A Robust Scheme to Detect SYN Flooding Attacks

Changhua Sun, Jindou Fan, Bin Liu
Department of Computer Science and Technology, Tsinghua University, China
sch04@mails.tsinghua.edu.cn, fanjindou@gmail.com, liub@tsinghua.edu.cn

***Abstract*—We propose a more robust scheme to detect SYN flooding attacks. Existing methods for detecting SYN flooding are based on the protocol behavior of TCP SYN–FIN (RST) or SYN–ACK pairs, as normally the number of SYN packets is equal to that of FIN (added with RST) packets, or ACK packets in the handshake. When SYN flood starts, there will be more SYN packets. However, the attacker can avoid the detection by sending the FIN or RST packets (ACK packets) in conjunction with the SYN packets. To make the detection scheme more robust, we record the flow information of SYN packets in a counting Bloom Filter, and count the FIN (RST) packets according to the Bloom Filter. In addition, the Change Point Detection method based on a non-parametric Cumulative Sum algorithm is applied to make the detection mechanism much more generally applicable. Through trace-driven simulations, we show our detection scheme is more efficient and robust in detecting various SYN flooding attacks. More importantly, our scheme can be easily deployed at ISP's edge routers.**

I. Introduction

Internet services become the necessity of daily life as millions of people around the world use these services frequently. Unfortunately, Distributed Denial-of-Service (DDoS) attacks can easily deny regular Internet services from being accessed by legitimate users. And these attacks remain serious problems on the Internet today, as they take advantage of the lack of authenticity in the IP protocol, destination oriented routing, and stateless nature of the Internet. It is very difficult to defend DDoS attacks.

Among various DDoS attacks, the TCP SYN flooding [1] is the most commonly-used attack. This attack exploits TCP's three-way handshake mechanism and its limitation in maintaining half-open connections. When a server receives a SYN packet, it interprets the packet as a request by the remote client to initiate a TCP connection, and allocates resources (typically backlog queue in the system memory) to track the TCP state. In addition, the server returns a SYN/ACK packet (a packet contains both the acknowledgement ACK and the synchronization request SYN) to the client and waits until either the half-open connection completes or the TCP connection times out. In the SYN flooding attack, the server will receive a large number of SYN packets but never receive the final ACK packets to complete the three-way handshake. Then the victim server's backlog queue can be easily exhausted, causing all the new incoming SYN requests to be dropped. Furthermore, many other system resources, such as CPU and network bandwidth used to retransmit the SYN/ACK packets, are occupied.

Previous SYN flooding defenses mainly include: 1) victim modifications, such as SYN cookies [2] and SYN cache [3]; 2) firewalls and proxies [4]; 3) router solutions, like SYN detection [5]. SYN cache is to allocate minimal state when the initial request is received, and only allocate all the resources required when the connection is completed. If the backlog queue is full, the oldest entry is removed. SYN cookies allocate no state for half-open connections. Instead, they encode most of the states and encrypt them into the sequence number transmitted in the SYN/ACK packet. The ACK packet that completes the handshake can be used to reconstruct the state to be put into the backlog queue. One problem with SYN cookies is not able to encode all the TCP options, and the other is that TCP protocol with SYN cookies would never retransmit the unacknowledged SYN/ACK packet. In addition, both SYN cache and cookies do not handle application data piggybacked on the SYN segment [4]. Moreover, both of them can not alleviate bandwidth exhaustion attacks resulting from SYN flooding, and not decrease the normal traffic's delay increased by the SYN flooding. Firewalls and proxies offload the connection establishment procedures from the end hosts, screen connection attempts until they are completed, and then proxy them back to protected end hosts. They often use SYN cookies and SYN cache, and have the same limitations.

