More Accurate and Fast SYN Flood Detection

Changhua Sun, Chengchen Hu, Yi Tang and Bin Liu
Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, China
{sch04,tangy05}@mails.tsinghua.edu.cn, {huc, liub}@tsinghua.edu.cn

***Abstract*—SYN flood attacks still dominate Distributed Denial of Service attacks. It is a great challenge to accurately detect the SYN flood attacks in high speed networks. An intelligent attacker would evade the public detection methods by suitably spoofing the attack to pretend to be benign. Keeping per-flow or per-connection state could eliminate such a spoofing, but meanwhile, it also consumes extremely huge resources. We propose a more accurate and fast SYN flood detection method, named SACK^2, which could detect all kinds of SYN flood attacks with limited implementation costs. SACK^2 exploits the behavior of the SYN/ACK-CliACK pair to identify the victim server and the TCP port being attacked, where a SYN/ACK packet is sent by a server when receiving a connection request and a CliACK packet is the ACK packet sent by the client to complete the three-way handshake. We utilize the space efficient data structure, counting Bloom filter, to recognize the CliACK packet. Comprehensive experiments demonstrate that, SACK^2 is the fastest and most accurate detection method compared with related methods which also leverage the packet pair's behavior. The memory cost of SACK^2 for a 10Gbps link is 364KB and can be easily accommodated in modern routers.**

I. Introduction

Distributed Denial of Service (DDoS) attack remains a serious problem on the Internet, as it takes advantage of the lack of authenticity in the IP protocol, destination oriented routing, and stateless nature of the Internet. Among various DDoS attacks, TCP SYN flooding [1] is the most commonly used one and known as one of the most powerful flooding methods [2]. It still dominates DDoS attacks according to the recent NANOG report [3] in 2008.

There are two main causes of SYN flood attacks. The first is the inherent asymmetry feature in TCP three-way handshake protocol, which enables an attacker to consume substantial resources at the server, while sparing its own resources. The other is that the server cannot control the packets it receives, especially the SYN packets can easily reach the server without its approval. Fig. 1 illustrates the three-way handshake protocol: 1) A client sends a SYN packet to a server to perform an active open request; 2) The server reserves connection resources (backlog queue) to track the TCP state on receiving a SYN packet and replies with a SYN/ACK packet in response; 3) Finally, the client sends an ACK back to the server as an acknowledgement, and the connection is established when receiving this ACK on the server side. We call the ACK packet in the third step to complete the three-way handshake as *CliACK* (Client ACK), and all the other ACK packets during the TCP life time as *DACK* (Data ACK). During SYN flood attacks, an attacker generates a large number of SYN requests but never sends the CliACK packets to complete the connections. Since the victim server allocates resources to track the TCP state for each received SYN packet, its backlog queue can be easily exhausted and all the new incoming SYN requests are dropped. Furthermore, many other system resources, such as CPU and network bandwidth, are occupied. Fig. 1 also shows the connection's release process. Either server or client can initiate a FIN packet and wait for the acknowledgement to close the connection. Under special conditions, a RST packet can also be used to close a connection. Similar to the CliACK packets, the corresponding FIN/RST packets to the SYN flood packets do not exist during SYN flood attacks.

A number of literatures investigate the mitigation or detection of the SYN flood attack. Victim modifications, *e.g.*, SYN cache [4] and SYN cookies [5], are the most viable techniques [6] up-to-date to mitigate SYN floods, as they only need to modify the victim servers and can be easily deployed. However, SYN cache is still possible to exhaust the backlog queue when the server is forced to drop the incomplete connection state at a time interval smaller than the round trip time, and SYN cookies are only able to encode the maximum segment size TCP option up to 8 unique values. It is impossible for SYN cookies to encode other TCP options, like window scale and selective acknowledgment, which are believed to improve TCP's performance and are widely supported in modern operating systems by default. Therefore, it is not recommended to use SYN cookies under normal conditions. Recently, leaf router-based SYN flood defenses, especially detection schemes [7]–[12], have received much attention in current literatures with the advantage of protecting a batch of servers in the ISP's networks. They try to utilize the unspoofable characteristic during the attack. Their ideas are interesting but they would miss some elaborated SYN flood attacks, especially when the attack is suitably spoofed to appear benign. In this paper, we concentrate on the accurate and fast router-based detection method for all kinds of SYN flood attacks.

