SYN-dog: Sniffing SYN Flooding Sources*

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 sources. We install SYN-dog as a software agent at leaf routers that connect stub networks to the Internet. The statelessness and low computation overhead of SYN-dog make itself immune to any flooding attacks. The core mechanism of SYN-dog is based on the protocol behavior of TCP SYN—SYN/ACK pairs, and is an instance of the Sequential Change Detection [1]. To make SYN-dog insensitive to site and access pattern, a non-parametric Cumulative Sum (CUSUM) method [4] is applied, thus making SYN-dog much more generally applicable and its deployment much easier. Due to its proximity to the flooding sources, SYN-dog can trace the flooding sources without resorting to expensive IP traceback.

1. INTRODUCTION

The growing DDoS attacks have imposed a significant threat on the availability of network services [12]. Due to the readily available tools and its simple nature, flooding packets is the most common and effective DoS attack. More than 90% of the DoS attacks use TCP [18], and TCP SYN flooding dominates in the available attacking tools and the number of known DoS attacks [16]. The SYN flooding consists of a stream of spoofed SYN packets directed to a listening TCP port of the victim, which exploits the TCP three-way handshake mechanism and its limitation in maintaining half-open connections.

Under the normal condition, when a server receives a SYN request, it sends a SYN/ACK packet back to the client and waits for client's acknowledgment. Before 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. The half-open connection is not closed until the failure of two retransmissions, which typically lasts for 75 seconds. The server has built in its system memory a backlog queue to maintain all half-open connections.

However, if a SYN request is spoofed, the victim server will never receive the final ACK packet from the client to complete the three-way handshake. Since this backlog queue is of finite size, the flooding of spoofed SYN requests can easily exhaust the victim server's backlog queue, causing all of new 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 its SYN packets. Therefore, under SYN flooding attacks, the victim server cannot respond only to legitimate connection requests while ignoring the spoofed. Note that the spoofed source address must be an invalid IP address so that it can't be reachable from the victim; otherwise, any endhost 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 received packet. The arrival of RST causes the connection to be reset, foiling the flooding attack.

Most of previous work in countering SYN flooding attacks focused on mitigating the flooding effect on the victim, such as Syn cookies [3], SynDefender [6], Syn proxying [19] and Synkill [24]. All of these defense mechanisms are stateful, i.e., states are maintained for each TCP connection or state computation is required like Syn cookies, which makes the defense mechanism itself vulnerable to SYN flooding attacks. Moreover, the defense mechanisms installed at the firewall of the victim server or inside the victim server can not give any hint about the SYN flooding sources, and hence, must rely on the expensive IP traceback [2, 20, 23, 26, 27, 32] to trace the flooding sources.

In this paper, we propose a simple and robust mechanism called *SYN-dog* to sniff SYN flooding sources without resorting to expensive IP traceback. The statelessness and low computation overhead of SYN-dog make itself immune to any flooding attacks. Instead of monitoring the ongoing traffic at the front end or the victim server itself, we install SYN-dog as a software agent at leaf routers that connect end hosts to the Internet. The key feature of SYN-dog is to utilize the inherent TCP SYN—SYN/ACK pair's behavior for sniffing SYN flooding sources.

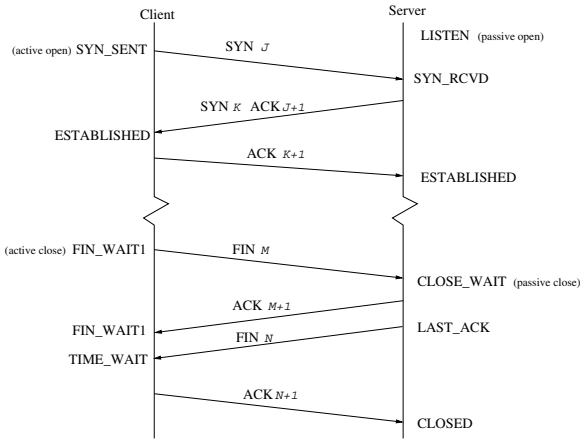


Figure 1: TCP states corresponding to normal connection establishment and teardown (from [29])

The SYN and SYN/ACK packets signal the start of a TCP connec-

*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.

tion establishment in each direction. As shown in Figure 1 that is borrowed from [29], in the ideal case, one appearance of a SYN packet results in the corresponding transmission of a SYN/ACK packet in the reverse direction within one round-trip time (RTT). Although there is no strict one-to-one match between SYN and SYN/ACK packets due to SYN losses and subsequent retransmissions, under the normal condition, a very strong positive correlation between SYN and SYN/ACK does exist as shown in Section 4.1. The discrepancy between the number of SYNs and SYN/ACKs mostly happens for the following two reasons.

