting engine at the edge router attempts to limit the failure rate of each host in the hash entry to at most λ. A leaky bucket algorithm is used to rate limit each host; a token is removed from the bucket for each failed connection and every λ second a new token is added to the bucket. Once the bucket for a particular host is empty, further outgoing connections originating from that host are dropped.

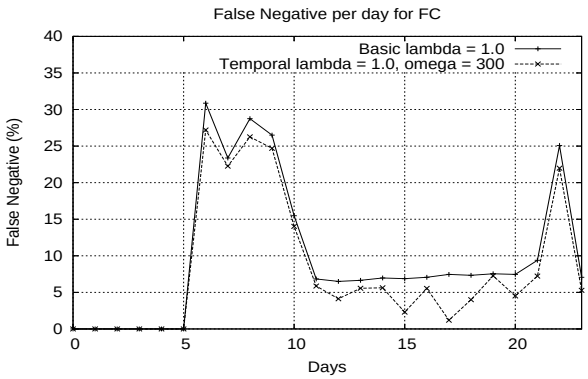
Temporal FC attempts to limit both the short term failure rate λ and a longer term rate Ω. In their algorithm, Ω is a daily rate while λ is a per second rate. The value of Ω is intended to be significantly smaller than λ * (total seconds in a day). Hosts in a hash table entry are subjected to rate limiting 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 rate of λ. One can also adjust λ to a higher value to accommodate temporary burstiness of failed connections while keeping the failure rate over a longer period relatively low.

To evaluate these two algorithms we conducted experiments with the aggregate Blaster/Welchia data set, with varying values of λ and Ω. The hash table size for the network is set to 256, approximately 5 hosts are mapped to the same entry. We note that the hash table is simply a way to reduce the amount of state. Since non-infected hosts rarely make more than one failed connection per second, hash collision has very little effect on false positives.

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) suggests an increase in the false positive rates during the first



(a) FP per day



(b) FN per day

Figure 4: Error rates of Basic and Temporal FC RL algorithms with $\lambda = 1.0$ & $\Omega = 300$. per day

Anon IP	# Benign Flows Dropped		Total # of Benign 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	Benign Flows from Infected Client

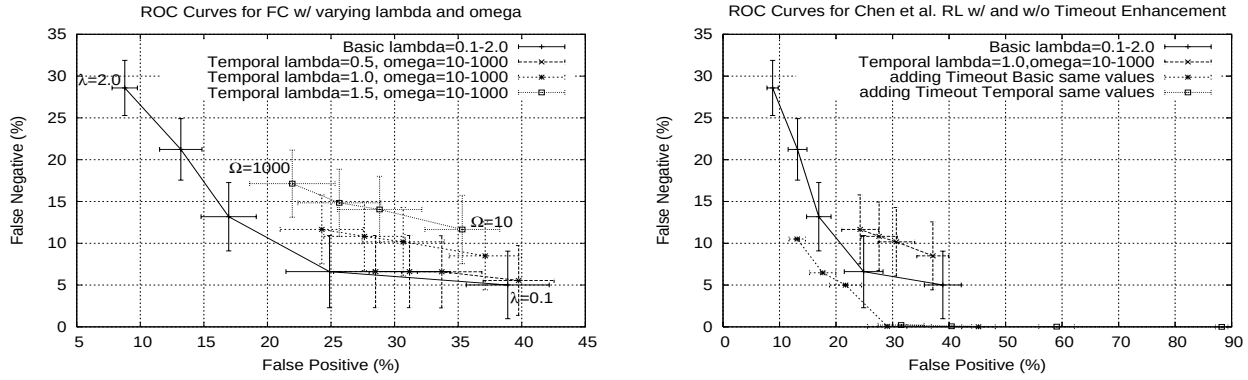
Table 2: Per Host False Positives and Cause for Day 6 for Basic & Temporal $\lambda = 1.0$ and $\Omega = 300$

week of infection. The increase in the false positive rates can be attributed to the fact that worm traffic on an infected host generats failed connections at such a rapid speed that they completely deplete the tokens in the bucket; a majority of the legitimate connections are thus dropped. This remains true even when the bucket size is increased from 10 to 100 tokens.

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 reach a more reasonable level. The bulk of false negatives can be attributed to the fact that Chen's schemes call for only a TCP_RST packet as an indication of a failed connection. Since many firewalls simply drop packets instead of responding with TCP_RSTs, using TCP_RSTs exclusively underestimates the number of failed connections. To confirm this observation, we also conducted another experiment that augments Chen's scheme such that TCP timeout was also considered a failure. The result of the experiment is shown in Figure 5(b), which indicates that considering TCP timeouts also as failures greatly reduced the false negatives.

