SYN-dog: Sniffing SYN Flooding Attacks*

Haining Wang Danlu Zhang Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122
{hxw,danlu,kgshin}@eecs.umich.edu

Abstract

This paper presents a simple and robust mechanism, called *SYN-dog*, to sniff SYN flooding attacks. The core of SYN-dog is based on the distinct protocol behavior of TCP connection establishment and teardown, and is an instance of the Sequential Change Point Detection [2]. To make the detection mechanism insensitive to site and access pattern, a non-parametric Cumulative Sum (CUSUM) method [6] is applied, thus making the detection mechanism robust, more generally applicable and its deployment much easier. The statelessness and low computation overhead of SYN-dog make itself immune to flooding attacks. The efficacy of SYN-dog is evaluated by extensive trace-driven simulations. The evaluation results show that SYN-dog has short detection latency and high detection accuracy. Moreover, due to its proximity to the flooding sources, SYN-dog not only sets alarm upon detection of ongoing SYN flooding attacks, but also reveals the location of the flooding sources without resorting to the IP traceback.

Keywords — TCP SYN flooding attack, protocol behavior, intrusion detection, CUSUM algorithm

1 Introduction

The growing number of Distributed Denial of Service (DDoS) attacks impose a significant threat on the availability of network services, and the vulnerability of the Internet to DDoS attacks has been witnessed by the recent attacks on popular web sites like Yahoo, eBay and E*Trade, and their resultant disruption of services [14]. Due to the readily available tools and its simple nature, flooding packets is the most common and effective DoS attack. The latest research results have shown that more than 90% of the DoS attacks use TCP [30], and TCP SYN flooding dominates in the available attacking tools and the number of DoS attacks known to date [27, 30]. TCP SYN flooding consists of a stream of spoofed TCP SYN packets directed to a listening TCP port of the victim. Not only the Web servers but also any system connected to the Internet providing TCP-based network services, such as FTP or mail servers, are susceptible to TCP SYN flooding attacks.

*Haining Wang and Kang G. Shin were supported in part by DARPA under the US Airforce contract AFRL F33615-02-C-4031, and by the Office of Naval Research under Grant No. N00014-99-1-0465.

The stateless and destination-based nature of Internet routing infrastructure cannot differentiate a legitimate SYN packet from a spoofed one, and TCP does not perform strong authentication on SYN packets. Therefore, under a SYN flooding attack, the victim server cannot single out, and respond only to, legitimate connection requests while dismissing the spoofed. Moreover, the spoofed source address conceals the true origin of flooding sources, thus making it difficult to identify, and trace back, the flooding sources.

Most of the previous related work — such as Syn cache [24], Syn cookies [4], SynDefender [25], Syn proxying [1], and Synkill [37] — focused on mitigating the effect of SYN flooding attacks on the victim in order to sustain service availability. These defense mechanisms can be characterized by two common features.

- They are installed at the firewall of, or inside, the victim server, hence providing no hint about the location of the SYN flooding source(s). They have to rely on the usually-expensive IP traceback [3, 31, 36, 40, 39, 45] to locate the flooding source(s).
- They require to maintain or compute states for each TCP connection, which not only makes themselves vulnerable to SYN flooding attacks but also degrades end-to-end TCP performance, e.g., longer delays in setting up TCP connections.

Recent experiments have shown that even a firewall designed specifically to resist SYN floods, became futile under a flood of 14,000 packets per second [9]. On the other hand, in the absence of SYN flooding attacks, such a defense mechanism just wastes resources. By checking only its backlog queue, the victim server cannot tell whether it is under a flooding attack or simply overloaded with legitimate SYN packets. Without an efficient detection mechanism, the defense mechanism installed at a server consumes additional server resources, making the already-overloaded server even less responsive to legitimate requests.

Furthermore, instead of consuming a server's resource, the latest reflected SYN/ACK attacks [15] seize the link bandwidth between the victim and its Internet Service Provider (ISP) by flooding SYN/ACK responses. The above-mentioned defense mechanisms cannot provide any protection for such a "bandwidth consumption" attack, and innocent reflectors will make it difficult to locate the true flooding sources with the IP traceback.

We, therefore, need a simple stateless mechanism to sniff these SYN (including reflected SYN/ACK) flooding attacks. It should also be immune to any type of flooding attacks, and should not degrade end-to-end TCP performance either. By "sniff" we mean not only detect an attack, but also easily uncover the flooding sources without resorting to IP traceback. The sniffing mechanism is orthogonal to the defense mechanisms mentioned earlier. The relationship between the two is analogous to the one between a surveillance radar and surface-to-air missiles within an anti-aircraft defense system, where the radar always checks for intruding enemy aircraft, but the missiles stay inactive until an enemy intrusion is detected.

In this paper, we propose a simple and robust mechanism, called *SYN-dog*, to sniff SYN flooding attacks. While SYN-dog monitors the TCP traffic constantly, the defense system can be turned off until a warning is issued by the SYN-dog. Instead of monitoring the ongoing traffic at the front end or the victim server itself, SYN-dog is installed at a leaf router that connects a stub network[1] to the Internet. The simplicity (hence the attractiveness) of SYN-dog lies in its statelessness and low computation overhead. Its key features include:

[1] A stub network only carries packets to and from local hosts.

- SYN-dog utilizes the distinct protocol behavior of TCP connection establishment and teardown, which consists of SYN–FIN pair and SYN–SYN/ACK pair, for SYN flooding detection.
- SYN-dog is insensitive to sites and access patterns: the non-parametric Cumulative Sum (CUSUM) method [6] is applied, making SYN-dog robust, much more generally applicable and its deployment easier.
- SYN-dog plays a dual role in sniffing SYN flooding attacks: the first-mile SYN-dog and the last-mile SYN-dog. Due to its close proximity to the flooding sources, the first-mile SYN-dog not only alarms on the ongoing SYN flooding attacks, but also reveals the location of the flooding sources.

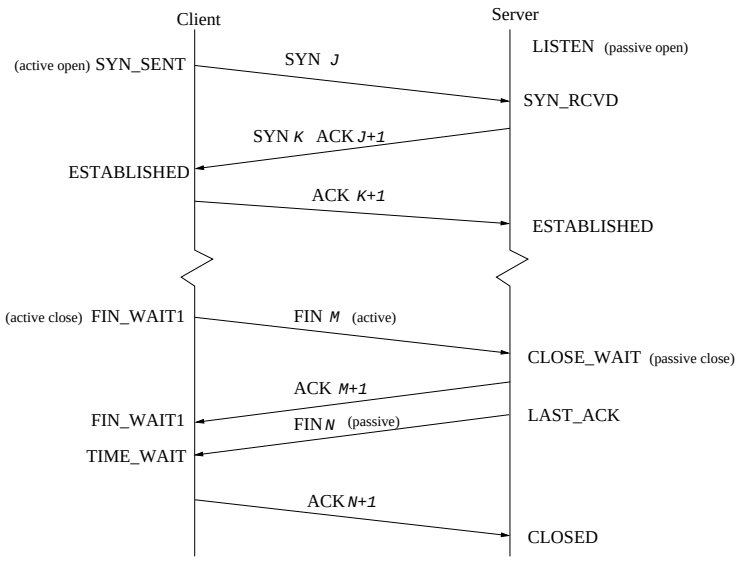


Figure 1: TCP states corresponding to normal connection establishment and teardown (from [42])

As shown in Figure 1 which is borrowed from [42], SYN and FIN packets delimit the beginning (SYN) and end (FIN) of each TCP connection in the same direction. In contrast, SYN and SYN/ACK packets signal the start of a TCP connection establishment in two opposing directions. Under the normal condition, one appearance of a SYN packet results in: (1) the eventual return of a FIN packet in the same direction; (2) the corresponding transmission of a SYN/ACK packet in the reverse direction within one round-trip time (RTT). Thus, the difference between the number of SYN and FIN (or SYN/ACK) packets can be utilized to sniff SYN flooding attacks, and the SYN-dog is an instance of the Sequential Change Point Detection [2].

Only a few additional variables are introduced to count the number of received TCP SYN, SYN/ACK, FIN and RST packets at each interface of a leaf router. We rely on packet classification to differentiate the various TCP control packets at leaf routers. Large-scale packet classification mechanisms [18, 23, 41] have been proposed and implemented, making it possible for routers to distinguish TCP control packets from others at a very high speed. Therefore, the SYN-dog's capability to withstand any flooding attacks depends on the ability of a leaf router in classifying and forwarding packets, typically at the rate of a million packets per second [23]. No per-connection state or state computation is involved in SYN-dog. Unlike the other network intrusion detection systems that maintain state for

each TCP connection like the TCP scrubber [28], SYN-dog does not have the cold-start problem[2] mentioned in [19].

SYN-dog is, in some sense, a by-product of the router infrastructure that can differentiate TCP control packets from data packets [44]. As SYN-dog tells TCP control packets from data packets, fine-grained service differentiation and resource isolation can be made on TCP flows. End-to-end TCP performance can be improved significantly as shown in [44]. Therefore, SYN-dog benefits not only victim servers but also the clients inside the stub network, making it attractive for wide deployment.

The efficacy of SYN-dog is evaluated by extensive trace-driven simulations. Traces taken from different sites at different times are used to test the sensitivity of SYN-dog. The evaluation results show that SYN-dog has short detection latency and high detection accuracy. Furthermore, SYN-dog is incrementally deployable and its implementation overhead is very low. The scheme of utilizing distinct protocol behaviors and the CUSUM algorithm can also be used for sniffing other types of flooding attacks.

The remainder of the paper is organized as follows. Section 2 briefly describes the working mechanisms of SYN and reflected SYN/ACK flooding attacks. Section 3 presents the SYN-dog framework, including the structure and placement of SYN-dog. Section 4 describes two different detection methods with respect to the different SYN pairs utilized. Section 5 details the proposed CUSUM algorithm for detecting abnormal traffic behaviors, and discusses its robustness against network anomalies. Section 6 evaluates the performance of SYN-dog using trace-driven simulations. The related work is discussed in Section 7. Finally, conclusions are drawn in Section 8.