- Some TCP servers are overloaded, and drop the SYN requests without generating SYN/ACK responses.
- The forwarding path of SYNs is congested, and as a result, some SYNs are dropped before they reach their destinations, so no corresponding SYN/ACKs are generated.

Due to its proximity to the flooding sources, SYN-dog can trace the flooding sources without resorting to expensive IP traceback. The flooding sources must be inside the subnet to which the leaf router is connected.

Neither state nor state computation is involved in our SYN-dog. Only two new variables are introduced to measure the number of received SYN and SYN/ACK packets at the inbound and outbound interfaces, 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—SYN/ACK pair's behavior, the dynamics of the difference between the number of SYN and SYN/ACK packets can be viewed as a stationary, ergodic random process, and SYN-dog is an instance of the Sequential Change Detection [1]. To make SYN-dog independent of sites and access patterns, the difference between the number of SYNs and SYN/ACKs is normalized by an estimated average number of SYN/ACKs. The non-parametric Cumulative Sum (CUSUM) method [4] is applied, making SYN-dog much more generally applicable and its deployment much easier.

The efficacy of SYN-dog is validated by trace-driven simulations. The evaluation results show that SYN-dog has short detection time and high detection accuracy. Due to its proximity to the flooding sources, SYN-dog mechanism not only alarms on the ongoing SYN flooding attacks but also reveals the location of the flooding sources. It is also incrementally deployable and works without requiring a wide installation of SYN-dogs.

The remainder of this paper is organized as follows. Section 2 discusses the issues related to SYN-dog. Section 3 describes the proposed detection algorithm based on the TCP SYN—SYN/ACK pair's behavior. Section 4 validates and evaluates the performance of SYN-dog using trace-driven simulations. Finally, conclusions are drawn in Section 5.

2. ISSUES RELATED TO SYN-DOG

Two issues closely related to the SYN-dog mechanism are discussed in this Section. One is the structure of SYN-dog, and the other is packet classification.

The SYN-dog consists of two *Sniffers*, which are installed at the inbound and outbound interfaces of a leaf router, respectively. The one installed at the outbound interface is the *outbound Sniffer* that counts the outgoing SYNs, while the one installed at the inbound interface is the *inbound Sniffer* that counts the incoming SYN/ACK. Figure 2 illustrates the structure of SYN-dog at a leaf router. The two sniffers coordinate with each other via shared memory, or IPC inside the router, and periodically exchange the counting information.

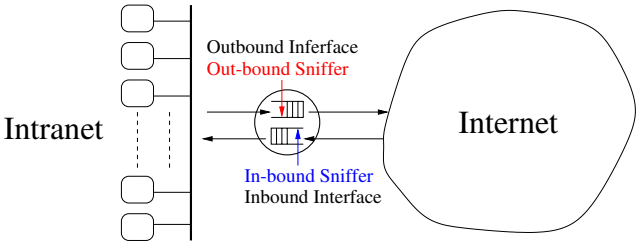


Figure 2: The structure of SYN-dog at a leaf router

SYN-dog is, in some sense, a by-product of the router infrastructure that differentiates TCP control packets from data packets [31]. This packet classification was originally motivated by the desire of providing fine-grained service differentiation to IP flows. Large-scale packet classification mechanisms [14, 15, 28] have been proposed, making it possible to distinguish the TCP SYN and SYN/ACK packets at leaf routers at a very high speed.

To identify TCP SYNs and SYN/ACKs, 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 [33] has been proposed, which allows trusted routers to access the transport-layer information. Therefore, the network-level security of IPSec can not be an obstacle to the identification and counting of TCP SYNs and SYN/ACKs at leaf routers.

