# Intentional Dropping: A Novel Scheme for SYN Flooding Mitigation

Basheer Al-Duwairi and G. Manimaran
Dependable Computing & Networking Laboratory
Dept. of Electrical and Computer Engineering
Iowa State University, Ames, IA 50011
Email: {dbasheer,gmani}@iastate.edu

***Abstract*— This paper presents a novel scheme to mitigate the effect of SYN flooding attacks. The scheme, called intentional dropping based filtering, is based on the observation of *client's persistence* (i.e., client's reaction to packet loss by subsequent retransmissions) which is very widespread as it is built in TCP's connection setup. The main idea is to *intentionally drop* the first SYN packet of each connection request. Subsequent SYN packet from a request is passed only if it adheres to the TCP's timeout mechanism. Our analysis shows that the proposed scheme reduces attacker's effective attack rate significantly with an acceptable increase in connection establishment latency.**

## I. Introduction

The Internet has witnessed enormous growth over the last few years. Internet services became a necessity of today's life as millions of people around the globe use these services very often. Unfortunately, it has been shown that it is easy to disturb the functionality of the Internet by attacking its infrastructure taking advantage of the vulnerable Internet elements and protocols. Denial of Service (DoS) attacks and their more complicated version known as distributed DoS attacks are among the top threats to the Internet infrastructure. These attacks deny regular Internet services from being accessed by legitimate users, taking advantage of the lack of authenticity in the IP protocol, destination oriented routing, stateless nature of the Internet, and the deterministic nature of Internet protocols.

TCP continues to be the dominant as the transport protocol used by several popular Internet applications (e.g., http, email, and ftp). Recent studies show that TCP carries 95% of today's Internet traffic and 80% of the total number of flows in the Internet [9]. Therefore, it is no wonder that TCP-based DoS attacks are the most common. TCP traffic itself can be generally classified as control traffic (e.g., SYN, SYN-ACK, FIN, etc..) and data traffic. Although an attacker can flood its target by any type of traffic, the magnitude of damage caused to the target and the difficulty of attack mitigation is maximized when TCP control packets are used as the weapon of an attack. Among TCP control packets, SYN packets are the most dangerous if used by an attacker in the form of SYN flooding, because every faked SYN packet, in addition to bandwidth consumption, can disproportionately consume a system's resources for a disproportional amount of time.

In SYN flooding attacks [2], the victim is overwhelmed by very large number of spoofed SYN packets (i.e., packets that hold spoofed IP source addresses). This consumes victim's bandwidth and forces it to allocate resources for connections that will never complete. A TCP listen port has a finite number of slots in its listen queue and normally that number of slots is relatively small. When an attacker sends enough faked SYN packets, the listen queue can be fully occupied and subsequently deny any legitimate SYN packet from entering into the listen queue. Therefore, until the connection establishment process times out, a disproportional amount of system resources are occupied: a slot in the attacked port's listen queue, memory to maintain connection information, and CPU and network bandwidth to retransmit the SYN-ACK packet.

The main challenge in SYN flooding mitigation is to accurately identify attack packets and filter them without causing collateral damage to legitimate traffic designated to the victim. In this paper, we propose a scheme, called intentional dropping based filtering, that takes the TCP client's persistence behavior into account to distinguish legitimate connection requests from attack connection requests far away from the victim, leading to protection from bandwidth exhaustion attacks as well as victim's exhaustion attacks. We perform simple analysis to evaluate the proposed scheme focusing on its effectiveness in reducing attacker's effective attack rate, and showing its impact on connection establishment latency.

The rest of the paper is organized as follows: Previous work is discussed in section II. Section III describes the proposed scheme. Section IV studies the performance of the proposed scheme. Finally, conclusion and future work are presented in section V.

## II. Previous work

Previous work on mitigating SYN flooding attacks aimed at avoiding allocating resources for TCP connections before proving their legitimacy. This can be seen in schemes such as SYN cookies [12], and SYN cache [7]. SYN cookies work to alleviate SYN floods by calculating cookies that are functions of the source address, source port, destination address, destination port, and a random secret seed. When a SYN packet is received, the server calculates a SYN cookie and sends it to the requesting client as part of the SYN-ACK packet without allocating resources for that request. When ACK packet is received, the connection is established if a valid cookie is included.

