A Comparison of SYN Flood Detection Algorithms

Matt Beaumont-Gay
UCLA Computer Science
mattb@cs.ucla.edu

Abstract

The problem of detecting distributed denial of service (DDoS) attacks, and particularly SYN flood attacks, has received much attention in current literature. A variety of algorithms for detecting such attacks have been published. Researchers have tested their own algorithms using traces containing real or synthetic attacks, and have reported good results based on those tests. However, the traces used and parameters of the attacks seen or generated vary greatly between published works. This paper compares three published SYN flood detection algorithms using traces collected from the UCLA Computer Science Department network and synthetic attacks in an Emulab network. The algorithms vary significantly in the speed at which they detect the start and end of attacks, their false positive and false negative rates, the types of non-DDoS activity they detect, and other properties. Their qualitative strengths and weaknesses are discussed, and suggestions are made for enhancements.

1. Introduction

A SYN flood attack is one in which an attacker sends a large number of TCP SYN packets to a victim. This causes the victim to use scarce resources (CPU time, bandwidth, and, in the absence of SYN cookies [1], memory) to respond to the attacker's SYNs. If the attack rate is high enough, the server will begin to drop excess SYNs, and legitimate clients will be unable to connect, leading to a denial of service.

The network security research community has proposed many methods of detecting and preventing such attacks. To test and validate their designs, researchers have used simulation, testbed networks, and real network traces. However, there is no single agreed-upon scenario, or even a single general methodology, for testing new defense systems. Simulations have many tunable parameters; testbed networks can be set up in arbitrary topologies and with any commercially available equipment; and traces used by researchers are not only highly variable in their contents but are also sometimes simply too old to be relevant to modern network traffic.

I selected and implemented three SYN flood detection algorithms from the network security literature. I then evaluated the three algorithms with attacks run in an Emulab [8] testbed network and with relatively recent network traces from the UCLA Computer Science Department network. I make no claim that these particular experiments are the gold standard of network security system evaluation. However, by testing the algorithms under identical conditions, I can make strong claims about their relative performance.

2. Descriptions of Algorithms Tested

I selected three different algorithms for this study. The algorithms were chosen because they each possess certain similar characteristics. The primary similarity is that each algorithm is intended to detect the same kind of attack, SYN flooding. Also, each is designed to be deployed at the edge of a leaf network, uses a constant amount of state, operates on a time scale of tens of seconds, and uses only TCP control packets (SYNs, SYN/ACKs, FINs, and RSTs) as its input.

All of the algorithms have a handful of tunable parameters. The authors of each paper discuss how the parameters are tuned and select particular values for their own experiments; I use these values where possible.

2.1. SynFinDiff

Wang, Zhang, and Shin present an algorithm in [7] which will be referred to as *SynFinDiff*. The core of their algorithm is the CUSUM method described in [2], which belongs to the class of sequential change point detection methods. Roughly speaking, the CUSUM method determines when the mean μ_0 of some independent Gaussian random variables changes to $\mu_1 \neq \mu_0$. The challenge, then, is to distill network traffic down to a Gaussian random variable with a mean that is stationary during normal operation but that changes when a SYN flood attack begins.

Wang et al. use the difference between the count of incoming SYNs and that of outbound FINs, Δ_n, over an observation period of length t_0. The collection period for FINs begins at an offset of t_d later than the start of the collection period for SYNs to allow for the longevity of TCP connections. Δ_n is normalized by an exponentially weighted moving average (EWMA) of the number of FINs seen in past observation periods, $\bar{F}$. They define $\tilde{X}_n = \Delta_n/\bar{F} - a$, where a is a constant chosen to make the mean of $\tilde{X}_n$ negative during normal operation. They then define $y_0 = 0$ and $y_n = (y_{n-1} + \tilde{X}_n)^+$ (for $n > 0$; x^+ is x if $x > 0$ and 0 otherwise). The algorithm reports an attack if $y_n > N$ for some threshold N.