Briefly, packets are classified as follows. First, we check if the IP packet contains a TCP header. The IP packet that contains the TCP header must have zero fragmentation offset. Then we compute the offset of TCP flag bits in the IP packet. Finally, the six TCP flag bits are read to determine the type of the TCP segment. The detailed description of the packet-classification algorithm is given in [31].

3. STATISTICAL FLOODING DETECTION

The rationale behind SYN-dog is that a flooding attacker's behavior is noticeably different from that of a legitimate user. It compares the observed sequence with the profile representing the user's normal behavior, and detects any significant discrepancy. Moreover, as will be seen in Section 3.2, SYN-dog can detect flooding attacks even when the normal connection arrivals are bursty and time-varying.

3.1 Sniffing Mechanism

The total number of outgoing SYNs and incoming SYN/ACKs are recorded during every observation period, t_0, at leaf routers. The setting of the observation period t_0 must balance the sniffing resolution and the algorithm's stability; t_0 is set to 20 seconds in our CUSUM algorithm. Note, however, that our algorithm is insensitive to this choice. At the end of each observation period, the number of outgoing SYNs counted by the outbound Sniffer and the number of incoming SYN/ACKs counted by the inbound Sniffer are reported to the SYN-dog's CUSUM algorithm.

Under the normal condition, the difference between the above collected numbers of outgoing SYNs and incoming SYN/ACKs is bounded compared to the total number of active TCP connections. This observation holds in spite of the fact that the total number of active TCP connections may be bursty on a small time scale, and slowly-varying on a large time scale. According to the specification of TCP/IP protocol [22, 29], an outgoing SYN is paired with an incoming SYN/ACK within one RTT. In other words, the strong correlation between the number of SYNs and SYN/ACKs is not sensitive to the request ar-

rival process. Figures in Section 4.1 clearly show that the consistent synchronization between the SYNs and SYN/ACKs is independent of the sample time, sites and time-of-day.

However, under SYN flooding attacks, there will be much more outgoing SYNs than incoming SYN/ACKs collected by the sniffers of SYN-dog. The SYN flooding traffic has significant regularity and semantics that can be filtered out. Recent experiments with SYN attacks on commercial platforms have shown that the minimum flooding rate to overwhelm an unprotected server is 500 SYN packets per second. With a specialized firewall designed to resist against SYN floods, a server can be disabled by a flood of 14,000 SYNs 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 SYN/ACKs counted by the inbound Sniffer remains largely unchanged. Therefore, the difference between the number of outgoing SYNs and incoming SYN/ACKs will dramatically increase, and remain large during the whole flooding period that typically lasts for several minutes [18]. Therefore, the occurrence of a large difference between the number of SYNs and SYN/ACKs implies the existence of flooding sources in its stub network.

3.2 The CUSUM Algorithm

Let $\{\Delta_n, n = 0, 1, \cdots\}$ be the number of outgoing SYNs minus that of incoming SYN/ACKs collected from the above sampling. To alleviate its dependence on the time, access pattern and size of the network, $\{\Delta_n\}$ is normalized by the average number $\bar{K}$ of incoming SYN/ACKs during the sampling period t_0. $\bar{K}$ can be estimated in real time and updated periodically. An example of recursive estimation and update of $\bar{K}$ is:

$$\bar{K}(n) = \alpha\bar{K}(n-1) + (1-\alpha)\text{SYN/ACK}(n), \tag{1}$$

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 $X_n = \Delta_n/\bar{K}$. The mean of X_n, denoted as c, is much less than 1. $\{X_n\}$ is not 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 flooding source sniffing algorithm is based on the Sequential Change Detection [1]. The objective of Change Detection is to determine if the observed time series is statistically homogeneous. It has been studied extensively by statisticians. See [1] and [4] for a good survey. The existing algorithms 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 about homogeneity 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 a quicker response when a flooding attack occurs. It also saves memory and computation. Despite of the numerous works on the modeling of the arrival process of TCP connection requests [5, 7, 10, 13, 21, 25], there is no consensus on whether it should be modeled as self-similar or Poisson. 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 sniffing. 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 Detection problems [1, 4].