Router-based detection and mitigation schemes can be complementary to the victim-based defenses [5]. SYN detection scheme in [5] is based on the fact that a normal TCP connection starts with a SYN packet and ends with a FIN or RST packet. When the SYN flood starts, there will be more SYN packets than FIN and RST packets. However, the attacker can avoid detection by sending a mixture of FIN (RST) and SYN packets. The ratio of SYN–ACK is also suggested in [5] and studied in [6] to detect SYN flooding. However, it is unclear how to determine whether a ACK packet is the ACK to complete the TCP connection in [6]. Moreover, the scheme in [6] is mainly to detect spoofed SYN flooding attacks, and can not detect a SYN flooding attack with a mixture of ACK and SYN packets.

In this paper, we propose a more robust scheme to detect SYN flooding at ISP's edge routers. The scheme is simple, stateless, and can be easily deployed. Our main idea is to

This work is supported by NSFC (No. 60373007, 60573121 and 60625201), the Cultivation Fund of the Key Scientific and Technical Innovation Project, Ministry of Education of China (No. 705003), the Specialized Research Fund for the Doctoral Program of Higher Education of China (No. 20040003048 and 20060003058), China/Ireland Science and Technology Collaboration Research Fund (2006DFA11170), and the Tsinghua Basic Research Foundation (JCpy2005054).

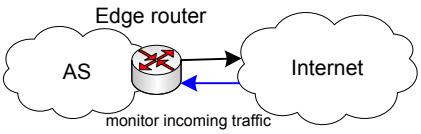


Fig. 1. Illustration of the detection scheme at an edge router.

use a counting Bloom Filter [7] to store the *4-tuple* (source and destination IP, source and destination Port) of counted SYN packets. Then a FIN or RST packet is counted only if its *4-tuple* is in the Bloom Filter. Therefore, any malicious FIN or RST packets would not be counted. In this case, the fact that there will be more SYN packets than FIN and RST packets during SYN flood would still hold. To make the detection scheme generally applicable, the Change Point Detection method based on a non-parametric Cumulative Sum (CUSUM) algorithm [8] is applied. Through trace-driven simulations, it is shown that our scheme can detect various SYN flooding attacks and is more robust.

The rest of paper is organized as follows. Section II describes the proposed scheme. Section III carries out the trace-driven simulations to study the performance of the proposed scheme. Finally, in section IV, we conclude the paper.

II. Detection Scheme

In this section, we first give an overview of the proposed scheme and show how it works. Then we explain Bloom Filter and Change-Point Detection method to further understand the detection scheme.

A. Overview

Under normal conditions, each TCP connection which completes the three-way handshake begins with a SYN or SYN/ACK packet, and ends with a FIN or RST packet in either traffic direction. We define *valid SYN packets* as the pure SYN and SYN/ACK packets. Also, we define *valid FIN packets* as the FIN and RST packets which close the TCP connections. Those connections either complete the three-way handshake or have a *valid SYN packet* in the same traffic direction before this FIN or RST packet. Our scheme utilizes the inherent TCP *valid SYN–FIN* pairs behavior for SYN flooding detection. Notably, the *valid SYN–FIN* pairs are different from the SYN–FIN pairs in [5], as in the latter, FIN packets include all pure FIN packets and most RST packets (75% RST packets).

As TCP SYN flooding attacks exploit TCP's three-way handshake, there are many half-open TCP connections. Therefore, under normal SYN flooding conditions, there would be more SYN packets than FIN packets. However, the attackers could easily overthrow this assumption by sending mixture FIN and SYN packets, leading to out-of-work of the detection scheme in [5]. By contrast, it is difficult to violate the fact that there are more *valid SYN packets* than *valid FIN packets* under SYN flooding.

We mainly use a counting Bloom filter [7] to record the TCP connection information for *valid SYN packets*. Then we classify whether a FIN or RST packet is a *valid FIN packet* according to the Bloom filter. A Bloom filter is a simple space-efficient data structure for representing a set in order to support membership queries. It is described in section II-B.