1-4244-0222-0/05/$20.00 (c)2005 IEEE

*This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the Proceedings IEEE Infocom.*

With the SYN cache, when the SYN backlog queue overflows, a minimal amount of state is stored for each SYN request in a data structure known as the SYN cache, and a SYN-ACK response is sent. If a valid ACK comes back, a complete connection is created. If there is no route or a TCP RST or ICMP Unreachable comes back, the entry is deleted. Otherwise, the entries will just time out. In both cases, since state is not kept, the SYN backlog queue is not exhausted, and normal TCP communications can continue. The major problem with these approaches is that they can not alleviate bandwidth exhaustion attacks resulting from SYN flooding.

Generally, filtering of spoofed IP packets leads indirectly to TCP flooding mitigation. For example, in ingress filtering [4], routers are configured to block packets that arrive at the edge router of the source network with illegitimate source addresses. This may violate some existing setups and protocols such as Mobile IP and multi-homing. It is also difficult to convince ISP administrators to support ingress filtering because the benefit is not felt directly by the deploying ISP. Another example is the SAVE protocol [8], which is designed to provide routers with the information needed for source address validation. The main problem of this protocol is that legitimate packets may be filtered even in the absence of an attack. This is due to routing instability which leads to errors in the source address validation tables maintained at the SAVE enabled routers. In general, such schemes require large scale deployment to prevent IP source address spoofing efficiently.

## III. Intentional dropping based filtering

### A. Intentional dropping concept

Consider the client server model shown in Fig. 1. It is assumed that packets from client $C$ to server $S$ pass through router $R$. When the client issues a "connect()" command, the stack sends a SYN packet, $P$. If $R$ is overloaded, it may drop packet, $P$. We call this dropping *accidental* because it it is not under the router's control. However, if $R$ is not overloaded and yet decides to drop packet, $P$, then we call this dropping *intentional* because it is under the router's control. The effect of dropping packet, $P$, either accidentally or intentionally, is the same from the client's viewpoint. Since the server does not get the SYN packet, the client never gets the corresponding SYN-ACK packet. So the client waits for some time, $T_0$, thinking that the packet might be lost on its way, and sends a SYN packet again. This behavior is known as *client persistence*. In this paper, we assume that endpoints adhere to a TCP-Reno style congestion control mechanism [5]. Therefore, the client initiates subsequent retransmissions before aborting its connection. According to [1], the initial timeout value before the first connection attempt is $T_0 = 3$ seconds. After each retry, the amount of time to wait is doubled (i.e., the time to wait after the $i$-th retransmission is $T_i = 2^i T_0$ seconds, $i = 1, 2, ...$).

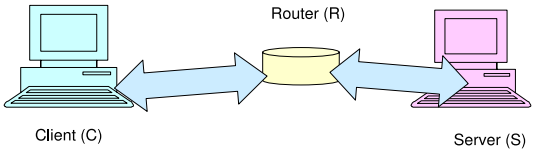


Fig. 1. Client-Server model. Bidirectional arrows represent communication channels between C, R, and S

### B. Intentional dropping based filtering

A recent study by Jamjoom et. al., [6] shows that client persistence is very widespread as it is built in TCP's congestion control. This behavior can be observed whenever network congestion occurs. For example, during flash crowd event (FCE), very large number of *legitimate* clients try to establish connections with a popular server. Theses clients keep retransmitting their SYN packets until they gain access or they give up. Also this behavior is observed during a SYN flooding attack, where very large number of *fake* SYN packets mixed with few legitimate SYN packets overload a given server. In this scenario, attackers continue to flood the server, and legitimate clients, due to their persistence behavior, keep trying to gain access to the targeted server before giving up. In both cases, we emphasize that SYN packets are the main contributing factor to network and end server congestion.

During SYN flooding attack, we apply intentional dropping on incoming SYN packets destined to the victim to prevent network congestion, and yet cause client persistence, which is a basic characteristic of legitimate clients. Typically, it is impossible to distinguish legitimate SYN packets from attack SYN packets by inspecting each packet alone. However, by taking advantage of the client persistence behavior, we can make an educated guess of whether a given connection request is malicious or legitimate. *We emphasize here that the distinction is made at the connection level rather than at the packet level.* The main idea is to *intentionally drop* the first SYN packet of each connection request. Subsequent SYN packet from a request is passed only if it adheres to the TCP's time out mechanism. Since attackers do not adhere to TCP's timeout mechanism, and in most cases they assign random addresses to their SYN packets, all SYN flooding packets are supposed to be filtered. In contrast, due to their persistence behavior, legitimate clients can be identified and allowed to pass one SYN packet per connection request.