2 SYN and Reflected SYN/ACK Flooding

A DDoS attack works as follows. An attacker sends control packets to the previously-compromised flooding sources, instructing them to target at a given victim. The flooding sources then collectively generate and send an excessive number of flooding packets to the victim, but with fake and randomized source addresses, so that the victim cannot locate the attackers.

SYN flooding attacks exploit the TCP's three-way handshake mechanism and its limitation in maintaining half-open connections. When a server receives a SYN request, it returns a SYN/ACK packet to the client. Until the SYN/ACK packet is acknowledged by the client, the connection remains in half-open state for a period of up to the TCP connection timeout, which is typically set to 75 seconds. The server has built in its system memory a backlog queue to maintain all half-open connections. Since this backlog queue is of finite size, once the backlog queue limit is reached, all connection requests will be dropped. If a SYN request is spoofed, the victim server will never receive the final ACK packet to complete the three-way handshake. Flooding spoofed SYN requests can easily exhaust the victim server's backlog queue, causing all the incoming SYN requests to be dropped. Note that the spoofed source address should be an invalid IP address (or the corresponding end-host is disconnected from the Internet at that time) so that it may be unreachable from the victim; otherwise, any end-host that receives the SYN/ACKs from the victim would send a RST to the victim. A RST packet is issued when the receiving host does not know what to do with the packet it received. Arrival of a RST causes the connection to be reset, foiling the flooding attack.

While the conventional SYN flooding is an attack of "system resource consumption," the recent reflected SYN/ACK flooding attacks [15] virtually "disconnect" a victim server from the Internet by hogging the link bandwidth between the victim and its ISP with an excessive number of SYN/ACK

[2] A "cold start" refers to the situation when a network intrusion detection system begins to run, or after it is restarted, it doesn't know how to deal with the incoming TCP traffic that belongs to the connections established earlier.

packets (a.k.a. bandwidth consumption attack). It is a kind of Distributed Reflection DoS (DRDoS) attacks [33]. In reflected SYN/ACK flooding attacks, a large number of innocent BGP routers (service port 179) and well-known TCP servers are exploited as the reflectors. The attacker "sprays" the spoofed SYN packets, whose source IP addresses are falsified as the victim's IP address, across a large number of reflectors. Each reflector alone only receives a moderate flux of spoofed SYN packets so that it can easily sustain the availability of its normal service. However, these innocent reflectors involuntarily reflect and amplify the malicious SYN packets. Their SYN/ACK responses, which are aggregated and flooded to the victim, are excessive, using up the link bandwidth between the victim and its ISP.

Note that in reflected SYN/ACK flooding attacks, all malicious SYN packets from the attacker must traverse the leaf router that connects the attacker to the Internet, in order to reach the Internet and then get sprayed across the numerous reflectors. The SYN-dog installed at this leaf router can detect the flow of these malicious SYNs, since no SYN/ACKs return to the attacker and the total number of malicious SYNs is still very large. So, the same method for sniffing SYN flooding attacks can be applied to sniff reflected SYN/ACK flooding attacks. To SYN-dog, the reflected SYN/ACK flooding attack is just a variation of conventional SYN flooding attack.

3 The Framework of SYN-dog

In this section, we first briefly describe the structure of SYN-dog, and then discuss where to place SYN-dog.

3.1 Structure of SYN-dog

SYN-dog consists of two *sniffers*, one at the in-bound interface and the other at the out-bound interface of a leaf router. We refer to the traffic from the Internet to the stub network as *in-bound*, and the traffic in the other direction as *out-bound*. The *in-bound sniffer* (*out-bound sniffer*) monitors the incoming (outgoing) TCP traffic. Figure 2 illustrates the structure of SYN-dog placed at a leaf router.

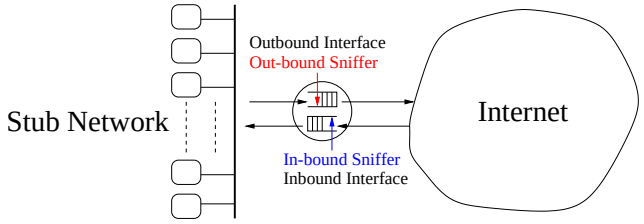


Figure 2: The structure of SYN-dog placed at a leaf router

Each leaf router can be either the first-mile or the last-mile router, depending on the direction of traffic between the stub network and the Internet. The SYN-dog at a leaf router, therefore, plays a dual role in detecting SYN flooding attacks:

- as the first-mile SYN-dog, it detects the flooding attacks that are originated from its local stub network and traces the flooding sources inside the local stub network; and
- as the last-mile SYN-dog, it detects flooding attacks on a server that resides inside the local stub network, and issues a warning signal upon detection of an attack.

The first-mile SYN-dog plays the primary role in sniffing a flooding attack, due mainly to its proximity to the flooding sources. Once an ongoing SYN flooding attack is detected, the first-mile SYN-dog's warning signal automatically indicates the flooding sources to be inside the stub network to which the SYN-dog is connected, without resorting to the IP traceback. However, the detection sensitivity of the first-mile SYN-dog may diminish as the number of attackers gets larger. In a large-scale DDoS attack, the flooding sources can be orchestrated so that each flooding traffic source may cause only an insignificant deviation from the normal traffic pattern.

In contrast, the last-mile SYN-dog can quickly detect the flooding attacks as all of the flooding traffic streams are merged at the last-mile router. Although it cannot provide any hint about the flooding sources, upon receipt of the last-mile SYN-dog's warning signal for an attack, the defense system like SynDefender can be triggered to protect the victim. To bring down the victim under protection, the flooding sources have to increase their flooding rates significantly, but this will make it easier for the first-mile SYN-dog to detect the flooding attack and locate its source(s). So, the last-mile SYN-dog plays an important complementary role in sniffing SYN flooding attacks.

3.2 Placement of SYN-dog

In addition to its installation at leaf routers, SYN-dog can also be placed at the firewall or the proxy server of a large organization which has only a single connection to the external world. All packets of a TCP session must pass through the same SYN-dog. However, we do not recommend the SYN-dog to be installed at core routers mainly because (1) it is close to neither flooding sources nor the victim; and (2) packets of the same flow could traverse different paths; (3) it is not always possible to accurately classify different TCP control packets at core routers due to the possible use of IPSec; (4) it cannot detect the reflected SYN/ACK flooding attacks easily, since malicious SYNs are diffused before reaching the core router.

As has been done in most intrusion detection (ID) systems, SYN-dog can be placed on the link that connects a stub network to the Internet by monitoring the bidirectional traffic on that link. However, besides the extra specialized equipment and manpower required, during high peak (near saturation) flow rates, almost no event of any kind would be logged by an ID system — they either have to drop packets at a very high rate or require a high-performance multi-CPU architecture for packet state analysis. As the link speed continues to improve, it will be more difficult for network flow monitors (that run on a typical PC) to pace with the network's packet transfer rate.

Recently, multi-homed ASs become necessary to improve availability, reliability and load-balancing. In such a case, the stub network is connected to the Internet by multiple leaf routers. However, as long as the packets that belong to the same TCP session go through the same leaf router, SYN-dog still works. If the packets of the same TCP session go through different leaf routers, we need a loose synchronization mechanism between the SYN-dogs in these leaf routers, which is the subject of our future work.

4 Two Detection Methods

Based on the distinct protocol behavior of TCP connection establishment and teardown, we utilize two types of packet pairs — SYN–FIN and SYN–SYN/ACK pairs — to detect SYN flooding attacks. According to the type of packet pairs used, we devise two different methods for detecting flooding attacks.

One may conceive a third way to detect a SYN flooding attack by using the connection state information and associating each SYN packet with the corresponding ACK that completes the three-way

handshake. However, as SYN-dog is stateless, it cannot tell if the ACK is for the received SYN/ACK or just for a data packet received. Therefore, the SYN–ACK pair scheme does not work in case of SYN-dog.

4.1 SYN–FIN pairs

In the first detection method that utilizes the SYN–FIN pairs, a SYN packet and the corresponding FIN, both in the same direction, are monitored by the same sniffer. No coordination and communication between these two sniffers are required. Therefore, in this detection method, the first-mile SYN-dog employs only the out-bound sniffer, while the last-mile SYN-dog uses only the in-bound sniffer. Although SYN packets can be distinguished from SYN/ACK packets, there is no way to discriminate active FINs from passive FINs, since each end-host behind a leaf router may be either a client or a server. So, the SYN–FIN pairs refer to both (SYN, FIN) and (SYN/ACK, FIN). In this detection method, the SYN packets are "generalized" to include the pure SYN and SYN/ACK packets.

Under a long-running normal condition, the TCP semantics has the one-to-one correspondence between SYNs and FINs. However, in reality there can always be a discrepancy between the number of SYNs and FINs. Besides the small number of long-lived TCP sessions, the other major cause of this discrepancy lies in the occurrence of RST packets. A single RST packet can terminate a TCP session without generating any FIN packet, which violates the SYN–FIN pair's protocol behavior. RSTs are generated for two reasons: (1) *passive*, or the RST is transmitted upon arrival of a packet at a closed port; (2) *active*, or the RST is initiated by a client to abort a TCP connection.[3] Each active RST is associated with the SYN from the same session, and both of them can be seen by the same sniffer. In contrast, a passive RST cannot be associated with any SYN seen by the same sniffer as the passive RST and its corresponding SYN must go through different sniffers. Furthermore, passive RSTs may even have nothing to do with SYNs. For instance, late arrival of a data packet at the port that has already been closed, will trigger the transmission of a RST. We treat passive RSTs as a background noise.