2.2. SynRate

The second algorithm is the CUSUM-based algorithm in [6] by Siris and Papagalou, hereafter termed *SynRate*. It is similar to SynFinDiff, and in fact Siris and Papagalou explicitly compare their algorithm to that of Wang et al., claiming better performance from their algorithm. The statistic that Siris and Papagalou feed to CUSUM is the number of incoming SYNs seen in a 10-second interval, x_n, minus an EWMA of SYN counts from past intervals, $\bar{\mu}_{n-1}$. The intuition is that the EWMA gives a likely value for the next SYN count, and a significant deviation from that likely value is defined as an anomaly.

The final equation derived by Siris and Papagalou for the test statistic g_n is

$$g_n = \left[g_{n-1} + \frac{\alpha\bar{\mu}_{n-1}}{\sigma^2}\left(x_n - \bar{\mu}_{n-1} - \frac{\alpha\bar{\mu}_{n-1}}{2} \right) \right]^+$$

where α is an "amplitude percentage parameter" corresponding to a "probable percentage of increase of the mean rate" after an attack begins, and σ^2 is the variance of the x_i. The superscript $+$ indicates that the bracketed expression is forced to 0 if it is less than 0. Similarly to SynFinDiff, SynRate signals an attack when $g_n > h$, for a fixed threshold h.

2.3. PCF

The *PCF*, or Partial Completion Filter, was introduced by Kompella, Singh, and Varghese in [4]. PCFs are a probabilistic method of detecting significant numbers of SYNs without corresponding FINs (or, generally, significant numbers of the first of any paired operations without the second of the pair). A PCF is built from a set of k hash tables, termed "stages" by Kompella et al., which use independent hash functions. Each bucket is a counter. When a SYN is seen, for each stage, the pair of the destination IP address and destination port (<dstIP, dstport> for short) is hashed and the corresponding bucket is incremented. Likewise, when a FIN is seen, the same pair is hashed and the corresponding bucket is decremented. If, at any point, all of the buckets to which a <dstIP, dstport> pair hashes are greater than some threshold, the PCF reports an attack. All of the buckets are zeroed out after a fixed measurement interval.

3. Synthetic Attacks

I implemented the three algorithms listed in Section 2 in a program which took input from `tcpdump`. I set up a small Emulab [8] network with two HTTP servers and several clients, and ran SYN flood attacks against one of the servers with the attack detection program running on the border router of the servers' LAN.

3.1. Test Environment

Figure 1 depicts the network used in the attack tests. All nodes ran Linux 2.6.12; the servers, in particular, had SYN cookies [1] enabled. Nodes n-0 and n-1 were the HTTP servers. Nodes n-4 and n-5 were clients, each requesting a 1MB file from each web server approximately once per second for the duration of each experiment. Nodes n-6 and n-7 were attackers; at a specified time in each experiment, they both began a SYN flood against n-0. Nodes n-2 and n-3 acted as the border routers for the server and client/attacker networks, respectively. The SYN flood detection program ran on n-2. Packet loss by the `tcpdump` process was negligible in all experiments.

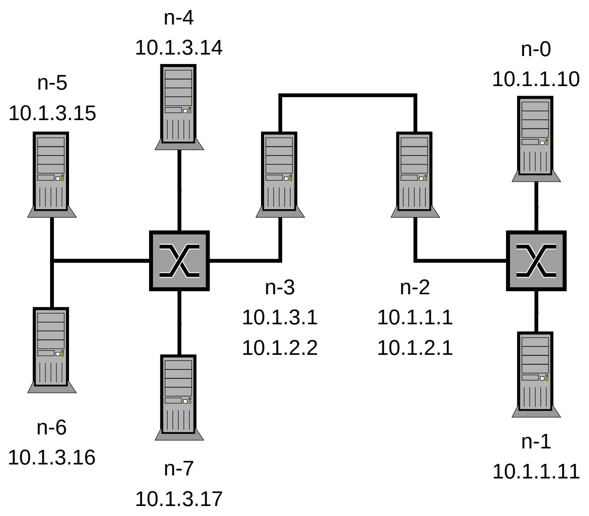

Figure 1. The Emulab network used for the synthetic tests.

