Detecting SYN Flooding Attacks

Haining Wang Danlu Zhang Kang G. Shin

EECS Department, The University of Michigan
Ann Arbor, MI 48109-2122
{hxw, danlu, kgshin}@eecs.umich.edu

Abstract— **We propose a simple and robust mechanism for detecting SYN flooding attacks. Instead of monitoring the ongoing traffic at the front end (like firewall or proxy) or a victim server itself, we detect the SYN flooding attacks at leaf routers that connect end hosts to the Internet. The simplicity of our detection mechanism lies in its statelessness and low computation overhead, which make the detection mechanism itself immune to flooding attacks. Our detection mechanism is based on the protocol behavior of TCP SYN–FIN (RST) pairs, and is an instance of the Sequential Change Point Detection [1]. To make the detection mechanism insensitive to site and access pattern, a non-parametric Cumulative Sum (CUSUM) method [4] is applied, thus making the detection mechanism much more generally applicable and its deployment much easier. The efficacy of this detection mechanism is validated by trace-driven simulations. The evaluation results show that the detection mechanism has short detection latency and high detection accuracy. Moreover, due to its proximity to the flooding sources, our mechanism not only sets alarms upon detection of ongoing SYN flooding attacks, but also reveals the location of the flooding sources without resorting to expensive IP traceback.**

I. Introduction

The recent attacks on popular web sites like Yahoo, eBay and E*Trade, and their consequent disruption of services have exposed the vulnerability of the Internet to Distributed Denial of Service (DDoS) attacks [12]. It has been shown that more than 90% of the DoS attacks use TCP [19]. The TCP SYN flooding is the most commonly-used attack. It 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 servers or Mail servers, are susceptible to the TCP SYN flooding attacks.

The 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. The stateless and destination-based nature of Internet routing infrastructure cannot differentiate a legitimate SYN from a spoofed one, and TCP does not offer strong authentication on SYN packets. Therefore, under SYN flooding attacks, the victim server cannot single out, and respond only to, legitimate connection requests while ignoring the spoofed.

To counter SYN flooding attacks, several defense mechanisms have been proposed, such as Syn cache [17], Syn cookies [3], SynDefender [6], Syn proxying [20], and Synkill [26]. All of these defense mechanisms are installed at the firewall of the victim server or inside the victim server, thereby providing no hints about the sources of the SYN flooding. They have to rely on the expensive IP traceback [2], [21], [25], [28], [29], [34] to locate the flooding sources. Because the defense line is at, or close to, the victim, the network resources are also wasted by transmitting the flooding packets.

Moreover, these defense mechanisms are stateful, i.e., states are maintained for each TCP connection or state computation is required. Such a solution makes the defense mechanism itself vulnerable to SYN flooding attacks. Recent experiments have shown that a specialized firewall, which is designed to resist SYN floods, became futile under a flood of 14,000 packets per second [8]. The stateful defense mechanisms also degrade the end-to-end TCP performance, e.g., incurring longer delays in setting up connections. In the absence of SYN flooding attacks, all the overheads introduced by the defense mechanism become superfluous. We, therefore, need a simple stateless mechanism to detect SYN flooding attacks, which is immune to the SYN flooding attacks. Also, it is preferred to detect an attack early near its source, so that one can easily trace the flooding source without resorting to expensive IP traceback.

In this paper, we propose a simple and robust mechanism to detect SYN flooding attacks, which is complementary to the defense systems mentioned above. The simplicity of this flooding detection system (FDS) lies in its statelessness[1] and low computation overhead. The FDS is, in some sense, a by-product of the router infrastructure that differentiates TCP control packets from data packets [33]. Instead of monitoring the ongoing traffic at the front end (like firewall or proxy) or the victim server

Haining Wang and Kang G. Shin were supported in part by Samsung Electronics, Inc. and by the Office of Naval Research under Grant No. N00014-99-1-0465.

[1] In a stricter sense, it is per-connection stateless, i.e., no per-connection state is kept.

itself, we detect SYN flooding attacks at leaf routers that connect end hosts to the Internet. The FDS can be deployed at the first-mile or last-mile leaf routers. The benefit of deploying the FDS at the first-mile leaf routers is their proximity to the flooding sources. If a SYN flooding attack is detected at the first-mile leaf router, information about the location of flooding sources is also captured. The flooding sources must be inside the subnet to which the leaf router is connected, hence saving most of the work required by IP traceback. We will discuss the placement of FDS in Section II-B.

The key feature of FDS is to utilize the inherent TCP SYN–FIN pairs' behavior for SYN flooding detection. The SYN/FIN packets delimit the beginning (SYN) and end (FIN) of each TCP connection. As shown in Figure 1 that is borrowed from [31], under the normal condition, one appearance of a SYN packet results in the eventual return of a FIN packet. Although we can distinguish SYNs from SYN/ACK packets, we have no means to discriminate active FINs from passive FINs since each end host behind a leaf router may be either a client or a server. Therefore, the SYN–FIN pairs refer to the pairs of (SYN, FIN) and (SYN/ACK, FIN). In this paper, the "SYN" packets are generalized to include the pure SYN and SYN/ACK packets. While the RST packet violates the SYN–FIN pair, for any RST that is generated to abort a TCP connection[2], we can still get a SYN-RST pair. The impact of RST upon SYN flooding detection is discussed further in Section II-C.