The dramatic drop of false negatives at day 10 can be correlated with the Welchia outbreak. Blaster scans by sending TCP packets to port 135. This behavior elicits numerous timeouts from hosts behind firewalls and few TCP_RSTs, as evidenced by Figure 5(b). However, due to the different scanning behavior of Welchia, the false negatives are reduced on day 10 when Welchia is unleashed. Welchia first attempts to identify hosts using ICMP ECHO requests. If a response is elicited, Welchia attempts to connect to the machine, which would effect a TCP_RST as opposed to a TIMEOUT (since the host does exist, as evidenced by the ping). As more machines are infected with Welchia, and Blaster machines are patched, a greater portion of worm traffic causes TCP_RST responses, which Chen's scheme considers a failed connection and thus lowers the false negatives. In addition, since Chen's mechanism was only applied to TCP flows, the ICMP scanning will not use up any tokens when "pinging" a host.

Figure 5 plots the false positive rates against the false negative rates in a Receiver Operator Curve with varying values for λ and Ω. The data points in this graph are averaged daily statistics over the trace period. Due to the fact that temporal is more restrictive towards the creation of failure replies, a significant amount of non-worm traffic from hosts identified as "infected" is dropped, as evidenced by Table 2. On average temporal drops nearly all benign traffic once the host is identified as infected.



(a) ROC Curve for varying values of λ and Ω

(b) ROC Curve w/ and w/o timeout enhancement

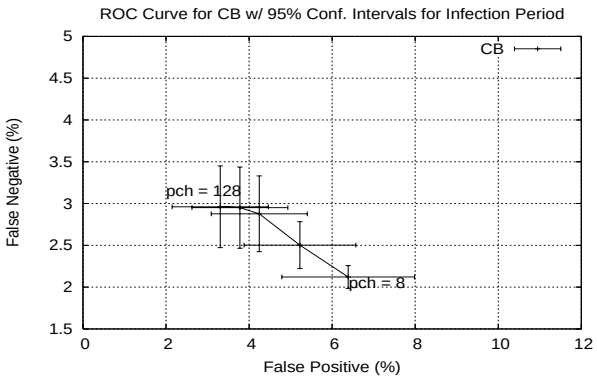
Figure 5: ROC Curve for different λ and Ω values for Basic and Temporal RL algorithms

Comparing FC results to host-based Williamson's, we can see that the two rendered similar false positive rates while FC has a higher false negative rate due to its inaccurate failure indications. However, FC affords a lower storage overhead by not keeping per-host state. A similar modification can be made to Williamson's to reduce the storage requirement. One can envision an edge-based implementation that keeps an aggregate active set for a group of hosts. Only when the delay queue exceeds a certain size, is a per-host active set instantiated (for that group of hosts). Such an augmentation will be straightforward to implement.

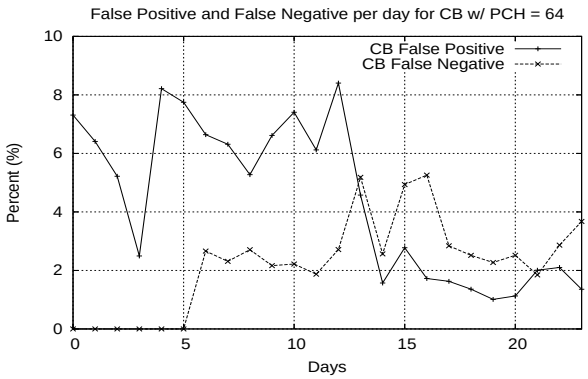
7 Credit-based Rate Limiting (CB)

Another rate limiting scheme based on failed connection statistics is the credit-based scheme by Schechter et. al. [16]. 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 addresses that have not been visited previously. The underlying observation here is that scanning worms produce a large volume of failed connections, but more specifically they produce failed first-contact connections. Consequently, anomalous first-contact statistics are indicative of scanning behavior and can be used as an impetus for rate limiting. The notion of first contact is fundamental to CB and as we show later is instrumental to its every aspect. 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 first contact adds one. 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, all non-first-contact connections (typically legitimate traffic) are permitted through irrespective of the credit balance. Consequently, a scanning worm producing a large number of unsuccessful TCP requests 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.



