2007

Stateful DDoS attacks and targeted filtering

Shigang Chen
University of Florida, Department of Computer Science and Engineerign, sgchen@cise.ufl.edu

Yong Tang
University of Florida, Department of Computer Science and Engineerign, yt1@cise.ufl.edu

Wenliang Du
Syracuse University, Department of Electrical Engineering and Computer Science, wedu@ecs.syr.edu

Stateful DDoS Attacks and Targeted Filtering

Shigang Chen† Yong Tang† Wenliang Du‡

† Dept. of Computer & Information Science & Engineering, University of Florida
{sgchen, yt1}@cise.ufl.edu
‡ Dept. of Electrical Engineering and Computer Science, Syracuse University
wedu@ecs.syr.edu

Abstract

The goal of a DDoS (distributed denial of service) attack is to completely tie up certain resources so that legitimate users are not able to access a service. It has long been an open security problem of the Internet. In this paper, we identify a class of stateful DDoS attacks that defeat the existing cookie-based solutions. To counter these attacks, we propose a new defense mechanism, called *targeted filtering*, which establishes filters at a firewall and automatically converges the filters to the flooding sources while leaving the rest of the Internet unblocked. We prove the correctness of the proposed defense mechanism, evaluate its efficiency by analysis and simulations, and establish its worst-case performance bounds in response to stateful DDoS attacks. We have also implemented a Linux-based prototype with experimental results that demonstrate the effectiveness of targeted filtering.

Keywords: Distributed Denial of Service, Network Security, Stateful Attacks

I. INTRODUCTION

The open design of the Internet is fundamental to its phenomenal success, but the universal accessibility, anonymous nature, and complexity also make the public networks subject to various network-based attacks. DDoS (Distributed Denial of Service) is among the most-threatening Internet security problems. There are a variety of DDoS attacks [1]. A few examples are the smurf attack [2], the SYN flooding attack [3], and the UDP flooding attack [4]. Besides the one-packet kills, most DoS attacks flood the servers with an overwhelming number of packets, which tie up the limited resources and prevent the servers from performing their normal functions. To make the problem worse, it is often difficult to distinguish the attacking traffic from the normal traffic. When the server drops the excessive incoming packets, the legitimate packets are dropped as well.

Moore, Voelker, and Savage's work demonstrated that DoS attacks was widespread in the Internet. By using a novel traffic-monitoring technique, called "backscatter analysis", they observed 12,805 attacks against over 5000 distinct Internet hosts belonging to more than 2000 organizations during a three-week period [5]. In February, 2000, Yahoo web servers were temporarily brought down by a brute-force DDoS attack, and during the following week, CNN, Ebay, Amazon, and other popular web sites were attacked. If these sites could be shut down, few could claim immune to DoS attacks, which continues to be true today [4]. Another source of DDoS is through Internet worms, which can direct tens of thousands of compromised machines to flood a server. In addition, the worm scan activities can cause widespread congestion, leading to *de facto* DoS attacks against network communication. In 2001, the CodeRedv2 worm was programmed to launch a DoS attack against the White House web server from the twentieth through the twenty-seventh of each month. In yet another example, the W32.Blaster worm [6] of 2003 was scheduled to attack the windowsupdate.com server for certain days of each month.

Much research against DDoS attacks has been carried out in recent years. Defense mechanisms are designed to be implemented on routers [7], [8], [9], [10], [11], [12], servers [13], [14], or both servers and clients [15], [16], [17], [18]. They will be surveyed shortly. Most of them require Internet-wide deployment, which has not been and may never be achieved. On the other hand, the cookie-based defense mechanisms [13], [19], [14] require the modification only to the local server, and therefore can be readily deployed, which makes them particularly useful. The purpose of using cookies is to prevent source-address spoofing, but they also have vulnerability. In this paper we point out that all cookie-based approaches are subject to *stateful DDoS attacks*, with the attackers residing on the routing paths from the server to the forged addresses and thus able to intercept the cookies. Furthermore, as a generalization, any anti-DoS mechanism based on restricting the source address space that can be forged [7], [8], [9] is subject to a similar problem.

We then propose a new defense mechanism, called *targeted filtering*, that blocks out stateful DoS attacks. Targeted filtering is complementary to the cookie-based defense or other mechanisms that restrict the forged address space. It is locally deployed at a firewall protecting a server behind. The basic idea is to use the attack packets themselves to

The correspondence author is Dr. Shigang Chen at sgchen@cise.ufl.edu, 1 352 392 2713.

drive a master blocking list to converge towards the forged address space and block out packets from there, which allows the rest of the Internet to access the server. We design a fast algorithm for updating the master blocking list in order to support on-line operations. We analyze the properties of the system and establish the worst-case bounds on the convergence time, i.e., how long it takes the firewall to block out the attack traffic after the attack starts. The effectiveness and the performance of targeted filtering are studied by extensive simulations that cover various combinations of system configurations and attack configurations. We also implemented a targeted-filtering prototype in the Linux kernel space and performed experiments.

The rest of the paper is organized as follows. Section II surveys the related work. Section III describes the stateful DDoS attacks. Section IV presents our new targeted-filtering defense mechanism. Section V discusses the simulation results. Section VI describes a prototype and presents experimental results. Section VII draws the conclusion.

II. Related Work

DoS attacks have received considerable attention in the research community. Much work has been done on the SYN flooding attack [13], where a server's listen queue is kept full by faked connection requests (SYN requests), which causes the rejection of legitimate requests. The SYN-cookie solution allows new connections to be established even when the listen queue is full [13]. Without using the listen queue, the server encodes the state information of a half-opened connection in the sequence-number field (*cookie*) of the SYN/ACK response, and later recovers the information from the ACK packet. The drawback is that not all TCP options (maximum segment size, window scale factor, etc.) can be encoded in the cookie. The recent work by Xu and Lee embedded cookies in the http redirection messages; this approach does not require any modification of the TCP implementation on the web servers [14]. The SYN cache solution minimizes the amount of state information for a half-opened connection at the server side [20]. It alleviates but not eliminates the SYN flooding problem. The client-puzzle solution requires the clients to solve cryptographic puzzles before making TCP connections [15], [16], [17], [18]. To achieve high-level security, it incurs significant computation overhead to the clients, which can be undesirable for certain applications, especially when mobile devices are involved.

