Sustaining Availability of Web Services under Severe Denial of Service Attacks

Jun Xu
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
jx@cc.gatech.edu
Technical Report GIT-CC-01-10

May 11, 2001

Abstract

Denial of service (DoS) is one of the most difficult security problems to address. While most existing techniques (e.g., IP traceback) focus on tracing the location of the attackers after-the-fact, little is done on how to mitigate the effect of an attack while it is raging on. We design a system that can sustain the availability of web services during severe DoS attacks. We observe that one of the major difficulties in doing this is that packets sent by attackers (bad traffic) can be completely indistinguishable from packets sent by legitimate users (good traffic), forcing a large percentage of good traffic to be dropped as a consequence. We develop a protocol that can effectively separate these two types of traffic in a statistical sense, and this separation process is secure and robust against various attacks. Therefore, by provisioning adequate resource (e.g., bandwidth) to "good traffic" separated by this process, we are able to provide fairly good service to a large percentage of users even during severe DoS attacks. For one example, during an attack where the incoming traffic rate is 5 times as high as the link rate (i.e., 80 percent of traffic has to be dropped), the system can continue to serve 59 percent of users, with only 39 percent increase to average end-to-end download time of web pages. In comparison, without such a defense, no user would receive any service due to the long retransmission timeouts caused by the heavy packet loss. Our system and protocol are completely compatible with HTTP (and HTTPS) protocols and do not require any modification to web server or client software.

1 Introduction

Denial of Service (DoS) and Distributed Denial of Service (DDoS) started to be reported frequently after 1996 [9]. One instance of a DDoS attack, deployed in mid February 2000, was directed against high-profile web sites such as Yahoo, CNN, Amazon, eBay, Datek, E*Trade, ZDNet, and Buy.com. These web sites were down for several hours as a result of this attack [20]. In a DDoS attack, a human *hacker* will first compromise a large number of hosts on the Internet and install DDoS software on them such as TFN2K, Trin00, and Stacheldraht (see rootshell.com). We refer to these hosts as "attackers" even though they are "victims" themselves. The hacker will then issue a "start" command to the attackers, asking them to send a large amount of traffic to a common target simultaneously. During such DDoS attacks, the attacked web sites are overwhelmed by so much traffic that they virtually can not provide any service to their real users.

We design a system that can sustain the availability of web services (including HTTP, HTTPS, and file/email services over the web interface) even under severe DDoS attacks. Here is an overview how this system works. In a web session, which typically consists of hundreds of packets (see Table 1) from the client to the server, our system makes sure that only the very first SYN packet may get delayed due to the attack. All other packets (including the SYN packets of other TCP connections in the same session) will be forwarded to the web server at normal throughput. This leads to significant performance gain. For one example, during an attack where the incoming traffic rate is 5 times as high as the link rate (i.e., 80 percent of traffic would have to be dropped), the system can continue to serve 59% percent of users and the end-to-end download time of web pages is only 39% longer than usual. In comparison, without such a defense, no user would receive any service since every packet in the session would be delayed in the same way the very first packet is.

The basic idea of our system is to upgrade the legitimate users' traffic to *protected traffic categories*. Traffic that belongs to the protected categories can be unambiguously recognized and separated using a security protocol we have developed. Though an attacker may also get its traffic upgraded, to do so it would have to reveal its real IP address, which makes the attacker more liable and controllable. Cryptography measures such as message authentication code (MAC) [36] are used to make sure that attackers (even collectively) can not disrupt this classification process. The scheme (including the protocol) is 100% compatible with existing protocols (HTTP, HTTPS, and TCP) and no change to web server and browser software (not even requiring Java capability) is needed.

In denial of service attacks, performance of the system becomes a security issue because it is exactly what the hacker aims to destroy. The performance analysis of the scheme is based on the conservative assumption that the hacker will, given its limited resource (attackers), tries to inflict as much damage as possible by choosing the most effective strategy at its disposal. Our system will, accordingly, choose a strategy that minimizes such damage. This natural adversarial relationship between the hacker and our system suggests that the system performance should be analyzed using a constrained min-max model in the context of game theory. Based on this model, we estimate the performance of the system and show that it is effective in protecting web services.

The rest of the paper is organized as follows. Section 2 discusses related work on (D)DoS. Section 3 and 4 present the design of the system and its implementation issues, respectively. Section 5 analyzes the performance of the system. Section 6 summarizes the contributions of this work and discusses future research.

2 Background and Related Work

DoS in the context of operating system (OS) security [21, 55, 37, 38] has been long studied. In that context all users are properly identified (by user id) and authenticated (through password)

and the system resources are under the centralized control of a resource allocator/scheduler. The problem to solve is to design a provably secure and implementable resource allocation/scheduling policy that can guarantee the liveness of all processes.

Network DoS, however, is more challenging than DoS in the OS context for two major reasons. First, in network DoS, users of a network in general can no longer be properly identified since the attackers can generate packets with arbitrary IP addresses. So when a packet arrives, there is no way to tell whether its IP address is genuine or spoofed. Authentication does not solve this problem for three reasons: (a) not all web sites require user registration and login, (b) user registration is typically not legally binding and the hacker can legitimately create a lot of bogus accounts (e.g., filling out a brief survey is all that is needed to obtain a Yahoo account), and (c) authentication protocols themselves may become a target for DoS attacks [35]. Second, the resource (e.g., bandwidth) may no longer be under the control of the attacked system. For example, it is shown in [34] that most of the traffic is dropped by upstream routers during a DoS attack before they ever reach the victim network.

Some previous works [17, 51, 9, 23, 48, 24, 27] focus on DoS attacks caused by the exploitation of system vulnerability (e.g., limitation on the number of half-open sockets) instead of DoS caused by the sheer volume of the attacking traffic. Work has been done recently on countering the latter type of DoS attack [14, 47, 50, 15, 7]. These works all focus on IP traceback, that is, to trace the origin(s) of an attack. Their common approach (except in [7]) is to require upstream routers to probabilistically write a stamp into a very-infrequently-used IP header field. The stamp is an encoding of the identity of an intermediate router or edge. Upon the receipt of the packets from an attacker, the victim decodes the stamps to reconstruct the nodes/edges that the packets have traversed. Different IP traceback schemes differ in their choices of encoding and decoding techniques. While they are valuable in finding the exact location of the attacker and (hopefully) punishing the hacker in the future, they are in general (or work very slowly) not able to mitigate the effect of a DoS attack while it is raging on. These schemes are orthogonal to our scheme and will complement each other if both are deployed.

The first work on mitigating the effect of DoS attacks is described in [34]. Based on the observation that most network traffic actually gets dropped at some bottleneck upstream routers during a (D)DoS attack before they ever reach the victim site, their approach is to ask a set of upstream routers that form a "cut" between the victim and the rest of the world, to perform rate limiting on traffic destined for the victim network. Each of these routers is far enough so that it is not a performance bottleneck. We refer to such routers *perimeter routers* since they form a line of defense. For example, suppose in Fig. 1, R3, R4, R5, and R6 are the perimeter routers that the corporate intranet asks for help. Web servers of the corporate intranet are behind the firewall and the firewall can only allow B packets per second to pass. One possible rate limiting scheme is that each router will only allow no more than 0.25*B to pass so that the firewall will not be overloaded. Suppose most of the bad traffic goes through R3, R4, and R5 and most of the good traffic goes through R5 and R6. Then legitimate users whose sessions go through R6 will benefit from the rate-limiting scheme because the the good traffic can fully enjoy the 0.25*B bandwidth. Situation for the legitimate users going through R5, on the other hand, will not benefit from the scheme because the router has no way to distinguish good traffic from large amount of bad traffic and has to drop (rate-limit) traffic indiscriminately. So this scheme is not very effective when the amount of bad traffic distributes among the perimeter routers in the same way as the amount of good traffic does [34].