Four objectives motivate us to explore a more accurate and fast detection method: 1) to deal with all kinds of SYN flood attacks, no matter what packets an attacker can spoof; 2) to report the start and the end of the SYN flood attack; 3) to discover the specific server and port under attack; and 4) to

This work is partly supported by NSFC (60625201, 60873250), the Specialized Research Fund for the Doctoral Program of Higher Education of China (20060003058), the Cultivation Fund of the Key Scientific and Technical Innovation Project, MOE, China (705003) and 863 high-tech (2007AA01Z216, 2007AA01Z468).

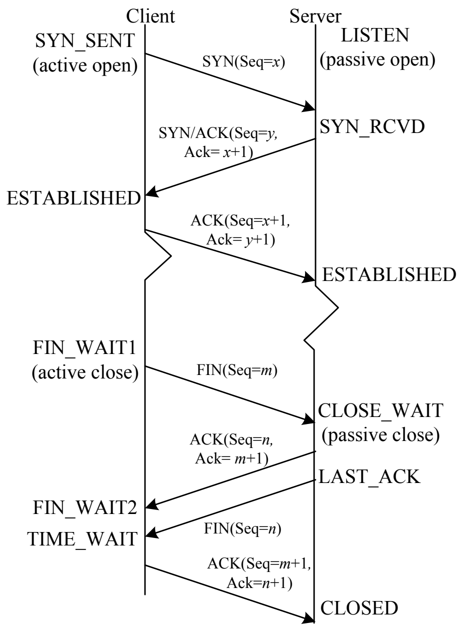

Fig. 1. TCP connection's establishment and release under normal conditions.

distinguish the SYN flood attack from the TCP port scan attack, which would send a large number of SYN packets to one or more interested ports of one or many servers.

We propose a more accurate and fast method, named *SACK*^{2}, the first to utilize SYN/ACK-CliACK pair's behavior to detect the SYN flood attacks (this paper is the extension of the poster paper [13]). SACK^2 can not only report the start and the end of the attack, but also disclose the specific server and port under attack. The basic idea of SACK^2 is to check the difference of SYN/ACK and CliACK packets in number. As shown in Fig. 1, it has only one SYN/ACK packet and one CliACK packet for each normal TCP connection. During the SYN flood attack, since the SYN/ACK packets have no corresponding CliACK packets, the number of SYN/ACK packets is much larger than the number of CliACK packets. Similar idea is used by previous work which checks the SYN-FIN or SYN-CliACK pair for detection, but an attacker can easily spoof the FIN/RST packet (the sequence number in FIN/RST (m) has little relationship with that in SYN (x), it is impossible to judge whether m is correct only according to x), and CliACK packet (it is impossible to judge whether the acknowledgment number $y + 1$ is correct according to x) to appear benign and evade their detections. By contrast, SACK^2 employs the space-efficient data structure, counting Bloom filter [14], to store the full information (*6-tuple*) of per TCP connection attached in the SYN/ACK packet, including client and server's IP addresses (<SIP, DIP>), ports (<SP, DP>), and initial sequence numbers (<SEQ, ASEQ>), and then to recognize the corresponding CliACK packet. In this manner, there is no way for an attacker to generate the spoofed CliACK packets to evade the detection. After that, we exploit another counting Bloom filter to report the server and port being attacked. The memory cost of SACK^2 is 364KB even for a 10Gbps link in the worst case, making it feasible and attractive for the hardware implementation in modern routers. Since our scheme can be deployed at the ISP's leaf router to report attacks for its high-end customers, which actually provides enough incentive for the deployment. To our best knowledge, SACK^2 is the first study which utilizes the SYN/ACK-CliACK pair's behavior to detect SYN flood attacks, and it is the first method that can detect all kinds of SYN flood attacks. Especially, SACK^2 possesses the following advantages compared with competitive approaches.

- SACK^2 can detect the start and the end of SYN flood attacks to any specific victim server and port;
- SACK^2 can deal with all elaborated SYN flood attacks;
- SACK^2 has the shortest response time to report the attack's start and end, and has low and controllable false positive (report attacks by mistake) and false negative (cannot report attacks by mistake) rate.

The rest of the paper is organized as follows. Section II presents related work. Section III describes SACK^2 in detail. Section IV presents the performance evaluations through real and synthetic traces. Finally Section V concludes the paper.