Attackers use forged source addresses to conceal their identities. Much work against DoS is on anti-address-spoofing. Ferguson and Senie proposed *ingress filtering* [7], which requires the routers of stub networks to inspect outbound packets and discard those packets whose source addresses do not belong to the stub networks. Park and Lee pioneered with the concept of *route-based distributed packet filtering* [8]. The idea is for a router to drop a packet if the packet is received from a link that is not on any routing path from the packet's source to the packet's destination. The paper demonstrated that a partial deployment (on 18% of Internet AS's) can effectively prevent spoofed IP packets from reaching their victims. Wang, Zhang and Shin presented a simple and effective mechanism, called *SYN-dog*, to identify SYN flooding sources [9]. It is a software agent installed at leaf routers connecting to stub networks. The agent detects SYN flooding from the attached networks by monitoring the differences between outbound SYN packets and inbound SYN/ACK packets. The major problem of the above approaches is that their effectiveness of preventing DoS comes only after the filters or the software are widely deployed across the Internet. An Internet-wide deployment can be difficult to achieve due to political, financial, and administrative reasons, or different technology preferences.

In recent years there has been a flourish of research work on *IP traceback* based on packet audit or route inference, whose goal is to find the origins of packets that have spoofed source addresses [21], [22], [23], [24], [25], [10], [26], [27], [28]. IP traceback is a reactive approach, which does not prevent spoofed packets from harming their victims. Sung and Xu used IP traceback to identify the network links that carry attack traffic and then preferentially filter out packets that are inscribed with the marks of those links [11]. Yaar, Perrig, and Song proposed *path identifier (Pi)*, a novel approach that assigns the same mark to packets traversing the same path and different marks to packets traversing different paths. Because the attack packets from the same source always carry the same mark, the victim is able to filter out those packets based on the mark. The effectiveness of these approaches also require wide deployment in order for most legitimate traffic to be marked differently from the attack traffic.

Mahajan et al. proposed aggregate-based congestion control (ACC) to rate-limit the identified attack traffic [12]. A congested router starts with local rate limit, and progressively pushes the rate limit to some neighbor routers and further out, forming a dynamic rate-limit tree. Routers in the tree perform filtering based on their shares of the rate limit. Chen and Song designed a class of *perimeter-based defense* mechanisms, which allows Internet service providers (ISP) to protect the communication between its customers against DoS attacks [29].

Keromytis, Misra, and Rubenstein proposed a novel architecture called *Secure Overlay Services* (SOS) [30], which proactively prevents DoS attacks. It is designed for emergency services. A certificate for accessing a protected server must be issued to each authorized client. Client requests are first authenticated and then routed via a Chord overlay

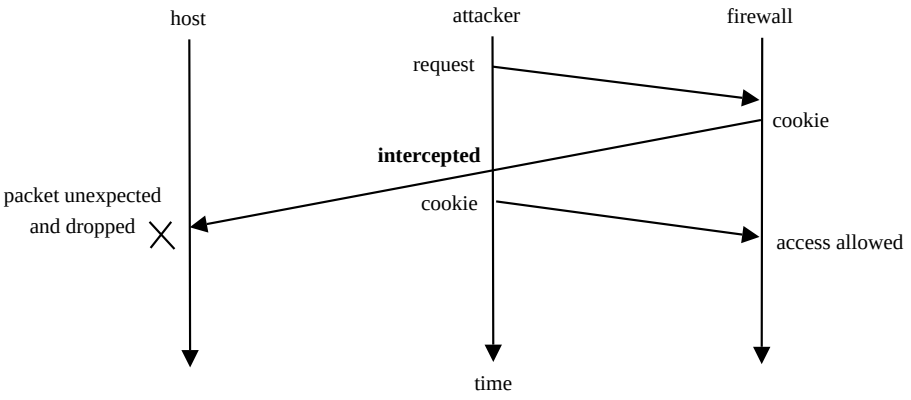

Fig. 1. Stateful DoS, with the attacker intercepting the cookie

network [31] to one of the servlets, which forward the requests to the target site. The defense against DoS relies on client authentication and the secrecy of the servlets' locations. Mayday [32] by Andersen is a generalization of SOS. It studies a variety of choices for authentication, routing, and filtering.

III. Stateful DDoS Attacks

A successful DoS attack achieves two objectives: overpowering the victim and concealing the attacker's identity. To overpower the victim, the attacker needs a strategy that small resource consumption at the attacker side causes much larger resource consumption at the victim side. For example, a small packet generated by the attacker causes a buffer space to be held for an extended period of time T at the victim. While the attacker can generate a large number of packets during T, the buffer space at the victim is going to overflow, which underlines the SYN flooding attack and the connection table overflow attack. To conceal the attacker's identity, forged source addresses must be used in the packets sent from the attacker.

The problem of address-spoofing can be partially solved by cookie exchange [19], [13], [14], which forces an attacker (or a compromised host) to use its real address in order to receive and return a cryptographic cookie before accessing any resource. The cookie approach fails under a stateful DDoS attack. As shown in Figure 1, an attacker keeps state information about each forged connection request and sniffs the network traffic for the responding cookie from the server, which allows it to complete the exchange even though forged source addresses are used. In order to perform a stateful DoS attack, an attacker (or zombie) has to reside on the routing path from the server to those forged addresses, which limits the address space that can be forged. Specifically, the attacker can forge source addresses belonging to the same LAN or downstream LANs. The combined address space that can be forged by all attackers and their zombies in a stateful DDoS attack is called the *attack address space* and denoted as A. The attackers may also simply use their real addresses or those of Zombies to launch DDoS attacks. This is a special case with A being the attackers' addresses.

All cookie-based approaches (such as SYN cookie [13] and http redirect cookie [14]) that only modify the server side are subject to the stateful DDoS attacks. The client-side address restriction approaches (such as ingress filtering [7] and route-based packet filtering [8]) are subject to a similar problem where the attackers are still able to forge source addresses from a restricted address space. We design a new defense mechanism to handle both.

IV. Targeted Filtering

Assume an existing defense mechanism has been implemented to restrict the attack address space A. For example, either a cookie-based approach is implemented on the server side or an address restriction approach is implemented on the client side. This section presents *targeted filtering*, which is deployed on a firewall to mitigate stateful DDoS attacks by automatically identifying and blocking the client requests from A, while letting the requests from the rest of the Internet go through. The system architecture is shown in Figure 2.

We will first describe the key element of targeted filtering, which is the maintenance of a master blocking list. We will then derive the upper bound on how fast our defense system can block out the attack packets, which is followed by optimization and algorithmic issues.