(a) ROC Curve w/ varying PCH sizes



(b) Error rates for PCH = 64 (recommended)

Figure 6: Results of Error Rates for CB RL

Anon IP	# Benign Flows Blocked	Total # of Benign 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 3: Per Host False Positives and Cause for Day 6 for PCH = 64

The CB scheme is an edge-router implementation that operates on per host statistics. The edge router instrumentation maintains a credit bank for each internal host. The level of initial failure credits per host (set as 10 in CB scheme) has minimal impact on the performance of the scheme, as that only approximates the number of failures that can occur throughout a given time period—a host can accrue more credits by initiating more successful first-contact connections. Our preliminary investigation shows that an initial credit level between 5 and 20 has negligible impact on the false positive and false negative rates.

To determine whether an outgoing TCP request is a first-contact connection, CB maintains a connection history PCH (Previously Contacted Host) for each host. The length of the PCH is a critical parameter, as it directly impacts the level of false positives—a shorter PCH results in more non-first-contact connections to be identified as first contacts, thereby permitting less normal traffic through when the worm exhausts the available credits. Schechter suggested a 64-address PCH to balance storage and false positive considerations. We conducted experiments with PCH sizes ranging between 8 and 128 entries. A LRU replacement algorithm is used to refresh addresses in the PCH.

Figure 6(b) shows the daily false positive and false negative rates for a PCH of 64 addresses. The data points in this graph are averages across all hosts. As shown, the average false positive and false negative rates are between 2% and 6%. These results outperform both FC and Williamson's scheme. This is primarily due to its strategy of rate limiting first contacts rather than distinct IPs or 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 3 shows the false positive data for the top twos hosts along with a HTTP client. The statistics show that the worst case false positive rate is rather high—nearly 40% for the host in row one of Table 3. Note however, the clients incur exceptionally high false positives are both P2P clients. For a normal HTTP client (row three in the table), the false positive rate is zero. This last row is a drastic improvement over the false positive rate the same host suffers under FC's schemes.

Figure 6(a) plots the average false positive rates against the corresponding false negative rates for PCH sizes 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 five pre-infection days). As shown, CB's error rates are not particularly sensitive to the length of the PCH's. A 3% increase in the false positive value is observed when PCH is reduced from 128 entries to 8. As the PCH size increases so does the false negative. This is because any connection is allowed to occur if its destination address is in the PCH. With a larger PCH, an infected client has a higher probability of generating an IP that already exists in the PCH. In our analysis, using PCH sizes of 8 and 64 respectively doubles the probability of collisions within the PCH. Although the sequential scanning manner of Blaster forbids duplicate address generation, a possible error generating the mutex could have allowed multiple instances of Blaster to execute on a single machine, generating the duplicate destinations found in the PCH since the random address generators will be seeded by the same value from *GetTickCount()*.

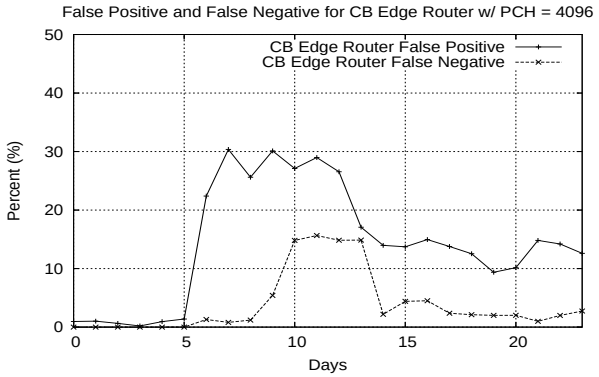


Figure 7: Error Rates for Aggregate CB with PCH = 4092

Note that since separate connection statistics are kept for each host, CB is fundamentally a centralized host-based scheme. Aggregating and correlating connection statistics across the network can reduce the storage overhead. 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 common scenario that legitimate applications (e.g., web browsing) on different hosts may visit identical external addresses (e.g., cnn.com). To test this scenario we evaluated CB from an edge router position with much larger PCHs. Figure 7 shows the false positives and false negatives for this edge based CB scheme with a PCH size of 4192. In addition, Figure 11 shows the ROC curve for the edge router based CB scheme with the PCH ranging from 128 to 8192 entries. As illustrated in Figure 11 the overall false positives increases compared to end host based CB. This is due to the fact that when using aggregate traffic statistics, is it much more difficult to isolate malicious flows from normal flows.

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 [27, 25, 8]. For instance, the non-existence of DNS lookups is a telltale sign for scanning activity.This observation was first made by Ganger et al. [7]. The scheme we analyze here is a modification of Ganger's NIC-based DNS detection scheme.