In general, three types of SYN pairs are considered as the normal behavior of TCP in the first detection method: (SYN, FIN), (SYN/ACK, FIN) and (SYN, $\text{RST}_{\text{active}}$). Unfortunately, SYN-dog cannot distinguish active RSTs from passive ones. There are two simple but extreme ways to resolve this problem: one is to treat all RSTs as active, and the other is to treat all RSTs as passive. The first approach degrades the SYN-dog detection sensitivity, while the second raises the SYN-dog false alarm rate. To make a trade-off between detection sensitivity and false alarm rate, it is necessary to set an appropriate threshold to filter most of the background noise. Based on our observation, under the normal condition: (1) SYNs and RSTs have a strong positive correlation; (2) the difference between the number of SYNs and that of FINs is close to the number of RSTs. These imply that passive RSTs constitute only a small percentage of the entire RSTs. So, we set the threshold to 75%, i.e., 3 out of 4 RSTs are treated as active. Moreover, for the following reason, SYN-dog can withstand the negative impact of passive RSTs that are incorrectly classified as active ones: at the end of each observation period, the CUSUM algorithm resets any negative difference between the number of SYNs and that of FINs (RSTs) to zero, so the spike of background noise is confined to one observation period only, preventing its cumulative effects.

The weakness of the SYN–FIN pairs scheme lies in its vulnerability to simple counter-measures. Once the attacker is aware of the presence of such a detection mechanism, it can paralyze the mechanism

[3] Active RSTs are issued mostly by clients. In its own best interest, a server rarely sends RST packets to clients once the TCP connection is established.

by flooding a mixture of SYNs and FINs (RSTs). Although one can argue that by doubling its flooding traffic, the attacker increases the possibility of being traced back, one may still wonder if there is a better way to overcome this shortcoming.

4.2 SYN–SYN/ACK pair

Fortunately, there is an alternative that is difficult for an attacker to counter. In the normal TCP three-way handshake, an out-bound SYN induces an in-bound SYN/ACK within a round-trip time. In contrast, for the flooded SYNs, because their spoofed IP source addresses are randomized, most of the corresponding SYN/ACKs will never return to the flooding sources, and hence, cannot go through the same leaf router as those flooding SYNs as shown in Figure 3 (mis-match part).

The second detection method makes use of SYN–SYN/ACK pairs to sniff flooding attacks. Since SYN/ACK packets are generated by the victim server, it is much more difficult for the flooding sources to evade the SYN-dog. Moreover, as compared to the SYN–FIN pair scheme, the interval between SYN and SYN/ACK is bounded by one RTT, not by the duration of a TCP session that has much larger variations.

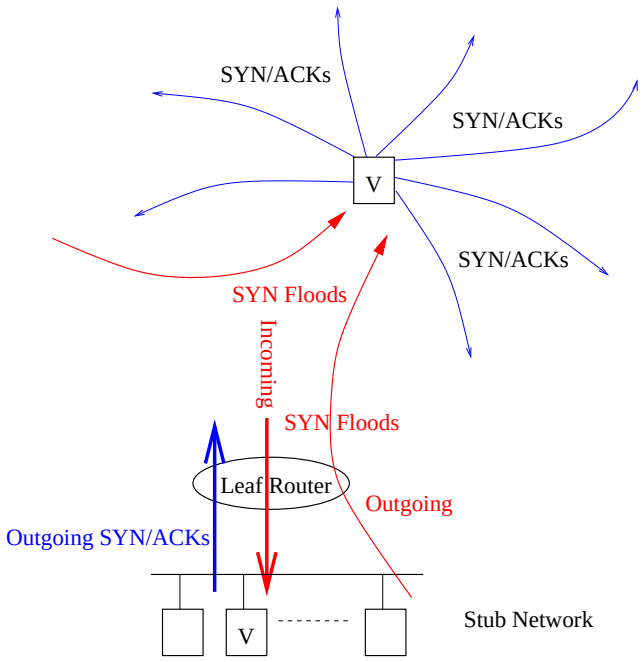


Figure 3: Match and mis-match between SYN–SYN/ACK pair at a leaf router

On the other hand, there are two disadvantages of the second detection method. First, unlike the first detection method, the out-bound sniffer and the in-bound sniffer must be coordinated. The out-bound sniffer maintains the count of outgoing SYNs and the in-bound sniffer keeps track of incoming SYN/ACK packets. At the end of each observation period, the count information must be exchanged between the two sniffers. Second, the SYN–SYN/ACK pair scheme is restricted to be used by the first-mile SYN-dog only, which sniffs the flooding sources inside the local stub network. It lacks the capability to issue a timely last-mile flooding warning to the network administrator of the local stub network that is under attack. The reason for this is that (1) each victim server generates a SYN/ACK in response to each SYN it received, regardless whether it is spoofed or not; (2) the incoming SYN flood and the outgoing SYN/ACKs pass through the same local leaf router. This phenomenon is illustrated

in Figure 3 (match part). So, there is no noticeable difference between the number of incoming SYN packets and that of the outgoing SYN/ACKs generated by the victim servers until the victim servers are totally shut down and no more SYN/ACKs are generated.

It is, however, very important for the last-mile SYN-dog to detect the ongoing flooding attack in a timely manner. Therefore, the last-mile SYN-dog will rely on SYN–FIN pairs for timely detection of an incoming SYN flooding attack.

4.3 Summary of SYN-dog

In summary, SYN-dog plays a dual role: one as the first-mile SYN-dog for sniffing flooding sources, and the other as the last-mile SYN-dog for issuing attack warnings. To build a robust and powerful SYN-dog, both flooding detection methods are included in the SYN-dog. The SYN–SYN/ACK pair method is employed by the first-mile SYN-dog to sniff flooding sources inside the local stub network, while the SYN–FIN pairs method is used by the last-mile SYN-dog to detect incipient flooding attacks, and issue a warning to the local network administrator.

Note that the methodology employed by the SYN-dog can be easily extended to detect other types of flooding attacks, such as TCP ACK flooding and ICMP flooding attacks. Based on their distinct protocol behaviors, the ACK-dog or ICMP-dog can detect an ongoing flooding attack by observing the violation of normal protocol behaviors: for ACK-dog, the imbalance of TCP ACK packets versus the corresponding TCP data packets in the reverse direction; for ICMP-dog, the imbalance of outgoing ICMP Echo requests versus incoming ICMP Echo replies.

5 Statistical Attack Detection

SYN-dog belongs to the commonly-known network-based ID system: an intruder will be singled out if its behavior is noticeably different from that of a legitimate user. Like most statistical anomaly-detection systems, we compare the observed sequence with the profile that represents the user's normal behavior, and detect any significant deviation from the normal behavior. However, unlike the traditional network ID system that *passively* monitors bidirectional traffic streams on network links, SYN-dog is transparently interposed at a leaf router and is implemented as an integrated component of the leaf router.

The burstiness of TCP connection request arrivals [12] makes the detection of attacks much harder, since there is no natural length of burst for self-similar traffic. Furthermore, the normal TCP-arrival pattern is site- and time-dependent. However, the strong positive correlation between SYN–FIN (RST) and SYN–SYN/ACK pairs clearly indicates the presence of SYN flooding. According to the specification of TCP/IP protocol [35, 42], in its normal operation, a FIN (RST) is paired with a SYN at the end of data transmission, and the arrival of a SYN/ACK in the reverse direction follows the transmission of a SYN within one RTT. However, under a SYN flooding attack, this SYN–FIN (RST) or SYN–SYN/ACK pair's behavior will deviate significantly from the normal operation.

5.1 Data Sampling and Detection Mechanism

We collect the numbers of SYN, SYN/ACK, and FIN (RST) packets during every observation period t_0 at leaf routers, which determines the detection resolution. In order to relate the SYN and FIN (RST) packets of the same connection, the sampling time of FIN (RST) is delayed by t_d after SYN is sampled, where t_d is so chosen that a significant portion of connections requested during the SYN sampling period terminate in the corresponding FIN (RST) sampling period. Recent Internet traffic

measurements [43] have shown that most of TCP connections last 12–19 seconds, so we set the sampling delay t_d to 10 seconds. In contrast, since most RTTs are less than 0.5 second, we start the collection of SYNs at the out-bound sniffer and SYN/ACKs at the in-bound sniffer simultaneously. To balance the detection resolution and the algorithm's stability and accuracy, we set t_0 to 10 seconds. Note, however, that both parameters are tunable and our algorithm is not very sensitive to this choice.

Under the normal condition, the difference between the collected number of SYNs and FINs (RSTs) or SYN/ACKs is very small, as compared to the total number of TCP connection requests. This observation holds in spite of the fact that the total number of TCP connection requests may be bursty on a small time scale, and slowly-varying on a large time scale. In other words, the correlation between the numbers of SYNs and FINs (RSTs) or SYN/ACKs is not sensitive to the request arrival process. The results presented in Section 6.1 clearly show that the consistent synchronization between SYNs and FINs (RSTs) or SYN/ACKs is independent of the sites and time-of-day.

Under SYN flooding attacks, the flooding SYN traffic has significant regularity and semantics that can be filtered out. Recent experiments with SYN attacks on commercial platforms [9] show that the minimum flooding rate to overwhelm an unprotected server is 500 SYN packets per second. Even with a specialized firewall designed to resist against SYN flooding, a server can be disabled by a flood of 14,000 packets per second [9]. To bring down the victim server for 10 minutes, for example, attackers must collectively inject at least 300,000 SYN packets. During the same time period, however, the numbers of counted FINs (RSTs) and SYN/ACKs remain largely unchanged. Therefore, there will be much more SYNs than FINs (RSTs) or SYN/ACKs collected during the flooding period. The difference between the numbers of SYNs and FINs (RSTs) or SYN/ACKs will increase dramatically, and remain large during the whole flooding period, which typically lasts for several minutes [30]. So, the occurrence of a large difference between the numbers of SYNs and FINs (RSTs) or SYN/ACKs indicates a possible SYN flooding attack. This will be used in our attack detection.

5.2 The CUSUM Algorithm

The CUSUM algorithm is applied to both of the two detection methods. For ease of presentation, we only show how it works in the SYN–FIN pair scheme, which is similar to the SYN–SYN/ACK pair scheme except that the collection of FINs (RSTs) is replaced with that of SYN/ACKs.

Let $\{\Delta_n, n = 0, 1, \cdots\}$ be the number of SYNs minus that of the corresponding FINs (RSTs) collected within one sampling period. To further alleviate its dependence on the time, access pattern and size of the network, $\{\Delta_n\}$ is normalized by the average number, $\bar{F}$,[4] of FINs (RSTs) during each sampling period. $\bar{F}$ can be estimated in real time and updated periodically. An example of recursive estimation and update of $\bar{F}$ is:

$$\bar{F}(n) = \alpha \bar{F}(n-1) + (1-\alpha)\text{FIN (RST)}(n), \tag{1}$$

where n is the discrete time index and α is a constant, whose default value is 0.01, lying strictly between 0 and 1 that represents the memory in the estimation.