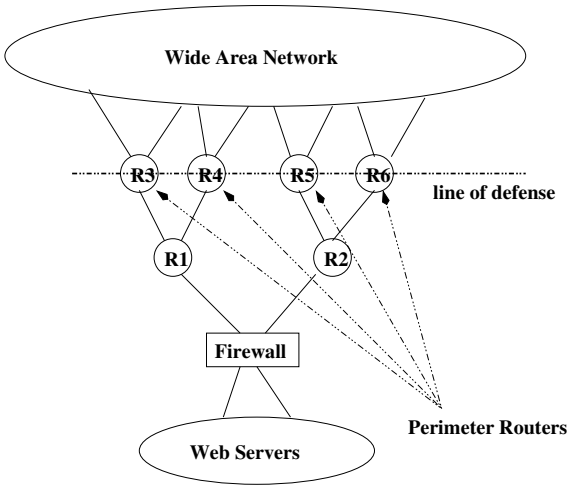

Figure 1: The System Model of Our Scheme

Site visited	Mission accomplished	#Packets the client sends
cnn.com	browse three pieces of headline news	1387
delta.com	search, reserve, and purchase a flight ticket	513
etrade.com	look up 5 stock quotes and access account balance	523

Table 1: Number of packets (HTTP and HTTPS) sent by a client during typical web sessions

3 System Design

As we explained before, it is not possible to defend against DoS without the help from upstream routers because little can be done at the web site if a large percentage of good traffic gets dropped before they ever reach the victim network. So we adopt a similar system model (shown in Fig. 1) as in [34]. The corporate intranet is connected to the wide area network (WAN) through a firewall. A set of perimeter routers will filter the packets going through them. We assume that the "line of defense" (perimeter) is pushed far enough such that, even during DoS attacks, none of the perimeter routers is a performance bottleneck. Compared to [34], the filtering operation in our system is more effective because the perimeter routers are able to distinguish good traffic from bad traffic. An attacker can still disguise bad traffic as good traffic, but to do so the attacker would have to reveal its genuine IP address, which makes the attacker more liable and controllable as we will show.

3.1 Motivation of Our Work

Our work is motivated by our observation (we collected the data shown in Table 1 using tcpdump) that a typical web transaction will result in hundreds or even thousands of packets (see Table 1) sent from a web client to a web server (there can be more packets in the other direction). This is explained by the fact that each web session typically consists of accessing multiple pages from the same server. Each access would result in multiple subsequent requests for fetching embedded objects (e.g., images, audio clips, and java applets). Each object in turn may be split into several packets (from the server to the client), and all these packets will have to be acknowledged by the client in TCP. During severe (D)DoS attacks, a large percentage of network traffic will be lost due

to congestion. So each packet would take several retransmissions in average to get through the network. Since in TCP [44], after each unsuccessful retransmission the retransmission timeout will be doubled (bounded by an upper limit of 64 seconds), these retransmissions may take a total of tens or even hundreds of seconds to accomplish. As every packet would experience such a delay in sequence in a TCP connection, the whole session would take thousands of seconds to finish[1].

In contrast, our system guarantees that among hundreds of packets sent by the client during a web session, only the very first packet (a TCP SYN packet) may be delayed as explained above. All other packets in the same session will receive normal service in the sense that the end-to-end download time of web pages will be comparable to when there are no DoS attacks. So from a user's perspective, it can be a little hard to get into the attacked web site at the very beginning due to the delay on the very first SYN packet. Once this "painful" period is over, the user will perceive the service as normal. In this way, our scheme helps sustain the availability of web services even during severe (D)DoS attacks. Our scheme supports web services including HTTP (port 80), HTTPS (secure HTTP at port 443), and email and file (download/upload) services that are provided through the web interface (e.g., yahoo email service). This not only covers majority [54, 19] of the Internet traffic, but also some of the most popular applications as well.

3.2 Assumptions

We make following basic assumptions on the capabilities of the attackers and our own systems (e.g., web servers and routers):

1. Defense measures in our system will only be triggered when a DoS attack is detected since these measures consume system resource and enforce a resource allocation policy that is not optimized for "peace-time" system performance.

2. The job of the proposed scheme is to protect web services (including HTTP, HTTPS, and web-based email/file services). The corporate intranet may also support other services, but they will be hurt by DoS attacks in the same way with or without our defense measures.

3. DoS attacks will not significantly reduce the available bandwidth of the network from server to client. Since most of today's routers use full-duplex links and switched architecture [43], congestion in one direction will not affect the available bandwidth in the other direction[2].

4. The number of "attackers" can be potentially large (say, several thousand), but we assume that it is upper-bounded by a number throughout the attack in the sense that the hacker is not able to manufacture new "attackers" (at reasonably high speed) while the attack is raging on. In essence, we view the attackers as a "nonrenewable" resource of the hacker.

5. We assume that there are two possible situations in which an attacker can receive packets destined for an IP address it claims to be. The first situation is that the attacker is a compromised host that uses its genuine IP to send and receive packets. The second situation is that the attacker is a compromised router in an intranet or a compromised host on a broadcast LAN (switched LAN is more secure), and therefore is able to spoof a set of IP addresses which belong to that intranet/LAN and receive packets sent to them. We do not distinguish these two situations: any IP address that is proved to be able to receive "callbacks" is considered to be the IP address of an attacker. In certain denial of service attacks (e.g., smurf), the hacker uses a set of machines as "reflectors" by sending them packets using victim's IP address. However, these machines are not compromised by the hacker and the hacker can not see the response from the victim to the

[1] Sublinear improvement (say, less than 4 times) on the total delay can be made when the web browser is able to support a few (typically up to 4) concurrent TCP connections, but the total delay is still too long for any meaningful service.

[2] In old days, when shared-bus architecture [10] were used instead, inbound and outbound traffic may affect each other.

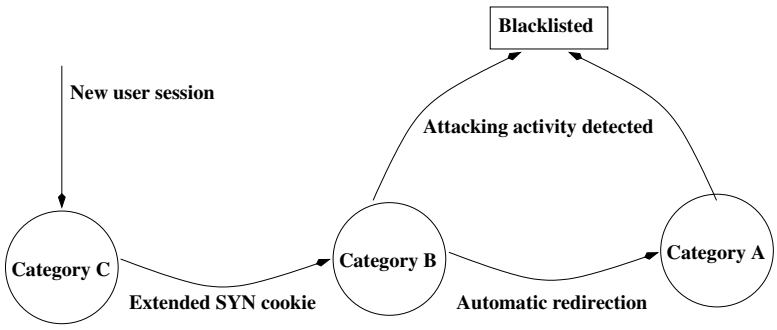

Figure 2: State migration diagram

"reflectors". So, in the context of this system, we do not view these machines as attackers.
6. All routers on or within the "line of defense" (Fig. 1) are assumed to be trustworthy and are not involved in the attacks. They accept requests from the corporate intranet to filter the incoming traffic during DoS attacks. Also, none of them is bottlenecked due to the attack.
7. In designing our system we use the worst-case estimate on what the hacker is capable of doing. We assume that the hacker tries to inflict as much damage as possible by using all resources and strategies at its disposal. This will be translated into a game theoretical context (see Section 5).