The high-level strategy of the rate limiting scheme is simple: for every outgoing TCP SYN, the rate

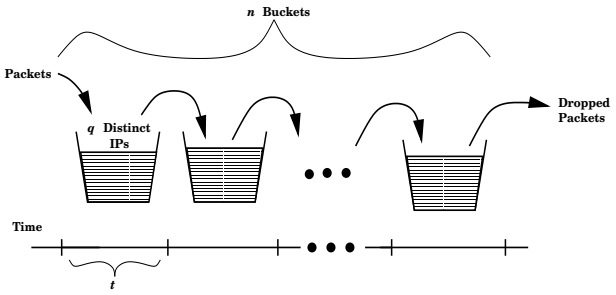


Figure 8: Cascading Bucket RL Scheme

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 8. 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. The n-th bucket, the last in line, has no overflow bucket and once it is full, new TCP SYN packets without DNS translations will simply be dropped. Packets in the i-th bucket are delayed until the beginning of the i+1 time interval. As all n buckets expire, they are reinstated 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 active sets or the failure rate of connections.

This DNS throttling scheme can be implemented at each host or at the edge router. A host-level implementation can be achieved by keeping DNS-related statistics on each host. Edge-router-based implementation would require a shadow DNS cache on the edge router to support access to DNS information.

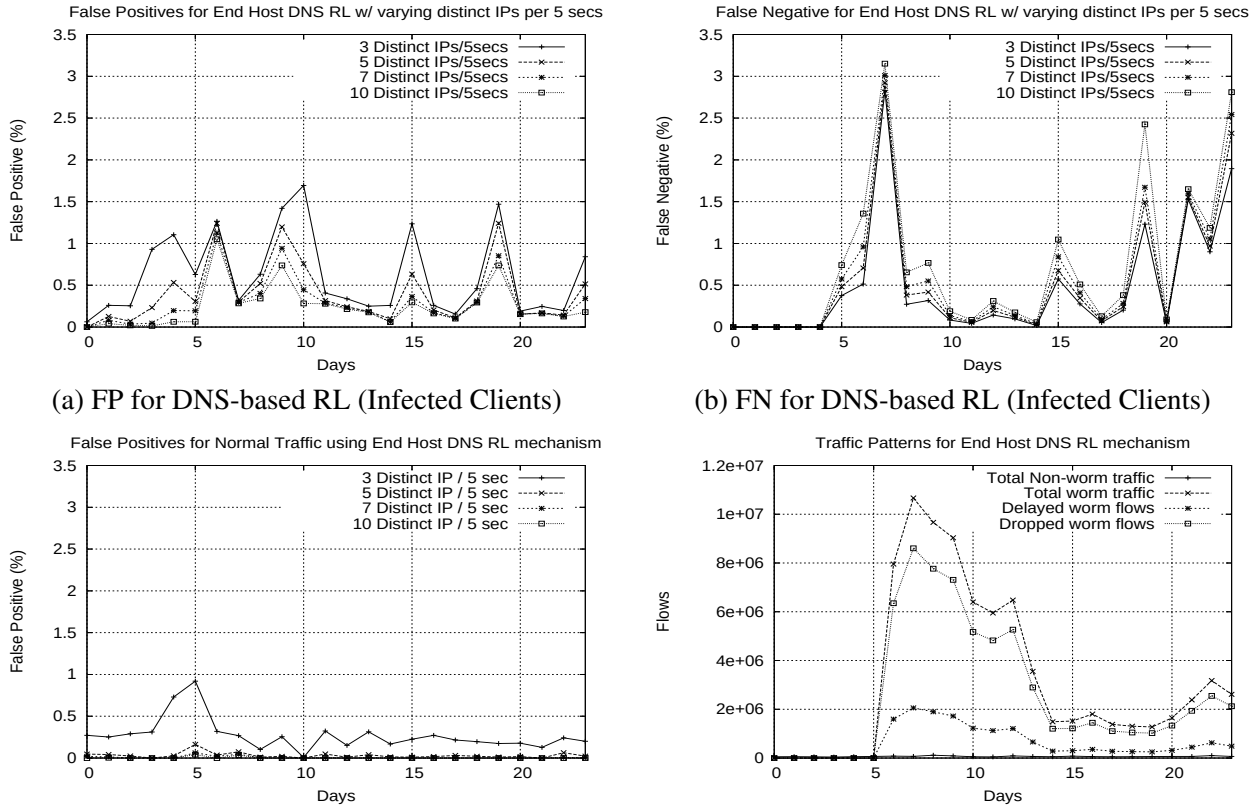
In our study, we implemented the throttling at the edge router, using DNS server cache information and all DNS traffic recorded at the network border[2]. More specifically, we mirrored the DNS cache at the edge and updated the cache as new DNS queries are recorded. For each DNS reply, we extracted the translated IP address, the host name, and the respective TTL from the incoming payload. This information is then entered into the mirrored DNS cache at the edge router. Traffic to destination addresses matching an unexpired DNS record was permitted through, while all other traffic was throttled.