Let $X_n = \Delta_n / \bar{F}$. Under normal condition, the mean of X_n, denoted as c, is much less than 1 and close to 0. $\{X_n\}$ is no longer dependent on the network size or time-of-day. Its dynamics are solely the consequence of the TCP protocol specification. So, we can consider $\{X_n\}$ as a stationary random process.

Our attack detection algorithm is based on the Sequential Change Point Detection [2]. The objective of Change Point Detection is to determine if the observed time series is statistically homogeneous, and if

[4]The average number of SYN/ACKs is represented as $\bar{K}$.

not, to find the point in time when the change happens. It has been studied extensively by statisticians. See [2] and [6] for a good survey. There have been various tests for different problems. They can be largely divided into two categories: posterior and sequential. Posterior tests are done off-line where the entire data is collected first and then a decision of homogeneity or a change point is made based on the analysis of all the collected data. On the other hand, sequential tests are done on-line with the data presented sequentially and the decisions are made on-the-fly.

We adopt a sequential test for quicker response when an attack occurs. It also saves memory and computation. One difficulty, however, is the modeling of $\{X_n\}$. Despite a number of previous results on the modeling of TCP connection request arrivals [7, 8, 12, 17, 34, 38], there is no consensus on whether it should be modeled as self-similar or Poisson. For dynamic and complex systems like the Internet, it may not be possible to model the total number of TCP connection request arrivals by a simple parametric description. So, we seek robust tests which are not model-specific. In fact, non-parametric methods fit this requirement very well. Specifically, we use the non-parametric CUSUM (Cumulative Sum) method [6] for detection of attacks. This method enjoys all the virtues of sequential and non-parametric tests, and the computation load is very light. When the time series is i.i.d. with a parametric model, CUSUM is asymptotically optimal for a wide range of Change Point Detection problems [2, 6].

$\{X_n\}$ is assumed to satisfy the following two conditions.

C1: $\{X_n\}$ is ψ-mixing, meaning that the $\psi(s)$ parameters, defined below, approach 0 as $s \to \infty$:

$$\psi(s) \stackrel{def}{=} \sup_{t \geq 1} \sup_{\substack{A \in \mathcal{F}_1^t, \\ B \in \mathcal{F}_{t+s}^{\infty}, \\ P(A)P(B) \neq 0}} \left| \frac{P(AB)}{P(A)P(B)} - 1 \right|, \tag{2}$$

where $\mathcal{F}_1^t$ is the σ-algebra generated by $\{X_1, X_2, \cdots, X_t\}$ and $\mathcal{F}_{t+s}^{\infty}$ is the σ-algebra generated by $\{X_{t+s}, X_{t+s+1}, \cdots\}$. $\psi(s)$ is affected by the dependency among the $\{X_n\}$ samples: highly dependent $\{X_n\}$ has $\psi(s)$ that decays slowly as $s \to 0$.

C2: The marginal distribution of $\{X_n\}$ satisfies the following regularity condition: $\exists t > 0$ such that $E(e^{tX_n}) < \infty$.

The details of these conditions can be found in [6]. Note that ψ-mixing is a much looser requirement than independence, and X_n being ψ-mixing only indicates that X_n is not "extremely" dependent. In practice, both conditions are mild and easily satisfiable, even for long-range dependent arrival processes. In general, $E(X_n) = c \ll 1$. We choose a parameter a that is the upper bound of c, i.e., $a > c$, and define $\tilde{X}_n = X_n - a$ so that it has a negative mean during normal operation. When an attack takes place, $\tilde{X}_n$ will suddenly become a large positive number. Suppose, during an attack, the increase in the mean of $\tilde{X}_n$ can be lower-bounded by h. Our change detection is based on the observation of $h \gg c$.

Let

$$\begin{aligned} y_n &= (y_{n-1} + \tilde{X}_n)^+, \\ y_0 &= 0, \end{aligned} \tag{3}$$

where x^+ is equal to x if $x > 0$ and 0 otherwise. The meaning of y_n can also be understood as follows: if we define $S_k = \sum_{i=1}^{k} \tilde{X}_i$, with $S_0 = 0$ at the beginning, it is straightforward to show that

$$y_n = S_n - \min_{1 \leq k \leq n} S_k, \tag{4}$$

i.e., the maximum continuous increment until time n. A large $\{y_n\}$ is a strong indication of an attack. Since Eq. (3) is recurrent and much easier to compute than Eq. (4), we will use it in making detection decisions.

Let $d_N(.)$ be the decision at time n: '0' for normal operation (homogeneity) and '1' for attack (a change occurs). Here N represents the flooding threshold:

$$d_N(y_n) = \begin{cases} 0 & \text{if } y_n \leq N, \\ 1 & \text{if } y_n > N. \end{cases} \tag{5}$$

In other words, $d_N(y_n) = I(Y_n > N)$, where $I(.)$ is the indicator function. The purpose of introducing a is to offset the possible positive mean in $\{X_n\}$ caused by network anomalies so that the test statistic y_n will be reset to zero frequently and will not accumulate with time.

Let P_m and E_m be the probability measure and the expected value generated by $\{\tilde{X}_n\}$ with the attack occurring at time m (change point at time m); let P_∞ and E_∞ be the counterparts without any attack (no change point). There are two fundamental performance measures for the sequential change point detection.

False alarm time (the time without false alarm): the time duration with no false alarm reported when there is no attack.

Detection time: the detection delay after the attack starts.

One would want the second measure to be as small as possible while keeping the first measure as large as possible. However, they are conflicting goals and cannot be simultaneously achieved. Therefore, the design philosophy of a statistical change point detection is to minimize the detection time subject to a certain false alarm tolerance. In order to compare the performance of different detection schemes, some criteria of false alarms must be specified, like average time between two consecutive false alarms, worst-case false alarm time, and so on. An algorithm is said to be *optimal* with respect to a certain criterion if it minimizes the detection time for an attack among all the detection schemes subject to the false alarm constraint. The CUSUM rule has been shown to be asymptotically optimal with respect to the worst-case mean false alarm time in the change point detection problems involving a known parametric model and independent observations [2].

Due to the lack of a complete model for $\{\tilde{X}_n\}$, it is difficult to discuss optimality. The choice of CUSUM is based on its simplicity in computation and non-parametric implementation, as well as its generally excellent performance. It has been shown in [6] that, with the choice of a, the upper bound in case of normal operation, and N, the flooding threshold, as $N \to \infty$, we have

$$\sup_n |\ln P_\infty(d_N(n) = 1)| = O(N), \tag{6}$$

which is equivalent to

$$P_\infty\{d_N(n) = 1\} = c_1 \exp(-c_2 N). \tag{7}$$

where c_1 and c_2 are constants, depending on the marginal distribution and mixing coefficients of $\{\tilde{X}_n\}$. In other words, the time between consecutive false alarms grows exponentially with N. The burstiness of traffic is reflected by the mixing coefficients $\psi(s)$, and thus, does impact the detection performance. However, the constants c_1 and c_2 only play a secondary role and can be ignored in practice.

In order to study the detection time, let us define

$$\begin{aligned} \tau_N &= \inf\{n : d_N(.) = 1\}, \\ \rho_N &= \frac{(\tau_N - m)^+}{N}, \end{aligned} \tag{8}$$

where ρ_N represents the normalized detection time after a change occurs and m represents the starting time of the attack. In CUSUM, for any $m \geq 1$, if h is the actual increase in the mean of $\{\tilde{X}_n\}$ during an attack, we have

$$\rho_N \rightarrow \gamma = \frac{1}{h - |c - a|}, \tag{9}$$

where $h - |c - a|$ is the mean of $\{\tilde{X}_n\}$ when $n > m$ (after an attack starts). However, since h is a bound rather than a true value, the above is a conservative estimation (upper bound) of the actual detection time.

5.3 Robustness against Network Anomalies

While there is no strict one-to-one match, under the normal condition, a very strong positive correlation between the numbers of SYNs and FINs (RSTs) or SYN/ACKs does exist as shown in Section 6.1. The discrepancy between the numbers of SYN and FIN (RST) or SYN/ACK packets lies in SYN losses and subsequent retransmissions. The SYN losses are caused by various network anomalies, including network congestion, routing loops and link failures. Clearly, these network anomalies reduce the sniffing sensitivity of SYN-dog.

Fortunately, these network anomalies are triggered by unrelated events; and to date there exist few evidences indicating that these different network anomalies are closely correlated. The recent Internet measurement studies have shown that: (1) there is very few congestions inside the core of Internet, where the bandwidth over-provisioning has been widely used and the link utilization varies from 3 to 60% [5]; (2) the majority of routing loops last shorter than 10 seconds [21]; and (3) link failures are fairly well spread even in the time scale of minutes [20]. Therefore, these randomly-occurring network anomalies can be treated as white noise. To offset the effect of white noise, a safety margin can be added when the normal upper bound, a, is set, without jeopardizing the sniffing sensitivity. Only a severe network congestion, long lasting routing loops, and bursty occurrence of link failures — which rarely happen — can confuse the SYN-dog to issue false alarms.

5.4 Parameter Specification

To implement the CUSUM algorithm, we only need to specify three tunable parameters: a, the upper bound in case of normal operation; h, the lower bound of the increase in case of an attack; and, N, the flooding threshold.

The CUSUM algorithm requires $E(\tilde{X}_n) < 0$ before the change point, and $E(\tilde{X}_n) > 0$ after it, i.e., $a > c$ and $h > a$. Based on the discussion in the previous section, to ensure a long false alarm time and make it independent of network size and access pattern, we set $h = 2a$ in our design. As the last-mile SYN-dog that utilizes SYN–FIN pairs for flooding detection monitors the incoming traffic, all the SYN flooding packets converge, and therefore, a large difference between the numbers of SYN and FIN (RST) packets is easily observable with $h \gg c$. In this case, the detection is not sensitive to the choice of a. With a large safe margin, we can simply choose $a = 1$ and $h = 2$.