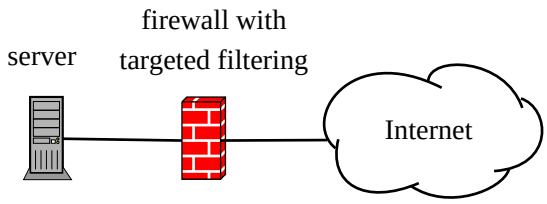

Fig. 2. System architecture

A. Master Blocking List

When a server is under a DDoS attack, an anti-DoS request is sent to the firewall, including the description of the service under attack and a rate limit for the client requests. The anti-DoS request may be generated by the management console, by the server, or by the firewall itself based on pre-configured policies. The firewall samples the incoming requests (e.g., SYN packets in case of SYN flooding attack), extracts the source addresses, inserts them to a *master blocking list* (MBL), and blocks the requests from those addresses. Note that the firewall should only sample among the requests that have passed MBL. Once a source address is in MBL, all requests from that address are blocked and thus not sampled. Therefore, any address may be inserted to MBL only once. The expansion of MBL continues until the rate of requests is reduced under the desired limit.

MBL cannot exceed a maximum number of entries, denoted as λ. If the maximum number of entries have been reached, an additional address x to be blocked will be merged with the existing entries M by the following **basic algorithm**: Find two entries in $M \cup \{x\}$ that share the longest common prefix and replace the two entries by the common prefix (which represents a network).

Although the above algorithm works very well in our simulations, it takes $O(\frac{|A|}{r})$ time in the worst case to converge on A and stop the attack, where $|A|$ is the size of the attack address space and r is the sampling rate. In order to improve the worst-case performance, we design a **revised algorithm**: Let c be a constant greater than one. Find two entries in $M \cup \{x\}$ that share the longest common prefix, denoted as P. Remove zero or more least-significant bits from P until the size of P is at least the combined size of the two entries multiplied by c. Replace the two entries by P.

With targeted filtering, the attacking traffic itself drives MBL to converge to those network addresses that stateful DDoS utilizes. If a DDoS attack is launched from a large number of compromised machines using their real addresses, targeted filtering defeats the attack in the same way: MBL converges to these addresses and blocks them out.

It is possible that the IP addresses of some normal users will be mistakenly placed in MBL. We will address this issue shortly.

B. Convergence Time

We analyze the *convergence time* of targeted filtering, which is the time it takes to stabilize MBL and block out all attack addresses. In particular, we show that the convergence time is $O(\sum_{a \in A} \frac{\lambda(\log_c |a|+1)}{(1-\alpha)r})$, which is proportional to the size λ of MBL, inverse proportional to the sampling rate r, and proportional to the summation of the logarithm over the sizes of all network addresses a in A.

The size of a network address (prefix) is denoted as $|a|$. Suppose a has l bits. Then $|a| = 2^{32-l}$. If a belongs to the attack address space A, after an entry in MBL (or multiple entries combined) grows big enough to cover a, all requests from a will be blocked and will not be sampled.

Lemma 1: Let a $(\in A)$ be a network address from which the attackers can forge addresses. After $\lambda(\log_c |a| + 1)$ different addresses from a are inserted into MBL, there must exist one entry in MBL that fully contains a.

Proof: Because there are only λ entries in MBL, after $\lambda(\log_c |a|+1)$ different addresses in a are inserted into MBL, there must exist one entry of MBL in which at least $(\log_c |a| + 1)$ addresses from a are merged. The size of the entry grows at least by a factor of c (> 1) after each merge. Therefore, it becomes no less than $c^{(\log_c |a|+1)-1} = |a|$. Both this entry and a are network addresses. Due to the construction of address space in IPv4, if two network addresses

share at least one address, one of them must fully contain the other. Since the MBL entry has the same size as a or bigger, it must fully contain a. □

Lemma 2: Let $a\ (\in A)$ be a network address from which the attackers can forge addresses. The number of addresses from a that are inserted to MBL can not exceed $\lambda(\log_c |a| + 1)$.

Proof: By Lemma 1, after $\lambda(\log_c |a| + 1)$ addresses from a are inserted to MBL, there exists one entry in MBL that fully contains a. This entry blocks all requests from a, and thus no subsequent request from a will be sampled. □

Let $\alpha\ (< 1)$ be the *server's utilization*, defined as the arrival rate of legitimate requests divided by the server's capacity.[1]

Theorem 1: Suppose A consists of a single network address. The convergence time of the revised algorithm is no more than

$$\frac{\lambda(\log_c |A| + 1)}{(1-\alpha)r}$$

Proof: For the attack to persist, the rate of attack requests passing through MBL must be no less than $(1-\alpha)$ of the server's capacity. Hence, the rate of attack requests being sampled is no less than $(1-\alpha)r$. Once a request is sampled, all other requests from the same address are blocked. Hence, all sampled requests have different source addresses. Suppose after T units of time MBL covers A and thus stops the attack. During T there are at least $m = (1-\alpha)rT$ addresses from A that are inserted into MBL. Because A consists of a single network address, by Lemma 2, m must not be greater than $\lambda(\log_c |A| + 1)$. Hence,

$$T \leq \frac{\lambda(\log_c |A| + 1)}{(1-\alpha)r}$$

□

Theorem 2: Suppose A consists of multiple network addresses. Let P be the common prefix of all network addresses in A. The convergence time of the revised algorithm is no more than

$$\min\{\frac{\lambda(\log_c |P| + 1)}{(1-\alpha)r}, \sum_{a\in A} \frac{\lambda(\log_c |a| + 1)}{(1-\alpha)r}\}$$

Proof: If the attackers could forge any address in P, by Theorem 1, the convergence time would be no more than $\frac{\lambda(\log_c |P|+1)}{(1-\alpha)r}$. Now since the attackers can only forge addresses from A, which is a subset of P, it follows that the worst-case convergence time will not exceed $\frac{\lambda(\log_c |P|+1)}{(1-\alpha)r}$.

Next we prove the convergence time is no more than $\sum_{a\in A} \frac{\lambda(\log_c |a|+1)}{(1-\alpha)r}$. For each network prefix $a \in A$, by Lemma 2, the number of addresses from a that are inserted to MBL can not exceed $\lambda(\log_c |a| + 1)$. Hence, the total number of addresses from A that are inserted to MBL can not exceed $\sum_{a\in A} \lambda(\log_c |a| + 1)$.

