

RC 22566 (Log# W0202-013) (09/25/2002)
Computer Science/Mathematics

IBM Research Report

Protecting Content Distribution Networks from Denial of Service Attacks

Kang-Won Lee, Suresh Chari, Anees Shaikh, Sambit Sahu, Pau-Chen Cheng

IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, New York 10598

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties). Some reports are available at <http://domino.watson.ibm.com/library/CyberDig.nsf/home>. Copies may be requested from IBM T.J. Watson Research Center, 16-220, P.O. Box 218, Yorktown Heights, NY 10598 or send email to reports@us.ibm.com.

This page intentionally left blank.

Protecting Content Distribution Networks from Denial of Service Attacks

Kang-Won Lee, Suresh Chari, Anees Shaikh, Sambit Sahu, Pau-Chen Cheng

IBM T. J. Watson Research Center
Hawthorne, NY 10532

Abstract— Denial of service (DoS) attacks continue to be a daunting challenge for service providers on the Internet. Recent work on countering these attacks has focused primarily on attacks at a single server location or on network resources. Increasingly, however, high-volume sites are distributed using content distribution networks (CDNs). In this paper, we develop two mechanisms to deter DoS attacks against CDN-hosted Web sites and CDN infrastructure servers. First, we propose a novel CDN request routing algorithm which allows CDN servers to effectively distinguish attack traffic from legitimate requests. Our scheme, based on a keyed hash function, significantly improves the resilience of CDNs to DoS attacks. Second, we introduce several site allocation algorithms based on binary codes which insure that an attack on one hosted Web site will have a limited impact on other hosted sites. Our scheme *guarantees* that a specified minimum number of servers remain available for other sites even when the intended victim is successfully attacked. Together, our schemes significantly improve the resilience of CDN-hosted Web sites, and complement other work on countering DoS and distributed DoS attacks.

I. INTRODUCTION

The problem of detecting and thwarting denial of service (DoS) attacks against Internet servers has recently drawn considerable attention. These attacks typically flood a network or server with bogus request packets, rendering it unavailable to handle legitimate requests. Despite increased awareness about security issues, denial of service attacks remain a challenging problem. According to a recent Computer Security Institute survey, for example, the number of respondents indicating their sites had been the victim of a DoS attack rose from 27% in 2000 to 38% in 2001 [1].

DoS attacks often target network resources by generating a large volume of bogus traffic that consumes network bandwidth on the links that connect to the target site. The impact of such brute-force attacks can be mitigated, however, by deploying network level mechanisms such as packet filtering and rate limiting [2], [3], [4]. While network level mechanisms provide a first level of protection, they cannot completely prevent attack traffic from reaching its targets unless ubiquitously deployed. Moreover,

certain types of attacks are intrinsically hard to counter at the network level. Therefore, considerable attention has been devoted to developing server-side measures to withstand DoS attacks on Internet servers [5], [6].

In today's Internet architecture, many high volume sites are distributed, either by replicating content in several data centers, or distributing content via a content distribution service provider (CDSP). Due to the increased server and network capacity available from their geographically distributed server infrastructure, CDSPs offer increased resilience to DoS attacks in addition to promising better performance. However, content distribution networks (CDNs) are not bullet-proof. For example, a determined attacker can infiltrate thousands of machines on the Internet using automated attack tools and launch a very large scale distributed DoS (DDoS) attack in order to bring down CDN servers in a specific region. Defense against such large scale DDoS attacks is most difficult when the attacker uses 'legitimate' packet types accepted by CDN servers, such as TCP SYN packets, to flood the target site. Hence, we focus on flooding attacks using TCP SYN packets on CDNs.

In addition, the shared nature of a CDN's infrastructure can be a weakness because an attack on a single CDN-hosted Web site can affect many (or all) of the other customer sites hosted by the same CDN. Without a careful site allocation strategy, the redundancy and replication provided by the CDN offer limited protection for the hosted customers.

In this paper, we develop deterrence mechanisms to DoS attacks that are suited for CDNs. Our proposed scheme makes the job of the attacker significantly more difficult by leveraging features unique to CDNs, namely the request routing system, which directs client requests to the most appropriate CDN server. We also introduce novel allocation algorithms to provide sufficient isolation among CDN-hosted Web sites to prevent an attack on one Web site from bringing down other sites.

More specifically, our contributions are:

- **Hash-based request routing:** We augment the CDN request routing scheme with a keyed hashing mechanism to help CDN servers distinguish legitimate requests from bogus requests. Using this mechanism, the total amount of attack traffic required to victimize a CDN increases *quadratically* with the number of CDN servers in the region. This significantly improves resilience over conventional CDNs, where the amount of traffic necessary to bring down a CDN increases *linearly* with the number of servers.
- **Site allocation policies and algorithms:** We propose an allocation strategy based on binary codes, which insures that a successful DDoS attack on any individual Web site that disables its assigned CDN servers, does not collaterally bring down other Web sites hosted by the same CDN. Our scheme *guarantees* that the non-target Web sites are served from at least a specified minimum number of servers, thereby ensuring a lower bound on availability and performance.