3.2. Experiments

Each experiment consisted of three consecutive phases. The first phase, "warmup," lasted for 180 seconds. In this phase, the only traffic was the client nodes making HTTP requests to the servers. The SynFinDiff and SynRate algorithms used this time to stabilize their respective EWMAs. Next was the "attack" phase; the attacking nodes perpetrated their SYN floods during this time. This phase lasted 360 seconds. Finally, after the attacks ended, there was a 180-second "cooldown" phase, with the clients continuing to make requests to the web servers. This phase was included to observe how each algorithm responded to the end of an attack. Traffic generation was started approximately 5 seconds after the start of the measurement program to avoid edge effects related to the algorithms' measurement periods.

The primary variable across the experiments was the rate of attack. This rate ranged from 10 to 1200 SYNs per second from each attacker. In addition, I ran a series of experiments with each attacker using a linear ramp-up in its attack rate. The final rate was set at 1000 SYNs per second per attacker and the ramp-up time was varied from 15 to 360 seconds. Table 1 shows the full set of values used for these parameters. The "SYN Rate" column shows the total attack rate, i.e., twice the per-attacker rate.

Expt. ID	SYN Rate (SYN/s)	Ramp-Up Time (s)
1	20	N/A
2	200	N/A
3	600	N/A
4	1200	N/A
5	1600	N/A
6	2000	N/A
7	2400	N/A
8	2000	15
9	2000	30
10	2000	60
11	2000	180
12	2000	360

Table 1. Experiment parameters

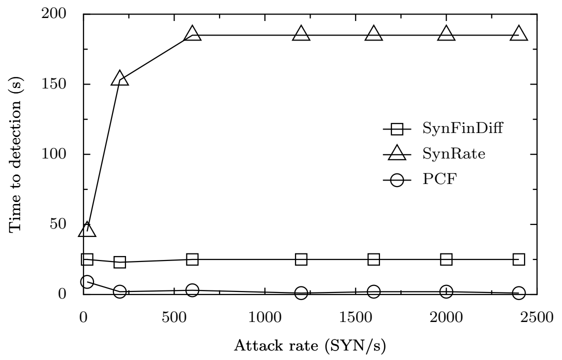

Figure 2. Time between attack start and attack detection as a function of attack rate.

3.3. Results

3.3.1. Detection Time. The speed with which an attack is reported is one of the most important metrics used to judge an attack detection algorithm. Figure 2 shows, for the attack rates used in Experiments 1 through 7 (the non-ramped experiments), the time between the start of the attack and the time at which each algorithm reported the attack. Both PCF and SynFinDiff show an essentially flat response time over most of the range tested, with SynFinDiff's response time being about 20 seconds slower than PCF's due to the former's 20-second sampling period. PCF takes 9 seconds to respond to the smallest (20 SYN/s) attack, versus 1 to 3 seconds for any higher rate.

The behavior of SynRate bears some explanation. In the equation for SynRate's test statistic g_n, the variance appears in the denominator of the term which is added to g_{n-1}. Thus, an increased variance results in a smaller increment to the test statistic. A larger jump in the traffic rate, as produced by the onset of a large attack, produces a much higher variance. The implications of this result will be discussed further in Section 5.

Figure 3 shows the time to detection for Experiments 8 through 12, in which the attackers increase their attack to 2000 SYN/s over a varying portion of the attack phase, as well as Experiment 6, in which the attackers immediately begin attacking at 2000 SYN/s (equivalent to a ramp-up time of 0). SynFinDiff is insensitive to the ramp-up time across the range tested, with a detection time of approximately 25 seconds in all experiments. PCF shows an approximately linear increase in detection time, from 3 seconds to detection for a 15-second ramp-up to 9 seconds for a 360-second ramp-up. SynRate again shows counterintuitive behavior: its detection time *decreases* dramatically as the ramp-up time increases. As with the results for the detection time in the non-ramped case, SynRate's behavior here is due to the variance term in the test statistic equation. A slower ramp-up gives a smaller variance, which in turn allows the test statistic to increase more quickly. Note, however, the rise in detection time as the ramp-up time goes from 0 to 15 seconds; here, the greater traffic volume seen with the 0-second ramp-up slightly outweighs the larger variance, leading to faster detection.