For the attack to persist, the rate of attack requests passing through MBL must be no less than $(1-\alpha)$ of the server's capacity. Hence, the rate of attack requests being sampled is no less than $(1-\alpha)r$. Once a request is sampled, all other requests from the same address are blocked. Hence, all sampled requests have different source addresses. Suppose after T units of time MBL covers A and thus stops the attack. During T there are at least $(1-\alpha)rT$ addresses from A that are inserted into MBL. Hence, we have

$$(1-\alpha)rT \leq \sum_{a\in A} \lambda(\log_c |a| + 1)$$

$$T \leq \sum_{a\in A} \frac{\lambda(\log_c |a| + 1)}{(1-\alpha)r}$$

□

Our simulations show that the average convergence time is much smaller than the upper bound given by Theorem 2. Even with a small c such as 1.25, MBL converges quickly to A, while the percentage of legitimate clients being mistakenly blocked is negligibly small (less than 1%).

[1] $\alpha < 1$ is required by Theorems 1 and 2 but not by the approach of targeted filtering. During temporary legitimate-traffic overload, targeted filtering can be triggered to block out a portion of legitimate traffic temporarily.

C. Optimization

The revised algorithm may expand an entry too large so that addresses adjacent to A are also blocked. To alleviate the problem, an optimization can be done as follows: once an entry reaches certain size s (e.g., Class B network), further merges involving this entry do not require a minimal growth factor of c, namely, the basic algorithm will be used for merging s with any other entry.

If A does not contain any network address bigger than s, it is obvious that the previous theorems still hold. Otherwise, we must use the following theorem.

Theorem 3: The convergence time of the revised algorithm with optimization is no more than

$$\sum_{x \in A} \frac{\lambda(\log_c |x| + 1)}{(1-\alpha)r}$$

where all network addresses in A whose sizes are larger than s are divided into subnets x whose sizes are not greater than s.

Theorem 3 can be trivially proved, in a way similar to the proof of Theorem 2. Another obvious optimization is to check whether one entry contains the other before the two are merged. If so, we simply remove the smaller one.

D. Periodic Updates of MBL

The MBL may take addresses from legitimate requests that happen to be sampled. Consequently some legitimate clients are blocked. To prevent legitimate clients from being blocked indefinitely, if one entry in MBL is not matched by any arrival request for a sufficiently long period of time (or the number of matched requests is under a small threshold), the entry is removed.

Assume the attack traffic from A has a higher density (number of requests per address) than the legitimate traffic from the entire Internet. We can use periodic update of MBL to reduce the chance of legitimate clients being inserted into MBL. In this approach, the sampled addresses are not merged into MBL immediately. Instead, they are placed in a temporary blocking list. Any requests that match either MBL or the temporary blocking list will be dropped. When the temporary blocking list is full, a new sample will replace an existing entry that has blocked the least number of requests. If there is a tie, the older entry is replaced. With such selection bias, the temporary blocking list is in favor of retaining the attack addresses from A due to the higher request density.

The firewall periodically merges the temporary blocking list to MBL and then empties the list for more addresses, until the rate of unblocked requests is under the desired limit.

E. An $O((\lambda + b)\log(\lambda + b))$ Algorithm for Updating MBL

We present an algorithm that merges the temporary blocking list with MBL in time $O((\lambda + b)\log(\lambda + b))$. It is an efficient implementation of the revised algorithm.

Let M be the master blocking list and B be the temporary blocking list. Let λ be the number of entries in M, b the number of entries in B, and c a constant greater than one. $|P| = 2^{32-l}$ for an address prefix P with l bits. The following algorithm merges B into M.

```
UpdateMBL(M, B)
(1)  M' ← sort(M, B)
(2)  compute common prefixes between adjacent entries in M'
(3)  Com ← the set of b longest common prefixes
(4)  for each P ∈ Com do
(5)      let E1 and E2 be the two entries sharing P
(6)      if E1 (or E2) is a prefix of E2 (or E1) then
(7)          remove E2 (or E1)
(8)      else
(9)          while (|P| ≤ (|E1| + |E2|) × c)
(10)             remove the least-significant bit of P
(11)         replace E1 and E2 by P
(12) M ← M'
```

The time complexity of Step 1 is $O((\lambda + b)\log(\lambda + b))$. The complexity of Step 2 is $O(\lambda + b)$. Step 3 can be done by sorting the common prefixes with a complexity of $O((\lambda + b)\log(\lambda + b))$. The complexity of Lines 5-11 is constant.

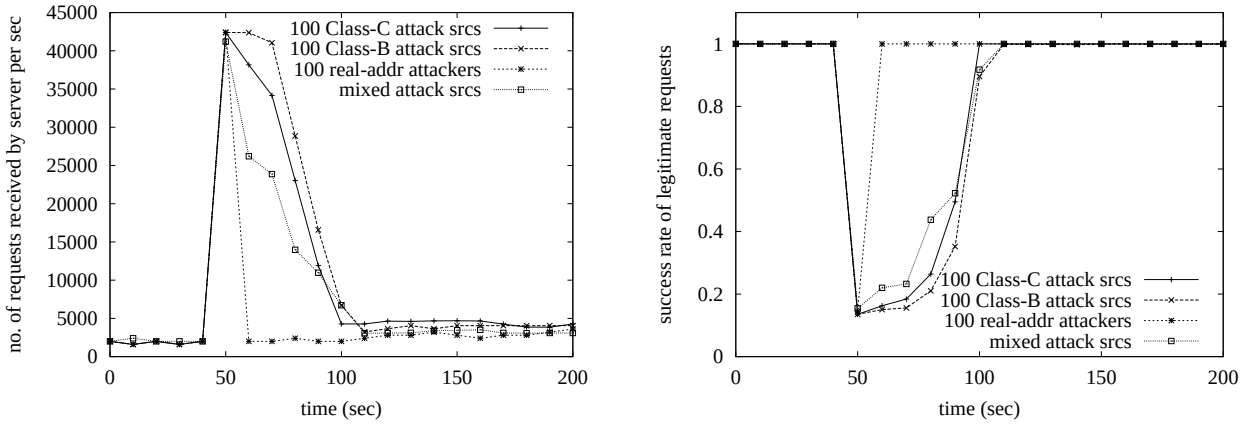

Fig. 3. Defend against DDoS by targeted filtering

Without losing generality, suppose E_2 is removed at Line 7. E_1 becomes adjacent to E_2's other neighboring entry in M' and inherits the common prefix of E_2 with that entry. After Line 11, P inherits the common prefixes of E_1 and E_2 with their other adjacent entries in M'. The inherited common prefixes may be too long, but the future execution of Lines 9-10 will enlarge those common prefixes to cover E_1 or P entirely. The complexity of Lines 4-11 is $O(b)$. The total complexity of the algorithm is thus $O((\lambda + b)\log(\lambda + b))$.

