A Comparison of SYN Flood Detection Algorithms

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

February 20, 2007

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.

These attacks have plagued the Internet since the mid-1990s. In the early 2000s, a series of attacks against high-profile web sites focused a great deal of attention to the problem. 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 of three SYN flood detection algorithms from the network security literature. I then evaluated the three algorithms with attacks run in an Emulab [24] 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.

The remainder of the paper is structured as follows. Related work is discussed in Section 2. Section 3 describes in detail the three algorithms selected for comparison. Section 4 presents the tests that were run on an Emulab network and their results, and Section 5 presents the results from the trace-driven experiments. In Section 6, offers some discussion of the strengths and weaknesses of each of the algorithms, and some suggestions for improving their performance. Finally, Section 7 concludes the paper.

2 Related Work

This paper, as a comparison study of published work, is heavily indebted to the contributions of previous researchers. The three algorithms evaluated here are

taken from [23], [21], and [11]. They are described in some depth in Section 3.

Many distributed denial of service (DDoS) detection mechanisms have been proposed by the network security research community. MULTOPS [7] is a clever data structure for detecting flooding sources; however, it is ineffective against spoofed traffic. Hop-Count Filtering [9] detects spoofed traffic at the destination. Source IP Monitoring [18] combines an observation in [10] and a statistical technique (also used in [23] and [21]) to determine when traffic is coming from unusual sources. Several authors have proposed other statistical and information-theoretic methods, including [12], [6], and [17]. [15] provides a major survey of DDoS defense mechanisms.

Network security metrics have only recently become a focus of research activity. Mirkovic et al. propose metrics for denial of service measurement in [16], and propose a DDoS defense evaluation methodology in [14]. In other security areas, particularly in operating system security, there are established evaluation procedures and metrics [4].

3 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. I used the parameter values selected by the original authors wherever possible.

3.1 SynFinDiff

Wang, Zhang, and Shin present an algorithm in "Detecting SYN Flood Attacks" [23] 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, $\bar{F}$. The authors 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.

Wang et al. set t_0 to 20 seconds and t_d to 10 seconds.[1] They choose $a = 1$ and calculate a corresponding $N = 1$. They do not specify the decay parameter for $\bar{F}$. I arbitrarily chose 0.9; that is, $\bar{F}(n) = 0.9\bar{F}(n-1) + 0.1F(n)$, where $F(n)$ is the count of FINs seen in the current observation period.

3.2 SynRate

The second algorithm is the CUSUM-based algorithm in "Application of Anomaly Detection Algorithms for Detecting SYN Flooding Attacks," by Siris and Papagalou [21], 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

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

[1] They justify the 10-second figure by referring to an empirical study performed in 1997 [22], which gives 12 to 19 seconds as the average observed connection duration. However, this figure says little about the *distribution* of connection durations. In the traces described in Section 5.1, I found that the 80th percentile of connection duration ranged from 10 to 16 seconds across different traces, though the average was, in some cases, much higher.

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. It is not explicitly stated in the paper whether the variance is estimated somehow or calculated exactly; I chose to calculate it exactly while calculating g_n. Similarly to SynFinDiff, SynRate signals an attack when $g_n > h$, for a fixed threshold h. The parameters chosen by Siris and Papagalou are $\alpha = 0.5$, $h = 5$, and 0.98 for the SYN-count EWMA decay rate.

3.3 PCF

The *PCF*, or Partial Completion Filter, was introduced by Kompella, Singh, and Varghese in "On Scalable Attack Detection in the Network" [11]. 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.

The experiments performed by Kompella et al. used $k = 3$ stages, 5000 buckets per stage, an alert threshold of 150, and a measurement interval of 60 seconds. For the independent hash functions, I paired the input tuple with the index of the stage and applied the object hash function built into Python [19].

4 Synthetic Attacks

I implemented the three algorithms listed in Section 3 in a program which took input from `tcpdump`. I set up a small Emulab [24] 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.