It is worth noting that we only throttle outgoing TCP SYNs instead of all packets. Thus, a host can carry on with incoming or established connections, but its ability to initiate connections is restricted. As a result, existing connections do not suffer unnecessary delays or drops. The applications that do experience false positives from DNS throttling 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 therefore not critical to the network operation. Note that we only tackle TCP traffic, as we use TCP SYNs as a basis for rate limiting. Out of the four mechanisms analyzed in this paper, only Williamson's can be applied to UDP traffic without

[2]As mentioned in Section 3, we recorded all DNS payloads.

modification, due to the fact that both FC and CB both utilize TCP behavior for characterization.

Using this implementation, we tested both per-host and aggregate throttling. In the next section, we analyze the results.



(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 9: Results for DNS-based End Host RL

8.1 Analysis

The critical parameter for the cascading-bucket throttling 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[3]. Additionally, the value of $n*t$ was set to 120 seconds to model the TCP timeout period.

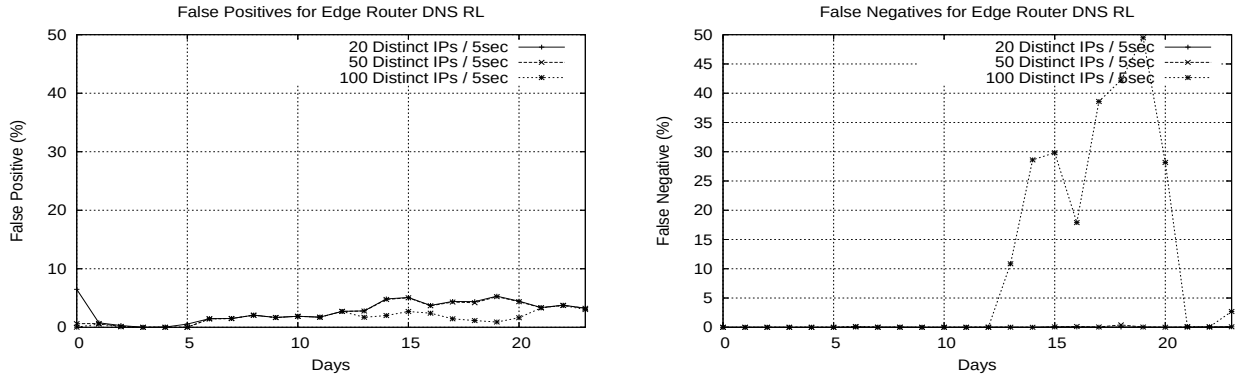
We note that the rate limits specify the number of untranslated IP connections allowed to exit the network. One could take a simpler approach and block all untranslated connections. Such an approach would be unsatisfactory for it fails to account for legitimate direct-IP connections. Examples include direct server-server communication and IP-embedding in HTML [25]. In our data set, we did not observe any embedded-IP in HTML, but direct IP server-server communications do occur. Peer-to-peer applications can also induce non-translated IP connections, which we observe is the primary cause of false positives for this scheme.

We first analyzed the host-level throttling scheme. For this, we maintain a set of cascading buckets for each host. Figure 9(a) and (b) show the false positive and false negative rates of host-based throttling on

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

Anon IP	# Benign Flows Delayed	Total # of Benign Flows	Cause
188.139.199.15	15572	57336	eDonkey Client
188.139.202.79	8958	33961	BearShare Client
188.139.173.123	0	15018	HTTP Client

Table 4: Per Host False Positives and Cause for Day 6 for $q = 7$



(a) FP for DNS-based RL

(b)FN for DNS-based RL

Figure 10: Results for DNS-based RL at the Edge Router

infected hosts. The data in these graphs are daily error rates averaged over all infected hosts. Figure 9(c) plots the analogous false positive rates on normal hosts.

These results yield a number of significant observations: First, host-level DNS throttling significantly outperforms the other mechanisms analyzed previously. As seen in Figure 9, 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. It is important to note that even at the peak of infection (which we recorded 11 million of daily scans from infected hosts), nearly all legitimate flows are permitted through. Figure 9(d) shows summarized statistics from our analysis. During the trace period, nearly 100% of the malicious traffic is rate limited (80% are dropped out right and the other 20% are delayed with an average delay of one minute per connection).