3.3.2. Quiescence Time. Another metric of interest is how fast an algorithm detects the end of an attack; that is, how long it takes to return to a non-alert or quiescent state after an attack has ended. For SynFinDiff and SynRate, a quiescent state is precisely when the test statistic is under the alert threshold. PCF has returned to a non-alert state when an entire measurement interval passes without an attack being detected.

SynFinDiff performs very poorly with regards to this metric. In none of the experiments did it return to a quiescent state before the end of the experiment. In each experiment, its test statistic rose to a level proportional to the volume of the attack, and then, after the attack had concluded, began to decline at a rate of 1 per 20-second observation period. As an extreme example, in the experiment with the largest attack volume, Experiment 7, extrapolation gives a time to quiescence of over six days.

Figure 4 shows the time to quiescence for SynRate in Experiments 2 through 7 and for PCF in Experiments 1 through 7. In Experiment 1, SynRate's test statistic was still above the detection threshold at the end of the experiment; it would have fallen below the threshold within a few minutes. Notably, PCF appears to detect the end of the attack in Experiment 1 in only 55 seconds. This is slightly misleading, as the end of the attack comes only 5 seconds after the beginning of a measurement interval, before PCF raises an alert for that interval. For the remainder of the experiments, PCF raises an alert less than 1 second after the beginning of the last interval containing a portion of the attack.

For Experiments 8 through 12, PCF has a constant quiescence time of 115 seconds, equal to the time it achieves for most of the non-ramped attacks. This is expected, as the attack rate at the end of the ramped attack is sufficient to cause immediate detection at the beginning of a measurement interval.

4. Trace-Driven Tests

The other major piece of work done for this study was to run each algorithm against a set of large

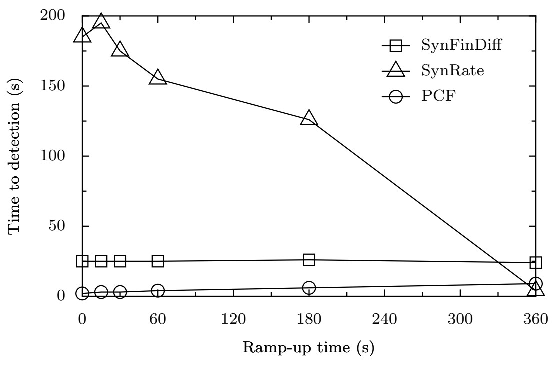

Figure 3. Time between attack start and attack detection as a function of the time taken for the attack to reach full strength (2000 SYN/s).

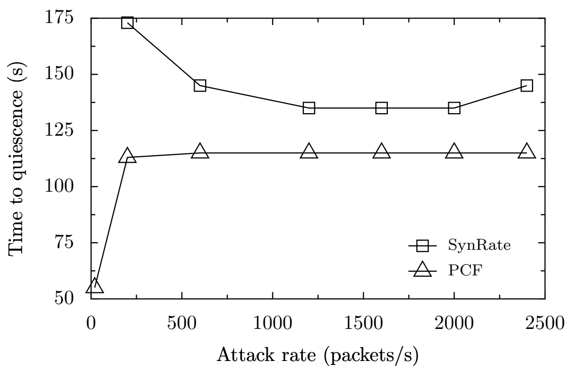

Figure 4. Time to quiescence.

network traces captured several years ago. The traces happen to contain several SYN flood attacks, as well as some other interesting features which affect the algorithms tested.