Under a long-running normal condition, the TCP semantics requires a one-to-one match between *valid SYN packets* and *valid FIN packets*. However, there is always a discrepancy between them in reality. One reason is the small number of long-lived TCP sessions, and the other is the retransmission of SYN packets. Through real trace analysis, it is shown that the discrepancy is almost constant. To make the detection scheme more flexible, we use the Change Point Detection method based on a non-parametric Cumulative Sum (CUSUM) algorithm [8], which is explained in section II-C.

In addition, our scheme is deployed at ISP's edge routers that connect end hosts to the Internet. We only need to monitor one traffic direction and count the *valid SYN and FIN packets* to detect SYN flooding attacks, which has advantages of being flexible and easily deployed. Moreover, as shown in Fig. 1, we prefer to monitor the traffic from Internet to the *AS* (incoming traffic), since, when detecting SYN flooding, it is easier to deploy SYN flooding mitigation scheme like *SynDefender* [9] at this traffic direction to protect the victims.

B. Bloom Filter

As mentioned before, a Bloom filter [7] is a simple space-efficient data structure for representing a set in order to support membership queries. It allows low and controlled false positives. A Bloom filter for representing a set $S = \{x_1, x_2, ..., x_n\}$ of n elements is composed by an array of m bits, initially all set to 0. We use k independent hash functions $h_1, h_2, ..., h_k$, each with range $\{0, 1, ..., m-1\}$. For each element $x \in S$, the bits $h_i(x)$ are set to 1 for $1 \leq i \leq k$. Note that a location can be set to 1 multiple times, but only the first change has an effect. Given a query on the existence of y in S, we check whether all $h_i(y)$ are set to 1. If not, then clearly y is not in S. If all $h_i(y)$ are set to 1, we assume that y is in S, although we are wrong with some probability. This probability is referred to as the *false positive* rate, where it suggests that an element x is in S even though it is not. We use p_f to indicate the false positive rate, which is approximately $(1-e^{-kn/m})^k$, when $m > kn$. If the number of hash functions is optimized with minimized false positive rate, we get:

$$p_f = (1/2)^k = (1/2)^{\ln 2 \cdot (m/n)}. \quad (1)$$

A variant of the original Bloom filter, called *counting Bloom filter* [10], [11], can be easily used for inserting and deleting items. In a counting Bloom filter, each entry in the Bloom filter is not a single bit but rather a small counter. When an item is inserted or deleted, the corresponding counters are incremented or decremented. If an item x is in S, the counters at locations $h_i(x)$, $1 \leq i \leq k$, should all be nonzero. In addition, the analysis from [10] reveals that 4 bits per counter should suffice for most applications to avoid counter overflow.

In our detection scheme, when we receive a SYN or SYN/ACK packet, the counter of *valid SYN packets* is increased. We extract *4-tuple* (source and destination IP, source

and destination Port) of this packet as an item, and insert this item to a counting Bloom filter. When we receive a FIN or RST packet, the item of its *4-tuple* is extracted and queried from the counting Bloom filter. If this item is in the Bloom filter, the counter of *valid FIN packets* is increased, and this item is deleted from the counting Bloom filter. If not, this packet is not a *valid FIN packet*, and nothing is needed. Through the counting Bloom filter, we can easily record the number of *valid FIN packets*. In our detection system, we use 11 hash functions and $m/n = 16$. According to (1), the false positive rate is as little as 0.000459. This is acceptable for detecting SYN flooding. In addition, we use 8 bits per counter, leading to the counter in the Bloom filter seldom overflows.

If the number of locations, at which the counters are nonzero, is greater than $1/2 \cdot m$, the false positive rate would increase. At this time, we reset the counting Bloom filter and the counters of *valid SYN and FIN packets*. Then we restart counting the number of *valid SYN and FIN packets*.

C. Change-Point Detection Method

To make our detection scheme work online, we collect the number of *valid SYN and FIN packets* during every observation period t_p. In addition, to relate the *valid SYN and FIN packets* of the same connection, the sampling time of *valid FIN packets* is t_d later than that of SYN, where t_d is so chosen that a significant portion of connections requested during the SYN sampling period end in the corresponding FIN sampling period. In our experiments, we choose t_d as 10 seconds, and t_p as 20 seconds, according to [5].