In contrast, as the first-mile SYN-dog that employs SYN–SYN/ACK pairs for flooding detection monitors the outgoing SYN and incoming SYN/ACK traffic, only part of the flooding SYN packets can be seen by each detector because the attack may be initiated from many sites simultaneously. Thus, a proper choice of a is more important. To balance the detection sensitivity and false alarm rate, we set $a = 0.35$ and $h = 0.7$. Note that the choices of a and h are independent of the network size and access pattern. In doing so, a universal false alarm rate can be realized for easy implementability of

our detection and sniffing mechanism. On the other hand, in practice, the network administrator of the involved edge router can incorporate site-specific information so that the algorithm can achieve higher detection performance.

Based on a and h, the flooding threshold N can be specified as follows: (i) assume $c = 0$, and γ can thus be obtained from Eq. (9); and (ii) specify a target detection time (i.e., the product of γ and N) such that the flooding threshold N is determined by Eq. (8). We choose t_0 as the designed detection time for the last-mile SYN-dog, hence $\gamma = 1$ and $N = 1$. In contrast, we choose $3t_0$ as the counterpart for the first-mile SYN-dog, hence $\gamma = 2.86$ and $N = 1.05$.[5] Compared to the last-mile SYN-dog, the short detection time of the first-mile SYN-dog is not so crucial to the victim: the revelation of the flooding sources is more valuable, although it may take longer time. Note that the value of N is partially determined by the designed detection time, so it may not be larger than the value of h.

It is worth noting that our algorithm is to check the cumulative effect of an attack. So, it can detect attacks with the SYN flooding rate less than h at the expense of a longer response time. The actual lower bound of detection sensitivity in terms of SYN flooding rate, f_{min}, can be given as

$$f_{min} = (a - c) \cdot \frac{\bar{F}}{t_0}. \tag{10}$$

Furthermore, the detection capability is not sensitive to the flooding pattern: it can detect the attacks with both constant and bursty flooding rates. The effectiveness of this detection is evaluated by trace-driven simulations.

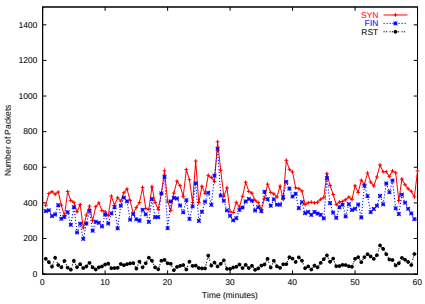
6 Performance Evaluation

To evaluate and validate the SYN-dog, we have conducted trace-driven simulation experiments. The trace data we used are collected from four different sites at different times. The first trace was gathered at DEC's (now HP) primary Internet access point, which is an Ethernet DMZ network. It contains an hour's worth of all wide-area traffic between DEC Western Research Lab and the Internet on March 9th, 1995. The second trace was taken on March 13th, 1997 on a 10 Mbps Ethernet connecting Harvard's main campus to the Internet, which is a half-hour trace. The third set was obtained by placing network monitors on the high-speed link (OC-12, 622 Mpbs) that connects the University of North Carolina at Chapel Hill (UNC) campus network to the rest of the world. The trace was collected on September 27th, 2000. The fourth set was collected at the Internet access link that connects the University of Auckland at New Zealand to the rest of the world. The tracing ran from 14:36 to 17:47 on Thursday, December 5th, 2000.

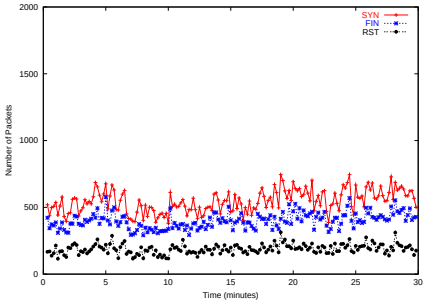
Table 1: A summary of the trace features

Trace	Starting time	Traffic type
DEC	2:00, Thu Mar 9, 1995	Bi-directional
Harvard	12:39, Thu Mar 13, 1997	Bi-directional
UNC-in	19:30, Wed Sept 27, 2000	Uni-directional
UNC-out	19:30, Wed Sept 27, 2000	Uni-directional
Auckland-in	14:36, Thur, Dec 5, 2000	Uni-directional
Auckland-out	14:36, Thur Dec 5, 2000	Uni-directional

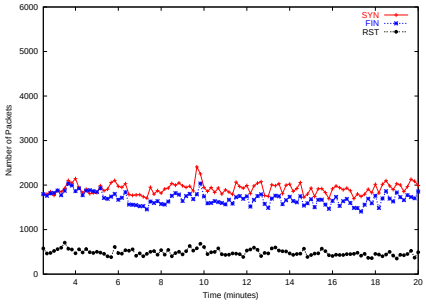
[5] N may not seem to be large but Eq. (9) can serve as an approximation.



(a) DEC

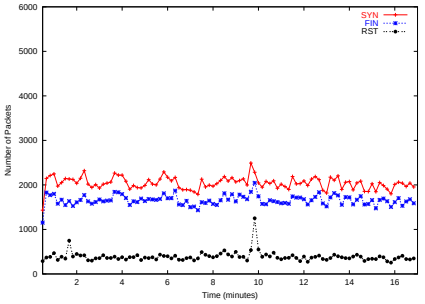


(b) Harvard

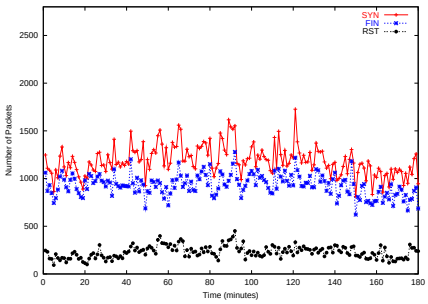


(c) UNC-in

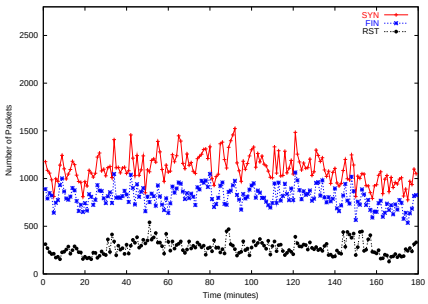
Figure 4: The dynamics of SYN and FIN (RST) packets (part I)



(a) UNC-out

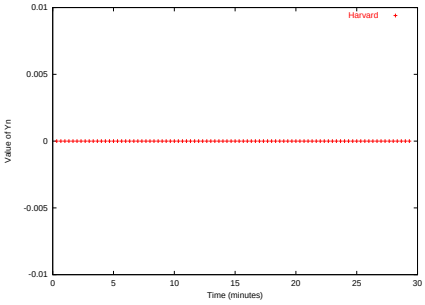


(b) Auckland-in

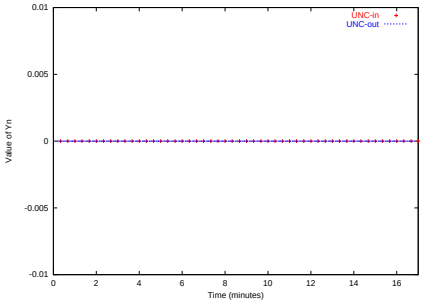


(c) Auckland-out

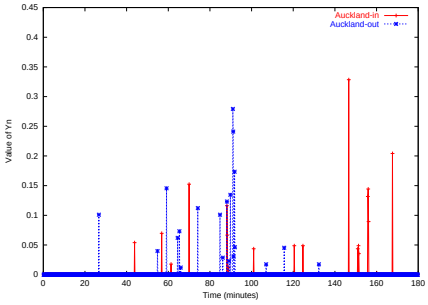
Figure 5: The dynamics of SYN and FIN (RST) packets (part II)



(a) Harvard

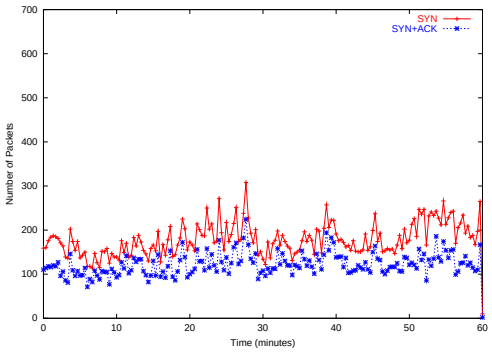


(b) UNC

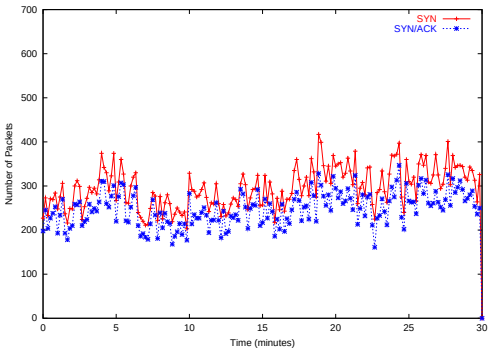


(c) Auckland

Figure 6: CUSUM test statistics under normal operation

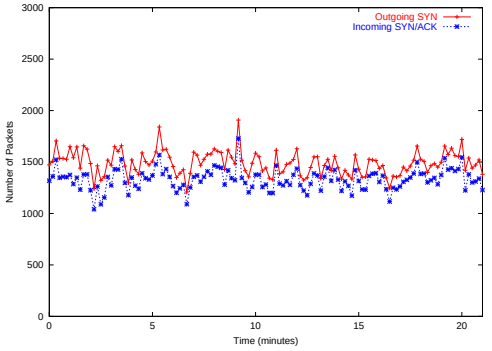


(a) DEC

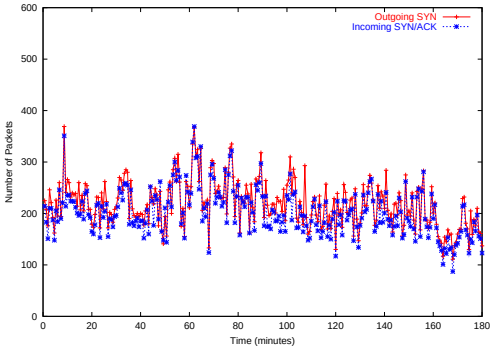


(b) Harvard

Figure 7: The dynamics of SYN and SYN/ACK packets at DEC and Harvard



(a) UNC



(b) Auckland

Figure 8: The dynamics of SYN and SYN/ACK packets at UNC and Auckland

The traces used in our experiments are summarized in Table 1. Note that the DEC and Harvard traces are mixed traffic collections in both directions, but the UNC and Auckland traces are uni-directional: UNC-in and Auckland-in collected the traffic data from the Internet to the UNC and Auckland campus networks, respectively, while UNC-out and Auckland-out collected the traffic data from the UNC and Auckland campus networks to the Internet, respectively.