3.3 Our Approach

The effectiveness of our scheme comes from its ability to distinguishing good from bad traffic, in a statistical sense. The general idea on how this can be accomplished is captured in Fig. 2. Our scheme employs a security protocol, which is seamlessly embedded into existing network protocols, to classify network traffic into different categories. These traffic categories are designed in such a way that traffic from legitimate users will be able to migrate smoothly from the least privileged category (category C) to more privileged categories (category B and A). Traffic generated by attackers, on the other hand, will be able to do so only after attackers reveal their genuine IP addresses. As we will show, revealing genuine IP addresses makes the attackers more "liable" and controllable because they are now subject to blacklisting (by the perimeter routers). We will show in Section 3.4 that this traffic classification protocol itself is robust and secure. Therefore, by strategically allocating resources (e.g., bandwidth) to more privileged categories, we can effectively protect good traffic from bad traffic. In the following, we will discuss these traffic categories (in the reverse alphabetical order) and how migration happens between them.
Category C: In Fig. 3, we show how traffic category in a typical HTTP session migrates. When a new user arrives, it sends a SYN packet in order to establish its first TCP connection with the system. An HTTP request for the URL "www.dos-free.com" is expected to be sent over this connection. This packet could come from any IP address and in general is not distinguishable from bogus SYN packets generated by attackers (with random IP addresses). So, during a DDoS attack, the attackers can send a large volume of such SYN packets so that the perimeter routers can only pass a certain percentage of them (otherwise the firewall will be congested). We refer to the first SYN packets from new users and bogus SYN packets from attackers as category C traffic, as we have no way to distinguish between them.
Category B: Once the first SYN packet of a legitimate user eventually gets through, after perhaps some failed attempts as shown in Fig. 3, the system guarantees that all future packets will get

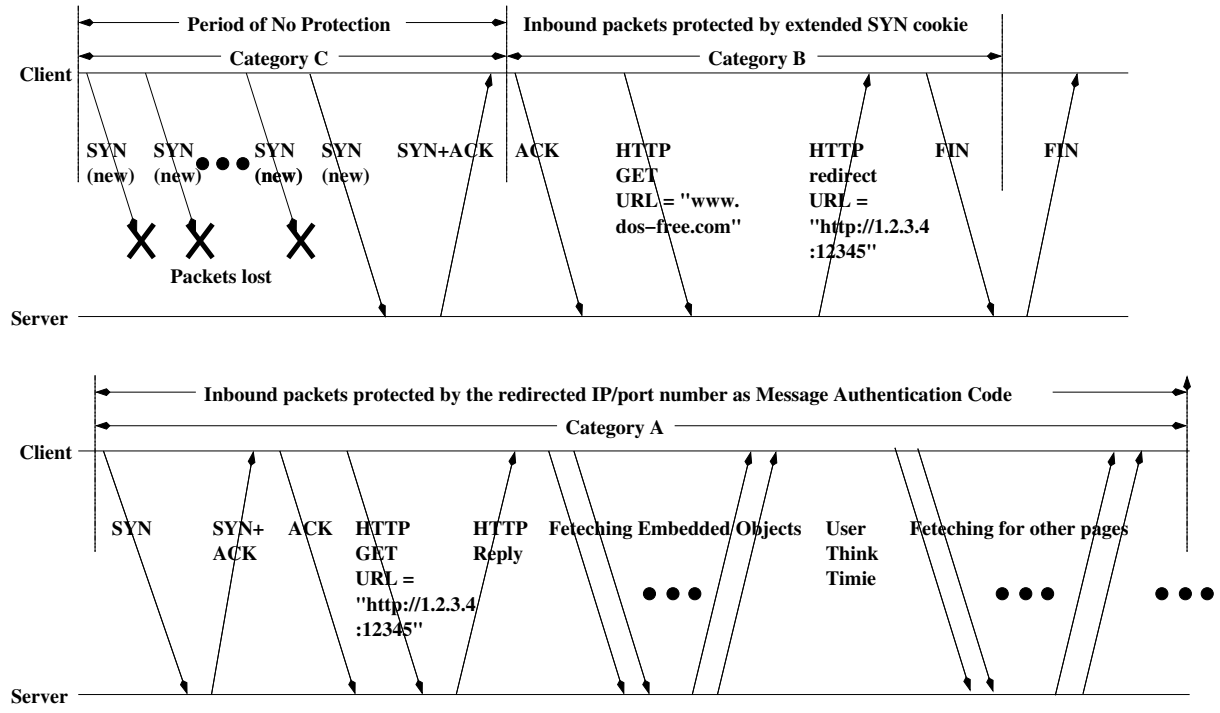

Figure 3: A typical HTTP session

through smoothly. This is achieved by redirecting the future correspondence to a pseudo IP address and port number through an HTTP URL redirect message. We will discuss this redirection process later. However, several packets need to be sent in order to even receive this redirect message, and these packets need to be protected as well. We refer to these packets as category B. Our technique to protect category B traffic is to use an extended form of SYN cookie, which was originally designed to counter SYN flood attacks. In SYN cookie [5], upon the receipt of a SYN packet from a client, instead of creating a half-open socket [52], the server simply responds with a SYN+ACK packet that contains a special TCP sequence number S-1. The lower 24 bits of S are the message authentication code (MAC) of the IP address and port number claimed by the client. Only when the server receives an ACK packet with TCP acknowledgment number S, will a full socket be opened. So a legitimate user, who is able to receive the SYN+ACK packet, will naturally respond with S according to the TCP protocol. On the other hand, an attacker using a spoofed IP address will not be able to receive the SYN+ACK packet, and can only guess about S. The probability of guessing it correctly is very small (2^{-24}) due to the security property of MAC.

In our system, SYN cookie will continue to be used against SYN flood attack. In addition, it will also be used to protect the packets involved in getting the HTTP URL redirect message as explained above. As shown in Fig. 3, they include the ACK packet to establish the first TCP connection, the HTTP GET request for "www.dos-free.com", and a FIN packet to tear down the connection. Our extended SYN cookie technique protects these packets by setting the first 23 bits of the TCP sequence number as a MAC and the last 9 bits to zero. Since all data that the client receives during this HTTP session is a text message (HTTP redirect) much shorter than 512 bytes, the TCP acknowledgment numbers of these three packets will share the same 23 bit prefix [44]. Therefore, perimeter routers are able to recognize such packets by checking whether the first 23

bits of the TCP acknowledgment number is the MAC of the port number and the source IP the client claims to be. We acknowledge that making the first 23 bits of the initial TCP sequence a random number (used for MAC) violates the recommendation made by the TCP protocol that the initial sequence number should increase slowly (SYN cookie conforms to that by making the last 24 bits a MAC). However, the problem of not conforming to this recommendation arises only when a delayed segment from a previous TCP connection is accepted as a legitimate segment in a latter connection [44]. This problem will not happen in our scheme because only the very first connection from a new user will violate this recommendation (there are simply no previous connections from the same user).