Supporting intentional dropping requires state information to be maintained about connection requests and their arrival times. It is clear that performing this scheme at the victim itself reduces resource exhaustion without eliminating bandwidth exhaustion. However, performing it at the edge routers[1] of the ISP network that contains the victim has the advantage of eliminating bandwidth exhaustion as well as victim's resource exhaustion[2]. This conclusion is based on the assumption of *routing stability*, referred to

[1]TCP header should be accessed by edge routers.

[2]In this paper, we assume that the scheme is to be implemented at the edge routers.

as the situation of having the same ingress router for all packets sent from a given source to a destination located in the ISP network, which implies that there is a mapping between attack nodes and the edge routers of the ISP's edge routers. By this mapping we mean that each subset of attack nodes have to forward their attack packets through the same edge router during the attack period.

### C. Illustrative example

To illustrate the operation of the proposed scheme, consider the two timing diagrams shown in Fig. 2. These diagrams reflect the reaction of a legitimate client (X) and an attacker (Y) to intentional dropping. It is assumed that both of them forward their traffic to server S through edge router R. Our objective is to show how router R can distinguish legitimate request from fake request. The following two cases are considered:

- Legitimate request: At time $t_1$, X initiates a TCP connection to S. The corresponding SYN packet, $P_X^1$ arrives the router R at time $at_1 = t_1 + t_{XR}^1$, where $t_{XR}^1$ represents the time it took the packet to travel from X to R. At this instant, R drops $P_X^1$ intentionally, and records connection request information, such as: connection ID and arrival time. As a result, X repeats its request by sending another identical SYN packet, $P_X^2$, at time $t_2 = t_1 + T_0$. When the packet arrives R at time $at_2 = t_2 + t_{XR}^2$, it is considered to be valid if the following inequality holds: $T_0 - \Delta \leq (at_2 - at_1)$. Theoretically, $t_{XR}^2 = t_{XR}^1$. However, due to variability of network load, the two values are expected to be different. $\Delta$, called the delay tolerance parameter, is introduced here to tolerate this difference. The inequality requires the interarrival time between the first two SYN packets of a given connection request to be at least $T_0 - \Delta$.
- Attack request: We assume that Y attacks S by sending randomly spoofed SYN packets at a constant rate of 1 packet/second. Applying intentional dropping to incoming SYN packets ($P_A^1$, $P_B^1$, $P_C^1$, etc. with spoofed source addresses $A$, $B$, $C$, etc.), at router R, leads to filtering of all of them because each SYN packet represents a new connection in R's viewpoint. Even if subsequent SYN packets hold the same spoofed source address, they are filtered because their interarrival time is $\leq T_0 - \Delta$.

### D. SYN packet filtering algorithm

By taking client's persistence into account, connection establishment request can be viewed as a sequence of SYN packets generated repeatedly at time instants specified by TCP's congestion control mechanism until a response is received or a maximum number of retransmissions is reached. Therefore, SYN packets can be generally categorized into *classes* that reflect the client's persistence level. A class of a packet is defined as the actual or estimated number of retransmissions of that packet. For example, class 0 SYN packets represent *new* connection requests,

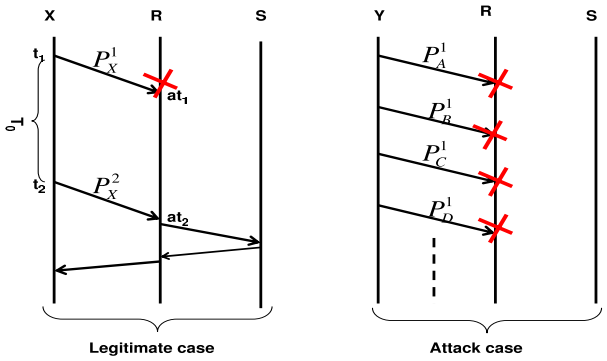


Fig. 2. Timing diagrams that show the impact of intentional dropping on connection establishment. Left: legitimate request. Right: attack requests. $P_S^i$ represents the $i$-th SYN packet with source address $S$

class 1 SYN packets represent *repeated* connection requests for the first time (i.e., first retransmissions), class 2 SYN packets represent *repeated* connection requests for the second time and, so on. Generating the $i$-th SYN packet for a particular connection, implicitly implies the loss of the previous $(i-1)$ SYN packets of that connection[3].