II. Related Work

There are many SYN flood detection methods which exploit the different characteristics between the attack and normal conditions. The relationship between the control packets in the connection establishment and release, like SYN, SYN/ACK, CliACK, FIN and RST, is widely used in the SYN flood detection. These methods, based on the control packet pair's behavior, can be roughly classified as follows.

1) SYN-FIN (SF) methods: Wang *et al.* [7] propose SF1 to count the number of incoming SYN and FIN/RST packets in each detection period, and use a non-parametric Cumulative SUM based method to detect the change of the normalized difference between the number of the SYN and FIN/RST packet. Sun *et al.* [10], [11] use a counting Bloom filter to recognize the corresponding FIN/RST packet with a same 4-tuple (<DIP, SIP, DP, SP>) as the SYN packet (SF2). Kompella *et al.* [12] propose PCF to check the difference of the incoming SYN and FIN packet with the same destination IP address and port pair. Unlike SF1 and SF2, PCF can report the specific server and port under attack. However, SF methods cannot detect the attacks mixed with spoofed FIN/RST packets, since it is impossible to judge whether the FIN packet is legitimate without maintaining the full connection state.

2) SYN-SYN/ACK (SS) method: Wang *et al.* [8] utilize the *SYN-SYN/ACK pair* which monitors the difference between the outgoing SYN packet and the incoming SYN/ACK packet at the attack sources. The incentive for deployment is very low, and the difference between SYN and SYN/ACK at one attack source may not be high enough for detection. We omit it.

3) SYN-CliACK (SC) method: Chen *et al.* [9] utilize the *SYN-CliACK pair* to detect and defend against SYN flood attacks under different types of IP spoofing. However, it is unclear in their paper how to distinguish whether an ACK packet is the CliACK to complete the TCP connection or a DACK. Furthermore, an attacker can send a mixture of SYN and spoofed CliACK packets, such as with wrong sequence numbers to evade the detection. It is reported that RBot, one type of Botnet, can mix SYN and CliACK packets to launch SYN flood attacks [15].

Instead, SACK^2 exploits the unspoofable feature of the SYN/ACK-CliACK pair and can deal with any SYN flood attacks. In addition, only PCF and SACK^2 can report the server and port under attack.

III. SACK^2

A. Deployment and Assumption

SACK^2 is proposed to be deployed at the ISP's leaf router to report SYN flood attacks at the early stage. To avoid an attacker's spoofing, SACK^2 needs to monitor the traffic in two directions, which includes the outgoing SYN/ACK and incoming CliACK packets.

A successful SYN flood attack is to insure that no new connection request can complete with the victim server, by forcing the server to drop incomplete connection state at a rate larger than RTT [4]. Suppose the backlog has N entries, and the oldest entry would be dropped when the backlog is full. If the attacked SYN flood rate (the number of SYN flood packets per second, SYN/s), denoted as r, is higher than N/RTT, any legitimate connection request cannot complete, since when the corresponding CliACK packet to complete the three-way handshake is received at the server, the connection request, SYN packet, is already dropped. By default, for current modern servers, N is 80 in Windows 2000 Server, 400 in Windows Server 2003 and above, 1024 in Linux, and 512×30 in FreeBSD. Besides, the backlog in FreeBSD is composed of 512 hash buckets, and each bucket has a limit of 30 entries [4]. An attacker may use the rate higher than $30/RTT$ to deny legitimate clients to a particular bucket in theory. Since most TCP connection's RTT is less than 500ms [16], we assume the upper bound of the flood rate to detect attacks against one victim server is 60 SYN/s.

In SACK^2, during a detection period t_p, if the number of SYN flood packets sending to a specific server is greater than $th = r \cdot t_p$, we report an attack. The lower bound of th, denoted as TH_{low}, is roughly the maximum SYN/ACK packet number per RTT of the <SIP, SP> pair under normal condition. The lower bound is used to distinguish flash crowds from SYN floods. In practice, TH_{low} can be obtained from history data and measurement. t_p determines the response time to report the start and the end of an attack, and is only required to be greater than RTT, and satisfy $60 \cdot t_p \geq TH_{low}$. And th/t_p pps (packet per second) represents the lowest flood rate SACK^2 can detect.

Our detection metrics are different from the previous work. Wang *et al.* [7] use the relative half-open connection rate as the detection metric, *i.e.*, the number of difference between the SYN and FIN/RST packet normalized against the average FIN/RST packet in each detection period. Kompella *et al.* [12] utilize the absolute number of half-open connections, *i.e.*, the number of difference between the SYN and FIN packet, and they choose the number as 150 to be a proper threshold from a probabilistic analysis. This threshold is fixed and the analysis does not consider flash crowds, such as larger number of new long live connections in one measurement period, which would lead to large false positive rate.