As with any Internet security measure, our scheme is not a comprehensive solution by itself. Rather, we propose a set of mechanisms to complement the existing techniques used to combat DDoS attacks. Our proposed mechanisms cannot protect, for example, against attackers who target the network bandwidth resources on the links connected to the CDN servers. Such attacks are better addressed by other DDoS countermeasures, such as packet filtering and rate limiting. It should be noted, however, that many observed attacks do not generate enough traffic to consume the bandwidth on today’s high-speed links, but are quite sufficient to overwhelm a server [7].

The improved resilience provided by the hash-based request routing scheme comes at the cost of selecting servers based on non-performance metrics *within a region*. Also our site allocation scheme assigns Web sites to a subset of the CDN servers, instead of using all servers to serve all sites. Our approach is flexible, however, in that it allows a CDSP to balance the trade-off between performance and security. We discuss this trade-off in Section VI.

The remainder of this paper is organized as follows: Section II reviews related work in the literature. Section III defines our CDN model and the types of attacks we consider. Section IV describes the hash-based request routing algorithm. Section V presents our site allocation strategy. Section VI discusses practical issues and further enhancements. Finally, Section VII concludes the paper.

II. RELATED WORK

This section provides a brief review of existing technologies to counter DoS attacks.

Network-level mechanisms such as packet filtering [2], [3] and rate-limiting [4], have been designed to prevent or mitigate the impact of DDoS attacks. These mechanisms are deployed in network routers to prevent potential attack packets from reaching their destination.

A number of recent studies have proposed techniques to determine the route to the origin of attacks as a deterrence mechanism. Examples of such techniques include “controlled flooding” [8], audit trails [9], [10], and traceback [11], [12]. In packet-based traceback, packets are specially marked as they are forwarded by the routers. When the end-host receives these marked packets, it can construct the path back to the origin of the attack, given a large enough number of marked packets [11], [12]. While traceback techniques deter potential attackers, they are typically applicable to only postmortem analysis, or while a site is under attack.

One of the most common types of attacks on end systems is SYN flooding [13], in which an attacker sends a large number of TCP SYN packets to the target without fully establishing the connection. Numerous defensive measures have been proposed and deployed against SYN flood attacks. They include SYN cookies [6], randomly dropping SYN packets, reducing the time allowed to complete TCP connection establishment. Among these techniques, SYN cookies are most popular in practice because of its simplicity and effectiveness. The basic idea of SYN cookies is to encode the information about the incoming SYN packets in the sequence number of the corresponding SYN-ACKs. In this way, the server does not have to maintain state for partially established connections, thereby substantially improving the resilience to SYN flood attacks. In Section IV-C, however, we show that SYN cookies alone may not provide enough protection from attacks with very high packet rates as reported in [7].

Unlike most previous work, the main focus of this paper is to develop security mechanisms tailored to CDNs. Recent work by Jung *et al.* that addresses the problem of differentiating attack traffic from surges due to flash crowds in CDNs [14] is closely related to our effort, though different in two key ways. First, our scheme differentiates attack traffic from legitimate requests *on-line* as they arrive, while Jung *et al.* provide a post-mortem analysis to characterize DDoS attack and flash crowd traffic patterns. Second, our scheme improves the resilience of a CDN-hosted site by filtering out attack packets whereas Jung *et al.* proposed a mechanism to dynamically redirect traffic

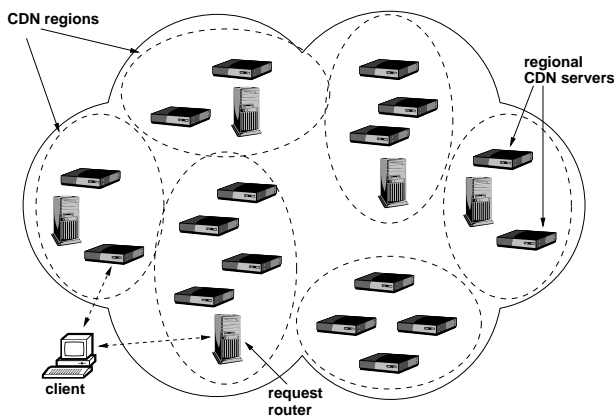


Fig. 1. Content distribution network model

from an overloaded server to less loaded ones.