Table 4 shows the example false positive statistics for the host DNS throttling. As shown, the top ranked false positives occurred with peer-to-peer clients. Note that the HTTP client in row three induced a 39% false positive rate with Williamson's scheme (see Table 1). Here the HTTP client is not subjected to rate limiting.

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 [24]. The results in Figure 9 show that, using values of $q = 3$ and $t = 5$, the scheme renders false positive and false negative rates lower than 1%. This means that a worm will have to scan slower than approximately 0.5 scans per second for it to evade rate limiting. Additionally, approximate TRW can detect a worm after 10 scans. If the DNS scheme simply blocks all non-translated traffic, it would have blocked the worm in less than 1 scan[4]. As discussed previously, this approach is suboptimal. We note that if the white-list is perfect; that is, all legitimate direct-IP connections are documented in the white list, there would be no need for rate limiting; one can simply block all untranslated IP connections and achieve

[4]Whyte et al. also observed this [25].

nearly zero error rates. However, since a perfect white-list is unattainable in practice, rate limiting offers a reasonable alternative. Note that the system administrator can set the rate parameters according to the particular applications running in the network, and potentially tighten the false negative rate.

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 20, 50, and 100 IPs per five second window. Figure 10 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. We note that a sweet spot exists for the aggregate case—it seems to lie somewhere between 50 and 100 IPs per five seconds. Also 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 due to the fact that the aggregate traffic limit is more restrictive than the collective limit in the host-based case. Although the aggregate false positive rates are slightly higher than the host-level case, we find the aggregate results extremely encouraging. The error rates of 5% false positive and $< 1\%$ false positive, means that DNS-based throttling is entirely suitable for an aggregate implementation.

We note that DNS statistics can be extended to contain mass-mailing worms by rate limiting MX lookups [25, 27]. In a tightly controlled network, however, port-25 filtering may be more effective. The extension of DNS-based throttling on MX lookups is out of the immediate scope of this paper.

9 Discussion

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 the general insights from the study.

DNS-based Rate Limiting vs. others: A summary comparison of the DNS-based scheme with the others is in Figure 11. The parameters here are consistent with the values used in the previous sections. Again, the false positives and false negatives are averaged daily statistics over the trace period. As shown, the ROC curve for DNS-based rate limiting is significantly closer to the origin than the others, which indicates vastly superior false positive and false negative rates. More specifically, host-based DNS throttling renders an average false positive and false negative rate below 1%. These results indicate an extremely strong case for DNS-based rate limiting. In addition, DNS-based rate limiting allows most benign traffic to pass through even at the peak of a worm outbreak. This is an extremely desirable characteristic as productivity of users will not be affected. In addition, it is also viable to utilize first contact statistics (such as CB). As this allows for previously contacted connections to pass through without delay and thus minimizing the impact on normal user traffic.

Recall that the q value in DNS throttling allows for q untranslated IPs per host to exit the network every t seconds. To put things in perspective, for the first day of infection, a total of 468,300 outbound legitimate flows are generated for the entire network. Out of these, a total of 463 flows are dropped when the value of q is set to 7, which yields a false positive rate of 0.099%. Note that this is less than 1 dropped flow per host per day. As a comparison, the CB scheme dropped a total of 3767 legitimate flows for the same day, which translates to a false positive rate of 7.8%.

The dramatic difference in the performance can be attributed primarily to the fact that DNS traffic patterns, compared to other statistics, more precisely delineate worm traffic from normal behavior. Most network applications utilize DNS translations—direct connections without prior IP translations are rare. DNS-based rate limiting 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 IP space extremely rapidly in their search for potential victims. Navigation through the IP space can be easily automated

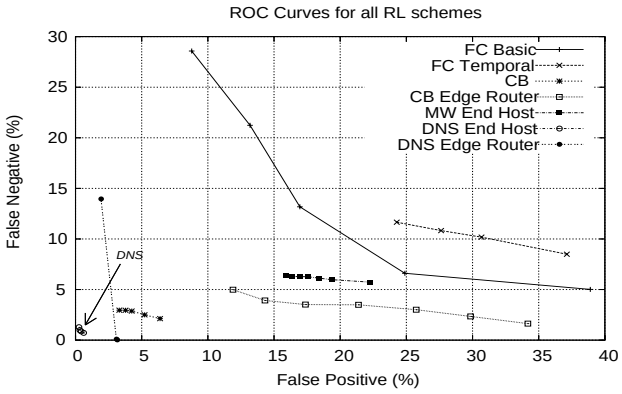


Figure 11: Average error rates for all RL schemes

because it contains numeric addresses only. The DNS name space, on the other hand, is less populated and has poorer locality properties. Navigating in this space is a far more difficult process to automate. 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 to propagate. A potential 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, then that in itself is a positive testimony for DNS-based throttling.