$\{X_n\}$ is assumed to satisfy some regularity conditions. The details can be found in [4]. In practice, they are very mild and easily satisfied even by long range dependent arrival processes. In general, $E(X_n) = c < 1$. We choose a parameter $a > c$ and define $\tilde{X}_n = X_n - a$ so that it has a negative mean during normal operation. When a flooding attack takes place, $\tilde{X}_n$ will quickly become a 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 $h \gg c$.

Define $S_k = \sum_{i=1}^{k} \tilde{X}_i$, with $S_0 = 0$ at the beginning. Let

$$\begin{aligned} y_n &= (y_{n-1} + \tilde{X}_n)^+, \\ y_0 &= 0, \end{aligned} \tag{2}$$

where x^+ is equal to x if $x > 0$ and 0 otherwise. It can be shown that

$$y_n = S_n - \min_{1 \le k \le n} S_k, \tag{3}$$

i.e., y_n is the maximum continuous increment until time n. A large $\{y_n\}$ is a strong indication of a flooding attack. Since Eq. (2) is iterative and much easier to compute than Eq. (3), 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:

$$d_N(y_n) = \begin{cases} 0 & \text{if } y_n \le N; \\ 1 & \text{if } y_n > N. \end{cases} \tag{4}$$

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 parameters involved: a, the upper bound in case of normal operation and N, the flooding threshold. These values affect the performance. 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 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 started.

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 met. Therefore, the design philosophy of a statistical change detection is to minimize the detection time subject to a certain false alarm tolerance, like average time between two consecutive false alarms, worst-case false alarm time, and so on. The CUSUM rule has been shown to be asymptotically optimal with respect to the worst-case mean false alarm time when the parametric model is known for the data and the observations are independent.

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

$$P_\infty\{d_N(n) = 1\} = c_1 \exp(-c_2 N). \quad (5)$$

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

$$\tau_N = \tau(d_N) = \inf\{n : d_N(.) = 1\}, \quad (6)$$
$$\rho_N = \frac{(\tau_N - m)^+}{N},$$

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 and a represent accurate values instead of bounds, we have

$$\rho_N \to \gamma = \frac{1}{h - |c - a|}, \quad (7)$$

where $h - |c - a|$ is the mean of $\{\tilde{X}_n\}$ when $n > m$ (after a flooding attack starts). The above is a conservative estimation (upper bound) of the actual detection time when a and h are bounds rather than the true values. To ensure a long false alarm time, we set $h = 2a$ in our design.

Since a potential attacker may initiate the attack from many sites simultaneously, only part of the flooding SYN packets can be seen by each SYN-dog. To balance the detection sensitivity and false alarm time, 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, ensuring wide implementability of our sniffing mechanism.

Based on a and h, the flooding threshold N can be specified as follows: assume $c = 0$, and γ can thus be obtained by Eq. (7); and Eq. (2) specify a target detection time (i.e., the product of γ and N) such that the flooding threshold N is determined. We choose $3t_0$ as the designed detection time when $h = 2a$ and therefore, $N = 1.05$.

It is worth noting that our algorithm is to check the cumulative effect of a flooding attack. So, it can sniff a flooding source 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

$$f_{min} = (a - c) \cdot \frac{\bar{K}}{t_0}. \quad (8)$$

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 SYN-dog is evaluated by trace-driven simulations.

4. PERFORMANCE EVALUATION

To evaluate and validate SYN-dog, we have conducted trace-driven simulation experiments. The trace data used in our study are collected from four different sites at different times. The first trace was gathered at Lawrence Berkeley Laboratory Internet access point, which contains one hour's worth of all wide-area traffic between the Lawrence Berkeley Laboratory and the rest of the world. The tracing time was from 14:00 to 15:00 on Friday, January 21, 1994. The second trace was taken on March 13, 1997 on a 10 Mbps Ethernet connecting Harvard's main campus to the Internet, which is a half-hour trace and starts at 12:39 EST. 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. 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 5, 2000. A summary of the traces used in our experiments is given in Table I.