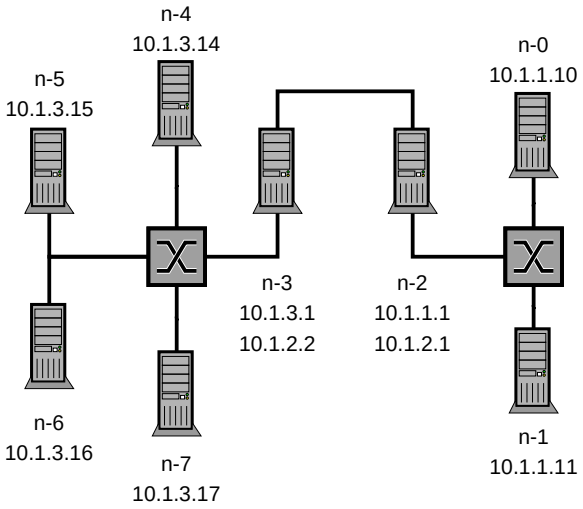

Figure 1: The Emulab network used for the synthetic tests.

4.1 Test Environment

Figure 1 depicts the network used in the attack tests. The network links were all 100Mb. The hardware assigned by Emulab varied somewhat between tests; the CPUs were 2GHz Intel Pentium 4s or 3GHz Intel Xeons with 512MB or 2GB of RAM, respectively. 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, running Apache 1.3.33. 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.

4.2 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 using a command-line HTTP client. The SynFinDiff and SynRate algorithms used this time to stabilize their respective EWMAs. Next was the "attack" phase. As implied by the name, the attacking nodes perpetrated

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.

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. Each experiment was run once; there was no significant stochastic element to the experiments, so repeated runs would not provide extra information. 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 SYNs per second from each attacker up to 1200 SYNs per second per 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; that is, the rate of increase varied from 133.3 SYN/s^2 to 5.55 SYN/s^2. 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.

4.3 Results

4.3.1 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

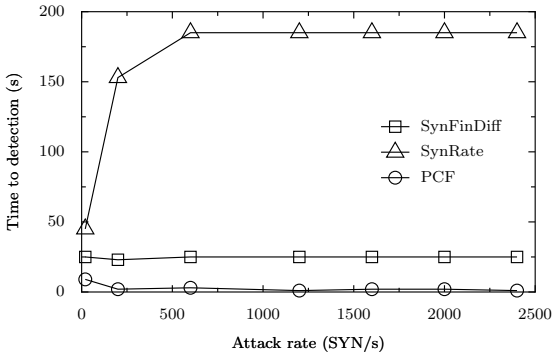

Figure 2: Time between attack start and attack detection as a function of attack rate.

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 is apparently counterintuitive; that is, one might expect that larger attacks would lead to faster detection, yet SynRate's detection time goes up with an increasing attack rate. The explanation lies in the equation which generates SynRate's test statistic g_n. In that equation, 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. For instance, in Experiment 1, with a 20 SYN/s attack rate, the variance at the measurement interval following the start of the attack is about 180. The variance at the corresponding interval in Experiment 7, the maximum tested attack rate, is about 4 million. The implications of this result will be discussed further in Section 6.

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 ap-

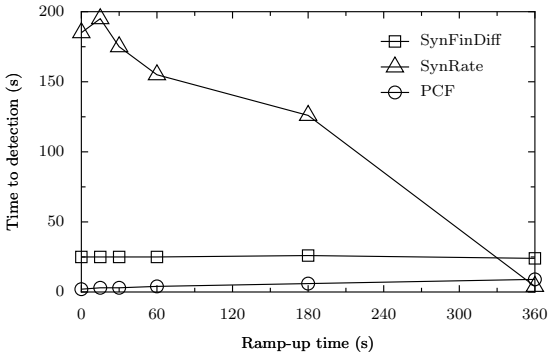

Figure 3: Time between attack start and attack detection as a function of the time taken for the attack to reach full strength (2000 SYN/s).