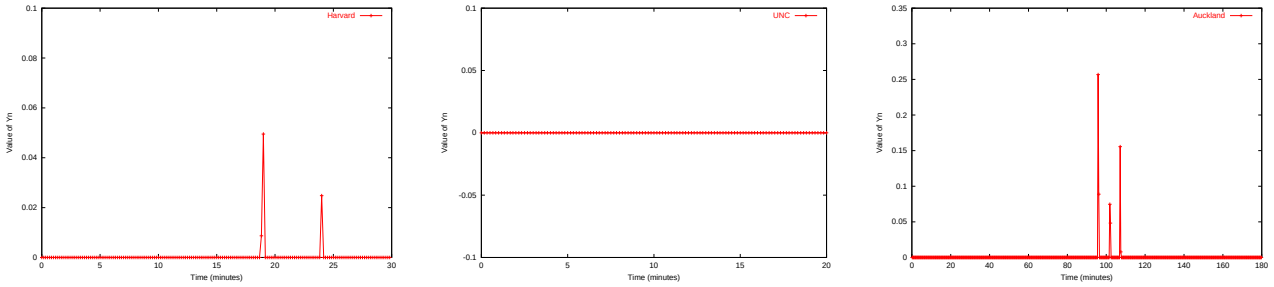
6.1 Normal Traffic Behavior

The three sets of traces represent the normal traffic behaviors at the exchange points between different stub networks and the Internet at different times. We parse the traces and extract the TCP SYN, SYN/ACK, FIN and RST packets from the TCP traffic.

6.1.1 SYN–FIN Pairs

The dynamics of SYN,[6] FIN and RST packets at the DEC site are illustrated in Figure 4 (a), and the corresponding result from the Harvard trace is illustrated in Figure 4 (b). Those from UNC-in and UNC-out are in Figures 4 (c) and 5 (a), and Auckland-in and Auckland-out are in Figures 5 (b) and (c), respectively. They clearly show the consistent synchronization between SYN and FIN (RST) packets.

[6]It is generalized SYNs, including SYNs and SYN/ACKs.



(a) Harvard (b) UNC (c) Auckland

Figure 9: CUSUM test statistics under normal operation at: Harvard, UNC and Auckland

The consistency indicates that the synchronization is an inherent traffic behavior and independent of time and sites.

We have applied the CUSUM algorithm on all the available traces without adding attacks. The test statistics, $\{y_n\}$, for all traces are plotted in Figure 6. For the Harvard and UNC traces, y_n's are constantly zeros. For the Auckland traces, more than 99% y_n's stay at zero. The isolated bursts in y_n are always much smaller than the threshold $N = 1.05$: the maximal spikes of y_n in Auckland-in and Auckland-out are 0.32 and 0.27, respectively. So, no false alarms are reported.

6.1.2 SYN—SYN/ACK pair

The dynamics of SYN and SYN/ACK packets at the DEC and Harvard sites are illustrated in Figures 7 (a) and (b), respectively. The outgoing SYNs and incoming SYN/ACKs from the UNC and Auckland traces are shown in Figures 8 (a) and (b). As with SYN–FIN pairs, these figures clearly demonstrate a consistent positive correlation between SYN and SYN/ACK packets. The consistency indicates that the strong positive correlation is also a distinct protocol behavior and independent of time and sites. Note that in the figures of the DEC and Harvard traces, SYNs and SYN/ACKs are collected from both directions, instead of "Outgoing SYN" and "Incoming SYN/ACK" as shown in the UNC and Auckland traces.

Also, we have applied the CUSUM algorithm on the Harvard, UNC and Auckland traces without adding flooding attacks. The test statistics, $\{y_n\}$, for the Harvard and UNC traces are plotted in Figures 9 (a) and (b); that for the Auckland trace is plotted in Figure 9 (c). As expected, for all the traces tested, y_n's are mostly zeros. Among the isolated spikes of y_n in the Harvard trace, the maximum is about 0.05; the maximal spike of y_n in the Auckland trace is about 0.26. Both are much smaller than the flooding threshold $N = 1.05$. So, no false alarms are reported.

6.2 SYN Flooding Detection

With the appearance of Trinoo, which only implements UDP packet flooding, many tools have been developed to create DDoS attacks. Most of them, such as Tribe Flood Network (TFN), TFN2K, Stacheldraht, Trinity, Plague and Shaft, generate TCP SYN flooding attacks and randomize all 32 bits of the source IP address [10, 11]. Although these DDoS attack tools employ different ways to coordinate attacks with the goal of achieving robust and covert DDoS attacks, their flooding behaviors are similar in that the SYN packets are continuously bombarded to the victim.

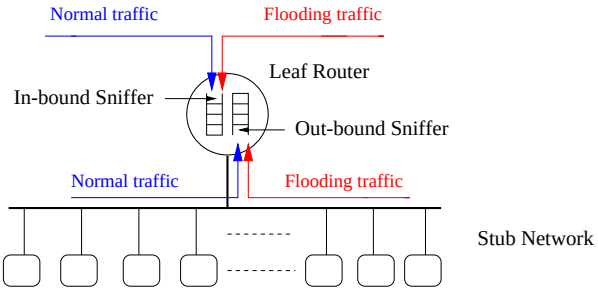


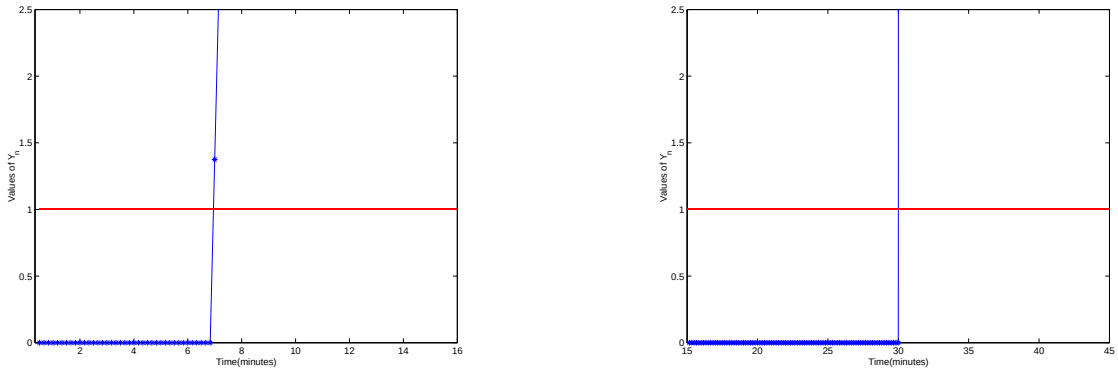
Figure 10: The trace-simulation flooding attack experiment

In the SYN flooding detection experiments, the UNC and Auckland 2000 traces are used as the normal background traffic. Among them, UNC-in or Auckland-in is used for incoming background traffic, and UNC-out or Auckland-out is for outgoing background traffic. The flooding traffic is mixed with the normal traffic, and the SYN-dog at the leaf router is simulated, as shown in Figure 10. Because the non-parametric CUSUM method is used for detection of flooding attacks, the flooding traffic pattern or its transient behavior (bursty or not) does not affect the detection sensitivity. Rather, the detection sensitivity depends only on the total volume of flooding traffic. So, without loss of generality, we assume that the flooding rate is constant.

In DDoS attacks, the flooding traffic seen by the first-mile and the last-mile SYN-dogs is quite different. The flooding traffic passing through the last-mile SYN-dog is the aggregation of the flooding traffic from all distributed attack sources, allowing for much easier detection of an attack. However, the flooding detection at the first-mile SYN-dog is much more difficult. In a large-scale DDoS attack, the flooding sources can be so coordinated that the traffic from each flooding source may not be noticeable at all. Suppose the minimum SYN flooding traffic to bring down a TCP server is V packets per second. Then, the flooding rate at the last-mile SYN-dog is V, but the flooding rate seen by the first-mile SYN-dog may be much smaller than this.

We assume that the flooding traffic is evenly distributed among different flooding sources and there is only one flooding source inside each stub network. The flooding rate seen by the first-mile SYN-dog, f_i, equals the individual flooding rate inside the same stub network. Therefore, f_i is determined by $\frac{V}{A_s}$, where A_s is the total number of the stub networks that contain flooding sources. This setting is intended to "hide" the attack from the first-mile SYN-dog. That is, the less the flooding sources inside the stub network, the less flooding traffic seen by the first-mile SYN-dog and the harder to detect the flooding attack. The flooding duration in all experiments is set to 10 minutes, a typical attack duration observed in the Internet [30]. The starting time of flooding attacks in the UNC traces is randomly chosen between 1 and 9 minutes, but the starting time in the Auckland traces lies between 3 and 166 minutes.

We first examine the detection sensitivity at the last-mile SYN-dog, which employs SYN–FIN pairs as its detection method. To demonstrate the high sensitivity of last-mile SYN-dog to SYN flooding, the flooding rate V is set to its minimum, 500 SYNs per second. The simulation results are plotted in Figures 11 (a) and (b), showing that the cumulative sum y_n exceeds the flooding threshold "1" in one observation period, i.e., the fastest response can be achieved, in the Auckland and UNC trace cases, respectively. So, the last-mile SYN-dog of the Auckland case can detect the SYN flooding attack in 10 seconds, and so does the last-mile of the UNC case. Once the flooding attack is detected, a defense system like SynDefender can be triggered to protect the victim from the flooding attack. To paralyze the defense system at the victim, attackers have to increase their flooding rate, and the first-mile SYN-dog will then be more likely to detect and locate the flooding sources inside the stub network.