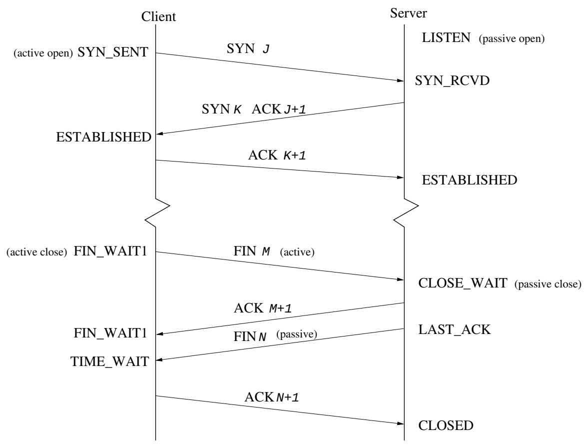


Fig. 1. TCP states corresponding to normal connection establishment and teardown

We rely on packet classification to differentiate the TCP SYN, FIN and RST packets at leaf routers. This packet classification was originally motivated by the desire of providing service differentiation to IP flows. Large-scale packet classification mechanisms [14], [16], [30] have been proposed, making it possible to distinguish TCP control packets at routers at a very high speed. At leaf routers, no state or state computation is involved in our FDS. Only three new variables are introduced to measure the number of received SYN, FIN and RST packets at the inbound and outbound interface, respectively. We refer to the traffic flowing from the Internet to the Intranet as *inbound*, and the traffic in the other direction as *outbound*. Based on this SYN–FIN (RST) pairs' behavior, the dynamics of the difference between the number of SYN and FIN (RST) packets can be modeled as a stationary, ergodic random process, and our FDS is an instance of the Sequential Change Point Detection [1]. To make the FDS independent of sites and access patterns, the difference between the number of SYNs and FINs (RSTs) is normalized by an estimated average number of FINs (RSTs). The non-parametric Cumulative Sum (CUSUM) method [4] is applied, making the FDS much more generally applicable and its deployment much easier.

The efficacy of our detection mechanism is validated by trace-driven simulations. The evaluation results show that our FDS has short detection time and high detection accuracy. Moreover, due to its close proximity to the flooding sources, our detection mechanism not only alarms on the ongoing SYN flooding attacks but also reveals the location of the flooding sources.

The remainder of the paper is organized as follows. Section 2 discusses the issues related to our detection system. Section 3 describes the proposed detection algorithm based on the TCP SYN–FIN (RST) pair's behavior. Section 4 validates and evaluates the performance of the FDS using trace-driven simulations. Section 5 discusses the related work. Finally, conclusions are drawn in Section 6.

II. Issues Related to Flooding Detection

Before describing the proposed flooding detection mechanism, we discuss the details of three closely-related issues: packet classification, placement of the FDS, and discrepancy between the number of SYNs and FINs.

A. *Packet Classification*

To identify TCP SYNs, FINs and RSTs, the TCP header needs to be accessed. This identification is performed at leaf routers, which are usually the trusted entities for the clients in the same intranet. A multi-layer IPSec protocol [35] has been proposed, which allows trusted routers to access the transport-layer information. Therefore, the network-level security of IPSec should not be an obstacle to the identification and counting of TCP SYNs, FINs and RSTs at leaf routers. A detailed description of the packet-classification algorithm is given in Figure 2.

The first two steps in Figure 2 guarantee that the IP packet contains a TCP header. The IP packet that contains the TCP header must have a zero fragmentation offset. Although IP options are included primarily for network testing or debugging, in order to accurately pinpoint the offset of TCP CODE BITs in an IP packet, the 4-bit header length field (measured in number of 32-bit words) in the IP header is read. This field is used to compute the offset of the 6-bit CODE BITs field of the TCP header in this IP packet as follows:

$$IP^{offset} = Hdr\_length^{IP} + TCP^{offset}.$$

It indicates that the offset of CODE BITs in the IP packet equals the sum of the length of IP header and the offset of CODE BITs in its TCP header. The 6-bit CODE BITs field of the TCP header is then read to determine the type of the TCP segment.

[2]Those RSTs are mostly issued by clients. In its own best interest, a server rarely sends the RST packets to its clients once their TCP connection have been established.

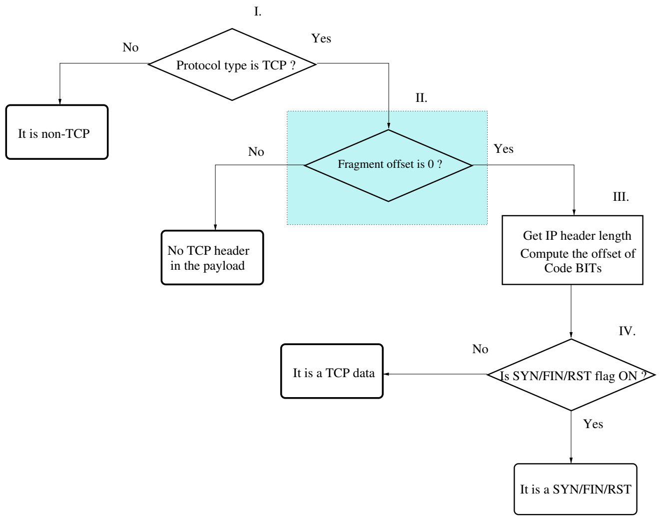


Fig. 2. The flowchart of the packet classification at leaf routers

B. *Placement of Detection Mechanism*

As mentioned before, the FDS is installed at either the first-mile or the last-mile leaf router, or both. However, each leaf router can be both the first-mile and last-mile router, depending on the direction of traffic flows between the intranet and the Internet. For the packets going out of the intranet, the leaf router is their first-mile router. On the other hand, for the incoming packets into the intranet, the leaf router is their last-mile router. Thus, we deploy the FDS at both the inbound and outbound interfaces. The one installed at the outbound interface is the *first-mile FDS*, while the one installed at the inbound interface is the *last-mile FDS*. Figure 3 illustrates the installation of FDS at a leaf router. The two FDSs can coordinate with each other via shared memory, or IPC inside the router.