For security purposes, the proposed scheme can be generalized such that intentional dropping can be applied to the first $k$ consecutive SYN packets from each connection request, where $k$ is called the *dropping parameter*. In this case, subsequent SYN packet from each request (i.e., the $(k+1)$-th) is passed only if it adheres to the TCP's time out mechanism. Obviously, the value of $k$ has direct impact on the connection establishment latency as it determines how many SYN packets to drop within each connection. The effect of $k$ is discussed in section IV.

The proposed scheme is initiated by the victim upon detection of SYN flooding attack. The victim informs its ISP's edge routers (e.g., through an authentic multicast message) to activate the filtering scheme. Fig. 3 shows the SYN packet Filtering Algorithm (SFA) to be performed by each edge router of the victim's ISP. It is important to mention that this algorithm is applied only to SYN packets destined to the victim. When a SYN packet, $P$, is received, its class, $i$, is determined (step 1). Clearly, if $P$ is a new connection request, it is classified as class 0. However, if $P$ is a repeated connection request, then its class is equal to the class of the preceding SYN packet of the same connection plus one[4]. Based on the class of the packet, $i$, and the dropping parameter, $k$, a decision is taken whether to pass the packet or to drop it. If $(i < k)$ (step 2), $P$ is dropped and request information, such as connection ID, arrival time, and class are recorded[5]. if $(i = k)$ (step 3), the packet is passed given that its interarrival time is at least $2^{i-1}T_0 - \Delta$. It is to be noted that the value $2^{i-1}T_0$ represents the timeout value after retransmitting a packet of class $i-1$. The packet is dropped if it does not adhere to the timing constraints (step 4).

[3]This claim is based on the assumption that packet loss is observed on the forward direction only.

[4]We assume that connection request information is already recorded at the edge router.

[5]We do not specify the storage mechanism in this paper.

**SFA (SYN packet $P$)**

1. $i$= class of packet $P$
2. if ($i < k$)
   a. drop $P$
   b. Record ($P$.ID, $P.at$, $i$)
3. else if ($i = k$ AND $P.iat \geq 2^{i-1}T_0 - \Delta$ )
   a. Pass $P$
4. else drop $P$

Fig. 3. SYN packet filtering algorithm. $P$ represents SYN packet destined to the victim, $V$. $P.ID$ stands for the packet's ID, $P.at$ stands for packet's arrival time, $P.iat$ stands for packet's interarrival time. $k$ is the dropping parameter and $\Delta$ is the delay tolerance parameter

### *E. TCP header inspection*

The proposed scheme requires that all edge routers of the ISP network that contains the victim to inspect each packet closely enough to determine if it is a TCP SYN packet, so the mechanism can more closely analyze the packet to see if it is a legitimate SYN. This requires, at least for all TCP packets destined to the victim, examining the TCP header fields. Although modern routers [3] have started implementing this capability, most of the existing edge routers may not have this capability. So installing this mechanism at those routers would slow them down. To reduce the overhead imposed on edge routers, a lightweight algorithm similar to the one proposed in [11] could be used to distinguish TCP control packets from TCP data packets. Based on that algorithm, a router can tell TCP control packets from data packets without accessing the TCP header by checking the "total length field" in the IP header. If the total length of an IP packet is 40, then it is most probably a TCP control packet (given that its protocol type is TCP and its fragmentation offset is zero). By following this approach, among the IP packets destined to the victim, only TCP control packets undergo TCP header inspection by edge routers. Once a TCP control packet is identified, the edge router has to inspect the corresponding flags in the TCP header to determine whether it is a SYN packet or not. This implementation based modification reduces the overhead of TCP header inspection. The actual TCP header inspection can be done by the router itself or by a special device attached to it.

## IV. Performance evaluation

The proposed scheme has a dual impact. On the positive side, it reduces the attack rate significantly. On the negative side, it increases the connection establishment latency for legitimate clients. We assume that clients would favor extra latency to gain access to a web server over a complete blocking of service. We quantify the performance of the proposed scheme in terms of the following performance metrics[6]:

### *A. Attacker's effective attack rate (EAR)*

EAR is defined as the number of attacker's SYN packets that pass filtering per unit time. This should be distinguished from attacker's actual attack rate (AAR) which is defined as the number of attacker's generated SYN packets per time unit. Theoretical maximums for AAR depend on the connection type (e.g., using analog modem, ISDN, T1, the maximum AAR is 87, 200, 2343 SYNs/sec, respectively [10]). Under intentional dropping scheme, what determines the EAR is the way the attack is performed. In our analysis, we consider the following types of SYN flooding attacks:

- Type 1: Attack SYN packets are generated at the rate specified by the attacker with randomly spoofed source addresses.
- Type 2: Attacker mimics the behavior of a legitimate client. Therefore, attack SYN packets that hold the same spoofed source address are generated at time instants as if the attacker is experiencing real packet loss.
- Type 3: Attacker mimics the behavior of a legitimate client for a group of spoofed source addresses. This type is similar to type 2 except that the attacker changes the source addresses alternatively.