4.1. Traces Used

The four traces used in this portion of the study were collected at the border of the UCLA Computer Science Department network in August 2001. The size of each trace was limited by compressing it with `gzip` and truncating the compressed file to exactly 2GB, save Trace 2, which is only 1.97GB compressed. Thus, the traces vary in actual length, depending on the degree of redundancy in the traffic, and in duration, depending on the average packet rate seen. Some pertinent statistics on the traces are shown in Table 2. At most 128 bytes of each Ethernet frame were captured; this was sufficient to record the IP and transport-layer headers, and some application data.

One notable feature of Traces 1 and 2 is that they were captured while the Code Red worm [3] was active. The scanning activity from Code Red-infected hosts raised the average SYN rate seen during these traces well above the expected level given the number of FINs and RSTs seen. This caused significant problems for SynFinDiff: in each trace, a traffic anomaly causes the test statistic to rise well above the detection threshold, after which it stays at an elevated level for the remainder of the trace. I filtered out traffic on port 80 (the port scanned by Code Red) from the two affected traces to create Traces 1a and 2a, and re-ran SynFinDiff on the new traces.

4.2. Methodology

I ran all three algorithms on each trace, producing lists of time intervals during which each algorithm raised an alarm. For each time interval, I examined the corresponding portion of the trace to determine what caused the alarm. For many anomalies, it was sufficient to generate a frequency count of source or destination IP addresses or ports, or of some combination of those. During a port scan, for instance, one source address might show up an order of magnitude more often than any other address. If that technique failed to show an obvious anomaly, I inspected the trace in a graphical trace analysis tool. This was sufficient to identify the anomaly or to classify the alarm as a false positive.

The major weakness of this approach is that there may be anomalies in the traces that were not flagged by any of the algorithms. Therefore, my analysis of false negatives is optimistic, since the set of anomalies detected by all of the algorithms is a subset (though not necessarily a proper subset) of the anomalies actually present in the traces.

4.3. Results

Table 3 summarizes the alerts produced by each algorithm for each trace. The "In Trace" rows show the number of distinct events flagged by the algorithms.

4.3.1. SYN Floods. There were three clearly identifiable SYN flood attacks seen over the total 55-plus hours worth of traces — one in Trace 1 and two in Trace 2. All three attacks used spoofed source addresses. Two of the attacks targeted port 1214 (the port used by the Kazaa peer-to-peer file sharing service); the other targeted port 80. The peak attack rate seen was over 600 SYN/s. All of the attack traffic came in short bursts, the longest of which was 1.2 seconds.

All three algorithms nominally detected the attack in Trace 1; however, PCF signaled an alert for significant time periods before and after the attack as a result of a high rate of apparently legitimate traffic to the target host. PCF detected both of the attacks in Trace 2. SynRate detected the first, but not the second; the bursts in that attack were too small and widely spaced to keep SynRate's test statistic at an elevated level. SynFinDiff did not detect the first attack, since it was targeted at port 80 and thus filtered out of Trace 2a, but did detect the second. While its test statistic also did not rise very much due to any given burst, the small rises seen were enough to push it above the alert threshold.

Trace	Duration	Size	Packets	SYNs/FINs/RSTs
1	15:09:57	2.872GB	33941098	1324158
1a	15:09:57	N/A	N/A	554180
2	14:20:26	2.625GB	30884752	1196948
2a	14:20:26	N/A	N/A	476072
3	16:19:10	2.974GB	33738416	1311138
4	9:32:52	2.942GB	33332981	1312611

Table 2. Summary statistics of the traces used in this section

4.3.2. Port Scans. The most common type of event detected was the port scan. All of the detected scans were horizontal; that is, a particular port was probed on each of a large number of addresses. These port scans were prominent within the traces. For example, the scan seen in Trace 4 comprised over 5600 SYNs in a little over three minutes, with peak rates of over 250 SYN/s. The average SYN rate due to other traffic over a 300-second interval surrounding the scan was 10 SYNs per second.