The first-mile FDS of the leaf router plays the primary role in detecting a flooding attack, due mainly to its proximity to the sources of the flooding attack. However, the detection sensitivity may decline with the increase of the size of the attack group. In a large-scale DDoS attack, the flooding sources can be orchestrated so that individual flooding traffic can cause only an insignificant deviation from the normal traffic pattern.

In contrast, the last-mile FDS can quickly detect the flooding attacks as all of the flooding traffic is aggregated at the last-mile router. Although it cannot provide any hint about the flooding sources, the defense system like SynDefender can be triggered to protect the victim, making the flooding attack harder to succeed. To bring down the victim under protection, the flooding sources have to significantly increase their flooding rates, but this increased flooding traffic makes it easier to detect the flooding attack and its sources at the first-mile routers.

However, the FDS is not recommended 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.

As has been done with most of intrusion detection (ID) systems, the FDS can be placed on the link that connects the intranet to the Internet by monitoring the bidirectional traffic on that link. However, besides the extra specialized equipment and manpower involved, 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 multi-CPU architecture in order to perform packet state analysis. As the link speed continues to grow, it will be more difficult for network flow monitors (that run on a typical PC) to pace with the network's packet rate.

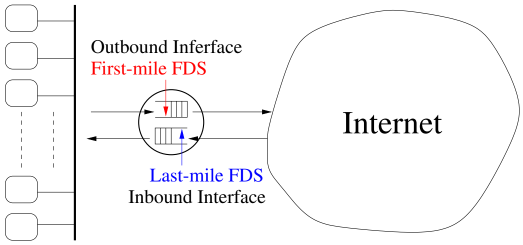


Fig. 3. The installation of FDS at a leaf router

Our last concern is with the wide deployment of the FDS at leaf routers. As the FDS provides differentiation between TCP control segments and data segments, fine-grained service differentiation and isolation can be made on TCP flows. The end-to-end TCP performance is significantly improved as shown in [33], instead of being undermined. Installation of the FDS benefits not only victim servers but also the clients inside the intranet. It greatly provokes the interest of wide deployment of the FDS. Furthermore, the FDS is incrementally deployable and its implementation overhead is low.

C. *Discrepancy between SYNs and FINs*

Under a long-running normal condition, the TCP semantics requires a one-to-one match between SYNs and FINs. However, in reality there is always 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 behavior. RSTs are generated for two reasons: (1) one is passive, i.e., the RST is transmitted in response to the arrival of a packet that is destined to a closed port; (2) the other is active, i.e., the RST is initiated to abort a TCP connection. Each active RST is associated with the SYN from the same session, since both of them can be seen by the same FDS. However, a passive RST can not be associated with any SYN seen at the same FDS due to the fact that the passive RST and its corresponding SYN must go through different FDSs. Furthermore, passive RSTs may even have nothing to do with SYNs. For instance, a late arrival of a data packet to the port that has been closed will lead to the transmission of a RST. We treat the passive RTSs as background noise.

In summary, three types of SYN pairs are considered as the normal behavior of TCP: (SYN, FIN), (SYN/ACK, FIN) and (SYN, RST_{active}). Unfortunately, the FDS can not distinguish the active RSTs from the passive ones. There are two simple but extreme ways to resolve this thorny problem: one is to treat all RSTs as active and the other is to treat all RSTs as passive. The first approach reduces the FDS detection sensitivity, while the second raises the FDS 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 the most of the background noise. Based on our observation, under the normal condition: (1) the 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 the passive RSTs are only a small percentage of the whole RSTs. So, we set the threshold to 75%, i.e., 3 out of 4 RSTs are treated as active. Moreover, for the following reason, the FDS can withstand the negative impact of the passive RSTs that are incorrectly classified as active RSTs: in CUSUM algorithm, the reset-to-zero for any negative difference[3] between the number of SYNs and that of FINs (RSTs) eliminates the cumulative effects, and thus the spike of background noise only degrades the detection sensitivity during one observation period.

III. Statistical Attack Detection

Basically, the FDS belongs to the commonly-known network-based intrusion detection system: an intruder is detected 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 in representing the user's normal behavior, and detect any significant deviation from the normal behavior. However, unlike the traditional network intrusion detection system that passively monitors bidirectional traffic streams on network links, the FDS is installed at a leaf router and can be viewed as a component integrated into the leaf router.

The burstiness of TCP connection request arrivals [10] makes the detection of attack signatures much harder, since the critical characteristic of self-similar traffic is that there is no natural length of a "burst". It is also site- and time-dependent. However, the strong positive correlation between SYN and FIN (RST) offers a clear indication for SYN flooding. According to the specification of TCP/IP protocol [24], [31], in normal operation, a FIN (RST) is paired with a SYN at the end of data transmission; but under SYN flooding attacks, this SYN–FIN (RST) pair's behavior will be violated, deviating from the normal operation.

A. Data Sampling and Detection Mechanism

We collect the number of SYN and FIN (RST) packets during every observation period t_0, which determines the detection resolution, at leaf routers. In order to relate the SYN and FIN (RST) packets of the same connection, the sampling time of FIN (RST) 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 (RST) sampling period. Recent Internet traffic measurements have shown that most of TCP connections last 12–19 seconds [32], so we set t_d to 10 seconds. To balance the detection resolution and the algorithm's stability and accuracy, we set t_0 to 20 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) is very small, as compared to the total number of TCP connection requests. This observation is true 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 number of SYNs and FINs (RSTs) is not sensitive to the request arrival process. The results presented in Section IV-A clearly show that the consistent synchronization between SYNs and FINs (RSTs) is independent of the sample time, 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 [8] 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 floods, a server can be disabled by a flood of 14,000 packets per second [8]. To shut down the victim server for 10 minutes, for example, the group of attackers need to inject at least a total of 300,000 SYN packets. During the same time period, however, the number of FINs (RSTs) remains largely unchanged. Therefore, there will be much more SYNs than FINs (RSTs) collected during the flooding period. The difference between the number of SYNs and FINs (RSTs) will dramatically increase, and remain large during the whole flooding period, which typically lasts for several minutes. So, the occurrence of a large difference between the number of SYNs and FINs (RSTs) in the order of minutes or tens of seconds indicates a SYN flooding attack. This will be used in our attack detection.