Currently, type 1 is the most common as attacker's are not aware of the proposed scheme. However, type 2 and type 3 are expected to appear if intentional dropping is deployed as a defense mechanism. It is easy to see that EAR is exactly zero for type 1. This is due to the fact that the first $k$ SYN packets are dropped from each connection request. In fact, it would be sufficient for $k$ to be one for the EAR to be zero. In type 2, the attacker is able to pass one SYN packet every $\sum_{j=0}^{k-1} 2^j T_0$ seconds. Therefore,

$$EAR = \frac{1}{\sum_{j=0}^{k-1} 2^j T_0} packets/second \qquad (1)$$

This explains why we introduced the dropping parameter $k$. Fig. 4 (a), supports our intuition that intentional dropping scheme results in drastically low EAR for type 2 attacks[7]. It also shows the effect of the dropping parameter, $k$, on EAR; $k$ denotes the number of SYN packets that are intentionally dropped, it is varied from 1 to 4 as most TCP implementations abort connection establishment after 4 to 6 failures. As expected, larger $k$ results in lower EAR, because a single SYN packet of a given connection request is passed on the $k_{th}$ attempt. In its current form, intentional dropping scheme does not mitigate type 3 attacks effectively. Due to space limitation, we do not discuss the extensions to the proposed scheme that would be effective against type 3 attacks.

### *B. Connection establishment latency increase (CELI)*

Originally, in addition to the round trip time, connection establishment latency is due to SYN packet loss in the forward direction (i.e., client to server), SYN-ACK packet loss in the reverse direction (i.e., server to client). The loss in both directions is accidental as it is caused by

[6]It is to be noted that the aggregate traffic before and after applying the scheme will increase by the same amount approximately due to packet retransmissions in both cases.

[7]It is to be noted that the EAR shown in the figure is for a single attacker.

network congestion. Obviously, intentional dropping of SYN packets in the forward direction increases the connection establishment latency. It can be seen that accidental dropping affects the operation of the SFA algorithm in the sense that SYN packet's class (defined in subsection III-D) can be inaccurate if SYN packets experience congestion loss. For example, if the second SYN packet of a given connection request is lost before reaching the intentional dropping filter, the sender will retransmit the packet after $2T_0$ seconds. The retransmitted packet is the third from sender's view point. However, when it reaches the intentional dropping filter it will be wrongly classified as the second packet. To avoid this confusion, we modify the SFA to become aware of congestion loss. The only modification required is to determine packet's class, $i$, based on the interarrival time, $T$, of successive SYN packets for the same connection attempt. Theoretically,

$$T = 2^{i-1}T_0 \tag{2}$$

Therefore,

$$i = 1 + \log_2(\frac{T}{T_0}) \tag{3}$$

However, since $T$ is not going to be exactly $2^{i-1}T_0$, the fraction $\frac{T}{T_0}$ must be rounded to the closest exponent of 2 before substituting in equation 3. It is to be noted that this analysis does not take into consideration consecutive packet loss from the same connection.

The maximum connection establishment latency increase, $CELI_{max}$, happens when there is no congestion loss for the first $k$ SYN packets of a given connection request. Therefore,

$$CELI_{max} = \sum_{j=0}^{k-1} 2^{j-1}T_0 seconds \tag{4}$$

Fig. 4 (b) shows the value of $CELI_{max}$ that correspond to $k$ intentional droppings of SYN packets that belong to certain connection request. It can be seen that as $k$ increases, $CELI_{max}$ increases drastically. This increase, in general, is within acceptable range (3 to 27 seconds) that are typically tolerable by legitimate clients. It is also important to observe the tradeoff introduced by $k$ between $EAR$ and $CELI_{max}$.