B. Architecture and Algorithm

SACK^2 exploits the behavior of the SYN/ACK-CliACK pair and the key point is to recognize the CliACK packet efficiently. The time interval between the SYN/ACK and its CliACK is a RTT (round trip time). The upper bound of RTT can be chosen as 0.5s [16], even under DDoS attacks [17]. The intuitive method is to store the 6-tuple of SYN packets for RTT, and then to recognize the right CliACK packet. One 6-tuple item takes 20 bytes. The total number of items, n, stored in one RTT mainly depends on the sum of maximum SYN packet rate. According to the reports on a 10Gbps backbone link from CAIDA's monitor [18], the maximum TCP flow number per second of the backbone link is 36.96K. In addition, the SYN flood rate reported [19] can reach to about 50Kpps. Therefore, it is very safe to suppose $n = 100000 \times 0.5 = 50000$ to report the SYN flood attack at the early stage. In this case, the memory cost of the intuitive method is about 2MB, which can only be accommodated by external memory. The challenge is how to devise a mechanism to find the right SYN packet item when receiving ACK packets. By contrast, SACK^2 can recognize the CliACK packet more efficiently with low cost.

Fig. 2 illustrates the logic architecture of SACK^2. It consists of two parts: 1) to recognize the right CliACK packets; and 2) to report the specific server and port under attack. SACK^2 utilizes counting Bloom filters in both parts. A Bloom filter [14] is a simple space-efficient data structure for representing a set in order to support membership queries. It consists of an array of m bits and k independent hash functions. A variant of the original Bloom filter, called *counting Bloom filter* [14], [20], can be easily used for inserting and deleting items. In a counting Bloom filter, each entry in the Bloom filter is not a single bit but a small counter.

First, a counting Bloom filter, CBF$_1$, is utilized to recognize the right CliACK packets. For each outgoing SYN/ACK packet, we extract its source and destination IP addresses, source and destination ports, sequence number and ACK sequence number and use the *6-tuple* <SIP, DIP, SP, DP, SEQ, ASEQ>, denoted as a, as the input for the k hash functions $h_1, h_2, ..., h_k$, hash value of each with range $\{0, 1, ..., m-1\}$. We increase the value by one for each bucket location $h_i(a)$. If the value of any $h_i(a)$ bucket would be overflowed, we do not increase it and simply keep its maximal value. For each incoming ACK packet, we extract and use its 6-tuple <DIP, SIP, DP, SP, ASEQ-1, SEQ>, denoted as b, as the input for the k hash functions of CBF$_1$. If the value of each bucket is greater than zero, *i.e.*, $h_i(b) > 0$ for $1 \leq i \leq k$, this ACK is recognized as the CliACK packet with some wrong probability, referred to as the *false positive* rate p_{fp}, since some buckets of $h_i(b)$ may be set to be greater than zero by other elements. For the recognized CliACK packet, we decrease the value by one for each bucket location $h_i(b)$, in

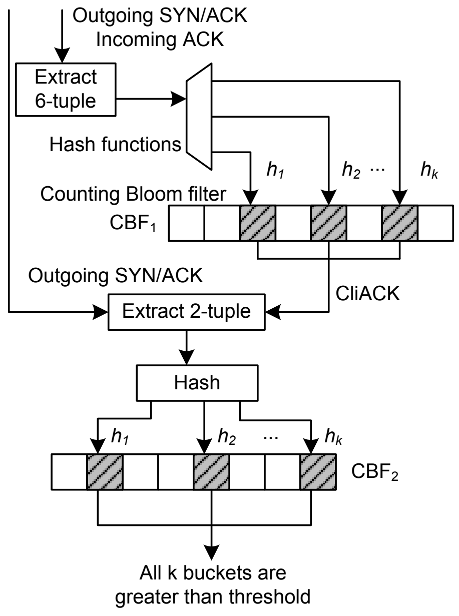

Fig. 2. The logic architecture of SACK^2.

order not to recognize the subsequent DACKs as the CliACK packet. When the number of nonzero buckets is greater than $m/2$, it is recommended to reset these buckets to zero to avoid large false positive rate. If CBF$_1$ needs to be reset, another counting Bloom filter, CBF$_{backup}$, the same as CBF$_1$, is used to recognize the CliACK packet. After CBF$_1$ is reset, we restore to use it.