Table 1: A summary of the trace features

Trace	Duration	Traffic type
LBL	One hour	Bi-directional
Harvard	Half hour	Bi-directional
UNC-in	Half hour	Uni-directional
UNC-out	Half hour	Uni-directional
Auckland-in	Three hours	Uni-directional
Auckland-out	Three hours	Uni-directional

4.1 Normal Traffic Behavior

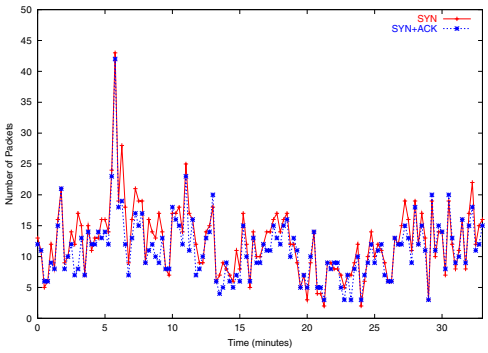
The four sets of traces represent the normal traffic behaviors at the exchange points between different stub networks and the Internet at different times. We parse the traces and extract the TCP SYN and SYN/ACK packets as the input to our leaf router simulator. The dynamics of SYN and SYN/ACK packets at the LBL site is illustrated in Figure 3 (a). The corresponding result from the Harvard trace is illustrated in Figure 3 (b). The outgoing SYNs and incoming SYN/ACKs from the UNC and Auckland traces are shown in Figures 4 (a) and (b). They clearly show the consistent synchronization between SYN and SYN/ACK packets. The consistency indicates that the synchronization is an inherent traffic behavior and independent of time and sites. Note that in the figures of the LBL and Harvard traces, the "SYN" and "SYN/ACK" are the collections from both directions, instead of "Outgoing SYN" and "Incoming SYN/ACK" as shown in the UNC and Auckland traces.

We have applied the proposed detection 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 5 (a) and (b); for the Auckland trace is plotted in Figure 5 (c). The flooding threshold is specified in last Section, i.e., $N = 1.05$. For all the traces tested, y_n's are mostly zeros. Among the isolated spikes of y_n in Harvard trace, the maximum is about 0.05; the maximal spike of y_n in Auckland trace is about 0.26. Both are much smaller than the flooding threshold N. So, no false alarms are reported.

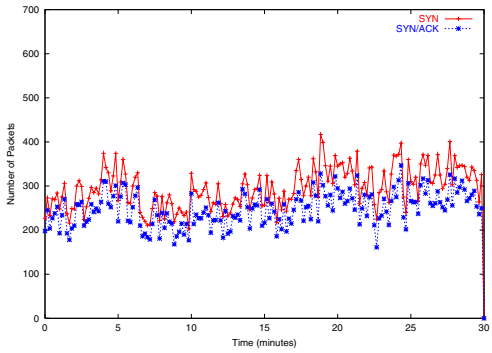
4.2 SYN Flooding Detection

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, 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