Issues with DNS-based rate limiting: There are various ways an attacker can attempt to circumvent the DNS rate limiting mechanism. We discuss a few of them here.

First, a malicious worm could use reverse DNS-lookups (PTR lookups) to "pretend" that it has received a DNS translation for a destination IP. Jung et. al. [10] reports that around 24 - 31% of DNS queries are PTR lookups. They characterize these 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 each infection attempt will significantly slow down the infection spread.

Second, an attacker could setup a fake external DNS server and issue a DNS query to the "DNS" server for each generated IP. This type of set up can be easily filtered at the border by establishing a "white-list" of legitimate external DNS servers. In addition, the attacker needs a server with a substantial bandwidth to accommodate the scan speed. Such a server is not trivial to obtain.

In order to accommodate SOHO (Small office Home office) users who use legitimate external DNS servers. One can implement a DNS query pairing mechanism, using open source packet scrubber such as Hogwash [9]. If the packet scrubber only sees a connection attempt without a DNS query at the router, this can be flagged as a non-dns lookup connection attempt. In addition, one could also use a whitelist for mobile users who do not want to use the local DNS servers.

Dynamic vs static rates: Rate limiting schemes generally impact the rate of both legitimate and malicious connections. We have already established that DNS-based throttling performs better than the others. One of the reasons is that DNS throttling has very little impact on legitimate traffic, it permits legitimate connections (as long as it has valid DNS translations) through at the same rate as the connections arrive at the rate limitor. For the other three schemes, however, the rate limits in effect imposed on legitimate traffic play an important

role in the performance of the scheme.

Williamson's imposes a strictly static rate, e.g., five distinct IPs per second, irrespective of the traffic demand. CB allows for a dynamic traffic rate by rewarding successful connection and penalizing failed connections. FC's mechanism renders a curious behavior; under normal operation, legitimate traffic is permitted through at its natural rate. As soon as infection hits, the outbound scans quickly exhausts the available tokens and blocks further connections altogether. As a result, a rate-limited host has an all-or-nothing behavior.

Figure 11 shows that CB outperforms both Williamson's and FC. This is primarily due to CB's dynamically changing 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.

A related point here is the use of first-contact statistics. CB confines rate limiting to first contact packets. Williamson's, however, attempts to rate limit each packet. As we discussed earlier, rate limiting packets for an established session is unnecessary and inconsistent with TCP semantic if the packets are delayed too long. Applying rate limiting on first contact flows rather than at the packet level allows a higher throughput of packets and potentially lower false positives.

Host vs aggregate: An issue of significance is host versus aggregate rate limiting at the subnet level. By aggregate rate limiting, we do not mean mechanisms that implement host-level throttling at the edge (such as CB [16]). Rather, we mean rate limiting based on aggregate traffic as seen at the edge router.

The general wisdom is that host-level throttling is more precise but is also 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. However, edge-based DNS throttling is an exception. Figure 11 shows that a carefully chosen rate limit, e.g., 50 IPs per five seconds, yields excellent precisions 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 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. We note that throttling at the edge imposes a stricter aggregate limit than the corresponding host-level scheme, therefore it can lead to a lower false negative rate as we observed with the DNS scheme. We remind the reader that our study did not include an analysis on processing overhead. Since edge-based schemes in general imply processing a larger amount of data per connection, a trade-off between storage and processing overhead exists. We also note that

A final point is 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 [18]) and apply the aggregate throttling at the border of each cell. Extending Whyte's ARP-based approach [25] to rate limiting is yet another option. Presently this work is primarily interested in mechanisms that deal with inter-network propagation. We leave the analysis of intra-network protection as future work.

Rate Limiting vs. Others: Many worm detection mechanisms have been proposed of late [11, 24, 25]. While some detection works are built on similar principles (e.g., TRW's failed connection statistics and Whyte's DNS-based detection) to rate limiting, the automated response component of rate limiting gives rise to a slightly different set of requirements. For example, a rate limiting scheme with high false positives

will severely disrupt network services. False positives for a detect-only scheme, while in a technical sense indicate the performance of the detector, in practice they may amount to nothing more than mere annoyance. As such, rate limiting and other automated response schemes have a more stringent error margin. Further, many detection schemes attempt to make host-level, coarse-grain decisions, i.e., a host is either infected or not. Rate limiting attempts to delineate malicious applications from legitimate ones so as to permit more fine-grained traffic filtering. As such, we find it beneficial to constrain our analysis to rate limiting mechanisms and not contrast explicitly with detection mechanisms in this work.