We will use this algorithm in our simulations and demonstrate that the overhead of executing the algorithm is very small.

V. Simulations

We use simulations to study how well the targeted-filtering mechanism performs against stateful DDoS attacks. Unless explicitly specified otherwise, the simulation parameters take the following default values. The normal workload of a server is 2,000 requests per second on average, with $\pm 20\%$ fluctuation over each second. The server's capacity is 6,000 requests per second[2]; it triggers the firewall to perform targeted filtering when the workload exceeds the capacity.[3] The maximum number of entries in MBL (master blocking list) is 5,000, and it is updated once every 10 seconds with the firewall randomly selecting 3,000 unblocked requests and inserting their source addresses to the temporary blocking list, which is then merged with MBL. There are 100 attack sources who coordinate a DDoS attack on the server. Each source sends 400 requests per second on average. An attack source represents all attach hosts on the same LAN, which have the same forged address space.

A. Performance of Targeted Filtering

Figure 3 shows the results of four simulation runs. In the first run (the curve of "100 Class-C attack srcs"), the attack sources perform a stateful DDoS attack, and each is able to forge addresses from a Class C network. In the second run ("100 Class-B attack srcs"), each source is able to forge addresses from a class B network. Note that an attack source represents all (tens or even hundreds) Zombie hosts from the network it forges addresses. In the third run ("100 real-addr attackers"), the attackers use their real addresses. In the fourth run ("mixed attack srcs"), 33 attack sources are able to forge addresses from Class C networks, 33 forge from Class B networks, and 33 use the real addresses. As shown in the left-hand graph, the number of requests surged after the attack began. After the firewall activated targeted filtering, the attack packets themselves drove MBL to block them out. As more and more faked requests were blocked, the service was restored for legitimate users. MBL stabilized when the rate of requests passing the firewall no longer exceeded the server's capacity. The right-hand graph shows that the percentage of legitimate requests that were successfully processed by the server dropped sharply after the attack began, but it recovered quickly to almost 100% after the firewall filtered most faked requests.

[2] Suppose each connection lasts for 10 seconds with 10 kbps throughput on average. The server's total throughput will be around 600 Mbps.

[3] In a normal overload, targeted filtering will also bring the workload under 6,000 requests per second. A temporarily-blocked legitimate user will be removed from the master blocking list once he stops accessing the server for a period of time.

B. Convergence Time with Respect to Firewall Configuration

The convergence time measures how long it takes for MBL to stabilize, i.e., from when the attack starts to when the traffic is reduced within the server's capacity. For example, the convergence time for "100 Class-C Attack srcs" in Figure 3 is 50 seconds (from 50 to 100 on the time axis). In the following we study how the firewall configuration affects the convergence time. Since the case of real-addr attackers is trivial, our explanation will focus on the other three cases.

The number of addresses inserted into MBL per update period equals the size of the temporary blocking list, denoted as b. The upper graph of Figure 4 shows that the average convergence time decreases as b increases. The decrement is non-linear. The curve demonstrates that, for the best performance/overhead tradeoff,[4] b should not be too large — around 3,000 achieves a favorable tradeoff. The middle graph shows that the convergence time increases linearly as the update period increases. The lower graph shows that the convergence time increases as the size of MBL increases. That is because, in a larger MBL, the addresses are closer to each other. Hence more address merges are needed to create network prefixes that are large enough to cover the attack address space, which implies more updates and thus longer convergence time. Another observation is that the increase in the convergence time is considerably faster when there are Class-B attack sources, but the exact number of Class-B sources does not matter much.

The update period is an ideal system parameter that can be tuned to control the convergence time because of their linear relation. While fixing the other system parameters, we can adjust the update period during run time to achieve a desirable convergence time. For example, we may cut the update period by half each time when the system does not converge after a threshold time period.

For all simulations presented in Figure 4, the success rate of legitimate requests (i.e., the percentage that is not blocked) was above 99.5% after MBL stabilizes. The detailed results on success ratio seem not to reveal additional information and therefore are omitted.

C. Impact of the Size of MBL

We found that the success rate suffered only when the size of MBL was very small, as demonstrated by Figure 5. The left-hand graph shows that when the size of MBL is less than 1,000, the success rate can be very bad. The dips between size 500 and size 1,000 can be explained as follows: Recall that during the last update MBL is merged with 3,000 new addresses from the temporary blocking list. If it happens that most flooding sources have already been blocked by previous updates, then most of those 3,000 addresses are in the legitimate address space. When the size of MBL is small, each merge of two legitimate addresses may create a large network prefix blocking a major portion of legitimate traffic, which reduces the success rate. This problem is greatly alleviated when the size of MBL becomes large (the small dip at size 1,500). Therefore, in order to ensure high success rate, the size of MBL should not be less than 1,000. From the right-hand graph, we learn that the average convergence time follows a staircase curve with respect to the size of MBL. This is the natural result of the "discrete" updates of MBL, with 3,000 new addresses each update. When the size of MBL is around 500, 1,500, or 2,500, one more update becomes necessary. Note that the staircase is not shown in the bottom graph of Figure 4 due to coarser granularity of the simulation.

D. Convergence Time with Respect to Attack Traffic Chacteristics

We now study how the attack traffic characteristics affect the convergence time. The left-hand graph of Figure 6 shows that the convergence time increases when the attackers send forged requests at a faster rate. As the attack rate is higher, more forged requests will be sampled. The attack addresses will occupy a larger portion of the temporary blocking list in each update, which decreases the gaps between them. Hence, more merges (thus more updates and longer convergence time) are needed to create network prefixes large enough to cover the attack address space. On the other hand, when the attack rate becomes sufficiently large (e.g., $> 1,400$) such that the attack addresses already occupy most of the temporary blocking list, then further increasing the attack rate will not significantly increase the convergence time. The right-hand graph shows that the convergence time increases as the number of attack sources increases for the same reason explained above.