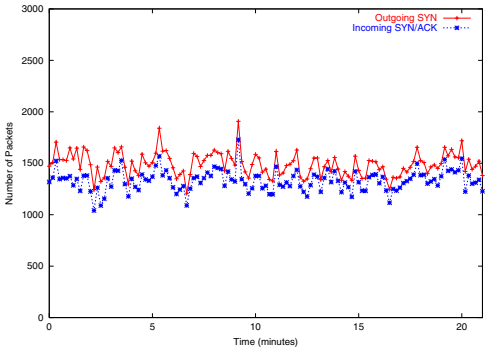


(a) LBL

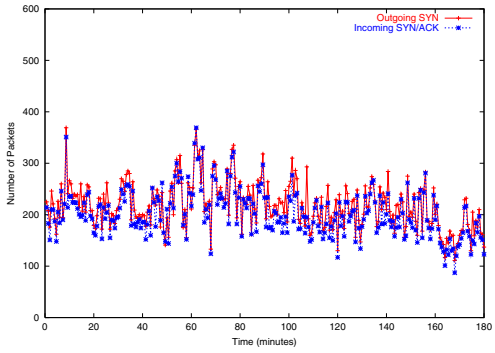


(b) Harvard

Figure 3: The dynamics of SYN and SYN/ACK packets at LBL and Harvard

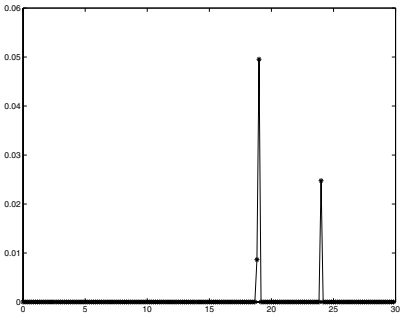


(a) UNC

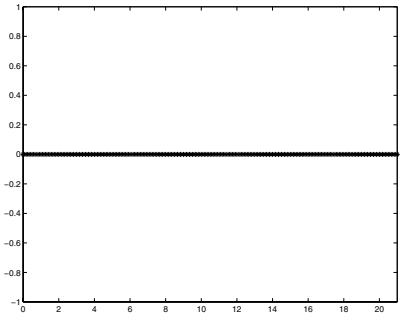


(b) Auckland

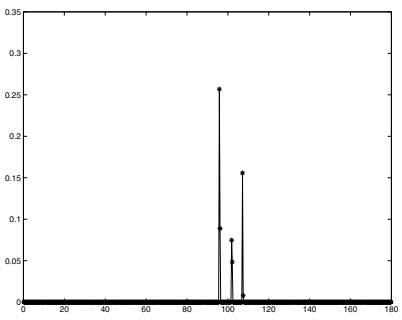
Figure 4: The dynamics of SYN and SYN/ACK packets at UNC and Auckland



(a) Harvard



(b) UNC



(c) Auckland

Figure 5: CUSUM test statistics under normal operation at: Harvard, UNC and Auckland

sends control packets to the previously-compromised slaves, 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 flooding sources.

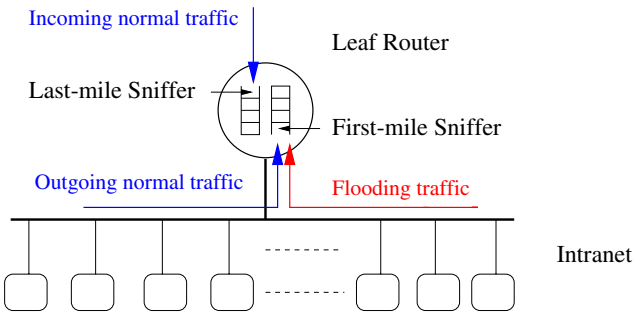


Figure 6: The trace-simulation flooding attack experiment

In the experiments of sniffing SYN flooding sources, the UNC and Auckland 2000 traces are used as the normal background traffic. Among them, UNC-in or Auckland-in is used for inbound background traffic, and UNC-out or Auckland-out is for outbound background traffic. The flooding traffic is mixed with the normal traffic, the SYN-dog at a leaf router is simulated, as shown in Figure 6. Because the non-parametric Cumulative Sum (CUSUM) method is used for sniffing flooding sources, 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 a large-scale DDoS attack, the flooding sources can be so coordinated that the traffic from each flooding source is not significant. We assume the flooding traffic is evenly distributed among different flooding sources and there is only one flooding source inside each stub network. Therefore, the flooding rate seen by the outbound Sniffer, f_i, equals the individual flooding rate inside the same stub network. This setting is intended to "hide" the flooding sources from SYN-dog. That is, the less the flooding sources inside the stub network, the less flooding traffic seen by the outbound Sniffer and the harder to detect the flooding attack. Assume that the minimum SYN flooding traffic to bring down a TCP server is V packets per second, then the flooding rate seen by the outbound Sniffer, f_i, is determined by

$$f_i = \frac{V}{A_s}$$

where A_s is the total number of the stub networks that contain a flooding source. The flooding duration in all experiments is set to 10 minutes, a typical attacking duration observed in the Internet [18]. The starting time of flooding attacks in the UNC traces is randomly chosen between 3 and 9 minutes, but the starting time in the Auckland traces lies between 3 and 136 minutes. To examine the detection sensitivity of SYN-dog, we conduct the flooding experiments in the UNC and Auckland traces and vary the flooding rate f_i seen by the outbound Sniffer.