Category A: Only a small number of packets can be protected as category B traffic because as soon as a server sends a client more than 512 bytes of data in a TCP connection, the first 23 bits of the sequence number will no longer be the same. So we need some form of protection that is more permanent, which category A provides. As explained above, when a user requests the URL "http://www.dos-free.com", it will actually get an HTTP redirect URL [4] that contains a pseudo IP address and port number pair (e.g., http://1.2.3.4:12345). For simplicity of discussion, we assume that the web site to be protected is a C Class network [52] (it owns 256 IP addresses). The scheme can work with smaller IP address space (bigger is certainly better) or even just one IP address, with proper adjustments on system parameters. Assume that the web site owns the network 1.2.3.*. In other words, all packets with the IP prefix 1.2.3.* will be forwarded to the web site. Our idea is to use the 8 bit host ID and 15 bits in the port number as the MAC of the source IP claimed by the client. Perimeter routers recognize and pass category A traffic by checking the MAC. Once a category A packet arrives at the firewall, network address translation (NAT) function of the firewall will translate its pseudo address and port number into real values. This process will be explained in detail in Section 4.

Since all packets from legitimate users will contain correct MACs (SYN cookie or pseudo IP+port) in their header fields, by verifying such MACs the perimeter routers effectively block most of DoS traffic sent using spoofed IP addresses. However, this does not prevent an attacker from performing DoS on our system using its genuine address, from which it can receive correct MACs. But once the attacker is using its real address, it becomes much more liable and controllable in the following way. The firewall will perform fair queuing and scheduling [56, 42, 3, 49, 32, 53] among all the users (IPs) so that an attacker that uses its genuine IP address can at most get its fair share of bandwidth. If an attacker sends much faster than its fair share the queuing discipline will drop its excess traffic. Moreover, the firewall will perform accounting on the number of packets that reaches the firewall and gets dropped by the scheduler for all genuine IP addresses. Once such a host is identified, its IP address (or IP prefix) will be blacklisted: the perimeter routers will be informed to drop all packets from that IP address. The system will only blacklist an IP if it has at least 16000 (note that the quota N can be much bigger than this) packets dropped by the system. In Section 3.4, we will show that this parameter (16000) is chosen to achieve a compromise among three conflicting security concerns. Since we assume that the number of subnets/hosts conquered by the hacker are limited, attackers sending excessive traffic using their real IP addresses act very much like "kamikaze" (Japanese , as they will be blacklisted one by one eventually. We will show in Section 3.4 that such "kamikaze" only lasts about 100 seconds in total. So it is not to the best interest of the hacker to attack our system this way.

However, an attacker can still use its genuine IP address to send legitimate web transactions just to consume our bandwidth. When there are a large number of attackers (say, a few thousand), they collectively could squat so much bandwidth that the end-to-end web page download time of a legitimate user (analyzed in Section 5) would be significantly increased. To counter this type of "nonviolent" attack, the system enforces a "no loitering" law. The idea is that the total number of packets a user will send during a web session is not very large (several hundred to a few thousand

as shown in Table 1). The firewall can identify and punish suspicious users by checking whether a user "loiters" in the system after its business with the system should be over. The idea is that the firewall can specify a threshold N so that the probability for a legitimate session to send more than N packets is smaller than a certain value (say 3 percent). Each user will be given a quota of N packets to be used in any way the user wants (lumpsum or installments). After that, the firewall will only give a user, whose quota is up, 1/20 of its fair share[3]. This effectively limits the amount of bandwidth attackers can squat, as shown in Section 5.

There is another possible way to limit the amount of bandwidth an attacker is able to squat if user registration and login is required for the web site and the registration is legally binding (e.g., E*Trade) in the sense that the true identity of the user is verified offline. By only allowing the authenticated users to "loiter" in the system, the system can also effectively limit the number of concurrent attackers in the system, assuming that no more than a small number of user accounts get compromised. Nevertheless, this method will not help if the site (e.g., CNN) does not require user registration/login or the user registration is not legally binding.

After all, it is already a triumph that an attacker has to reveal its real IP address to be able to attack the system effectively, since what all IP traceback schemes try so hard to get is exactly this IP address (tracing attacking paths are rather means to this end). Now the attackers voluntarily reveal this information if they want to generate Category A or B traffic.

3.4 Security Analysis

We mentioned above that the system will only blacklist an IP address if the fair scheduler drops more than 16000 packets from the IP. This number represents a compromise among the following three conflicting security issues.

1. This would allow the hacker to send 16000 packets over their fair share without being punished. Fortunately, the possible damage this may cause is very small. For example, even when there are 2000 attackers, only 32 million (2000*16000) packets will be allowed through the perimeter routers. Suppose the firewall can pass 300K packets per second (translated into 100 Mbps bandwidth with minimum packet size), the hacker can only sustain about 100 seconds of "Kamikaze".

2. Since 16000 is much larger than the total number of packets in a typical legitimate web session, this prevents a legitimate user from being "accidentally" convicted of DoS and blacklisted. In fact, even if a legitimate user has more than 16000 packets to send, it will only have a small percentage of its packets dropped [18, 32, 53] as long as they conform to TCP congestion control mechanisms.

3. Is it possible for a hacker to "frame" an innocent IP address by sending a lot of junk traffic using that IP? The answer is: the hacker has to send a huge amount of traffic to be able to frame a single IP address. The MACs used in the TCP acknowledgment number and in pseudo IP+port are both 23 bits. This would only give the hacker a probability of approximately 2^{-22} to be able to send packets with random IPs but at least one correct MAC. So the hacker would have to send an average of $1/(2^{-22}) * 16000 \approx 64$ *billion* packets to be able to get 16000 of them right on MACs in order to "frame" a single IP address. Even when the hacker's total attacking bandwidth is as large as 3.2 Gbps, which translates into 10 million pps with a minimum packet size, this requires approximately 6400 seconds (close to two hours) to "frame" a single innocent IP address.

4 Implementation Issues

Our scheme consists of operations performed at two components of the system (shown in Fig. 1), namely, the perimeter routers and the firewall.

[3]We make a worst-case estimate of several thousand on the number of attackers and the number 20 reflects this conservative estimate.