III. PROBLEM FORMULATION

A. CDN model

We consider a content distribution network model consisting of CDN servers or server clusters distributed in multiple regions and serving replicated Web content. Regions may be arbitrarily defined, though they typically have some topological or geographic significance. Servers deployed in a single ISP network or autonomous system, for example, could be part of a single CDN region.

Each CDN server in a region is shared in that it serves content of multiple Web sites (i.e., customers of the CDSP). There also may be an origin server which acts as the authoritative source of content for a particular content provider. We do not make any assumptions about how content is distributed to servers or which mechanisms are used to deal with application-level issues such as expiration or consistency. Figure 1 illustrates the model.

Clients access content from the CDN servers by first contacting a request router which directs the client to a CDN server within the appropriate region (where the region is chosen based on client proximity, for example). We assume that, within a region, the performance received by the client will be roughly equivalent for any CDN server in the region. This assumption is consistent with recent work showing that fine-grained load balancing does not generally result in improved performance [15]. Moreover, in certain types of CDNs, each region may have only a few clusters of servers. In this case, all servers in the same cluster provide similar performance.

We assume the request router bases its decision on the client IP address, perhaps along with other information about the state of the network or the candidate servers, or about the content being requested. In practice, the

request router may be a specialized DNS server or Web server which chooses a proximal server when the client makes a name resolution request or HTTP request, respectively [16]. If the request router is a DNS server, then additional modifications (e.g., as proposed in [17]) or techniques (e.g., in [18]) are necessary to expose the client IP address.

We note that our proposed mechanisms are designed to operate for the CDN infrastructure in each region. In other words, the proposed request routing algorithm selects a target CDN server only from the local region. Similarly, our site allocation algorithm makes allocation decision per region.

B. Assumptions about the attack

In this paper, we assume that the primary target of DDoS attacks on a CDN is the CDN servers. This assumption is based on observations that the attacker can overwhelm a server with a relatively small volume of attack traffic [7], [19]. The attacker need not generate enough packets to consume the network bandwidth on the links connecting the CDN server to make it unavailable.

We also assume that the request routers are less susceptible to DoS attacks than the CDN servers. Unlike CDN servers, which may perform complex operations, such as dynamic page creation and database query processing, the request routers typically handle simple request and response operations without having to establish connections or maintain much state (e.g., when using DNS-based request routing). Thus, we consider attacks against CDN servers to pose a more immediate threat.

While certain types of DDoS attacks utilize ICMP or UDP packets, recent empirical observations reveal that the majority (more than 90% in the study) of attacks use TCP packets [7]. Also, the observed attack packets commonly had source IP addresses following a random uniform distribution, indicating that source address spoofing is widely used in DDoS attacks. Hence, our focus is on flooding attacks using TCP SYN packets with spoofed source IP addresses. By spoofing the source IP address, the attacker will try to hide the true origin of the attack [10], [11] and increase the effectiveness of the attack. Note that this is an extremely difficult type of attack to address, since TCP SYNs are ‘legitimate’ packet types that must be handled by the CDN server – there is no distinction between a genuine user request and a bogus one. Attacks which use actual (i.e., not spoofed) IP addresses are not handled by our scheme. If the set of actual IP addresses is not very large, it may be possible to detect a surge of traffic from certain IP address groups, and block those address regions using stateful packet filtering. Very large-scale attacks using

legitimate addresses (e.g., attacks started by worms like ‘‘Code Red’’ [20]), however, are much more challenging, and countering such attacks is not part of the goals of this paper.

C. Quantifying resilience

We begin with a simple notion of *resilience* of a hosting environment. Intuitively, we say server A is more resilient than server B if the attacker must send more attack traffic to bring down server A than to bring down server B [7]. Here, ‘‘bring down’’ refers to the same unavailability condition at both servers (e.g., 95% of the requests to each site experience time-outs). Thus, we can define the relative resilience of server A with respect to server B as follows:

Definition 1—Resilience of a server: Server A is k times more resilient than server B if k times more attack traffic is required to make server A unavailable than to make server B unavailable, where unavailability is defined identically for servers A and B .

When we view the CDN as a network of n servers that replicate the full content of the origin server, it is straightforward to show that the CDN provides $O(n)$ resilience compared to a single server because it takes roughly n times more attack traffic to bring down all n CDN servers than to bring down a single server. Some existing CDN architectures do not provide $O(n)$ resilience, however, since they require the index page of a site to be retrieved from the origin server. In this case, the origin server becomes a single point of failure. Nevertheless, many CDSPs are now providing ‘‘whole-site’’ hosting which achieves $O(n)$ resilience.