4.2.1 *UNC Case*

Using the UNC traces as the background traffic, we observe the dynamics of y_n. Figures 7 (a), (b) and (c) plot the dynamic behaviors of y_n when f_i is set to 45, 60 and 80 SYNs per second, respectively. The accumulative effects of SYN flooding are clearly shown in the figures. In the cases of 60 and 80 SYNs per second, SYN-dog can detect the SYN flooding attack in 4 and 2 observation periods, respectively. However, in the case of 45 SYNs per second, SYN-dog takes a much longer time (about 9 observation periods, i.e., 3 minutes) to exceed the flooding threshold of 1.05. The detection performance of SYN-dog in the context of the UNC traces is summarized in Table 2, which lists the detection probability and detection time for different f_i values. The unit of detection time is the observation period t_0, which is 20 seconds.

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

f_i	Detection Prob.	Detection Time
37	0.8	19.8
40	1.0	13.25
45	1.0	8.65
60	1.0	4
80	1.0	2
120	1.0	1

Clearly, larger flooding rates lead to faster and easier detection of attacks. According to Eq. (8), 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, $\bar{K}$ will be smaller, so we can achieve more "sensitive" detection. This is confirmed by the simulation study of the Auckland traces, which is presented in the next section.

4.2.2 *Auckland Case*

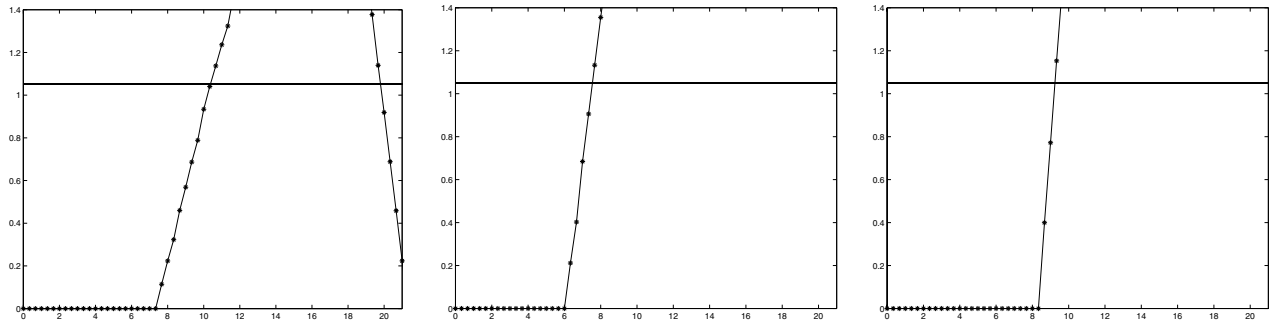
In the environment of the Auckland traces, the dynamic behaviors of y_n are illustrated in Figure 8 when f_i is set to 2, 5 and 10 SYNs per second, respectively. In the case of 2 SYNs per second, SYN-dog can detect the SYN flooding attack in about 8 observation periods. In contrast, at the flooding rate of 5 or 10 SYNs per second, SYN-dog takes a much shorter time (2 or 1 observation period, respectively) to detect the ongoing flooding. The detection performance of SYN-dog in the context of Auckland traces is summarized in Table 3. Since the $\bar{K}$ of Auckland trace is much smaller than that of UNC trace, the lower detection bound is significantly reduced from 37 to 1.75 SYNs per second.

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

f_i	Detection Prob.	Detection Time
1.5	0.55	20.64
1.75	0.95	12.95
2	1.0	7.85
5	1.0	2
10	1.0	< 1