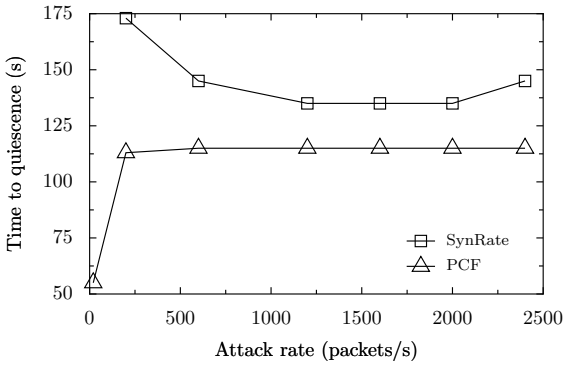

Figure 4: Time to quiescence.

proximately 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.

4.3.2 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. In Experiment 1, which had the smallest attack volume, the test statistic peaked at approximately 187, and declined to approximately 180 by the end of the experiment. Extrapolation yields a time to quiescence of over an hour. For the experiment with the largest attack volume, Experiment 7, the peak value of the test statistic was approximately 26362, giving 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.

SynRate's behavior in Experiments 8 through 12 is somewhat more interesting. The time its test statistic takes to fall below the alert threshold is directly proportional to the maximum value to which the test statistic rises during the attack. This maximum is affected positively by the overall attack volume, and negatively by the variance of the traffic rate, as discussed above. The smallest maximum value across the experiments with ramped attacks is seen in Experiment 9, though Experiment 8's maximum is only slightly higher. As shown in Figure 5, the test statistic in Experiment 8 both rises and falls faster than it does in Experiment 9, though the former's initial jump at the start of the attack is smaller. The shape

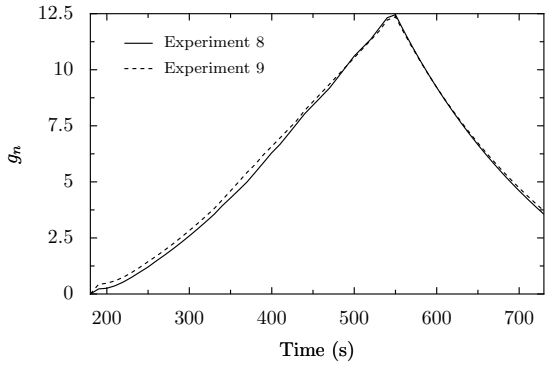

Figure 5: The SynRate test statistic g_n from the start of the attack phase through the end of the experiment in Experiments 8 and 9.

of the curves for the test statistic in Experiments 10–12 is very similar to that seen in Figure 5; those experiments show a faster increase and slower decrease than seen in either Experiment 8 or Experiment 9.

5 Trace-Driven Tests

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

5.1 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 thus the compression ratio), 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. In Trace 1, the test statistic rises nearly monotonically. 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.

5.2 Methodology

I ran all three algorithms on each trace (with exceptions due to the Code Red-induced problems in Traces 1 and 2, as noted above), 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 Ethereal [5]. 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. The only way to establish a ground truth is to manually inspect each trace. Of course, the amount of effort necessary to inspect nearly 100 million packets is prohibitive, to say the least.

5.3 Results

Tables 3–6 summarize the alerts produced by each algorithm for each trace. The "In Trace" row in each table shows the number of distinct events flagged by the algorithms.

5.3.1 SYN Floods

There were three clearly identifiable SYN flood attacks seen over the total 55-plus hours worth of traces

Trace	Date	Start	Duration	Size	Packets	TCP Control Pkts
1	2001-08-02	16:54:18	15:09:57	2.872GB	33941098	1324158
1a	2001-08-02	16:54:18	15:09:57	N/A	N/A	554180
2	2001-08-03	14:12:37	14:20:26	2.625GB	30884752	1196948
2a	2001-08-03	14:12:37	14:20:26	N/A	N/A	476072
3	2001-08-21	10:22:41	16:19:10	2.974GB	33738416	1311138
4	2001-08-23	10:12:21	9:32:52	2.942GB	33332981	1312611

Table 2: Summary statistics of the traces used in this section. Start times are PDT. "TCP Control Pkts" counts the number of TCP packets seen with the SYN, FIN, or RST bit set. Traces 1a and 2a are versions of Traces 1 and 2, respectively, with traffic on port 80 removed.