## V. Conclusions and future work

SYN flooding attacks are among the most difficult DoS attacks, since every faked SYN packet, in addition to bandwidth consumption, can disproportionately consume a system's resources for a disproportional amount of time. This paper proposed a novel scheme to mitigate these attacks. The scheme, called intentional dropping based filtering, enables edge routers of a given ISP to distinguish between legitimate and fake connection requests taking into account the persistent behavior of legitimate clients. The main idea is to intentionally drop the first SYN packet of each connection request. Subsequent SYN packet from a request is passed only if it adheres to the TCP's timeout mechanism. We showed that the proposed scheme

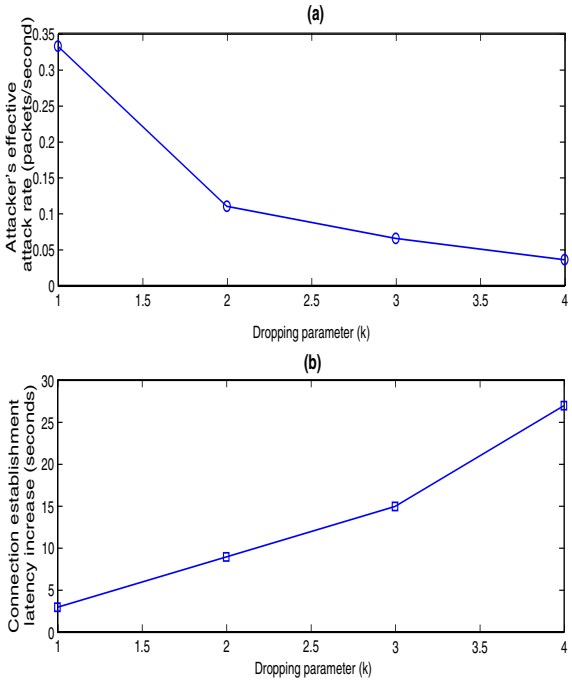


Fig. 4. (a) Effect of the dropping parameter, $k$, on the attacker's effective attack rate. (b) Effect of the dropping parameter, $k$, on the maximum connection establishment latency increase

reduces the effective attack rate significantly. However, it imposes additional delay on connection establishment. Several issues need to be addressed in the future. Our focus will be on state maintenance at edge routers to support the proposed scheme, improvements in the scheme to mitigate type 3 attacks and their analysis, and adaptive activation and termination of the scheme by each edge router independently.

## References


[1] R. Braden, "Requirements for Internet Hosts- Communication Layers," RFC 1122, October 1989.

[2] CERT Advisory CA-1996-21, "TCP SYN Flooding and IP Spoofing Attacks," available at http://www.cert.org/advisories/CA-1996-21.html

[3] Cisco systems, "Configuring TCP Intercept(Prevent Denial-of-Service Attacks)," available at: http://www.cisco.com/univercd/cc/td/doc/product/software/ios113ed/113ed_cr/secur_c/scprt3/scdenial.htm

[4] P. Ferguson and D. Senie, "Network ingress filtering: Defeating denial-of-service attacks which employ IP source address spoofing," RFC 2827, 2000.

[5] V. Jacobson, "Congestion Avoidance and Control," in *Proc. ACM SIGCOMM 1988,* August 1988.

[6] H. Jamjoom and K. G. Shin, "Persistent Dropping: An Efficient Control of Traffic Aggregates," In *Proc. of SIGCOMM'03*, Karlsruhe, Germany, August 25-29, 2003.

[7] J. Lemon,"Resisting SYN flooding DoS attacks with a SYN cache," in *Proc. USENIX BSDCon2002*, pp.8998, February 2002.

[8] J. Li, J. Mirkovic, M. Wang, P. Reiher, and L. Zhang., "SAVE: Source address validity enforcement protocol," in *Proc. of IEEE INFOCOMM 2002*, April, 2002.

[9] M. Mellia, I. Stoica, and H. Zhang, "TCP Model for Short Lived Flows," in *Proc. IEEE Communication Letters*, Feb. 2002.

[10] R. Oliver,"Countering SYN Flood Denial-of-Service (DoS) Attacks," invited talk at *The* $10_{th}$ *USENIX Security Symposium,* Washington, D.C. August, 2001

[11] H. Wang and K. G. Shin, "Transport-Aware IP Routers: A Built-In Protection Mechanism to Counter DDoS Attacks," in *Proc. of IEEE Transactions on Paralle and Distributed Systems,* VOL. 14, NO. 9, SEP. 2003.

[12] A. Zuquete, "Improving the functionality of SYN cookies," in *Proc. of 6th IFIP Communications and Multimedia Security Conference,* pp.5777, September 2002.