After identifying the corresponding CliACK packets, another counting Bloom filter, CBF$_2$ (the parameters of CBF$_2$, including m, k, bits per bucket, and hash functions, are all independent of the parameters of CBF$_1$), is employed to detect the specific SYN flood attack. During one detection period, t_p, for each outgoing SYN/ACK packet, we extract its source IP address and port, <SIP, SP>, denoted as c, as the input for the k hash functions of CBF$_2$. We increase the value by one for each bucket location $h_i(c)$, and if at any point the values of all the buckets are larger than th, we report that the <SIP, SP> pair is being SYN flooded. For each incoming CliACK packet, we extract its destination IP address and port, <DIP, DP>, denoted as d, as the input for the k hash functions of CBF$_2$. We decrease the value by one for each bucket location $h_i(d)$, and if the value of any bucket location is equal to th, we return to a non-alert state for the <DIP, DP> pair. At the end of the detection period, the nonzero buckets of CBF$_2$ are reset to zero. If we do not detect any attack for an entire detection period, we turn to a non-alert state.

C. Parameters Configuration

The traditional Bloom filter has only false positive rate, p_{fp}, which indicates the probability we determine an element belonging to the Bloom filter though it does really not, and is shown as (1), if it supports n elements, and $m > kn$. By optimizing k with minimized p_{fp}, we get (2).

$$p_{fp} \approx (1 - e^{-kn/m})^k. \qquad (1)$$

$$k = \ln 2 \cdot (m/n). \qquad (2)$$

The number of hash functions, k, is chosen according to (2) and the trade-off between the memory cost and false positive. For the counting Bloom filter, since we add and delete elements from it, it would also bring false negative rate. It is difficult to provide theoretical analysis for this rate since it depends on distribution of addition and deletion to the same bucket, and the number of bits for each bucket. If these operations are random, the rate would be very low [14]. The analysis from [20] reveals that 4 bits per counter should suffice for most network applications to avoid counter overflow. In this case, the probability of overflow is no more than $1.37m \cdot 10^{-15}$. As CBF$_1$ is just used for general purpose, *i.e.*, adding, querying and deleting elements, we choose 4 bits for each bucket in CBF$_1$. Since we rely on the value of the buckets in CBF$_2$ to report the attack, we need to choose a larger number of bits per bucket. We choose 32 bits per counter for CBF$_2$, leading to the seldom overflow.

D. Implementation Cost

First, we discuss the memory cost. Generally, n is only required to be greater than the number of maximum SYN packets per RTT. In the worst case, $n = 50000$ for a 10Gbps link as discussed in Section III-B. To make a trade-off between memory cost and false positive rate, we choose $k = 4$, and then $m = 6n$, $p_{fp} = 0.0561$ from (2). Since we use 4 bits per counter, the memory needed by CBF$_1$ and CBF$_{backup}$ (denoted as M) is 300KB. For CBF$_2$, we choose 4 hash functions and 32 bits per counter. The number of buckets is dependent on how many the server and port (<DIP, DP>) pairs are being attacked. We choose 16 000 buckets similar to PCF's parameters [12]. The memory cost of CBF$_2$ is 64KB. Therefore, the total memory cost needed by our scheme is 364KB for the 10Gbps link, which can be easily accommodated using the on-chip memory in modern routers.

For computation cost, we choose the hash functions from the H_3 category [21] to configure the Bloom filters. H_3 can be easily implemented by hardware (as well as software) since it mainly contains XOR operations, proportional to the number of output bits. The computation cost is very low.

IV. Performance Evaluation

In this section, we evaluate SACK^2 and related work including SF1, SF2, PCF and SC. Since it is not clear how to determine whether an ACK packet is the CliACK packet to complete the TCP connection by SC [9], we compare it with SACK^2 by qualitative analysis. The following performance measures are studied: 1) *False positive* is defined as the reported attacks by mistake; 2) *False negative* is defined as the unreported attacks by mistake; and 3) *Response time* consists of the detection and quiescence time. The detection time is the time interval between the attack's start and scheme's detection of its start, while the quiescence time is the time interval between the attack's end and scheme's detection of its end.