	Total Events	SYN Floods	Port Scans	P2P	False Positives	Other
SynFinDiff	4	1	2	1	0	1
SynRate	5	1	1	1	2	0
PCF	2	1	0	0	0	1
In Trace	4	1	2	1	-	-

Table 3: Results for Trace 1.

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

Table 4: Results for Trace 2.

	Total Events	SYN Floods	Port Scans	P2P	False Positives	Other
SynFinDiff	8	0	2	1	0	5
SynRate	4	0	2	1	0	1
PCF	2	0	0	2	0	0
In Trace	5	0	2	3	-	-

Table 5: Results for Trace 3.

	Total Events	SYN Floods	Port Scans	P2P	False Positives	Other
SynFinDiff	1	0	1	0	0	0
SynRate	18	0	1	4	6	7
PCF	4	0	0	2	0	2
In Trace	5	0	1	4	-	-

Table 6: Results for Trace 4.

— one in Trace 1 and two in Trace 2. The attack in Trace 1 was targeted at TCP port 1214 (the port used by the Kazaa peer-to-peer file sharing service) on a host that was actively participating in the Kazaa network. The attack consisted of three large bursts of approximately 580 to 670 packets apiece, lasting from 0.4 seconds to 1.2 seconds; the bursts occurred approximately one minute apart. The source address on each attack packet was from a private (RFC1918 [20]) address block. The TCP headers on all of the attack packets in each burst were identical, even down to the checksum; since the source address varied per packet, the TCP checksum for each packet was incorrect.

The first of the attacks in Trace 2 was targeted at TCP port 80 at an address from which no packets were seen during the trace (i.e., there was probably no host at that address). The attack consisted of three bursts of over 400 packets each; each burst lasted approximately half a second. As with the attack in Trace 1, all of the attack packets bore source addresses from a private address block, and had identical TCP headers with incorrect checksums.

The second SYN flood seen during Trace 2 was, like the attack in Trace 1, targeted at the Kazaa service on a host that was active in the Kazaa network. The attack consisted of 27 bursts of approximately 60 packets each, the packets in each burst arriving over a span of only 20 milliseconds. Again, the nominal sources were RFC1918 addresses, and the TCP checksum on each packet was incorrect. The first few bursts came just 3 to 6 seconds apart, but later bursts were spaced by minutes to tens of minutes.

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 long delay between bursts in the second attack kept the test statistic from rising above a negligible amount before falling back to 0. 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.

5.3.2 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. All but one of the scans were of ports 111 and 515; the one exception was a scan of port 53 in Trace 3. At least one scan of either port 111 or port 515 was seen in every single trace. These port scans were prominent within the traces. As a representative example, take the single scan seen in Trace 4, targeted at port 111. The scanning host sent SYNs to 211 of the addresses on a particular /24 network, mostly in numerical order, in slightly over 20 milliseconds. 12 seconds later it probed 254 hosts in a nearby /24, again in a little over 20 milliseconds. This was repeated several times, with 25 different /24 prefixes receiving at least one probe each in three minutes, and 23 of those receiving over 100 probes. Some /24s were even scanned twice. 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.[2] 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.[3] The two algorithms detected the same scans in Traces 3 and 4.

5.3.3 Peer-to-Peer (P2P) Activity

Another major category of detected anomalies is related to various peer-to-peer file-sharing applications, particularly Gnutella [8] 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 con-

[2] As mentioned in [11], PCF can be easily applied to port scan detection, by hashing the source IP address instead of the <destination IP, destination port> pair.

[3] Even when run on the reduced data set of Trace 2a, SynRate did not detect those smaller scans.

nections 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. For instance, in Trace 3, an event flagged by SynFinDiff was caused by a local client connecting to the Gnutella network by trying to open many connections, causing spikes in the SYN rate of up to 20 SYN/s, but with only about a 10% rate of returned SYN/ACKs. On the other side, nearly all of the alerts raised by PCF in Trace 1 were due to a single host receiving nearly 19,000 SYNs, with only 5000 FINs seen.