4.1 Operations Performed at Perimeter Routers

The following is the algorithm of the operation performed at the perimeter routers. The algorithm performs two major functions. First, it identifies the traffic that belongs to category A and B by verifying the correctness of MACs. Second, it blacklists the attackers that attack our system using genuine IP addresses.

```
Upon the arrival of a packet ``pkt'' destined for the victim
IF pkt.SOURCE_IP in blacklist_table THEN
    drop the packet and exit;
    /* the source IP of pkt is blacklisted */
ENDIF
code_string := MAC(pkt.SOURCE_IP, k)
/* MAC stands for message authentication and ``k'' is the MAC key */
/* ``||'' stands for concatenation */
/* as the convention in MAC, the first parameter will be ``padded'' to
   the proper size (say, 128 bytes) */
IF code_string[1:23] = (last_8_bits(pkt.DEST_IP) ||
    last_15_bits(pkt.DEST_PORT)) THEN
    pass the packet and exit;
/* this is category A traffic */
/* the MSB of pkt.DEST_PORT is used to distinguish between HTTP and HTTPS */
ENDIF
IF (code_string[24:46] XOR last_23_bits(pkt.SOURCE_PORT)) =
    first_23_bits(pkt.TCP_SEQUENCE)
THEN pass the packet and exit;
/* this is category B traffic */
ENDIF
IF pkt.SYN = 1 THEN   /* pkt is a SYN packet */
    pass it with probability p;
    /* this is category C traffic */
    /* how p is determined will be discussed in Section 5 */
ENDIF
drop the packet;
/* the packet does not match anything */
```

When a packet arrives, we first check if its source IP address is blacklisted. If the answer is yes, the packet will be dropped. Otherwise, we first check whether the packet belongs to category A by checking whether the last 8 bits of its destination IP address and the last 15 bits of its port number matches the first 23 bits of MAC(pkt.SOURCE_IP, k), which is the message authentication code of the message pkt.SOURCE_IP (concatenated with a pad as appropriate) with a key "k". Note that here we reserve the first bit of the port number to distinguish between HTTP and HTTPS when stateless network address translation (described later) is performed. The packet will be passed if there is there is a match. If not, we proceed to check whether the packet belongs to category B by checking whether the first 23 bits of its TCP acknowledgment number matches the next 23 bits of MAC(pkt.SOURCE_IP, k) XOR-ed with the last 23 bits of its source port number. The reason for this XOR operation is that we would like the second MAC to depend also on the source port of the user (an implicit requirement of the SYN cookie). However, to perform a separate MAC operation could be too computationally expensive for a high speed router, as we will show next. If the packet belongs to neither category A nor category B, it belongs to category C if it is a SYN packet. In this

case the packet will be passed with probability p, which will be strategically chosen to maximize the performance of our system (discussed in Section 5). The packet will be dropped if it does not belong to any category[4].

HMAC [30] was initially considered as the MAC function to be used in the system. HMAC is a generic hash-based MAC algorithm that can be used with any cryptographically strong one-way hash functions such as MD5 [46] and SHA1 [40]. However, we found that both HMAC-MD5 and HMAC-SHA1 are too slow for a high speed router. We measured their performance on a Pentium III 800 MHz desktop PC running Linux, using Dai's crypto++ 4.0 [13] library. Generating a 16-byte MAC of 128-byte minimum size data using a 16-byte key takes 13.8 microseconds for HMAC-SHA1 and 5.3 microseconds for HMAC-MD5[5]. Since modern routers typically need to forward packets at the speed up to one million packets per second per port, neither algorithm would catch up with that speed. However, the good news is that UMAC [6], a hash-based MAC algorithm developed recently, runs five times as fast as HMAC-MD5 and is shown to be provably secure (based on the assumption that the underlying hash function is secure). We measured on the same platform and found that each MAC operation would only take 1.1 microseconds to execute. Each dedicated CPU in the router would be able to perform close to 1 million MAC operations per second, which is comparable to the per-port packet forwarding rate of a high-end router. Also, network processors (e.g., Intel IXP1200 [25]) developed for routers contain several RISC CPU modules to process packets in parallel. It may be feasible to dedicate a subset of them to MAC operations since the MAC operations are completely independent of other header processing steps.

The second operation a perimeter router needs to perform in our scheme is to blacklist the IP clusters/addresses (how a cluster is identified will be discussed later) that are known to be controlled by attackers. The blacklist could be implemented as a hash table. Each hash node represents a conquered cluster or host (viewed as a single-node cluster), coded as a variable-length prefix. For example, a conquered C class network 1.2.3.4 will be represented as 1.2.3.0/16. The index of a hash node will be the hash value of the first 16 bits of the IP prefix of a conquered host or cluster. By using 16 bits we assume that no conquered cluster will be bigger than Class B network. So the probe process is to inspect whether the source IP address of an incoming packet matches the prefix contained in a hash node along the linked list pointed to by the hash index pointer. We measured on the aforementioned platform that probing a hash table with an average list length of 4 would take about 0.25 microseconds. Since we assume that the total number of conquered networks and hosts are limited, the total size of the hash table will be bounded. Therefore, the total amount of processing per packet at a perimeter router will not be larger than 1.5 microseconds (including the aforementioned MAC operation) assuming that a router uses processors comparable to Pentium III 800 MHz.

4.2 Operations Performed at the Firewall

In our system model (Fig. 1), the firewall is shown as one box. In reality, it can be implemented as a number of boxes operating in parallel with same functionalities or as several boxes with different functionalities. How that is exactly implemented is outside the scope of this paper. So here we refer to the firewall as an abstract entity. The firewall will be enhanced to provide the following five functionalities.

1. The firewall will perform standard connection interception [45] (also called TCP interception) when a category C packet arrives. It should send back a SYN cookie with correct MAC as part

[4] A small percentage of such packets may have to be passed if non-web services also have to be supported.

[5] Note that this is much slower compared to the throughput data measured in [13] because there the throughput is measured on hashing big chunks (e.g., hundreds of megabytes) of data so that the high initialization overhead is amortized.

of the TCP sequence number as explained before. When a SYN+ACK arrives with correct SYN cookie inside it, the firewall will establish a connection between a web server and the user. Also, the firewall should intercept HTTP requests for the default URL of the site ("http://www.dos-free.com") and respond with a URL redirect message containing the pseudo IP+port pair, which contains correct MAC for the IP address. The MAC key 'k' and the pad is shared among the firewall and perimeter routers.

2. The firewall should apply fair queuing and scheduling disciplines among users, identifiable by their IP addresses. There are a number of fair queuing and scheduling disciplines available [56, 42, 3, 49, 32, 53], and most of them are readily implementable on a corporate gateway where the number of concurrent flows are relatively small (compared to backbone routers). We refer readers to [56] for a comprehensive treatment of the subject. In our context, a flow is actually an IP flow (instead of TCP flow). So the number of concurrent flows will be even smaller than the number of flows in the usual sense (TCP flow), making the fair queuing and scheduling operation even lower in time and space complexity. Each IP will be given a fair share of bandwidth so that an attacker, disguised as a legitimate user, will not be able to get more service than a legitimate user. The scheduler will drop packets from a flow if it is sending faster than its fair share. Legitimate users will adjust its sending rate around this fair share due to the TCP congestion control algorithm [32, 53].

3. The firewall will perform network address translation (NAT) so that the pseudo IP+port that serves as MAC for category A traffic will be translated into the actual IP address of a web server. The system can make this process completely "stateless" by using hash functions to map pseudo IP+port to the actual IP address of a web server (similar techniques are used in [8] for network load-balancing purposes). Here we assume that the web servers are identical to each other in terms of the content serviced and functionalities provided. As explained before, the most significant bit (MSB) of the pseudo port number is used to indicate whether the actual port number is HTTP (80) or HTTPS (443). In the other direction, when a web server sends a packet back to a client, the source IP address of the packet will be overwritten by the pseudo IP+port (the MAC can be calculated from the destination IP of this packet) before it leaves the web site.