There are other events that may cause the increase of the difference between the number of SYN and FIN (RST) packets as follows.

- There has been a steady increase of on-line users and, at the same time, most of them issue long-lived TCP sessions. Thus, the number of established long-lived TCP connections is constantly increasing.
- Some well-known servers or the links connected to them are down. The SYN requests are retransmitted three times automatically before the request times out.

All of these cases are considered as exceptional situations, and rarely occur as the difference between the number of SYNs and FINs (RSTs) becomes very large.

B. The CUSUM Algorithm

Let $\{\Delta_n, n = 0, 1, \cdots\}$ be the number of SYNs minus that of the corresponding FINs (RSTs) collected within one sampling period. In general, the mean of $\{\Delta_n\}$ is dependent on the size of the subset. It may also vary with time of the day or week, depending on the access pattern. To make our algorithm more general, we should alleviate these dependencies. Thus, $\{\Delta_n\}$ is normalized by the average number $\bar{F}$ of FINs (RSTs) during the sampling period t_0. $\bar{F}$ can be estimated in real time and updated periodically. An example of recursive estimation and update of $\bar{F}$ is:

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

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

Define $X_n = \Delta_n/\bar{F}$. 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

[3] See Eq. 3 in Section III-B.

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 [1]. The objective of Change Point Detection is to determine if the observed time series is statistically homogeneous, and if not, to find the point in time when the change happens. It has been studied extensively by statisticians. See [1] and [4] 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 whole data segment 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 run.

We adopt a sequential test for a quicker response when an attack occurs. It also saves memory and computation. One difficulty, however, is the modeling of $\{X_n\}$. Recently, there has been considerable work on the modeling of the arrival process of TCP connection requests. It is reported in [10] that the statistics of TCP connection request arrivals have shown significant changes in the past few years, along with the Internet traffic itself: in early 90's, the dominant TCP connections are FTP and Telnet sessions, and the arrival process is Poisson [5]. However, after the Web became the predominant source of TCP connections, the arrival process displays heavy-tails in its inter-arrival times [23]. The newly-emerging Persistent-HTTP also has an impact on the TCP arrival pattern [27]. Furthermore, recent Internet traffic analyses have shown that the arrival process is not even stationary and dependent on the average arrival rate [7] (bursty, or long-range dependent at low rate, but approximately Poisson at high rate) and time scale [13] (bursty on a small time scale, but Poisson on a large time scale). For such a dynamic and complicated entity like the Internet, it may not be possible to model the total number of TCP connections at all times by a simple parametric model. Therefore, we seek robust tests which are not model-specific. Non-parametric methods fit this requirement very well. In particular, we apply the non-parametric CUSUM (Cumulative Sum) method [4] to our attack detection. This method enjoys all the virtues of sequential and non-parametric test, 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 [1], [4].

$\{X_n\}$ 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 $s \to \infty$:

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

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 [4]. In practice, they are very mild and easily satisfiable, even by 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 large positive. 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

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

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

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

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:

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

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

In this algorithm, there are two design parameters involved: a, the upper bound in case of normal operation, and N, the flooding threshold. Let $P_m(E_m)$ be the probability measure (expectations) of $\{\tilde{X}_n\}$ with the attack occurring at time m and $P_\infty(E_\infty)$ be the counterparts of $\{\tilde{X}_n\}$ without any attack. There are two fundamental performance measures for the sequential change point detection.

1) False alarm time (the time without false alarm): the time duration with no false alarm reported when there is no attack.
2) Detection time: the detection delay after the attack starts.

One would want the second measure to be as short as possible while keeping the first measure as long 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 [1].

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 [4] that, with the choice of a and N, as $N \to \infty$, we have

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

which is equivalent to

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

In other words, the time between consecutive false alarms grows exponentially with N. c_1 and c_2 are constants, depending on the marginal distribution and mixing coefficients of $\{\tilde{X}_n\}$. The burstiness of the 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's 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

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

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.

C. Parameter Specification

To implement the CUSUM algorithm, we first need to specify the two tunable parameters: a, the upper bound in case of normal operation, and h, the lower bound of the increase in case of an attack. 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.

In monitoring the incoming traffic (the 'last mile'), all the flooding SYN packets converge, and therefore, a large difference in the number 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 monitoring the outgoing traffic (the 'first mile'), since the attack may be initiated from many sites simultaneously, only part of the flooding SYN packets can be seen by each detector. To balance the detection sensitivity and false alarm time, we set $a = 0.2$ and $h = 0.4$. 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 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, hence $\gamma = 1$ and $N = 1$; we choose $3t_0$ as the counterpart for the first mile, hence $\gamma = 5$ and $N = 0.6$.[4]

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 limit of detection in terms of SYN flooding rate is a if $c \approx 0$. 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.