E. Mixed Attack Sources

Figure 7 presents the simulation results for a mix of Class-B/C attack sources. Generally speaking, a larger number of Class C sources causes a larger convergence time. That is because the addresses from Class C networks are closer to each other and will be merged first in MBL. Before Class C networks are completely blocked out, the addresses from

[4] The execution time per update is $O((\lambda + b)\log(\lambda + b))$. See Appendix A.

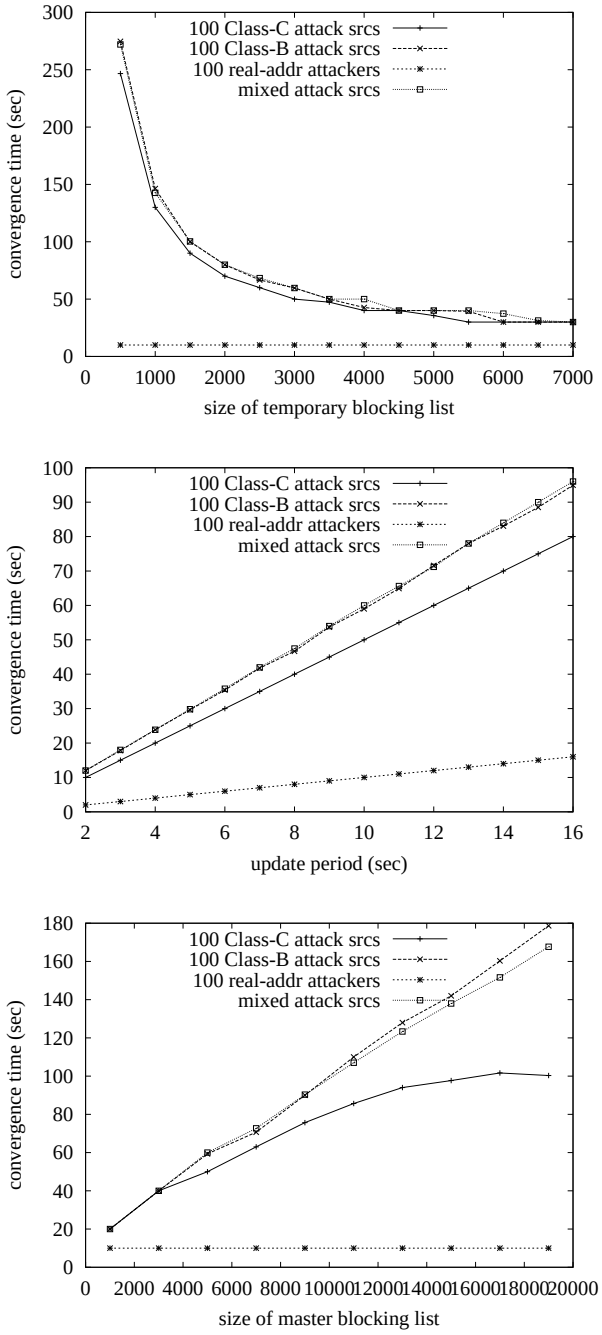

Fig. 4. Convergence time with respect to firewall configuration

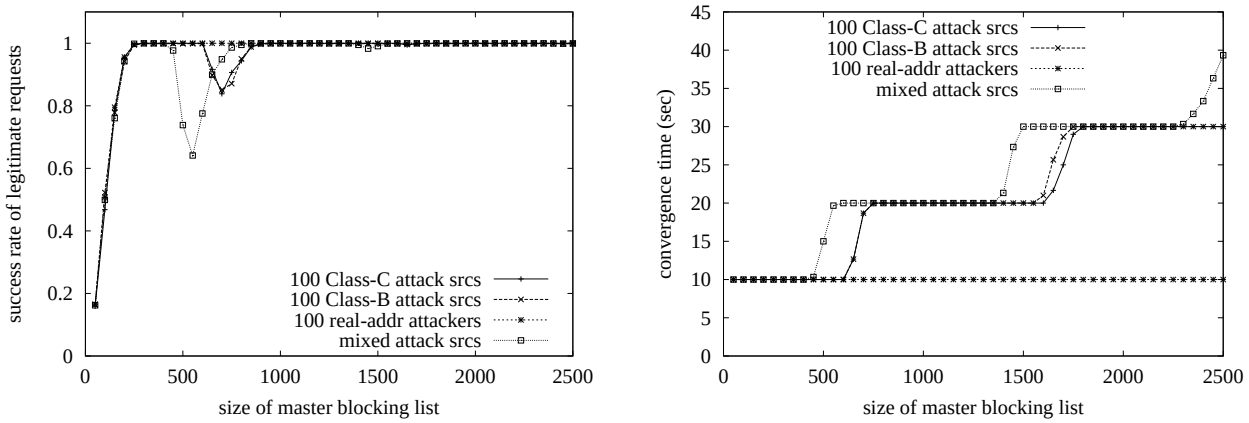

Fig. 5. Impact of different sizes of master blocking list

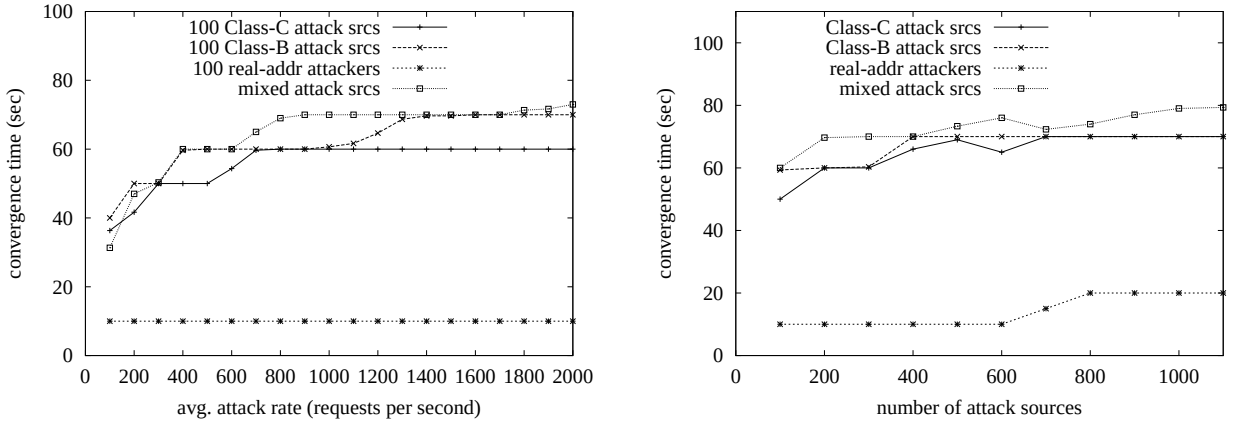

Fig. 6. Convergence time with respect to attack traffic characteristics

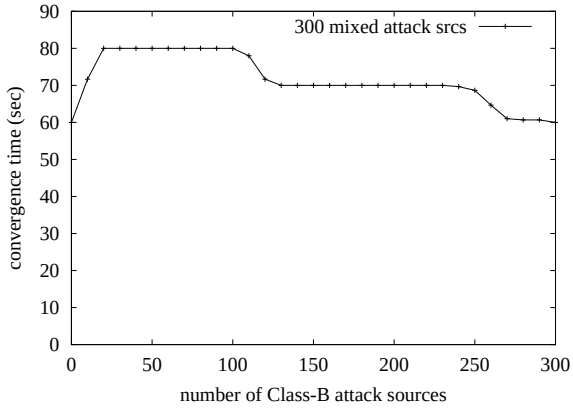

Fig. 7. Mixed attack sources

TABLE I

EXECUTION TIME (IN MILLISECONDS) PER MBL UPDATE

b	size of MBL	100 Class-C att. srcs	100 Class-B att. srcs	mixed att. srcs
1,000	1,000	1.30	1.69	0.91
1,000	2,000	1.13	2.17	1.14
1,000	3,000	3.70	3.47	3.07
1,000	4,000	6.27	5.67	4.17
1,000	5,000	7.92	8.18	5.38
2,000	2,000	2.47	3.39	2.47
2,000	3,000	5.52	3.23	2.88
2,000	4,000	6.94	6.87	4.34
2,000	5,000	9.15	8.92	6.38
2,000	6,000	12.38	11.86	8.74
4,000	4,000	9.24	10.03	7.68
4,000	5,000	12.89	12.42	9.48
4,000	6,000	16.77	15.00	11.93
4,000	7,000	21.25	19.27	13.89
4,000	8,000	25.54	23.36	16.23

Class B networks can be merged only when their gaps are comparable to those in Class C networks, which means the common prefixes after the merges are relatively small. Therefore additional merges are needed to create those Class B networks in MBL.

F. Execution Time Per MBL Update

Table I shows the execution time for inserting the temporary blocking list to MBL (Section IV-E). The insertion took less than 26 milliseconds on a DELL Inspiron 2650 with 1GHz P-IV and 256M main memory. A more powerful firewall will take less time, which makes it a practical solution, allowing a single firewall to support many servers simultaneously.

VI. PROTOTYPE AND EXPERIMENT

We have implemented targeted filtering on a Linux-based prototype firewall. It consists of three kernel modules: a targeted-filtering module, a control module, and a log module. The targeted-filtering module registers to a hook point of the IPv4 protocol via Netfilter (Linux Kernel V2.4), which allows it to inspect each passing packet, update MBL, and drop packets that match MBL. The control module is a device driver, to which a user-level process writes the control information, which is then read by the targeted-filtering module. The log module is also a device driver, to which the targeted-filtering module writes reports, which is then read by a user-level process.