4. The firewall will store a record for each genuine (passing the MAC test) IP address. The amount of time a record will be stored for an IP will be proportional to the amount of traffic the IP has sent. We apply the following heuristics: (a) an IP address will be stored indefinitely (until DoS attack has been cleared) if it has sent more than its quota, and (b) the record for an IP address can be removed if the IP has not been active for the amount of time that is 20 times its total web page download time. Since the page download time of most web transactions is short (typically no more than 20 seconds [33, 11, 22]), this method effectively limit the amount of information the firewall has to store.

5. The firewall needs to help perimeter routers perform blacklisting in an efficient way. Consider the situation where hacker may actually control all IP addresses in a subnet. Blacklisting such IP addresses individually is not storage-efficient for the perimeter routers. So instead the firewall "cluster" the compromised IPs using the method introduced in [31]. There the purpose is to cluster web clients for better web caching and content delivery. The method is to merge the lists of IP prefix entries in all publicly available BGP routing tables into a giant IP prefix table. Source IP address of a client packet will be searched against this giant table to find the longest matching IP prefix [16]. This IP prefix will be regarded as the domain or cluster the client belongs to[6]. It is reported in [31] that the accuracy of this method is 99.9%. We use the same method and apply the following heuristics to when a clustering operation should be triggered. When 16 addresses from

[6]Note that due to Classless Inter-Domain Routing (CIDR) [2], the size of the network an IP address belongs to can no longer be figured out from the first 4 bits of the IP address.

the same 24-bit IP prefix are blacklisted, we will perform a clustering operation to find the prefix of the domain such addresses belong to. Then, instead of blacklisting these addresses individually, we blacklist the whole domain that is involved. This may hurt legitimate users from that domain. However, if there are 16 hosts from a given domain that are compromised, that domain has a deeper problem than not being able to receive service from a web site.

Most of the functionalities mentioned above, including connection (TCP) interception, fair queuing and scheduling, and network address translation are standard router and firewall functions. The record keeping for IP addresses (to enforce "no loitering") can be implemented using a hash table in a straightforward way. The clustering algorithm only needs to be done once in a while (say every 20 minutes), and will not incur much overhead to the firewall.

One important detail remains to be addressed. In a page hosted by the protected web site, links to URLs hosted by the same site will cause problem to our scheme because they contain "http://www.dos-free.com:80" and will point to the "old" (unprotected) IP+port. However, this problem will not arise if relative URLs are used instead because the web browser of the user will add the proper HTTP prefix (http://1.2.3.4:12345). So the simplest solution is to use only relative URLs for links hosted by the same site. Another solution is a HTML translation (a simple string match and replacement) from absolute URLs to relative URLs[7]. Web proxy software in a firewall can be configured to perform this function. However, for better performance, a web switch or proxy box optimized for this purpose should be used instead.

4.3 Discussion

We assume that there is a protocol that facilitates the communication between the firewall and the perimeter routers. The design of this protocol is not complicated but is outside the scope of this paper[8]. The amount of information that needs to be conveyed is very small, which only includes a message authentication key and pad, and the IP addresses/prefixes that need to be blacklisted. Since sensitive information will be conveyed, such packets need to be authenticated and encrypted. For example, they can run on top of IPSEC protocol [29].

This scheme is backward compatible with all existing network protocol standards and their implementation base. It is fully compatible with HTTP and HTTPS protocols and standard web browsers. It does not even require the browser to have Java capability. No change to web server software is needed as long as all the web pages the site hosts are linked through relative URLs or the firewall performs the proper HTML translation. This scheme does not assume that there is user registration and authentication built into the system so that it is applicable to any web site.

5 Performance Analysis

The performance of the system is analyzed based on the following worst-case threat model. A hacker, given his resource constraint (compromised hosts/networks), tries to minimize the performance of the system by choosing the best strategy at his disposal. Our system, on the other hand, by choosing its best strategy, tries to maximize the performance. This adversarial relationship naturally fits into the following constrained min-max model from the game theory [26].

"System performance" being too general, we need to specify what is the metric to maximize (by the system) or to minimize (by the hacker) in a strict mathematical sense. Different choices of metrics necessarily lead to different results. The metric we use here is the total user satisfaction rate, defined as the number of new users (per second) that eventually makes their way to the

[7]This also handles dynamically generated URLs by Javascript as long as "www.dos-free.com" appears as a whole in the script.

[8]The same protocol as proposed in [34] may be adopted with packet format modifications.

system despite the long delay that occurred to the very first SYN packet, multiplied by the average satisfaction of each user. The satisfaction of each user is measured by the inverse of the total end-to-end web page download time of a session. The end-to-end web page download time is introduced in [12] to gauge the web performance. Following are the notations that will be used in the analysis.

Notations:
A: arrival rate of legitimate users
B: total bandwidth of the firewall
Y: bandwidth given to category C traffic
B-Y: bandwidth given to category A and B traffic
N: total number of attackers
X: number of attackers sending SYN packets using random IPs
Z: number of attackers sending "semantically correct" packets using genuine IPs
α: the average sending rate of an attacker using random IPs
W: average amount of traffic a user sends during the whole web session
b: effective per-user bandwidth when there is no DoS
R: effective per-user bandwidth when there is DoS
p: percentage of Category C traffic that passes through the perimeter routers
f(p): given p, the percentage of the users that eventually get into the system
d(p): given p, the average initial delay (to the first SYN packet) a user has experienced before getting into the system
T: total end-to-end download time of web pages during a user session in average

We assume that the firewall is the performance bottleneck of the system and it has bandwidth B. The arrival rate of new users is A and the amount of traffic each user will send during the whole session is W. When there is no DoS, each user still has an upper limit on its effective bandwidth (denoted as b) due to the transient behavior of TCP and the fact that most web transfers are small in size [41, 22]. So, without DoS attacks, $\frac{A*W}{B}$ is the utilization of the system and $\frac{W}{b}$ is the total end-to-end download time of web pages in a web session. The perimeter routers have a parameter Y to tune, which is the amount of Category C traffic that they should allow to pass. Note that if Y is not bounded above, the attackers could just send a lot of bogus SYN packets to swarm the system. However, if Y is set to 0, no legitimate users can get their first SYN packet through. So Y needs to be finely tuned to allow enough legitimate users through without consuming too much of the firewall bandwidth. We assume that the hacker has compromised N hosts, all of which will be used as attackers. Let X and Z denote the number of (effective) attackers that attack our system using SYN packets with faked IP address (category C) and "semantically correct" packets with genuine IP address (category A and B), respectively. Let α be the average rate an attacker can generate category C traffic and p be the percentage of the category C traffic that will be received by the firewall. We calculate p as $p = \frac{Y}{X*\alpha}$ because among the $X * \alpha$ category C packets (per second) that arrive only Y will be allowed to pass. Here we do not count the arrival rate of the first SYN packets of legitimate new users because they are negligible compared to $X * \alpha$.