TABLE I

A SUMMARY OF THE TRACE FEATURES

Trace	Starting time	Traffic type
DEC-1	2:00, Thu Mar 9, 95	Bi-directional
DEC-2	10:00, Thu Mar 9, 95	Bi-directional
Harvard-1	12:39, Thu Mar 13, 97	Bi-directional
Harvard-2	16:39, Thu Mar 13, 97	Bi-directional
UNC-in	19:30, Wed Sept 27, 00	Uni-directional
UNC-out	19:30, Wed Sept 27, 00	Uni-directional

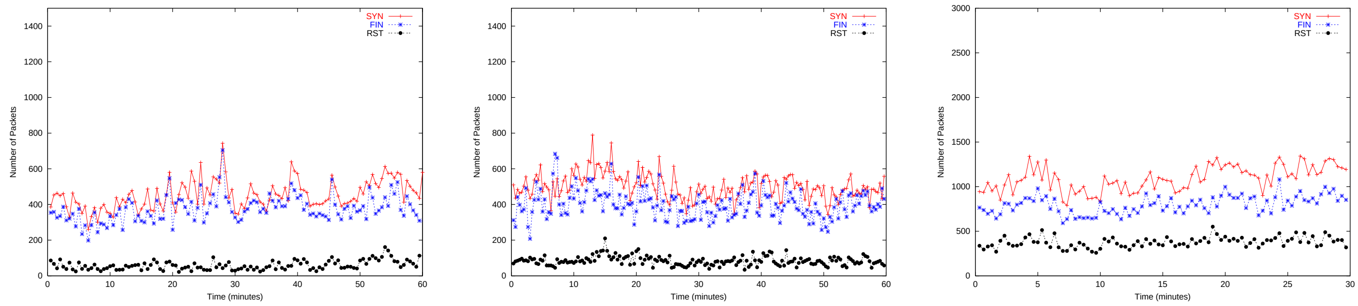
IV. Performance Evaluation

To evaluate and validate the FDS, we have conducted trace-driven simulation experiments. The trace data used in our study are collected at different times from three different sources. The first set was gathered at Digital's primary Internet access point, which is an Ethernet DMZ network. Each trace in this set contains an hour's worth of all wide-area traffic between DEC (now Campaq) Western Research Lab and the Internet. The second set was taken on March 13, 1997 on a 10 megabit Ethernet connecting Harvard's main campus to the Internet, which includes two half-hour traces. 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 27, 2000. A summary of the traces used in our experiments is given in Table I. Note the UNC traces are uni-directional: UNC_in collects the traffic data from the Internet to UNC campus network and UNC_out collects the traffic data from UNC campus network to the Internet.

A. Normal Traffic Behavior

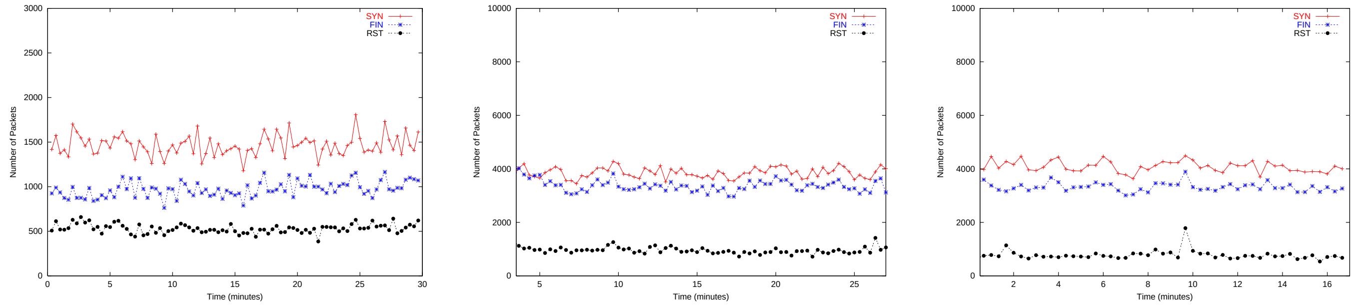
The three sets of traces represent the normal traffic behaviors at the exchange points between different intranets and the

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



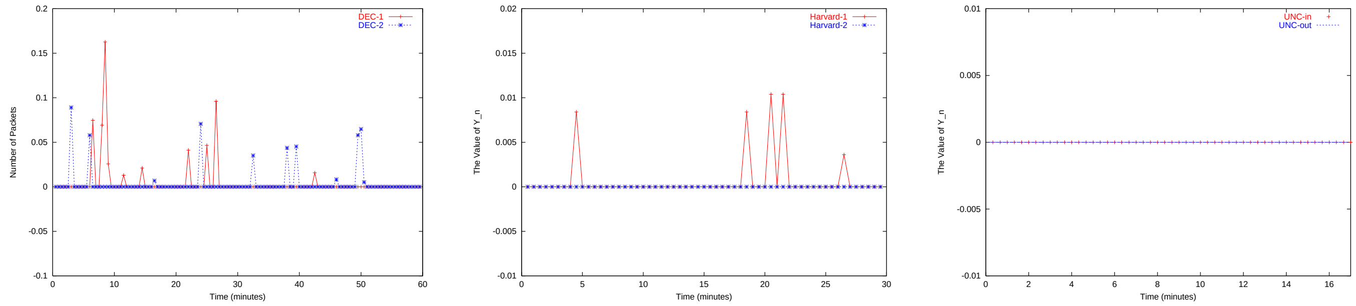
(a) DEC-1 (b) DEC-2 (c) Harvard-1

Fig. 4. The dynamics of SYN and FIN (RST) packets (part I)



(a) Harvard-2 (b) UNC-in (c) UNC-out

Fig. 5. The dynamics of SYN and FIN (RST) packets (part II)