We assume $\{\delta_n, n = 0, 1, ...\}$ are the number of *valid SYN packets* minus that of *valid FIN packets* within one sampling period. It is clear that the mean of $\{\delta_n\}$ is dependent on the size of the subnet (AS), time of the day, and even access pattern. To alleviate these dependencies, we normalize $\{\delta_n\}$ by the average number of $\{F_n\}$ during the the same sampling period t_p, where $\{F_n, n = 0, 1, ...\}$ is the number of *valid FIN packets* within one t_p. The average number of $\{F_n\}$, $\overline{F}$, can be estimated in real time and updated periodically. For example, we can update $\overline{F}$ as:

$$\overline{F}(n) = \alpha \cdot \overline{F}(n-1) + (1-\alpha) \cdot F_n. \tag{2}$$

where n is the time index and α is a constant, $0 \leq \alpha \leq 1$, representing the memory in the estimation of the average. We choose $\overline{F}(0) = F_0$ and $\alpha = 0.5$ in the experiments.

Let $X_n = \delta_n / \overline{F}$. $\{X_n, n = 0, 1, ...\}$ is independent on the network size or time-of-day. It can be considered as a stationary random process. When SYN flooding attacks occur, the value of $\{X_n\}$ after this point is changed. An observer must make a decision as quickly as possible to decide whether or not a change point has happened, while keeping the false alarm rate as low as possible. We use the Change Point Detection method based on a non-parametric Cumulative Sum (CUSUM) algorithm [8] to detect the change of $\{X_n\}$.

Suppose $E(X_n) = c$. We choose a parameter a which is the upper bound of c, i.e., $a > c$, and define $x_n = X_n - a$ so that it has a negative mean during normal operation. The nonparametric version of the CUSUM statistic can be expressed:

$$y_0 = 0, y_k = (y_{k-1} + x_k)^+. \tag{3}$$

where x^+ is equal to x if $x > 0$, and 0 otherwise. The corresponding decision rule is

$$d_N(\cdot) = d(y_k) = I(y_k > N). \tag{4}$$

where $I(\cdot)$ is the indicator function and N is the threshold. The function d_N represents the decision at time k, which give a value of 1 to indicate a change point and 0 to indicate a normal condition.

When an attack takes place, x_n will suddenly become large positive. Therefore, under normal condition, y_n is 0. Under SYN flooding attacks, y_n becomes positive. If $y_n > N$, we report an attack.

The two parameters, a, the upper bound in case of normal operation, and N, the flooding threshold, influence the performance of our detection scheme. It is important to make a good trade-off between the detection delay and the false alarm tolerance level. In our detection scheme, we choose $a = 2$ and $N = 1$, according to the real trace analysis and parameters in [5].

TABLE I
TRACE INFORMATION

Trace	Start Time	Duration	Load	Repeat SYNs
THU-1	14:29, Fri Jun 23, 06	347s	28.82%	25.4%
THU-2	14:19, Wed Jul 12, 06	516s	23.04%	26.4%
THU-3	15:56, Wed Jul 12, 06	469s	26.74%	22.4%