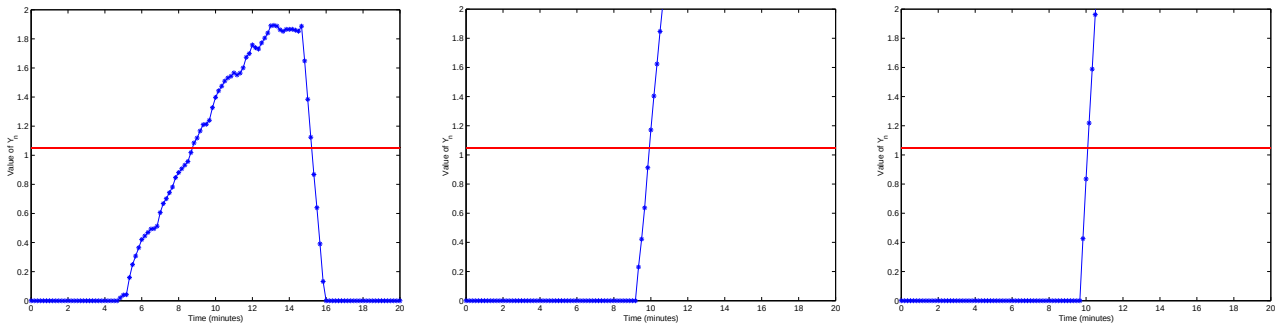
(a) UNC case (b) Auckland case

Figure 11: SYN flooding detection sensitivity at the last-mile SYN-dog

To examine the detection sensitivity of the first-mile SYN-dog, which employs SYN–SYN/ACK pairs to detect attacks, we vary the flooding rate f_i seen by the first-mile SYN-dog, i.e., the individual flooding rate inside the stub network. As the last-mile detection is much easier than the first mile, we only study the detection probability and detection time for the latter. We conduct the SYN flooding detection experiments on the UNC and Auckland traces.

6.2.1 The UNC Case

Using the UNC traces as the background traffic, we observe the dynamics of y_n. Figures 12 (a), (b) and (c) plot the dynamic behaviors of y_n when f_i is set to 35, 60 and 80 SYNs per second, respectively. The accumulative effects of SYN flooding are clearly shown in these figures. In the cases of 60 and 80 SYNs per second, the first-mile SYN-dog can detect the SYN flooding attack in 4 and 2 observation periods, respectively. However, in the case of 35 SYNs per second, the first-mile takes a much longer time (about 24 observation periods, i.e., 4 minutes) to exceed the flooding threshold of 1.05. The detection performance of the first-mile SYN-dog in the context of the UNC traces is summarized in Table 2, which lists the detection probabilities and detection times for different f_i values. The detection time is measured in number of the observation period t_0, which is set to 10 seconds.



(a) 35 SYNs per second (b) 60 SYNs per second (c) 80 SYNs per second

Figure 12: SYN flooding detection sensitivity of the SYN-dog at UNC

Clearly, larger flooding rates lead to faster and easier detection of attacks. According to Eq. (10), the lower detection bound is about 37 SYNs per second in this simulation scenario. If we implement the same SYN-dog at a smaller subnet, $\bar{K}$ — the average number of incoming SYN/ACKs — will be

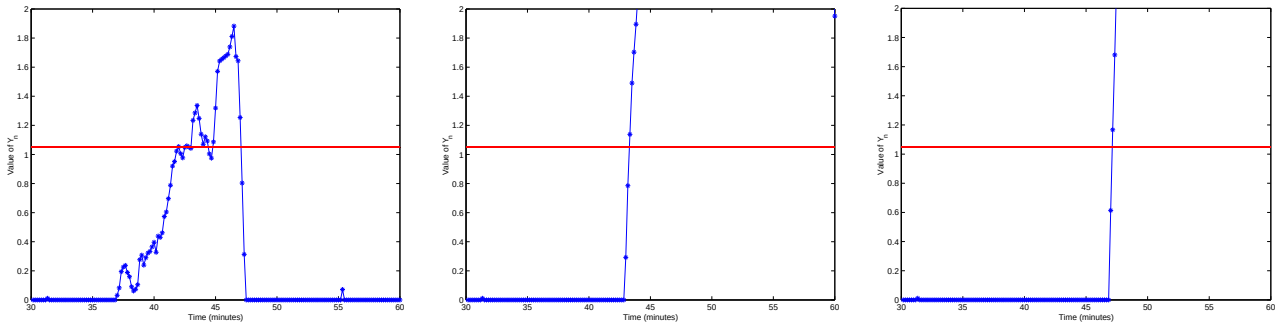
Table 2: Detection Performance of the first-mile SYN-dog at UNC

f_i	Detection Prob.	Detection Time
35	0.8	24.43
37	1.0	18.75
40	1.0	12.25
50	1.0	5.95
60	1.0	4.05
70	1.0	2.80
80	1.0	2.05
100	1.0	1.45
120	1.0	1.0

smaller, so we can achieve higher detection sensitivity. This is confirmed by the study of the Auckland traces, which is presented in the next section.

6.2.2 The Auckland Case

In the case of Auckland traces, the dynamic behaviors of y_n are illustrated in Figure 13 when f_i is set to 1.5, 3 and 4 SYNs per second, respectively. In the case of 1.5 SYNs per second, the first-mile SYN-dog can detect the SYN flooding attack in about 27 observation periods. In contrast, at the flooding rate of 3 or 4 SYNs per second, the first-mile SYN-dog takes a much shorter time (3 or 2 observation periods, respectively) to detect the ongoing flooding. The detection performance of the first-mile SYN-dog for the context of Auckland traces is summarized in Table 3. Since $\bar{K}$ of the Auckland trace is much smaller than that of the UNC trace, the lower detection bound is reduced significantly from 35 to 1.5 SYNs per second.



(a) 1.5 SYNs per second (b) 3 SYNs per second (c) 4 SYNs per second

Figure 13: SYN flooding detection sensitivity of the first-mile SYN-dog at Auckland

6.2.3 Discussion

Due to its proximity to the flooding sources, once the first-mile SYN-dog detects the ongoing flooding traffic, it can take further steps to pinpoint the flooding sources inside the stub network, for example, by filtering the MAC addresses of IP packets whose source addresses are spoofed.

From the detectable flooding rate, we can determine the efficacy of our algorithm in detecting distributed SYN flooding attacks. To attack a protected server, the aggregate flooding rate V should

Table 3: Detection Performance of the first-mile SYN-dog at Auckland

f_i	Detection Prob.	Detection Time
1.5	0.65	27.1
1.75	1.0	13.8
2	1.0	8.0
2.5	1.0	4.1
3	1.0	2.5
4	1.0	1.5
5	1.0	1.0

be larger than 14,000 requests per second [9]. In the UNC case, the lower detection bound is 35, and A_s can be as large as 400 stub networks like the UNC case. Considering the fact that the UNC stub network consists of over 35,000 users [38], it clearly demonstrates the utility and power of SYN-dog. In the Auckland case, the lower detection bound is 1.5, and hence, A_s has to be as large as 9,333 medium size stub networks like the Auckland case. Source address spoofing requires that the attack software open a *raw* network socket, so the attacker must have root access on end hosts. Although the attacker can simultaneously initiate the flooding attacks from (possibly many) machines in several ISPs, it is much harder to launch the attacks from a similar large number of stub networks due to access limit.

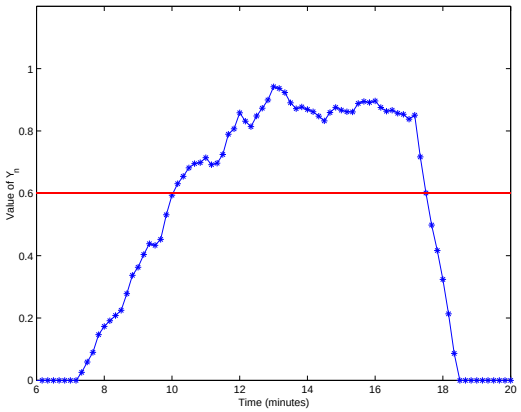


Figure 14: Improvement of flooding detection sensitivity

For the time being, we set the parameters to be independent of network size and access pattern. In practice, the network administrator of the involved leaf router can incorporate site-specific information so that the algorithm can achieve higher detection performance. For instance, in the UNC case, we can reduce a, the upper bound in case of normal operation, from 0.35 to 0.2 and N, the flooding threshold, from 1.05 to 0.6 without incurring additional false alarms. Then, the lower detection bound f_{min} decreases from 35 to 15 SYNs per second, and the detection sensitivity is greatly improved. The dynamics of y_n for the case $f_i = 15$ is shown in Figure 14.

In summary, SYN-dog not only achieves fast detection and high detection accuracy, but also can be easily implementable and broadly applicable.

7 Related Work

All the effective solutions of countering SYN flooding attacks can be roughly classified into four categories: firewall-based, server-based, agent-based, and router-based. As firewalls have been installed at almost all sites, several SYN flooding protection systems are available at these firewalls, such as SynDefender [25] and Syn proxying [1]. The firewall before the protected server plays a key role in protection mechanisms, which act on behalf of the server before a connection is actually established. It intercepts the TCP traffic between clients and the server, and maintains state for each TCP connection. The drawback of these approaches is the longer delay incurred to each TCP connection setup.

Syn cache [24] and Syn cookies [4] belong to the server-based mechanism. Syn cache also maintains states for each SYN request, but its state structure is much smaller. Syn cache employs a global hash table to keep the incomplete queued connections, instead of the per-socket linear list. The listen queue is split among hash buckets. In the Syn cookies mechanism, when the server receives a SYN packet, it responds with a SYN/ACK packet and the ACK sequence number calculated from the source address, source port, source sequence, destination address, destination port and a secret seed. Then, the server releases all states. If an ACK comes from the client, the server can recalculate the ACK sequence number to determine if it is a response to the previous SYN/ACK. If it is, the server can directly enter the "established" state and open the connection. So, the server removes the need to watch for half-open connections. Syn cookies have been implemented as a standard part of Linux kernel.