5.3.4 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.

5.3.5 Other Anomalies

There were several anomalies in the traces that did not fit into the above categories. The most interesting of these was an anomaly flagged by SynFinDiff in Trace 1a. It was apparently the result of a brief spike in the observed SYN rate, followed by a lengthy period during which the rate of FINs and RSTs was significantly lower than during surrounding time intervals. Upon further investigation, I determined that the drop in TCP traffic was the result of a major increase in the rate of incoming DNS queries. During a two-hour period during which the FIN and RST rates were clearly depressed, over 254,000 DNS queries were seen. A total of about 324,000 queries were seen over the entire 15-hour trace. Of the queries in the anomaly, the majority (167,000) were A-type queries for a single hostname. The next most common queries were A queries for two other hosts, with 10,000 queries seen for each.

Several of the alerts raised by SynRate in Trace 4 were caused by a host making connections to a web server at a rate of 5–10 SYN/s for three to ten seconds. This behavior could be the result of "download accelerator" software, or just a web browser that does not use the HTTP keep-alive extension. Even though these were small events in absolute terms, they occurred during periods where the prevailing SYN rate was under 5 SYN/s; thus, an additional 5–10 SYN/s was a large percentage increase, which caused SynRate's test statistic to rise briefly. Likewise, in Trace 3, one single host was persistently trying to connect to an FTP server which was not available. It was the single largest source of SYNs in that trace, outstripping the next external source by nearly a factor of 5. While its SYN rate was small in absolute terms, during periods of low activity, it raised the aggregate SYN rate enough relative to the aggregate FIN rate to cause a number of brief SynFinDiff alerts.

5.3.6 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.[4] 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.

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.

6 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.

6.1 Non-DDoS Activities Detected

Stated in detail, the problem of network-based SYN flood detection is to determine when a host on the

[4] Though the attack in Trace 1 was in the middle of a larger event flagged by PCF, the alert for the measurement interval containing the beginning of the attack occurred within less than a second of the arrival of the first attack packet.

network is being forced to waste resources on illegitimate TCP connections. Precisely determining this has been shown to require per-flow state [13]. Constant-space algorithms, then, can only give approximate answers. As particularly brought to light in Section 5, 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.

6.2 Second-Order Attacks

The results presented in Sections 4 and 5 suggest some weaknesses in the algorithms that might be exploited by an attacker. One attack exploits SynRate's dependency on the variance of the inbound traffic rate. As seen in Section 4, a higher variance in the traffic rate increases the time before SynRate raises an alarm. By sending attack traffic in a pattern such as to deliberately increase the variance of the traffic rate seen at the target network, the attacker may be able to effect a denial of service without SynRate raising an alarm, or at least delaying the alarm by a significant time.

A weakness of quite a different character lies in the large interval at which PCF clears its state. Assume that the attacker actually wants to trigger PCF, e.g., to trigger a defensive mechanism with undesirable side effects. The attacker can send a relatively small burst of traffic (a few hundred packets) at the beginning of a measurement period, thus causing PCF to raise an alarm and trigger the defensive system. The attacker only needs to repeat this once every 60 seconds to keep PCF in a perpetual state of alert.

6.3 Proposed Enhancements

Given the flaws described above, it is clear that there is room for improvement in each of the algorithms. One notable property of PCF as presented in [11] 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.[5] 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, for instance, 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.

6.4 Broader Applicability

SYN floods are by no means the only type of DDoS attack seen on the Internet today. UDP floods, and to a lesser extent ICMP floods, are also of concern. The three algorithms discussed in this paper may be modifiable to detect these attacks. In fact, both Siris and Papagalou and Kompella et al. suggest that their methods can be applied to other types of attacks.

[5] Decrementing the destination's counters would lead to a trivial evasion attack: the attacker could open a large number of connections to the target, and then send RSTs to the target with source ports not used for the actual connections.