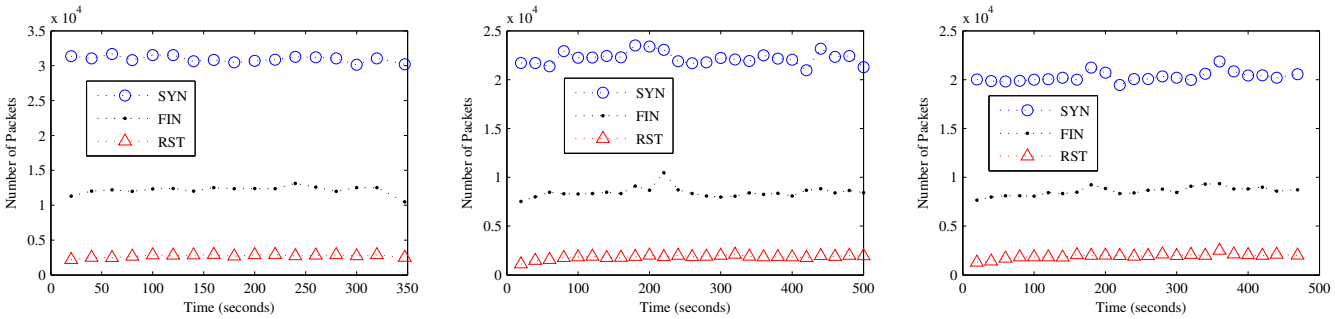
III. PERFORMANCE EVALUATION

To evaluate our detection scheme, we carried out trace driven simulations. The trace data used in our study are collected at the egress point of Tsinghua University [12], which is a GE (1 Gbps) link that connects Tsinghua University to the rest of world. A summary of the traces is given in Table I. As either direction of traffic can be used to detect SYN flooding by our scheme, though these traces are from *AS* to Internet, they are suitable for detecting SYN flooding. The column of Repeat SYNs in Table I indicates the percentage of the retransmission SYN packets, towards the total SYN packets. This is one of the reasons for the discrepancy between *valid SYN and FIN packets*.

For simplicity, we name the detection scheme in [5] as *SFD* (SYN flooding detection), and our scheme as *SFD-BF* (SYN flooding detection based on Bloom filter). As mentioned in section II-B, we use 11 H_3 hash functions [13] to implement the Bloom filter.

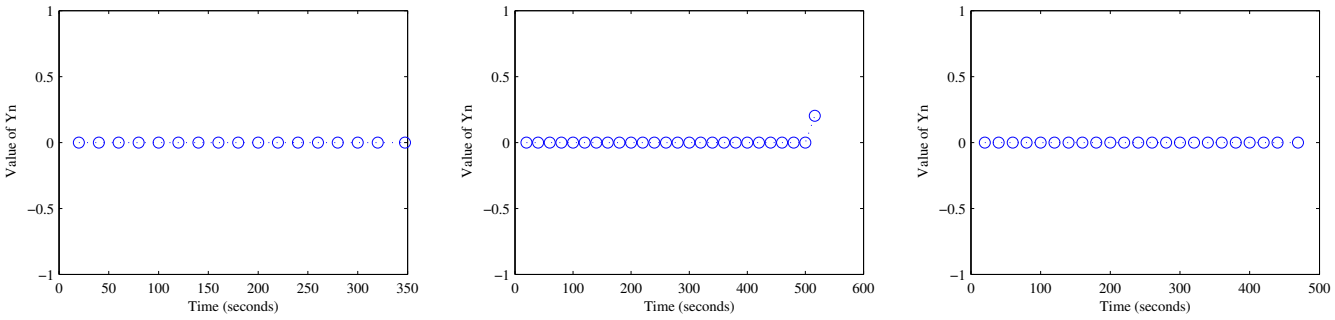
A. Normal Traffic Behavior

We analyze the *valid SYN and FIN packets* of the three traces, and the results are shown in Fig. 2. They clearly



(a) THU-1 (b) THU-2 (c) THU-3

Fig. 2. The dynamics of *valid SYN and FIN packets*.



(a) THU-1 (b) THU-2 (c) THU-3

Fig. 3. SFD-BF detection scheme under normal operation.

show consistent synchronization between *valid SYN and FIN packets*, matching with the inherent traffic behavior.

Fig. 3 shows the test statistics, $\{y_n\}$, for all the three traces, on which our detection scheme are applied without adding attacks. The flooding threshold, N, is set to 1, as specified in section II-C. It is shown that all y_n's are mostly zero, and always much smaller than the threshold. Therefore, no false alarms are reported.