A software agent [37], called *synkill*, has been developed for mitigating the impact of SYN flooding attacks. Working in a LAN environment, *synkill* continuously monitors TCP's three-way handshake messages. If a SYN packet is not acknowledged in a certain amount of time, *synkill* will inject a matching RST packet to free the resources occupied by the illegitimate half-open connection. Moreover, based on network observation and administrator-provided input, *synkill* classifies IP source addresses, with a high probability, to have been spoofed or not.

To counter DDoS attacks including SYN flooding attacks, many router-based defense system have been proposed [13, 16, 22, 29, 32, 26, 46]. Of these, D-WARD [29], Ingress filtering [13], and MULTOPS [16] are close to SYN-dog. D-WARD [29] is deployed at source-end networks, and monitors two-way traffic between the the network and the rest of the Internet. Based on the comparison with normal traffic models, D-WARD detects and throttles the ongoing flooding attacks. Its normal traffic models are simply based on flow rates. However, due to the diversity of user behaviors and the emergence of new network applications, it is difficult to obtain a robust and general model for describing the normal traffic flow rates.

Ingress filtering [13], in which the internal router interface is configured to block packets that have source addresses from outside the internal network. This limits the attacker's ability to launch a SYN flooding from that network, since the attacker would only be able to generate packets with internal addresses. A data structure called MULTOPS [16] is a tree of nodes that keeps packet-rate statistics for subnets at different aggregation levels. Based on the observation of a significant disproportional difference between the traffic flowing in and out of the victim, routers use MULTOPS to detect ongoing bandwidth attacks. Note that all these previous efforts were used solely for countering DDoS attacks, and do not benefit end-hosts inside stub networks, thus giving little incentive for wide deployment.

8 Conclusion

We developed and evaluated a simple and robust mechanism to sniff SYN (including reflected SYN/ACK) flooding attacks, which is installed at leaf routers. SYN-dog utilizes the SYN–FIN or SYN–SYN/ACK

pair's behavior that is invariant under various arrival models and independent of sites and time-of-day. The distinct features of SYN-dog include:

- it is stateless and requires low computation overhead, making itself immune to flooding attacks;
- the non-parametric CUSUM method is employed, making the detection robust;
- it is insensitive to site and access pattern; and
- it does not degrade the end-to-end TCP performance.

The efficacy of SYN-dog is evaluated and validated by trace-driven simulations. The evaluation results show that the SYN-dog achieves high detection accuracy and short detection time. Moreover, once the first-mile SYN-dog detects the ongoing flooding traffic, information about the location of flooding sources is also revealed, thus saving most of IP traceback efforts that might otherwise be needed.

Acknowledgment

We would like to thank Dong Lin for Harvard traces, Kevin Jeffay for UNC traces and Klaus Mochalski for Auckland traces. Also, comments from Cheng Jin at CalTech are gratefully acknowledged.

References

[1] Netscreen 100 firewall appliance. http://www.netscreen.com/.

[2] M. Basseville and I. V. Nikiforov. *Detection of Abrupt Changes : Theory and Application.* Prentice Hall, 1993.

[3] S. M. Bellovin. Icmp traceback messages. In *Internet Draft: draft-bellovin-itrace-00.txt (work in progress)*, March 2000.

[4] D. J. Bernstein and Eric Schenk. Linux kernel syn cookies firewall project. http://www.bronzesoft.org/projects/scfw.

[5] S. Bhattacharyya, C. Diot, J. Jetcheva, and N. Taft. Pop-level and access-link-level traffic dynamic in a tier-1 pop. In *Proceedings of ACM Internet Measurement Workshop'2001*, San Francisco, CA, November 2001.

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

[7] R. Caceres, P. B. Danzig, S. Jamin, and D. J. Mitzel. Characteristics of wide-area tcp/ip conversations. In *Proceedings of ACM SIGCOMM '91*, Zurich, Switzerland, September 1991.

[8] W. S. Cleveland, D. Lin, and D. Sun. Ip packet generation: statistical models for tcp start times based on connection-rate superposition. In *Proceedings of ACM SIGMETRICS '2000*, Santa Clara, CA, June 2000.

[9] T. Darmohray and R. Oliver. Hot spares for dos attacks. *;login*, 25(7), July 2000.

[10] S. Dietrich, N. Long, and D. Dittrich. Analyzing distributed denial of service tools: The shaft case. In *Proceedings of USENIX LISA'2000*, New Orleans, LA, December 2000.

[11] D. Dittrich. Distributed denial of service (ddos) attacks/tools page. http://staff.washington.edu/dittrich/misc/ddos/.

[12] A. Feldmann. Characteristics of tcp connection arrivals. In *ATT Technical Report*, December 1998.

[13] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing. In *RFC 2267*, January 1998.

[14] L. Garber. Denial-of-service attack rip the internet. *Computer*, April 2000.

[15] S. Gibson. Distributed reflection denial of service. In *Technical Report, Gibson Research Corporation*, February 2002. http://grc.com/dos/drdos.htm.

[16] T. M. Gil and M. Poletter. Multops: a data-structure for bandwidth attack detection. In *Proceedings of USENIX Security Symposium'2001*, Washington D.C, August 2001.

[17] S. D. Gribble and E. A. Brewer. System design issues for internet middleware services: Deductions from a large client trace. In *Proceedings of USENIX Symposium on Internet Technologies and Systems'97*, December 1997.

[18] P. Gupta and N. McKeown. Packet classification on multiple fields. In *Proceedings of ACM SIGCOMM '99*, Cambridge, MA, September 1999.

[19] M. Handley, V. Paxson, and C. Kreibich. Network intrusion detection: evasion, traffic normalization, and end-to-end protocol semantics. In *Proceedings of USENIX Security Symposium'2001*, Washington D.C, August 2001.

[20] U. Hengartner, S. Moon, R. Mortier, and C. Diot. Detection and analysis of routing loops in packet traces. In *Proceedings of ACM Internet Measurement Workshop'2002*, Marseille, France, November 2002.

[21] G. Iannaccone, C.-N. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot. Analysis of link failures in an ip backbone. In *Proceedings of ACM Internet Measurement Workshop'2002*, Marseille, France, November 2002.

[22] J. Ioannidis and S. M. Bellovin. Implementing pushback: Router-based defense against ddos attacks. In *Proceedings of NDSS'2002*, San Diego, CA, February 2002.

[23] T.V. Lakshman and D. Stiliadis. High speed policy-based packet forwarding using efficient multi-dimensional range matching. In *Proceedings of ACM SIGCOMM '98*, Vancouver, Canada, September 1998.

[24] J. Lemon. Resisting syn flooding dos attacks with a syn cache. In *Proceedings of USENIX BSDCon'2002*, San Francisco, CA, February 2002.

[25] Check Point Software Technologies Ltd. Syndefender. http://www.checkpoint.com/products/firewall-1.

[26] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *ACM Computer Communication Review*, 32(3), July 2002.

[27] G. R. Malan and et al. Observations and experiences tracking denial-of-service attacks across a large regional isp. In *Technical Report, Arbor Networks*, 2001.

[28] G. R. Malan, D. Waston, F. Jahanian, and P. Howell. Transport and application protocol scrubbing. In *Proceedings of IEEE INFOCOM '2001*, Anchorage, Alaska, March 2001.

[29] J. Mirkovic, G. Prier, and P. Reiher. Attacking ddos at the source. In *Proceedings of IEEE ICNP'2002*, Paris, France, November 2002.

[30] D. Moore, G. Voelker, and S. Savage. Inferring internet denial of service activity. In *Proceedings of USENIX Security Symposium'2001*, Washington D.C., August 2001.

[31] K. Park and H. Lee. On the effectiveness of probabilistic packet marking for ip traceback under denial of service attack. In *Proceedings of IEEE INFOCOM '2001*, Anchorage, Alaska, March 2001.

[32] K. Park and H. Lee. On the effectiveness of route-based packet filtering for distributed dos attack prevention in power-law internets. In *Proceedings of ACM SIGCOMM '2001*, San Diego, CA, August 2001.

[33] V. Paxson. An analysis of using reflectors for distributed denial-of-service attacks. *ACM Computer Communication Review*, 31(3), July 2001.

[34] V. Paxson and S. Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3), June 1995.

[35] J. Postel. Transmission control protocol. In *Request for Comments 793*, DDN Network Information Center, SRI International, September 1981.

[36] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for ip traceback. In *Proceedings of ACM SIGCOMM '2000*, Stockholm, Sweden, August 2000.

[37] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni. Analysis of a denial of service attack on tcp. In *Proceedings of IEEE Symposium on Security and Privacy*, May 1997.

[38] F. D. Smith, F. H. Campos, K. Jeffay, and D. Ott. What tcp/ip protocol header can tell us about the web. In *Proceedings of ACM SIGMETRICS '2001*, Cambridge, MA, June 2001.

[39] A. C. Snoren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-based ip traceback. In *Proceedings of ACM SIGCOMM '2001*, San Diego, CA, August 2001.

[40] D. Song and A. Perrig. Advanced and authenticated marking schemes for ip traceback. In *Proceedings of IEEE INFOCOM '2001*, Anchorage, Alaska, March 2001.

[41] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. In *Proceedings of ACM SIGCOMM '98*, Vancouver, Canada, September 1998.

[42] W. R. Stevens. *TCP/IP Illustrated, Volume 1.* Addison-Wesley Publishing Company, 1994.

[43] K. Thompson, G. J. Miller, and R. Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network*, 11(6), November/December 1997.

[44] H. Wang and K. G. Shin. Layer-4 service differentiation and resource isolation. In *Proceedings of IEEE RTAS'2002*, San Jose, CA, September 2002.

[45] S. F. Wu, L. Zhang, D. Massey, and A. Mankin. Intention-driven icmp traceback. In *Internet Draft: draft-wu-itrace-intention-00.txt (work in progess)*, February 2001.

[46] D. Yau, J. Lui, and F. Liang. Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles. In *Proceedings of IWQoS'2002*, Miami Beach, FL, May 2002.