The experiment testbed is shown in Figure 8. H1 is responsible for generating legitimate requests from random source addresses. H2 is responsible for generating forged requests. The experimental parameters are chosen as follows. There are 150 attack sources, among which 50 can forge addresses from Class B networks in a stateful DDoS attack, 50 can forge from Class C networks, and 50 use their real addresses. The average attack rate per source is denoted as r, which will be varied in the experiment. The server's capacity is 10,000 requests per second, and the normal load is 5,000 requests per second. The size of MBL is 1500, and the size of the temporary blocking list is 1000. The result of a typical run of the experiment is given in Table II. It shows that, as the time progresses, most legitimate requests are continuously accepted by the firewall, while more and more attack requests are blocked. The few blocked legitimate requests (e.g., when time = 56) mostly come from the attack address space, which makes them indistinguishable from the attack traffic.

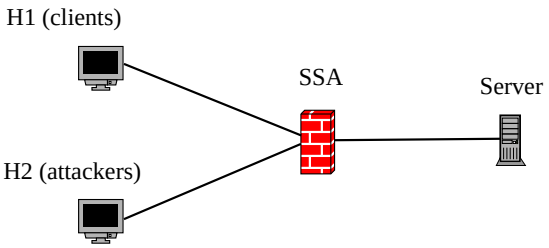

Fig. 8. Experiment testbed

TABLE II

EXPERIMENT ON TARGETED-FILTERING IMPLEMENTATION (NO. OF REQUESTS PER SECOND), $r = 100000/150$

elapsed time (sec)	arrival rate of legitimate requests at the firewall	rate of legitimate requests passing the firewall	arrival rate of forged requests at the firewall	rate of forged requests passing the firewall
0	5001	5001	0	0
2	4975	4975	0	0
4	4988	4988	93342	93342
6	5156	5156	102509	102509
8	5252	5252	109148	109148
10	4807	4807	101260	101260
12	5095	5095	93213	93213
14	4714	4714	96318	62329
16	4824	4824	103172	66773
18	5066	5066	95648	61880
20	5214	5214	106098	68610
22	4542	4542	95396	61762
24	5088	5088	107219	58026
26	4919	4919	95637	51731
28	4635	4635	103950	56381
30	5227	5227	95713	51843
32	5054	5054	101791	55207
34	4570	4569	97677	27572
36	5568	5567	96459	27287
38	5339	5338	104351	29615
40	4694	4693	95544	26951
42	5120	5120	106869	30146
44	5046	5039	95090	9579
46	4848	4839	101873	10282
48	5344	5335	100600	10079
50	5300	5292	94764	9650
52	4960	4959	108244	10847
54	5226	5217	95766	105
56	5403	5392	95494	119
58	4968	4961	109159	145
60	5224	5214	96626	141
62	4722	4715	95217	111
64	5050	5040	108695	146
66	5087	5080	96119	135

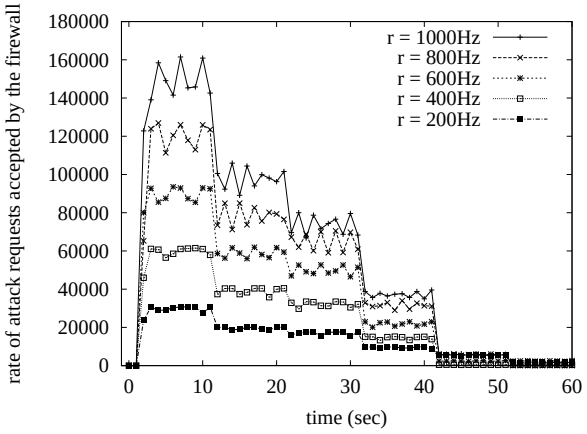

Fig. 9. Experiments with varied attack rates (r)

Figure 9 shows the results of experiment runs with different attack rates. During each experiment run, the firewall reacts quickly to the attack and selectively blocks out the attack traffic, while the legitimate traffic is virtually unaffected. The time it takes the firewall to stabilize the traffic does not vary much for different attack rates.