Let f(p) denote the percentage of new users that eventually have their first SYN packet get through and receive service from the system afterwards. We assume that a user is willing to wait for 32 seconds for response. In TCP, this is translated into the willingness to tolerate six consecutive losses and retransmissions of the first SYN packet. In the default TCP setting, the timeout value for these six retransmissions are 0.5, 1, 2, 4, 8, and 16 seconds, respectively [52]. Assuming that the round-trip-time (RTT) is bounded by 0.5 seconds, they (0.5+1+2+4+8+16+0.5) add up to 32 seconds. So $f(p) = \sum_{i=0}^{6} p * (1-p)^i = 1 - (1-p)^7$. Let d(p) be the average delay of the very first SYN packet of new users who eventually reaches the victim. Then $d(p) = \frac{\sum_{i=0}^{6} 0.5*2^i*p*(1-p)^i}{f(p)} = \frac{1-(2-2p)^7}{2*p*(2p-1)*f(p)}$ according to the aforementioned default timeout setting.

Let R and T be the average bandwidth and total download time of a user session during the DoS attack. T and R are related by $T = \frac{W}{R}$. If a session lasts T seconds, there are A*T*f(p) concurrent good users according to Little's law (due to the DoS attack, the arrival rate of good users becomes A*f(p)). Since there are also Z attackers who will take a fair share of the bandwidth, each user will receive up to $\frac{B-Y}{A*T*f(p)+Z}$ bandwidth thanks to the fair queuing and scheduling performed at the firewall (Y is reserved for category C traffic). Since a user's bandwidth is limited by b, we get $R = min\{\frac{B-Y}{A*T*f(p)+Z}, b\}$. So $W = \frac{W}{R} * R = T * min\{\frac{B-Y}{A*T*f(p)+Z}, b\}$. Solving for T, we get

$$T = \begin{cases} \frac{W*Z}{B-Y-W*f(p)*A} & : \quad b \geq \frac{B-Y}{Z+f(p)*A*\frac{W*Z}{B-Y-W*f(p)*A}} \\ \frac{W}{b} & : \quad otherwise \end{cases} \tag{1}$$

The total user satisfaction rate, which is the metric to optimize, is $\frac{f(p)*A}{d(p)+T}$. Here d(p)+T is the total delay the user has experienced in downloading the web pages in a session. Since both p (function of X and Y) and T (function of Z and Y) are functions of X, Y, and Z, we can use g(X,Y,Z) to denote $\frac{f(p)*A}{d(p)+T}$, the metric to be optimized. The according to the minmax theorem of the game theory [26], the optimal value is:

$$\underset{Y}{max}\ \underset{X,Z}{min} g(X,Y,Z) = \underset{X,Z}{min}\ \underset{Y}{max}\ g(X,Y,Z) \tag{2}$$

The hacker's strategies are different choices of X and Z and the system's strategies are different choices of Y. The solution to (2) is called Nash equilibrium [39] in the game theory. Neither party has the incentive to unilaterally deviate from the Nash equilibrium because if he does, the other party can gain more by choosing a strategy that takes advantage of this deviation.

In the interest of space, we only discuss the solution of (2) in one real-world scenario. Each scenario would corresponds to a set of constraints on the parameters X and Z. Here we assume that the hacker can send both category C packets and Category A/B packets. In this situation, the constraint on X and Z are $X \leq N$ and $Z \leq N/20$. The N/20 comes from the "no loitering" law to limit the amount of bandwidth an attacker can steal, assuming that after a while all attackers have used up their quota. We can see that, given any fixed Y, g(X,Y,Z) decreases when either X or Z becomes bigger. So the hacker's strategy will always be X=N and Z=N/20. Our system will then choose Y such that g(Y,N,N/20) is maximized. So in this scenario, the minmax formula degenerates into a single variable optimization problem. In the following two examples, we estimate the performance of our system when it is bombarded with traffic that is (a) 5 times or (b) 10 times of the firewall bandwidth.

Example 1: We assume that the bandwidth of the firewall is 300000 inbound packets per second (pps), which is equivalent to 100 Mbps when every packet is of the minimum size (40 bytes). The average arrival rate of new users are 300 per second and each user will send 700 packets (representative of real-world data shown in Table 1), without counting retransmissions. So when there are no DoS attacks, the firewall is 700*300/300000 = 70% loaded as explained before. A user's average effective bandwidth is assumed to be 40 pps (translated into an average of 17.5 seconds of total download time for the whole web session). Both are close to real-world data observed in literature [33, 11, 41, 22]. There are 1290 attackers, the attacking rate of which is assumed to be 1000 packets per second (translated into 320 kbps with minimum packet size of 40 bytes). These two parameters are so chosen that the total traffic (good+bad) is 5 times of the firewall bandwidth (i.e., 80% of packets have to be randomly dropped without our defense scheme).

We solved (2) for these parameters using numerical method. The optimal customer satisfaction is achieved when Y is set to 168720 pps, 56% of the total bandwidth. In this optimal situation, our system is able to provide service to 59% of the users and the total end-to-end download time

(including the delay of the first SYN packet) during the whole session for these users is 39% longer than usual. The fact that 56% of the bandwidth has to be reserved for SYN packets is not surprising because even so only 11.2% percent (0.56*0.2) of the first SYN packets sent by new users reaches the victim.

Example 2: All other parameters being the same as in example 1, now there are 2930 attackers. In this situation, 90% of packets have to be dropped. Solving (2) for this case, we found that our system can still provide service to 40% of users and the web page downloading time is increased by 44%. This optimal situation is achieved when Y reaches 210210, approximately 70% of the total bandwidth.

We can see that even during severe (D)DoS attacks (20% and 10% packet survival probability), our system still can render service to a large percentage (59% and 40%) of users, with a tolerable (39% and 44%) increase on the average end-to-end download time of web pages.

In reality, we do not know some of these parameters exactly. However, they can be estimated in an adaptive way. The system can measure and store B, W, and b when there are no DoS attacks. When a (D)DoS attack happens, the system can estimate N and α by measuring the amount of traffic that arrives at the perimeter routers during the attack. These measurements do not need to be accurate at the beginning. The system will adapt to the optimal strategy by trying different Y's in "cautious" steps.

6 Contributions and Future Work

The major contributions of this work can be summarized as follows:

- We designed a system that effectively sustains the availability of web services even during severe (D)DoS attacks. Our scheme is easily deployable since (a) it does not entail any modifications to web server design and implementation, (b) it is fully compatible with any web browsers speaking HTTP/HTTPS (not even requiring the browser to support Java or cookie capability), and (c) it does not require any user registration and authentication. The existence and execution of the scheme is completely transparent to web clients. Since the web is the most popular Internet application, which accounts for the majority [54, 19] of the total Internet traffic, and most E-commerce transaction are conducted through web interface, this work will strengthen the Internet infrastructure against (D)DoS attacks.

- The design of our system is well engineered to address various security and performance considerations. The design is highly implementable since it uses and customizes standard techniques (e.g., fair queuing/scheduling, MAC, NAT, SYN cookie) that have been well developed and validated.

- We proposed a game theoretical framework that accurately models the performance of our system as the Nash equilibrium between conflicting goals of the hacker to maximize the damage and our system to minimize it. Since all (D)DoS problems contain such an adversarial relationship in nature, we expect this model to be useful for analyzing the performance of other (D)DoS and defense systems.

Our future work consists of two major components. First, we will simulate the performance of the system using ns-2 [1] to verify the accuracy of our analysis. Second, a prototype system is under development. It will be built on open-source components such as BSD ALTQ [28], which can be modified to add the functionalities our scheme needs.