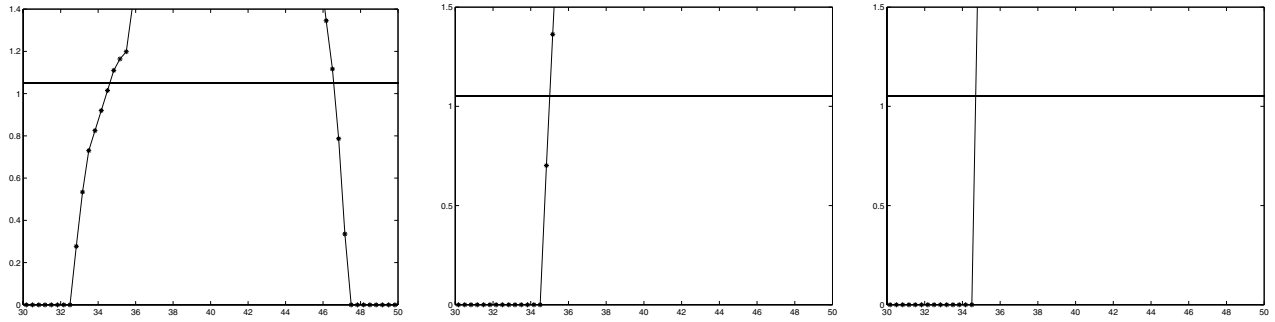
4.2.3 *Discussion*

Due to its proximity to the flooding sources, once SYN-dog detects the ongoing flooding traffic, it can further locate the flooding source inside the stub network, for example, by triggering the ingress filtering mechanism [11] and checking the MAC addresses of IP packets whose source addresses are spoofed.



(a) 45 SYNs per second (b) 60 SYNs per second (c) 80 SYNs per second

Figure 7: SYN flooding detection sensitivity at the SYN-dog of UNC



(a) 2 SYNs per second (b) 5 SYNs per second (c) 10 SYNs per second

Figure 8: SYN flooding detection sensitivity at the SYN-dog of Auckland

From the detectable flooding rate, we can specify the efficacy of our algorithm in detecting distributed SYN flooding attacks. To attack a protected server, the aggregate flooding rate V should be larger than 14,000 [8]. In the UNC case, the lower detection bound is 37, and A_s can be as large as 378 stub networks like the UNC case. Considering that the UNC stub network consists of over 35,000 users [25], it clearly demonstrates the utility and power of our SYN-dog mechanism. In the Auckland case, the lower detection bound is 1.75, and hence A_s can be as large as 8,000 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 numerous machines, it is much harder to launch the attacks from a similar large number of stub networks due to access limits.

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 37 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 9.

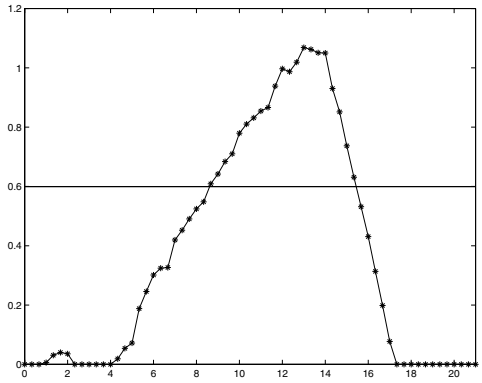


Figure 9: The improvement of flooding detection sensitivity

5. CONCLUSION

We presented a simple and robust mechanism to sniff SYN flooding sources, which is installed at leaf routers. SYN-dog utilizes the SYN—SYN/ACK pair's behavior that is invariant under various arrival models and independent of sites and time-of-day. Due to the employment of the non-parametric CUSUM method, SYN-dog is robust and stateless, resulting in the low computation overhead. Moreover, SYN-dog does not undermine end-to-end TCP performance. The efficacy of SYN-dog is evaluated and validated by trace-driven simulations. The simulation results show that SYN-dog is sensitive to the SYN flooding attacks. Once SYN-dog detects an ongoing flooding traffic, the location of flooding sources is revealed and further action can be taken to pinpoint the flooding sources.

Acknowledgment

We would like to thank Dong Lin for Harvard traces, Kevin Jeffay for UNC traces and Klaus Mochalski for Auckland 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.V. Lakshman and D. Stiliadis, "High Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching", *Proceedings of ACM SIGCOMM'98*, September 1998.

[16] G. R. Malan *et. al.*, "Observations and Experiences Tracking Denial-Of-Service Attacks Across a Large Regional ISP", Technical Report, Arbor Netorks, 2001.

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

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

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

[20] 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.

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

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

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

[24] 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.

[25] 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.

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

[27] 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.

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

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

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

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

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

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