(a) DEC (b) Harvard (c) UNC

Fig. 6. CUSUM test statistics under normal operation

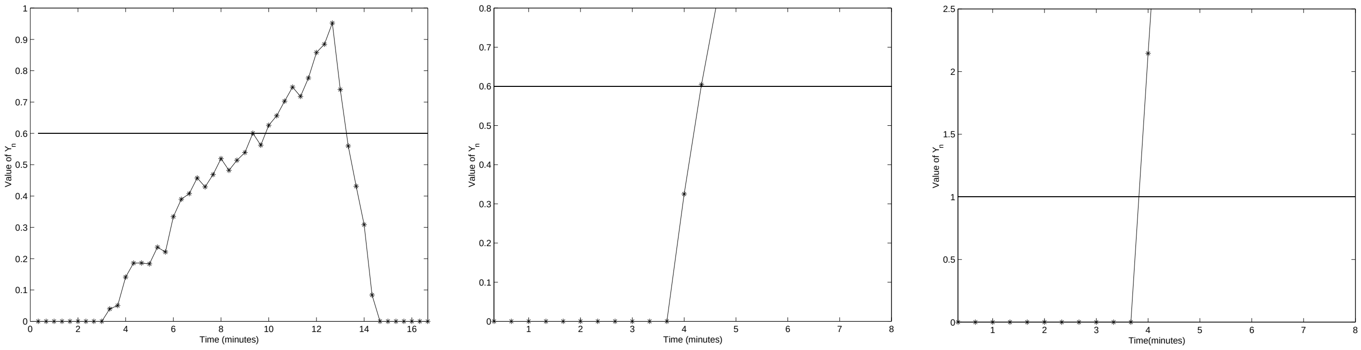
Internet at different times. We parse the traces and extract the TCP SYN, FIN and RST packets as the input to our leaf router simulator. The dynamics of SYN, FIN and RST packets at DEC site are illustrated in Figures 4 (a) and (b). The corresponding result from Harvard traces are illustrated in Figures 4 (c) and 5 (a) and those from UNC_in and UNC_out are in Figure 5 (b) and (c). They clearly show consistent synchronization between SYN and FIN (RST) packets. The consistency indicates that the synchronization is an inherent traffic behavior and independent of time and sites.

We have applied the proposed detection algorithm on all the available traces without adding attacks. The test statistics, $\{y_n\}$, for all traces are plotted in Figure 6. The flooding thresholds are specified in Section III-C, i.e., $N = 0.6$ for the first mile and $N = 1$ for the last mile. For all the traces tested, y_n's are mostly zeros. The isolated bursts in y_n are always much smaller than the threshold. So, no false alarms are reported.

B. *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, Trinity, Plague and Shaft, generate TCP SYN flooding attacks [9]. Although these DDoS attack tools employ different ways to coordinate the attacks with the goal of achieving robust and covert DDoS attacks, their flooding behaviors are similar in that the SYN packets are continuously sent to the victim.

The mechanism of DDoS attacks works as follows: the master sends control packets to the previously-compromised slaves,



(a) 35 SYNs per second

(b) 80 SYNs per second

(c) 500 SYNs per second

Fig. 7. SYN flooding detection sensitivity (a) and (b) at first-mile FDS; (c) at last-mile FDS

instructing them to target at a given victim. The slaves then generate and send high-volume streams of flooding messages to the victim, but with fake or randomized source addresses, so that the victim cannot locate the attackers.

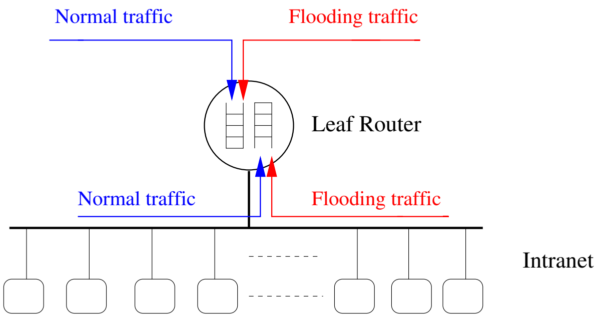


Fig. 8. The trace-simulation flooding attack experiment

In the SYN flooding detection experiments, the UNC 2000 traces are used as the normal background traffic. Among them, UNC_in is used for inbound, i.e., last-mile monitoring, and UNC_out is for outbound, i.e., first-mile monitoring. The flooding traffic is mixed with the normal traffic, and the FDS at the leaf router is simulated, as shown in Figure 8. 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. The detection sensitivity depends only on the total volume of flooding traffic. Therefore, 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 FDSs is quite different. The flooding traffic passing through the last-mile FDS is the aggregation of the flooding traffic from all distributed attackers, allowing a much easier detection. However, the flooding detection at the first-mile FDS is much more difficult. In a large-scale DDoS attack, the flooding sources can be so coordinated that the traffic from each flooding source is not significant. Assume that the minimum SYN flooding traffic to bring down a TCP server is V packets per second. Then, the flooding rate at the last-mile FDS is V, but the flooding rate seen by the first-mile FDS may be quite different.

We assume 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 FDS, 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 FDS. That is, the less the flooding sources inside the stub network, the less flooding traffic seen by the first-mile FDS and the harder to detect the flooding attack. The flooding duration in all experiments is set to 10 minutes, a typical attacking duration observed in the Internet [19]. The starting time of flooding attacks is randomly chosen between 1 and 5 minutes.

We first examine the detection sensitivity at the last-mile FDS. To demonstrate the high sensitivity of last-mile FDS to SYN flooding, the flooding rate V is set to its minimum, 500 SYNs per second. The simulation results are plotted in Figure 7 (c), showing that the cumulative sum y_n exceeds the flooding threshold "1" in a single observation period, i.e., the fastest response can be achieved. So, the last-mile FDS can detect the SYN flooding attack in 20 seconds. Once the flooding attack is detected, the protection system like SynDefender can be triggered to defend the victim from the flooding attack. To paralyze the protection system at the victim, the attackers have to increase their flooding rate, and the first-mile FDS will then be more likely to detect and locate the flooding sources inside the stub network.