For protocols which have a simple request–response behavior, such as DNS or ICMP Echo, the analogy to TCP's SYN–FIN pair makes the modifications straightforward. For instance, SynFinDiff's inputs could be the numbers of DNS requests and responses seen during a measurement period.[6] PCF could be modified in a similar manner, and either of those two could be just as easily modified to count ICMP Echo Request and Echo Reply packets.

SynRate is a slightly different case, as it only uses incoming requests (SYNs) as its input. This gives it more flexibility; for instance, its input could be simply the total number of bytes inbound to a network during a measurement interval, which would allow it to detect bandwidth attacks no matter what the protocol. It could also be applied to more specific inputs such as DNS or ICMP Echo; however, the discussion above regarding minimum traffic thresholds would hold just as much as it does for the original SYN input. In fact, such a threshold might be more important for ICMP, given that the average rate of ICMP traffic seen is much lower than that of TCP SYNs.

7 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.

[6] Recall that the measurement intervals for SYNs and FINs are offset by a certain duration to account for longevity of TCP connections. If the algorithm is used to detect DNS flooding, the measurement periods should not be offset.

8 Acknowledgements

Thanks to Kevin Eustice and Matt Schnaider for many valuable discussions, and to my adviser, Dr. Peter Reiher, for his support and patience.

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] The Center for Internet Security. `http://www.cisecurity.org/`.

[5] Ethereal. `http://www.ethereal.com/`.

[6] Laura Feinstein, Dan Schnackenberg, Ravinda Balupari, and Darrell Kindred. Statistical approaches to DDoS attack detection and response. In *Proc. of DISCEX*, 2003.

[7] Thomer M. Gil and Massimiliano Poletto. MULTOPS: a data structure for bandwidth attack detection. In *Proc. of USENIX Security*. USENIX, Aug 2001.

[8] The Gnutella protocol. `http://www.the-gdf.org/`.

[9] Cheng Jin, Haining Wang, and Kang G. Shin. Hop-count filtering: an effective defense against spoofed DDoS traffic. In *Proc. of CCS*. ACM SIGSAC, 2003.

[10] Jaeyeon Jung, Balachander Krishnamurthy, and Michael Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites. In *Proc. of the World Wide Web Conference*, May 2002.

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

[12] Amit Kulkarni and Stephen Bush. Detecting distributed denial-of-service attacks using Kolmogorov complexity metrics. Technical Report 2002GRC086, General Electric Global Research Center, May 2002.

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

[14] Jelena Mirkovic, E. Arikan, S. Wei, Sonia Fahmy, Roshan Thomas, and Peter Reiher. Benchmarks for DDoS defense evaluation. In *Proc. of Milcom*, Oct 2006.

[15] Jelena Mirkovic and Peter Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *Computer Communications Review*, 24(2), Apr 2004.

[16] Jelena Mirkovic, Peter Reiher, Sonia Fahmy, Roshan Thomas, Alefiya Hussain, Steven Schwab, and Calvin Ko. Measuring denial-of-service. In *Proc. of Quality of Protection Workshop*, Oct 2006.

[17] Yuichi Ohsita, Shingo Ata, and Masayuki Murata. Detecting distributed denial-of-service attacks by analyzing TCP SYN packets statistically. In *Proc. of Globecom*. IEEE Communications Society, 2004.

[18] Tao Peng, Cristopher Leckie, and Kotagiri Ramamohanarao. Proactively detecting distributed denial of service attacks using source IP address monitoring. In *Proc. of Networking 2004*. IFIP, 2004.

[19] The Python programming language. `http://www.python.org/`.

[20] Yakov Rekhter, Robert G. Moskowitz, Daniel Karrenberg, Geert Jan de Groot, and Eliot Lear. RFC 1918: Address allocation for private internets, Feb 1996.

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

[22] Kevin Thompson, Gregory J. Miller, and Rick Wilder. Wide-area Internet traffic patterns and characteristics. *IEEE Network*, 11(6), Nov 1997.

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

[24] 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.