For SynFinDiff and SynRate, scans like this are extremely obvious. However, since any individual destination address received at most two SYNs from the probing host, PCF did not report any of the scans. In Trace 1, SynFinDiff reported one more scan than SynRate; that scan was much smaller than most others seen, only raising the SynRate test statistic to about half of its detection threshold. In Trace 2, SynRate and SynFinDiff reported significantly different sets of scans. Four large scans over an interval of approximately 100 minutes were reported as three distinct events by SynRate, but as only one event by SynFinDiff. As seen in the Emulab experiments, SynFinDiff takes much longer to reach quiescence than does SynRate; this is clearly borne out by Trace 2. On the other hand, SynFinDiff detected two smaller scans which were missed by SynRate. The two algorithms detected the same scans in Traces 3 and 4.

4.3.3. Peer-to-Peer (P2P) Activity. Another major category of detected anomalies is related to various peer-to-peer file-sharing applications, particularly Gnutella and Kazaa, as identified by the ports used. P2P-related anomalies appeared both inbound and outbound; that is, some alerts were caused by P2P clients on the UCLA CS network making (or attempting to make) a large number of connections to peers elsewhere on the Internet, and some were caused by many P2P clients from the Internet at large trying to connect to a client on the UCLA network.

4.3.4. False Positives. SynRate raised several alarms for which there were no apparent events visible in the trace. These generally appeared during periods of light network activity, where an increase in the average SYN rate from, say, 5 SYN/s to 10 SYN/s for a brief period would cause SynRate's test statistic to spike above the detection threshold. All of these alarms lasted only one or two measurement periods.

4.3.5. Detection and Quiescence Time. When any of the algorithms detected a SYN flood, it did so very quickly. SynFinDiff and SynRate raised alerts at the end of the measurement periods within which the attacks began, and PCF raised alerts within a few seconds of the start of the attacks. Across the numerous port scans detected by both SynFinDiff and SynRate, neither of those algorithms showed a clear advantage in detection time; in all cases, the algorithms' detection times were within 40 seconds of each other.

	SYN Floods	Port Scans	P2P	False Positives	Other
SynFinDiff	1	2	1	0	1
SynRate	1	1	1	2	0
PCF	1	0	0	0	1
In Trace 1	1	2	1	-	-
SynFinDiff	1	4	0	0	0
SynRate	1	5	0	1	0
PCF	2	0	0	0	0
In Trace 2	2	5	0	-	-
SynFinDiff	0	2	1	0	5
SynRate	0	2	1	0	1
PCF	0	0	2	0	0
In Trace 3	0	2	3	-	-
SynFinDiff	0	1	0	0	0
SynRate	0	1	4	6	7
PCF	0	0	2	0	2
In Trace 4	0	1	4	-	-

Table 3. Results from trace-driven experiments

The trace-driven tests generally support the results from the Emulab experiments regarding SynRate's advantage over SynFinDiff in time to quiescence. For the one SYN flood detected by both algorithms, SynFinDiff actually reported the end of the attack 200 seconds before SynRate did. However, after many of the port scans detected by both algorithms, SynFinDiff's test statistic remained above the alert threshold for anywhere from 15 minutes to nearly an hour after SynRate had become quiescent.

5. Discussion

Over the course of the experiments described above, several interesting qualitative behaviors of the algorithms became apparent. In this section, as well as discussing those behaviors, I consider possible attacks against the algorithms themselves, and offer some suggestions for improvements.

5.1. Non-DDoS Activities Detected

Stated in detail, the problem of network-based SYN flood detection is to determine when a host on the network is being forced to waste resources on illegitimate TCP connections. Precisely determining this has been shown to require per-flow state [5]. Constant-space algorithms, then, can only give approximate answers. As particularly brought to light in Section 4, for the approximations provided by each of the algorithms studied in this paper, there are conditions for which the approximation breaks down.

A major lesson from the trace-driven tests is that there are many other network events which look to some extent like a SYN flood. The most egregious example of this is port scans. The algorithms which disregard packets' destination addresses in effect aggregate a port scan's probes into one very large event. Worm activity can be viewed as a distributed port scan; as seen in Traces 1 and 2, worm probes significantly affect SynFinDiff's performance.