To examine the detection sensitivity of the first-mile FDS, we vary the flooding rate f_i seen by the first-mile FDS. Figures 7 (a) and (b) plot the simulation results when f_i is set to 35 and 80 SYNs per second, respectively. The accumulative effects of SYN flooding are clearly shown in these figures. In the case of 80 SYNs per second, the first-mile FDS can detect the SYN flooding in 2 observation periods, i.e., 40 seconds. However, in the case of 35 SYNs per second, the first-mile FDS takes much longer (around 6 minutes) to exceed the flooding threshold of 0.6, since the lower detection bound is about 35 SYNs per second in this simulation scenario. However, due to its proximity to the flooding sources, once the first-mile FDS detects the ongoing flooding traffic, it can further locate the flooding source inside the stub network. In some sense, the detection time becomes a secondary issue, because the revelation of the flooding sources is more important to the victim server.

As the last-mile detection is much easier than the first mile, we only study the detection probability and detection time for the latter. The simulation results for different f_i values are listed in Table II. The unit of detection time is the observation period

TABLE II

DETECTION PERFORMANCE OF THE FIRST-MILE FDS

f_i (SYNs/s)	Detection Prob.	Detection Time
33	70%	24.36
35	100%	17.25
40	100%	9.2
50	100%	4.75
60	100%	3.0
70	100%	2.4
80	100%	1.8
90	100%	1.2
100	100%	1.0

t_0, which is set to 20 seconds. Clearly, larger flooding rates lead to faster and easier detection of attacks. Note that for the UNC_out trace, $\bar{F} \approx 3764$ which is equivalent to 188 FINs (RSTs) per second. Theoretically, the lower detection bound of the flooding rate is $a \times 188 = 37.6$, where $a = 0.2$, while our simulation results give the lower bound around 35.

From the detectable flooding rate, we can specify the efficacy of our algorithm in detecting distributed flooding attacks at the first mile. To attack a protected server, the aggregate flooding rate V should be larger than 14,000. Using our lower detection bound, suppose $f_i = 35$ SYNs per second, then A_s can be as large as 400 stub networks. Although the attacker can simultaneously initiate the flooding attack from numerous machines, it is much harder to launch the attack from the same large number of subnets due to the access limit. Considering the size of the UNC stub network that has over 35,000 users [27], it clearly demonstrates the utility and power of our detection mechanism.

If we implement this detection mechanism at a smaller subnet, $\bar{F}$ will be smaller, so we can achieve more "sensitive" detection. We have conducted the simulation experiments based on the traces that are collected from the stub network of University of Auckland at New Zealand on December 5, 2000. Because its detection lower bound is 5 SYNs per second, A_s has to increase to 2,800 medium-size stub networks. In summary, our FDS not only achieves fast detection and high detection accuracy, but also can be easily implemented and broadly applied.

V. 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 Syn-Defender [6] and Syn proxying [20]. The firewall before the protected server plays a key role in protection mechanisms, which acts on behalf of the server before the connection is actually established. It intercepts the TCP traffic between clients and the server, and maintains state for each TCP connection. The drawbacks of this approach are delays on every packet for additional processing.

Syn cache [17] and Syn cookies [3] belong to the server-based mechanism. Syn cache still maintains states for each SYN request, but the allocated state structure is much smaller. The 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 received a SYN packet, it responds with a SYN/ACK packet with 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. The Syn cookies have been implemented as a standard part of Linux kernel.

A software agent [26], called *synkill*, has been developed for mitigating the impact of SYN flooding attacks. Working in a local area network environment, *synkill* continuously monitors TCP three-way handshake messages. If a SYN packet is not acknowledged after 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 administratively given input, *synkill* classifies IP source addresses, with high probability, to be spoofed or genuine.

A data-structure called MULTOPS [15] 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 into and out from the victim, routers use MULTOPS to detect ongoing bandwidth attacks. Ingress filtering [11], in which the internal router interface is configured to block packets that have source addresses from outside the internal network. This limits the ability to launch a SYN flooding attack from that network, since the attacker would only be able to generate packets with internal addresses.

Given the reachability constraints imposed by routing and network topology, route-based distributed packet filtering (DPF) [22] exploits routing information to determine if a packet arriving at the router is valid with respect to its inscribed source/destination addresses. The experimental results in [22] show that a significant fraction of spoofed packets are filtered out, and the spoofed packets that escaped the filtering can be localized into 5 candidate sites which are easy to trace back.

VI. FUTURE WORK

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 system, it can paralyze the SYN–FIN detection mechanism by flooding a mixture of SYNs and FINs (RSTs). Although we 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.

Fortunately, there is an alternative that is not easy for the attacker to counter. In the normal TCP three-way handshake, an outbound SYN always induces an inbound SYN/ACK within a round-trip time. The key feature of this alternative is to utilize this SYN–SYN/ACK pair for SYN flooding detection, instead of the SYN–FIN pairs. Compared to the SYN-FIN pairs scheme, this requires the coordination between the two FDSs. The first-mile FDS maintains the count of outgoing SYNs and the last-mile FDS keeps track of incoming SYN/ACK packets. At the end of each observation period, the count information must be exchanged between the two FDSs. Since SYN/ACK packets are generated by the other side (i.e., the victim of flooding attacks), it is harder for the flooding sources to evade the detection.

Moreover, as compared to the SYN-FIN pair, the interval between SYN and SYN/ACK is bounded by a RTT, instead of the duration of a TCP session that has a much larger variation. Note that the SYN–SYN/ACK pair detection scheme itself is not immune to counter-measures. If the spoofed source address is in the same stub network as a flooding source, it cannot detect the ongoing flooding attack.

Recently, Multihomed 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, our detection scheme still works. If the packets of the same TCP session go through different leaf routers, we need a loose synchronization mechanism between the FDSs in these leaf routers, which will be addressed in our future work.