It is reported [22] that there exist some low rate SYN flood attacks in Auckland-VIII trace set [23], which is public

TABLE I
TRACE INFORMATION AND THE NUMBER OF CONCERNED PACKETS

Trace	Start	SYN	FIN/RST	SYN/ACK	CliACK
A1	08:00 Wed	521,347	190,372	48,389	42,431
A2	16:00 Wed	571,483	284,161	78,035	69,367
A3	08:00 Thu	415,791	188,801	50,726	44,402
A4	14:00 Thu	721,281	305,046	77,261	71,118

available from NLANR project. We use this trace set to evaluate SACK^2. This trace set is a two-direction and two-week anonymized trace collected at the Internet access link of University of Auckland in December 2003. Each daily trace is divided into 24 one-hour traces. We analyze 48 traces from Dec. 3 to Dec. 4. The maximum SYN/ACK packet number per RTT per <SIP, SP> pair in most of these traces is less than 50, but some reaches 187. We choose the lower bound of the threshold as 190, *i.e.*, $TH_{low} = 190$, and $t_p = 60$ to detect flood rate higher than 3.2 SYN/s. Since the maximum SYN packet per RTT in these traces is less than 201, and the <SIP, SP> pairs is less than 500 in the University, we choose 2048 buckets for CBF$_1$ and CBF$_{backup}$, and 3000 for CBF$_2$. The total memory cost of SACK^2 is 18KB to detect SYN flood attacks in these traces. And the false rate to recognize the CliACK packet is maximum 5.61% in theory.

We randomly choose 4 traces as illustrated in Table I to describe the detailed results. SYN, FIN/RST and CliACK are the incoming direction, while SYN/ACK is the outgoing direction. The CliACK packet is identified using the full state approach. It is noticed that the number of incoming SYN is much larger than the one of the outgoing SYN/ACK. The reason is that there are many TCP port scans in there traces. We use trace A1 as the background traffic (remove TCP port scan traffic) and the modified TFN2K [24] to generate synthetic traces when needed.

A. *False Positives and Negatives*

The false rate of SYN flood detections lies in two aspects: 1) the basic assumption of the algorithm, and 2) the special data structure like Bloom filter to reduce the memory cost.

We implement the basic assumption, like SYN-FIN or SYN/ACK-CliACK pair's behavior, of SF1, SF2, PCF and SACK^2 using the full state approach. Since only PCF and SACK^2 can report the specific attack, we first compare them on the real traces. We adjust PCF to use the same detection metrics, $t_p = 60$ and $th = 150$, as SACK^2. Table II shows the detection results. PCF mistakes 6 port scan <IP, Port> pairs as SYN flood attacks. In the TCP port scan, an attacker sends a few number of SYN packets to interested server and port, and the server would reply with RST packets or ignore the unwanted SYN requests. Since the client or the server can use one RST to close the two-side TCP connection, PCF reports three pairs as false positives. 10.0.0.125 : 80 is reported under attack in A1 by SACK^2, but PCF does not report it. The reason is that SYN/ACK packets are retransmitted.

We also generate the following five attacking scenarios to evaluate the basic assumption:

- **S1**: An attacker sends a mixture of FIN and SYN flood packets, and the FIN packets have no relationship with the SYN packets;
- **S2**: An attacker sends each SYN flood packet, followed by one FIN packet, with the same <DIP, DP> pairs as the SYN flood packet;
- **S3**: An attacker sends each SYN flood packet, followed by one FIN packet, with the same 4-tuple, but wrong sequence number as the SYN flood packet;
- **S4**: An attacker sends each SYN flood packet, followed by one ACK packet, with the same <SIP, DIP> pairs as the SYN flood packet;
- **S5**: An attacker sends each SYN flood packet, followed by one ACK packet, with the same 4-tuple, but wrong sequence number as the SYN flood packet.

TABLE II
COMPARISON BETWEEN PCF AND SACK^2 ON REAL TRACES

<IP, Port>	Trace	PCF	SACK^2
10.0.0.125 : 80	A1		Attack
10.5.112.150 : 80	A2	Attack	Attack
10.0.7.75 : 80	A2	Attack	Attack
10.0.0.70 : 80	A3,A4	Attack	Attack
10.0.0.26 : 443	A1,A2,A3	False Positive	
10.1.81.73 : 80	A2	False Positive	
10.0.0.1 : 1080	A3,A4	False Positive	
10.0.0.29 : 25	A1,A3	Port Scan	
10.0.0.206 : 25	A1,A2,A3,A4	Port Scan	
10.0.0.151 : 25	A1,A2,A3,A4	Port Scan	
10.0.0.156 : 25	A3,A4	Port Scan	
10.0.0.1 : 1081	A4	Port Scan	
10.0.2.72 : 25	A4	Port Scan	