10 Summary

A number of rate limiting schemes have been recently been proposed to mitigate random scanning worms. In this paper, we present an empirical analysis of the different schemes, using real traffic and attack traces. We evalutate and contrast the false positive and false negative rates for each scheme. Our analysis reveals these high-level insights. First, DNS behavior-based rate limiting renders by far the most accurate results. Second, it is feasible to implement effective rate limiting on aggregate traffic at the network border, as indicated by the DNS analysis. This is encouraging because aggregate rate limiting alleviates the universal participation requirement thought necessary for worm containment. Third, schemes that offer dynamic rate restrictions in general offer lower false positive and false negative rates than those that impose static rates.

We note that the conclusions and results from this analysis are invariably affected by the particulars of our trace data. We plan to further our study by including other independent data sets. Other future work of interest includes designing streaming algorithms for worm detection and rate limiting to improve upon current results.

11 Acknowledgments

We would like to thank Greg Ganger for comments on an earlier draft of this paper. And thanks to Srinivasan Seshan for discussions on DNS-based rate limiting.

References

[1] Network Associates and 2003-08. W32/Sobig.f@MM.

[2] Network Associates and 2004-01. W32/Mydoom@MM.

[3] CERT. CERT Advisory CA-2003-04 MS-SQL Server Worm, January 25, 2003.

[4] Shigang Chen and Yong Tang. Slowing Down Internet Worms.

[5] Zesheng Chen, Lixin Gao, and Kevin Kwiat. Modeling the Spread of Active Worms.

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

[7] Gregory R Ganger, Gregg Economou, and Stanley M Bielski. *Self-Securing Network Interfaces: What, Why and How*. Technical report.

[8] Gregory R Ganger, Gregg Economou, and Stanley M Bielski. Self-securing Network Interfaces: What, Why and How.

[9] Hogwash. Inline packet scrubber.

[10] Hari Balakrishnan Jaeyeon Jung, Emil Sit and Robert Morris. DNS Performance and the Effectiveness of Caching.

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

[12] Jeffrey O Kephart and Steve R White. Directed-Graph Epidemiological Models of Computer Viruses. Pages 343-359.

[13] Hyang-Ah Kim and Brad Karp. Autograph: Toward Automated, Distributed Worm Signature Detection.

[14] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. Inside the Slammer Worm.

[15] David Moore, Colleen Shannon, Geoffrey Voelker, and Stefan Savage. Internet Quarantine: Requirements for Containing Self-Propagating Code.

[16] Stuart E. Schechter, Jaeyeon Jung, and Arthur W. Berger. Fast Detection of Scanning Worm Infections.

[17] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. Automated Worm Fingerprinting. Proceedings of the 6th ACM/USENIX Symposium on Operating System Design and Implementation.

[18] Stuart Staniford. Containment of Scanning Worms in Enterprise Networks. *Journal of Computer Science*.

[19] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to 0wn the Internet in Your Spare Time.

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

[21] Helen J. Wang, Chuanxiong Guo, Daniel R. Simon, and Alf Zugenmaier. Shield: vulnerability-driven network filters for preventing known vulnerability exploits. Pages 193–204. ACM Press.

[22] Yang Wang, Deepayan Chakrabarti, Chenxi Wang, and Christos Faloutsos. Epidemic Spreading in Real Networks: An Eigenvalue Viewpoint.

[23] Yang Wang and Chenxi Wang. Modeling the effects of timing parameters on virus propagation. Pages 61–66. ACM Press.

[24] Nicholas Weaver, Stuart Staniford, and Vern Paxson. Very Fast Containment of Scanning Worms.

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

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

[27] Cynthia Wong, Stanley M Bielski, Jonathan McCune, and Chenxi Wang. A Study of Mass-mailing Worms. ACM Press.

[28] Cynthia Wong, Chenxi Wang, Dawn Song, Stanley M Bielski, and Gregory R Ganger. Dynamic Quarantine of Internet Worms.

[29] PC World. HP Shelves Virus Throttler, August 24, 2004.

[30] Cliff Changchun Zou, Weibo Gong, and Don Towsley. Code Red Worm Propagation Modeling and Analysis.