B. Simple SYN Flooding Detection

In the SYN flooding detection experiments, the THU-2 trace is used as the normal background traffic (using other traces can get the similar results). First, we simulate the simple SYN flooding attacks. The attackers send a large number of SYN packets in a small period, but never acknowledge them, which is the most common SYN flooding attack.

The end hosts in Tsinghua University are connected at least by a 100Mb Ethernet. As the maximum SYN flood rate of a 100Mb Ethernet is about 148,800 SYNs/sec [14], we set the aggregate SYN flood attack rate as 1500 SYNs/sec, which is close to the average SYN packets rate of the traces. Normally, a successful SYN flood attack rate would be higher than this rate. The higher the rate is, the easier our scheme can detect. The flooding traffic is mixed with the normal traffic, whose source IP address is set according to the trace and destination IP address is set randomly. Under this assumption, we simulate a SYN flooding attack toward certain end hosts in Tsinghua University. The flooding duration is set to 300 seconds. The starting time of flooding is set at 200 seconds. Note that these parameters do not affect the results of the experiments.

Fig. 4 shows the simulation results using the detection scheme in [5] (SFD) and our scheme (SFD-BF). Both two schemes show fast response time, as the cumulative sum y_n exceeds the flooding threshold 1 in a short observation period. SFD-BF has a faster response time, since it takes 20 seconds to detect flood, while SFD takes 60 seconds.

C. Complex SYN Flooding Detection

If the attacker is aware of the SFD detection scheme, he can avoid detection by sending a mixture of SYNs and FINs (RSTs). In this section, we simulate this situation. The attackers send a larger number of FIN and SYN packets, but never acknowledge the SYN packets. Also, FIN and SYN packets are mixed, and none of a FIN packet closes the TCP connection opened by the flooding SYN packets. In our simulations, this can be easily achieved by randomly choosing destination IP address for SYN and FIN packets.

Fig. 5 shows the value of y_n. SFD can not detect the SYN flooding attack, as all the y_n's are zero, less than the flooding threshold 1. On the other side, SFD-BF detects the flooding at a single observation period, i.e., in 20 seconds. This shows our detection scheme is more robust than the scheme in [5], and can detect various complicate SYN flooding attacks.

In summary, our detection scheme not only achieves fast

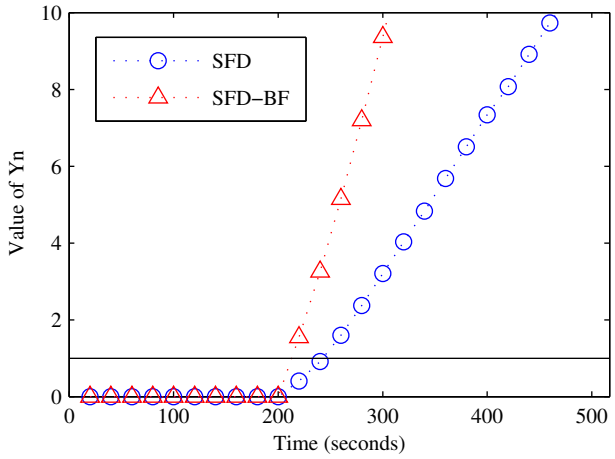


Fig. 4. SYN flooding detection under simple attacks.

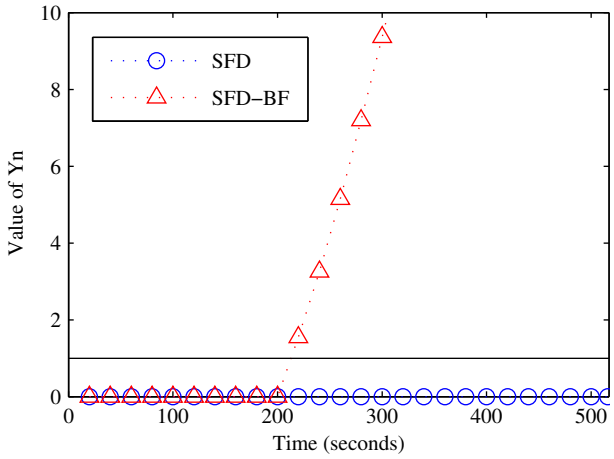


Fig. 5. SYN flooding detection under complex attacks.

detection and high accuracy, but also can detect various SYN flooding and be easily implemented in edge routers.