VII. Conclusion

We described a class of stateful DDoS attacks that defeat the existing cookie-based solutions and proposed a new defense mechanism, called targeted filtering, to block such attacks. One advantage of the new mechanism is that it can be deployed at a local firewall. We thoroughly discussed various algorithms and optimizations, and proved the worst-case convergence time with respect to a number of system/attack parameters. Both analytical and simulation results showed the effectiveness of this mechanism in defending against stateful DoS attacks. We also implemented a Linux-based prototype, demonstrating the technical feasibility of operating such a defense mechanism in real time.

References

[1] S. Northcutt and J. Novak, "Network Intrusion Detection (3rd Edition)," *New Riders Publishing, ISBN: 0735712654*, 2002.
[2] Computer Emergency Response Team, "CERT Advisory CA-1998-01 Smurf IP Denial-of-Service Attacks," *http://www.cert.org/advisories/CA-1998-01.html*, January 1998.
[3] C. Schuba, I. Krsul, M. Kuhn, E. Spafford, A. Sundaram, and D. Zamboni, "Analysis of A Denial of Service Attack on TCP," *Proc. of IEEE Symposium on Security and Privacy*, 1997.
[4] S. Gibson, "The Strange Tale of the Denial of Service Attacks Against GRC.COM," *http://grc.com/dos/grcdos.htm*, 2002.
[5] D. Moore, G. Voelker, and S. Savage, "Inferring Internet Denial of Service Activity," *Proc. of USENIX Security Symposium'2001*, August 2001.
[6] Computer Emergency Response Team, "CERT Advisory CA-2003-20 W32/Blaster worm," *http://www.cert.org/advisories/CA-2003-20.html*, August 2003.
[7] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing," *IETF, RFC 2267*, Janurary 1998.
[8] K. Park and H. Lee, "On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets," *Proc. of ACM SIGCOMM'2001*, August 2001.
[9] H. Wang, D. Zhang, and K. G. Shin, "SYN-dog: Sniffing SYN Flooding Sources," *Proc. of 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, July 2002.
[10] S. F. Wu, L. Zhang, D. Massey, and A. Mankin, "Intention-Driven ICMP Trace-Back," *IETF, Internet Draft, draft-wu-itrace-intention-00.txt (work in progress)*, Febrary 2001.
[11] M. Sung and J. Xu, "IP Traceback-based Intelligent Packet Filtering: A Novel Technique for Defending Against Internet DDoS Attacks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 9, pp. 861–872, September 2003.
[12] P. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling High Bandwidth Aggregates in the Network," *Computer Communications Review*, vol. 32, no. 3, pp. 62–73, July 2002.
[13] D. J. Bernstein, "SYN cookies," *http://cr.yp.to/syncookies.html*, 1997.
[14] J. Xu and W. Lee, "Sustaining Availability of Web Services under Distributed Denial of Service Attacks," *IEEE Transactions on Computers*, vol. 52, no. 2, pp. 195–208, 2003.
[15] A. Juel and J. Brainard, "Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks," *Proc. of Network and Distributed System Security Symposium (NDSS'99)*, February 1999.
[16] T. Aura, P. Nikander, and J. Leiwo, "DoS-Resistant Authentication with Client Puzzles," *Cambridge Security Protocols Workshop 2000. LNCS, Springer-Verlag*, 2000.
[17] D. Dean and A. Stubblefield, "Using Client Puzzles to Protect TLS," *10th Annual USENIX Security Symposium*, 2001.
[18] X. Wang and M. K. Reiter, "Defending Against Denial-of-Service Attacks with Puzzle Auctions," *2003 IEEE Symposium on Security and Privacy*, May 2003.
[19] C. Kaufman, R. Perlman, and M. Speciner, "Network Security (2nd Edition)," *Prentice Hall PTR, ISBN: 0-13-046019-2*, 2002.

[20] J. Lemon, "Resisting SYN Flood DoS Attacks with A SYN Cache," *Proc. of USENIX BSDCON2002*, February 2002.
[21] S. M. Bellovin, "ICMP Traceback Messages," *IETF, draft-bellovin-itrace-05.txt (work in progress)*, March 2000.
[22] H. Burch and B. Cheswick, "Tracing Anonymous Packets to Their Approximate Source," *Proc. of USENIX LISA'00*, December 2000.
[23] D. Schnackenberg, K. Djahandari, and D. Sterne, "Infrastructure for Intrusion Detection and Response," *Proc. of First DARPA Information Survivability Conference and Exposition*, January 2000.
[24] R. Stone, "CenterTrack: An IP Overlay Network for Tracking DoS Floods," *Proc. of USENIX Security Symposium'00*, August 2000.
[25] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical Network Support for IP Traceback," *Proc. of ACM SIGCOMM'2000*, August 2000.
[26] D. Song and A. Perrig, "Advanced and Authenticated Marking Schemes for IP Traceback," *Proc. 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," *Proc. of ACM SIGCOMM'2001*, August 2001.
[28] K. Park and H. Lee, "On the Effectiveness of Probabilistic Packet Marking for IP Traceback under Denial of Service Attacks," *Proc. of IEEE INFOCOM'2001*, March 2001.
[29] S. Chen and Q. Song, "Perimeter-Based Defense against High Bandwidth DDoS Attacks," *to appear IEEE Transactions on Parallel and Distributed Systems.*
[30] A. D. Keromytis, V. Misra, and D. Rubenstein, "SOS: Secure Overlay Services," *Proc. of ACM SIGCOMM'2002*, August 2002.
[31] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications ," *ACM SIGCOMM'2001*, 2001.
[32] D. G. Andersen, "Mayday: Distributed Filtering for Internet Services," *Proc. of 4th USENIX Symposium on Internet Technologies and Systems*, March 2003.