References

[1] Ucb network simulator - ns (version 2).

[2] Rfc1518: An architecture for ip address allocation with cidr. Technical report, Network Working Group, September 1993.

[3] J. Bennett and H. Zhang. wf^2q: worst-case fair weighted fair queuing. In *IEEE INFOCOM'96*, March 1996.

[4] T. Berners-Lee, R. Fielding, and H. Frystyk. Rfc1945: Hypertext transfer protocol – http/1.0. Technical report, Network Working Group, May 1996.

[5] D. Bernstein. Syn cookies, September 1996.

[6] J. Black, S. Halevi, H. Krawczyk, Ted Krovetz, and P. Rogaway. Umac: Fast and provably secure message authentication. In *Advances in Cryptography – Crypto'99*, pages 216–233, 1999.

[7] H. Burch and B. Cheswick. Tracing anonymous packets to their approximate source. In *Prof. Usenix LISA 2000*, December 2000.

[8] Z. Cao, Z. Wang, and E. Zegura. Performance of hashing-based schemes for internet load balancing. In *Proc. Infocom'2000*, March 2000.

[9] CERT. Tcp syn flooding and ip spoofing attacks. Advisory CA-96.21, September 1996.

[10] T. Chen and S. Liu. *ATM Switching Systems*. Artech House, Boston, MA, 1995.

[11] H. Choi and J. Limb. A behavior model of a web traffic. In *Proc. ICNP'99*, September 1999.

[12] M. Christiansen, K. Jeffay, D. Ott, and F. Smith. Tuning red for web traffic. In *Proc. of Sigcomm'00*, pages 139–150, September 2000.

[13] W. Dai. *Crypto++ 4.0 Benchmarks*, June 2000.

[14] D. Dean, M. Franklin, and A. Stubblefield. An algebraic approach to ip traceback. In *Proc. NDSS 2001*, pages 3–12, February 2001.

[15] T. Doeppner, P. Klein, and A. Koyfman. Using router stamping to identify the source of ip packets. In *Proc. ACM CCS-7*, pages 184–189, November 2000.

[16] W. Doeringer, G. Karjoth, and M. Nassehi. Routing on longest-matching prefixes. *IEEE/ACM Trans. Networking*, 4(1):86–97, February 1996.

[17] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In E. Brickell, editor, *Proceedings of Crypto'92*, pages 139–147. LNCS 740, 1992.

[18] S. Floyd and V. Jacobson. Random early detection gateway for congestion avoidance. *IEEE/ACM Transaction on Networking*, 1(4):397–413, August 1993.

[19] National Laboratory for Applied Network Research (NLANR). Flow statistics analysis for fix-west. http://www.nlanr.net/, June 1998.

[20] L. Garber. Denial-of-service attacks rip the internet. *IEEE Computer*, 33(4):12–17, April 2000.

[21] V. Gligor. A note on the denial-of-service problem. In *Proceedings of the 1983 IEEE Symposium on Security and Privacy*, pages 139–149. IEEE Computer Society Press, 1983.

[22] J. Heidemann, K. Obraczka, and J. Touch. Modeling the performance of http over several transport protocols. *IEEE/ACM Transaction on Networking*, 5(5):616–630, October 1997.

[23] IETF. *Photuris: Session-Key Management Protocol*, March 1999.

[24] Checkpoint Inc. Tcp syn flooding attack and the firewall-1 syndefender. http://www.checkpoint.com/products/firewall-1/syndefender.html, 1997.

[25] Intel. Intel ixp1200 network processor. http://www.intel.com.

[26] A. Jones. *Game Theory: mathematical models of conflict.* John Wiley & Sons, 1980.

[27] A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proc. of NDSS'99*. Internet Society, March 1999.

[28] C. Kenjiro. A framework for alternative queuing: Toward traffic management by pc-unix based routers. In *Proc. Usenix 1998 Annual Technical Conf.*, June 1998.

[29] S. Kent and R. Atkinson. *Security Architecture for the Internet Protocol.* IPSEC Working Group, May 1998.

[30] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication.* Network Working Group, February 1997.

[31] B. Krishnamurthy and J. Wang. On network-aware clustering of web clients. In *Proc. ACM Sigcomm 2000*, pages 97–110, September 2000.

[32] D. Lin and R. Morris. Dynamics of random early detection. In *Proc. of Sigcomm'97*, September 1997.

[33] B. Mah. An empirical model of http network traffic. In *Proc. Infocom'97*, April 1997.

[34] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. Technical report, ACIRI and AT&T Labs Research, February 2001.

[35] C. Meadows. A formal framework and evaluation method for network denial of service. In *Proceedings of the 1999 IEEE Computer Security Foundations Workshop*, Mordano, Italy, June 1999.

[36] A. Menezes, P. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[37] J. Millen. A resource allocation model for denial of service. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 137–147. IEEE Computer Society Press, 1992.

[38] J. Millen. Denial of service: A perspective. *Dependable Computing for Critical Applications 4*, pages 93–108, 1995.

[39] J. Nash. Non-cooperative games. *Annals of Math.*, (54):286–295, 1951.

[40] NIST. Secure hash standard (shs). Technical report, FIPS 180-1, April 1995.

[41] V. Padmanabhan and J. Mogul. Improving http latency. *Computer Networks and ISDN Systems*, 28(1&2), December 1995.

[42] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single node case. *IEEE/ACM Transaction on Networking*, 1(3):344–357, June 1993.

[43] C. Partridge et al. A 50-gb/s ip router. *IEEE/ACM Trans. on Networking*, 6(3):237–248, June 1998.

[44] J. Postel. Rfc 793: Transmission control protocol. Technical report, Internet Society, September 1980.

[45] A. Rice. Defending networks from syn flooding in depth. Technical report, Sans Institute, December 2000.

[46] R. Rivest. Rfc1321: The md5 message-digest algorithm. Technical report, IETF, April 1992.

[47] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for ip traceback. In *Proc. ACM SIGCOMM 2000*, pages 295–306, August 2000.

[48] C. Schuba et al. Analysis of a denial of service attack on tcp. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1997.

[49] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round robin. In *Proc. of ACM SIGCOMM'95*, pages 231–242, August 1995.

[50] D. Song and A. Perrig. Advanced and authenticated marking schemes for ip traceback. In *Proc. Infocom 2001*, April 2001.

[51] E. Spafford. The internet worm incident. In *1989 European Software Engineering Conference (ESEC 89)*. Springer-Verlag, 1989.

[52] W. Stevens. *TCP/IP Illustrated Volume 1, The Protocols*. Addison-Wesley, 1994.

[53] B. Suter, T. Lakshman, D. Stiliadis, and A. Choudhury. Design considerations for supporting tcp with per-flow queueing. In *Proc. of IEEE INFOCOM'98*, March 1998.

[54] K. Thompson, G. Miller, and R. Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network*, pages 10–23, November 1997.

[55] C. Yu and V. Gligor. A specification and verification method for preventing denial of service. *IEEE Transaction on Software Engineering*, 16(6):581–592, June 1990.

[56] H. Zhang. Service disciplines for guaranteed performance service in packet switching networks. *Proceedings of the IEEE*, 83(10), October 1995.