The exceedingly noisy behavior of various peer-to-peer file sharing applications was initially surprising. However, it is worth noting that Kazaa, in particular, is a commercial application which speaks a proprietary protocol; the protocol design was not reviewed by any Internet standards body which may have taken exception to its high connection rate.

Both SynFinDiff and SynRate raised a number of alarms for events so small that the alarms might less charitably be called false positives. These algorithms both seek to not require tuning to site-specific traffic parameters, so they define anomalous activity by a proportional increase over the ambient traffic level. However, at very low traffic levels, a small, perhaps coincidental increase in traffic can be a proportionally large increase, thus causing an alarm. Suggestions for mitigating this problem are discussed below.

5.2. Proposed Enhancements

One notable property of PCF as presented in [4] is that it does not account for connections closed by RSTs. While the majority of TCP connections may close neatly with FINs, there were several instances in the trace-driven experiments where one host closed many connections via RSTs, which PCF ignored. This led to PCF overestimating the number of open connections and raising an alarm. Thus, a simple but highly beneficial modification to PCF is, when a RST packet is seen, to decrement the source's counters. When this modified PCF is run against the traces used in this study, all of the SYN floods are still detected, and of the many alarms due to overactive P2P clients, only two remain.

Finally, many of the false alarms and minor anomalies seen by SynFinDiff and SynRate spring from small increases in traffic during periods of light network activity. For modern operating systems running on modern hardware, there is a threshold traffic rate below which any supposed attack is irrelevant. Setting this threshold at, say, 20 SYN/s (a fairly conservative figure), and not reporting an attack for any period during which traffic is under the threshold would eliminate many of the false positives from SynFinDiff and SynRate.

6. Conclusion

I implemented three SYN flood detection algorithms from the literature: SynFinDiff, SynRate, and PCF. I tested the algorithms by running them on a simple Emulab network and by using traces from the UCLA Computer Science Department network as input. The Emulab tests showed the following: SynFinDiff has good detection speed but takes a very long time to return to a non-alert state; SynRate is significantly and negatively affected by attacks that create high variance in the traffic rate, but is faster than SynFinDiff at signaling the end of an attack; and PCF performs very well with regards to both detection time and quiescence time. The main lesson to take away from the trace-driven tests is that there are many significant and common network events other than SYN floods which are detected by these algorithms. Designers of network attack detection algorithms should keep in mind the noisy, hostile network conditions seen throughout the Internet today.

Acknowledgements

Thanks to Kevin Eustice and Matt Schnaider for many valuable discussions, and to my adviser, Dr. Peter Reiher, for his support and patience. Thanks also to the anonymous reviewers for their comments and suggestions.

References

[1] Dan J. Bernstein. SYN cookies. http://cr.yp.to/syncookies.html, 1997.

[2] B. E. Brodsky and B. S. Darkhovsky. *Nonparametric Methods in Change-Point Problems*. Kluwer Academic Publishers, 1993.

[3] CERT Coordination Center. CERT Advisory CA-2001-9 "Code Red" worm exploiting buffer overflow in IIS indexing service DLL. http://www.cert.org/advisories/CA-2001-19.html, Jul 2001.

[4] Ramana Rao Kompella, Sumeet Singh, and George Varghese. On scalable attack detection in the network. In *Proc. of Internet Measurement Conference*. ACM SIGCOMM, 2004.

[5] Kirill Levchenko, Ramamohan Paturi, and George Varghese. On the difficulty of scalably detecting network attacks. In *Proc. of CCS*. ACM SIGSAC, Oct 2004.

[6] Vasilios A. Siris and Fotini Papagalou. Application of anomaly detection algorithms for detecting SYN flooding attacks. In *Proc. of Globecom*. IEEE Communications Society, 2004.

[7] Haining Wang, Danlu Zhang, and Kang G. Shin. Detecting SYN flooding attacks. In *Proc. of INFOCOM*. IEEE Communications Society, 2002.

[8] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of OSDI*, Boston, MA, Dec 2002. USENIX.