IV. Conclusion

Existing SYN flooding detection scheme [5] is based on the TCP SYN–FIN pairs behavior. A normal TCP connection starts with a SYN packet and ends with a FIN or RST packet. When the SYN flood starts, there will be more SYN packets than FIN and RST packets. However, the attacker can simply overthrow the assumption by sending a mixture of SYN and FIN (RST) packets. By contrast, our detection scheme is based on the TCP *valid SYN–FIN* pairs behavior, where *valid SYN packets* are the pure SYN and SYN/ACK packets, the same as SYN packets in [5]; *valid FIN packets* are the FIN and RST packets that close the TCP connections which either complete the three-way handshake or have a *valid SYN packet* in the same traffic direction before this FIN or RST packet. It is almost impossible for attackers to violate this fact. Furthermore, we use a storage-efficient data structure, *counting Bloom filter*, which only requires a fixed-length memory for recording relevant traffic information, to recognize the *valid FIN packets*. To make the detection scheme more flexible, we applied a Change Point Detection method based on CUSUM to detect the abrupt changes in the traffic characteristics which correspond to the occurrence of SYN flooding attacks.

Through trace-driven simulations, it is shown that our scheme is more robust and efficient to detect various SYN flooding attacks. It achieves high detection accuracy and short detection time. Moreover, our scheme is stateless and requires low computation overhead (Bloom filter can be easily implemented by H_3 hash functions by hardware or software [13]), making itself immune to SYN flooding attacks.

After detecting SYN flooding, we now rely on other schemes like *SynDefender* to mitigate SYN flooding. It is helpful to develop a more effective SYN flooding mitigation scheme in the future [15].

References

[1] CERT Advisory CA-1996-21 TCP SYN flooding and IP spoofing attacks. [Online]. Available: http://www.cert.org/advisories/CA-1996-21.html

[2] "SYN cookies." [Online]. Available: http://cr.yp.to/syncookies.html

[3] J. Lemon, "Resisting SYN flood DoS attacks with a SYN cache," in *Proc. USENIX BSDCon*, 2002.

[4] W. Eddy, "TCP SYN flooding attacks and common mitigations," May 2007. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-tcpm-syn-flood-05.txt

[5] H. Wang, D. Zhang, and K. G. Shin, "Detecting SYN flooding attacks," in *Proc. IEEE INFOCOM*, 2002.

[6] W. Chen and D. Y. Yeung, "Defending against TCP SYN flooding attacks under different types of IP spoofing," in *International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICN/ICONS/MCL)*, 2006.

[7] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

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

[9] "Syndefender." [Online]. Available: http://www.checkpoint.com/products/firewall-1/

[10] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," in *Proc. ACM SIGCOMM*, 1998.

[11] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," in *Allerton*, 2002.

[12] "Tsinghua dragonlab." [Online]. Available: http://dragonlab.org/traffic/

[13] M. V. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient hardware hashing functions for high performance computers," *IEEE Transactions on Computers*, vol. 46, no. 12, pp. 1378–1381, 1997.

[14] T. Darmohray and R. Oliver, "Hot spares for DoS attacks," *Login: The Magazine of USENIX and SAGE*, 2000. [Online]. Available: http://www.usenix.org/publications/login/2000-7/apropos.html

[15] C. Sun, J. Fan, L. Shi, and B. Liu, "A novel router-based scheme to mitigate SYN flooding DDoS attacks," in *Proc. IEEE INFOCOM (Poster)*, Anchorage, Alaska, USA, 2007.