TABLE III
COMPARISON OF DIFFERENT DETECTION SCHEMES ON FIVE SCENARIOS

Scheme	**S1**	**S2**	**S3**	**S4**	**S5**
SF1	X	X	X	$\surd$	$\surd$
PCF	$\surd$	X	X	$\surd$	$\surd$
SF2	$\surd$	$\surd$	X	$\surd$	$\surd$
SC	$\surd$	$\surd$	$\surd$	X	X
SACK^2	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$

The first three scenarios aim to evade the detection utilizing the SYN-FIN pair's behavior, while **S4** and **S5** try to bypass the detection utilizing the SYN-ACK pair's behavior. Table III shows whether a schemes can detect the attacks or not under **S1-S5**. "X" indicates that a scheme does not report the attack by mistake, which would lead to false negative rate, while "$\surd$" means that the scheme reports the specific attack correctly. Since the detection schemes, including SF1, PCF, SF2, and SC, utilize the partial information of the SYN-FIN or SYN-CliACK pair's behaviors, the intelligent attacker can evade the detection by using some specific attacking scenarios, which would bring false negative rate. By contrast, SACK^2 exploits the full information of the SYN/ACK-ACK pair's behavior, and thus can detect any of these attacks. It is impossible for the attacker to bypass our detection.

Next, we study the false rate brought by the special data structure. Since SACK^2 uses two counting Bloom filters to store the full TCP connection information, it has larger false rate at this aspect. Fig. 3 illustrates the false rate to recognize the CliACK packet in each detection period. Except four

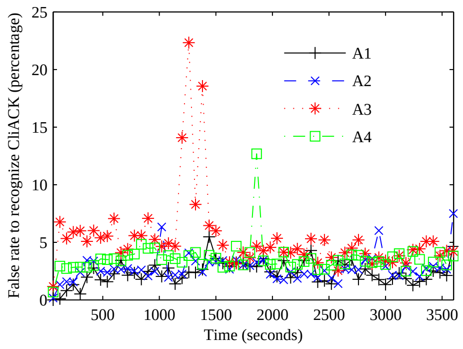

Fig. 3. False rate to recognize CliACK packets.

periods, the false rate is roughly 5%, which agrees with theoretical analysis of the Bloom filter and does not affect the final detection results. In summary, only SACK^2 can deal with SYN flood attacks with any spoofing.

B. Response Time

The response time of both PCF and SACK^2 is dependent on the detection period t_p, threshold th and the average SYN flood rate r. They only report attacks when $r \geq th/t_p$. The detection time is roughly th/r, which is less than t_p. Suppose the time interval between the attack end point and its detection period's start point is Δ. If $\Delta < th/r$, the quiescence time is $t_p - \Delta$. Otherwise, the time is $2 \cdot t_p - \Delta$. Since the time interval of SYN/ACK-CliACK is RTT, which is much shorter than that of SYN-FIN, SACK^2 can use shorter detection period than PCF to achieve shorter response time. For the aggregated level algorithms, it is pointed in [25] that the quiescence time of SF would be very long for long attack period and high attack rate. Our trace-driven simulations agree with this analysis. Due to the page limit, we ignore the experiment results.

V. Conclusion

In this paper, we propose a leaf router-based method, SACK^2, to detect the start and the end of the SYN flood attack, and also report the specific victim server and port being attacked. Under normal conditions, the number of the outgoing SYN/ACK packets is almost equal to the number of the incoming CliACK packets. However, during the SYN flood attacks, an attacker would not send the corresponding CliACK packets and the number of SYN/ACK packets would be greater than the number of CliACK packets. SACK^2 exploits the SYN/ACK-CliACK pair's behavior to discover SYN flood attacks. We keep the full information of per TCP connection attached in the SYN/ACK packet in the space-efficient data structure, counting Bloom filter, including client and server's IP addresses, client and server's ports, and client and server's initial sequence numbers. In this case, SACK^2 does not leave any chance for an attacker to evade the detection, *i.e.*, SACK^2 can detect all kinds of SYN flood attacks. Furthermore, the time to report the attack's start is less than one detection period, while the time to report the attack's end is less than two detection periods. Through experiments on real and synthetic traces, it is demonstrated that SACK^2 is the fastest and most accurate detection method compared with the methods which exploit the packet pair's behavior. The memory cost of SACK^2 is 364KB for a 10Gbps link in the worst case, which can be easily accommodated in modern routers.