VII. Conclusion

This paper presented a simple and robust SYN flooding detection mechanism to be installed at leaf routers. The detection utilizes the SYN–FIN pairs' behavior that is invariant under the various arrival models and independent of sites and time-of-day. The distinguishing features of FDS include: (1) it is stateless and requires low computation overhead, making itself immune to SYN flooding attacks; (2) the non-parametric CUSUM method is employed, making the detection robust; (3) it is insensitive to site and access pattern; and (4) it does not undermine the end-to-end TCP performance. The efficacy of FDS is evaluated and validated by trace-driven simulation. The simulation results show that the FDS achieves high detection accuracy and short detection time. Moreover, once the first-mile FDS detects the ongoing flooding traffic, information about the location of flooding sources is also revealed, thus saving most of IP traceback efforts.

Acknowledgment

We would like to thank Dong Lin for Harvard traces and Kevin Jeffay for UNC traces.

References

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

[2] S. M. Bellovin, "ICMP Traceback Messages", *Internet Draft: draft-bellovin-itrace-00.txt*, March 2000.

[3] D. J. Bernstein and Eric Schenk, "Linux Kernel SYN Cookies Firewall Project", http://www.bronzesoft.org/projects/scfw.

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

[5] R. Caceres, P. B. Danzig, S. Jamin and D. J. Mitzel, "Characteristics of wide-area TCP/IP conversations", *Proceedings of ACM SIGCOMM'91*, September 1991.

[6] Check Point Software Technologies Ltd. *SynDefender*: http://www.checkpoint.com/products/firewall-1.

[7] W. S. Cleveland, D. Lin and D. Sun, "IP packet generation: statistical models for TCP start times based on connection-rate superposition", *Proceedings of ACM SIGMETRICS'2000*, June 2000.

[8] T. Darmohray and R. Oliver, "Hot Spares for DoS attacks", *;login*, 25(7), July 2000.

[9] D. Dittrich, "Distributed Denial of Service (DDoS) Attacks/Tools Page", http://staff.washington.edu/dittrich/misc/ddos/.

[10] A. Feldmann, "Characteristics of TCP Connection Arrivals", ATT Technical Report, December 1998.

[11] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing", *RFC 2267*, January 1998.

[12] L. Garber, "Denial-of-Service Attack Rip the Internet", *Computer*, April 2000.

[13] S. D. Gribble and E. A. Brewer, "System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace", *Proceedings of USENIX Symposium on Internet Technologies and Systems'97*, December 1997.

[14] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields", *Proceedings of ACM SIGCOMM'99*, September 1999.

[15] T. M. Gil and M. Poletter, "MULTOPS: a data-structure for bandwidth attack detection", *Proceedings of USENIX Security Symposium'2001*, August 2001.

[16] T.V. Lakshman and D. Stiliadis, "High Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching", *Proceedings of ACM SIGCOMM'98*, September 1998.

[17] J. Lemon, "Resisting SYN Flooding DoS Attacks with a SYN Cache", *Proceedings of USENIX BSDCon'2002*, February, 2002.

[18] S. McCreary and K. Claffy, "Trends in Wide Area IP Traffic Patterns — A View from Ames Internet Exchange", *Proceedings of ITC'2000*, September 2000.

[19] D. Moore, G. Voelker and S. Savage, "Inferring Internet Denial of Service Activity", *Proceedings of USENIX Security Symposium'2001*, August 2001.

[20] Netscreen 100 Firewall Appliance, http://www.netscreen.com/.

[21] K. Park and H. Lee, "On the Effectiveness of Probabilistic Packet Marking for IP Traceback under Denial of Service Attack", *Proceedings of IEEE INFOCOM 2001*, March 2001.

[22] K. Park and H. Lee, "On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets", *Proceedings of ACM SIGCOMM'2001*, August 2001.

[23] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling", *IEEE/ACM Transactions on Networking*, Vol. 3, No. 3, June 1995

[24] J. Postel, Transmission Control Protocol, Request for Comments 793, DDN Network Information Center, SRI International, September 1981.

[25] S. Savage, D. Wetherall, A. Karlin and T. Anderson, "Practical Network Support for IP Traceback", *Proceedings of ACM SIGCOMM'2000*, August 2000.

[26] 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", *Proceedings of IEEE Symposium on Security and Privacy*, May 1997.

[27] F. D. Smith, F. H. Campos, K. Jeffay and D. Ott, "What TCP/IP Protocol Header Can Tell Us About the Web", *Proceedings of ACM SIGMETRICS'2001*, June 2001.

[28] D. Song and A. Perrig "Advanced and Authenticated Marking Schemes for IP Traceback", *Proceedings of IEEE INFOCOM'2001*, March 2001.

[29] A. C. Snoren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent and W. T. Strayer, "Hash-Based IP Traceback", *Proceedings of ACM SIGCOMM'2001*, August 2001.

[30] V. Srinivasan, G. Varghese, S. Suri, M. Waldvogel, "Fast and Scalable Layer Four Switching", *Proceedings of ACM SIGCOMM'98*, September 1998.

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

[32] K. Thompson, G. J. Miller, and R. Wilder, "Wide-Area Internet Traffic Patterns and Characteristics", *IEEE Network*, Vol. 11, No. 6, November/December 1997.

[33] H. Wang and K. G. Shin, "Layer-4 Service Differentiation and Isolation", Technical Report, University of Michigan, June 2001.

[34] S. F. Wu, L. Zhang, D. Massey, and A. Mankin, "Intention-driven ICMP traceback", *Internet Draft: draft-wu-itrace-intention-00.txt*, February 2001.

[35] Y. Zhang and B. Singh, "A Multi-layer IPsec Protocol", *Proceedings of 9th USENIX Security Symposium*, August 2000.