References

[1] "CERT advisory CA-1996-21 TCP SYN flooding and IP spoofing attacks," 1996. [Online]. Available: http://www.cert.org/advisories/CA-1996-21.html
[2] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the DoS and DDoS problems," *ACM Computing Surveys*, vol. 39, no. 1, 2007.
[3] C. Labovitz, D. McPherson, S. Iekel-Johnson, and M. Hollyman, "Internet Traffic Trends - A View from 67 ISPs," NANOG, 2008. [Online]. Available: http://www.nanog.org/meetings/nanog43/presentations/Labovitz_internetstats_N43.pdf
[4] J. Lemon, "Resisting SYN flood DoS attacks with a SYN cache," in *USENIX BSDCon*, 2002.
[5] "SYN cookies." [Online]. Available: http://cr.yp.to/syncookies.html
[6] W. Eddy, "TCP SYN flooding attacks and common mitigations," RFC 4987, 2007. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4987.txt
[7] H. Wang, D. Zhang, and K. G. Shin, "Detecting SYN flooding attacks," in *IEEE INFOCOM*, 2002.
[8] ——, "SYN-dog: sniffing SYN flooding sources," in *IEEE ICDCS*, 2002.
[9] W. Chen and D. Y. Yeung, "Defending against TCP SYN flooding attacks under different types of IP spoofing," in *Fifth International Conference on Networking (ICN)*, 2006.
[10] C. Sun, J. Fan, and B. Liu, "A robust scheme to detect SYN flooding attacks," in *International Conference on Communications and Networking in China (ChinaCom)*, Shanghai, China, August 22-24 2007.
[11] C. Sun, J. Fan, L. Shi, and B. Liu, "A novel router-based scheme to mitigate SYN flooding DDoS attacks," in *IEEE INFOCOM (Student Poster)*, Anchorage, Alaska, USA, May 2007.
[12] R. R. Kompella, S. Singh, and G. Varghese, "On scalable attack detection in the network," *IEEE/ACM Transactions on Networking*, vol. 15, no. 1, pp. 14–25, 2007.
[13] C. Sun, C. Hu, Y. Zhou, X. Xiao, and B. Liu, "A more accurate scheme to detect SYN flood attacks," in *IEEE INFOCOM Student Workshop*, Rio de Janeiro, Brazil, Apr. 20 2009.
[14] A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: a survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2004.
[15] V. Thing, M. Sloman, and N. Dulay, "A survey of Bots used for distributed denial of service attacks," *IFIP International Information Security Conference*, vol. 232, p. 229, 2007.
[16] H. Jiang and C. Dovrolis, "Passive estimation of TCP round-trip times," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 3, pp. 75–88, 2002.
[17] K. chan Lan, A. Hussain, and D. Dutta, "Effect of malicious traffic on the network," in *Passive and Active Measurement (PAM)*, 2003.
[18] "equinix-chicago monitor Realtime reports," 2008. [Online]. Available: http://www.caida.org/data/passive/monitors/equinix-chicago.xml
[19] "SCO Offline from Denial-of-Service Attack," 2003. [Online]. Available: http://www.caida.org/research/security/sco-dos/
[20] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," in *ACM SIGCOMM*, 1998.
[21] M. V. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient hardware hashing functions for high performance computers," *IEEE Transactions on Computers*, vol. 46, no. 12, pp. 1378–1381, 1997.
[22] J. Mirkovic, S. Wei, A. Hussain, B. Wilson, R. Thomas, S. Schwab, S. Fahmy, R. Chertov, and P. Reiher, "DDoS benchmarks and experimenter's workbench for the DETER testbed," in *3rd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities(Tridentcom)*, 2007.
[23] "Auckland-VIII trace data," NLANR Project, 2003. [Online]. Available: http://pma.nlanr.net/Special/auck8.html
[24] D. Dittrich, "Distributed denial of service (DDoS) attacks/tools." [Online]. Available: http://staff.washington.edu/dittrich/misc/ddos/
[25] M. Beaumont-Gay, "A comparison of SYN flood detection algorithms," in *International Conference on Internet Monitoring and Protection (ICIMP)*, 2007.