Our second metric quantifies the degree of *isolation*, or protection, of a Web site from an attack on another site hosted by the same CDN. For example, consider a CDSP hosting two customer Web sites A and B . Ideally, a DDoS attack on site A should not affect the performance or availability of site B , which is true when the two sites are not assigned to any common CDN servers. In practice, though, a given CDN server is shared among multiple Web sites in order to provide good performance to clients and to ensure that the CDN servers are well-utilized.

Hence, there is a tension between achieving site isolation and providing good performance. Our goal is to maximize the number of servers hosting each site while *guaranteeing* a specified degree of isolation among them. One simple way to quantify the degree of isolation is to count the number of CDN servers that are not shared by the Web sites of interest:

Definition 2—Isolation between two sites: Let A and B denote two independent Web sites, and $S_A =$

$\{s_1, \dots, s_l\}$ and $S_B = \{s'_1, \dots, s'_k\}$ denote the sets of CDN servers allocated to A and B , respectively. We define the *degree of isolation* between A and B to be $\min(|S_A - S_B|, |S_B - S_A|)$. For example, if $S_A = \{s_1, s_2, s_3\}$ and $S_B = \{s_2, s_3, s_4, s_5\}$, the degree of isolation is 1 because $|S_A - S_B| = |\{s_1\}| = 1$ and $|S_B - S_A| = |\{s_4, s_5\}| = 2$.

In Section V, we show that if sites are assigned to an equal number of servers, then the degree of isolation between two sites is $\frac{d}{2}$ where d (which is always even) denotes the number of disjoint CDN servers.

IV. HASH-BASED REQUEST ROUTING

In this section, we present a novel algorithm, called hash-based request routing, that can significantly improve the resilience of the CDN against DDoS attacks. The key idea of hash-based request routing is to treat requests with legitimate source IP addresses differently from bogus requests with spoofed source IP addresses so that most of the attack packets are preferentially dropped when the CDN is overloaded. In particular, the proposed request routing scheme helps the server to filter out $\frac{n-1}{n}$ fraction of the attack traffic without inhibiting legitimate requests, when n CDN servers are hosting the Web site in a region.

A. Algorithm description

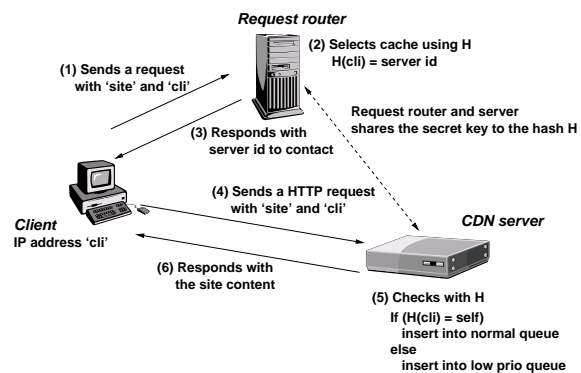


Fig. 2. Operation of hash-based request routing

When a client wants to access a CDN-hosted Web site, it first contacts a request router to find the IP address of the appropriate CDN server to contact. In general, the request routing decision is based on performance or load-balancing metrics. In our approach, however, the hash-based request routing aims to differentiate legitimate requests from potential attack traffic with spoofed IP addresses. This goal is achieved with simple keyed hashing using a secret key shared between request routers and CDN servers [21]. The basic idea behind keyed hashing

is to generate a uniformly distributed pseudo-random sequence that cannot be predicted by an attacker who does not know the secret key. The hash function output is then mapped to identifiers of CDN servers in the region. The operation of the hash-based request routing algorithm is described below (see Figure 2).

- 1) The client first sends a request to a request router to learn the address of the CDN server to contact in order to access a particular Web site.
- 2) The request router selects a CDN server based on the requested Web site and the source IP address of the client using a keyed hash function H , and the secret key K which is shared with the CDN servers. We assume a *simple uniform* keyed hash function $H : \text{IP addr} \rightarrow \text{CDN server id}$. In other words, any given IP address is equally likely to hash into any CDN server id in the region.
- 3) The request router responds with the address of the CDN server to contact. Note that the attacker *must* use a legitimate source address to query the request router because it will not receive the response otherwise. This also implies that the attacker can discover mappings only for the IP addresses of the compromised hosts that it has a full control over.
- 4) Upon receiving the response from the request router, the client sends a TCP SYN packet to the server with CDN server id. When the CDN server receives the SYN packet it verifies if the source IP address in the SYN packet hashes to its own address using the hash function H and the shared key K .
- 5) If the hash value matches its own address, the CDN server inserts the SYN packet into the “normal” service queue. Otherwise, the SYN packet is inserted into the “low priority” queue. The low priority buffer is small, and is meant for packets that may be misdirected due to temporary inconsistencies in K between the request router and the server. Packets in the normal queue are always served before those in the low priority queue.
- 6) Once the SYN packet is processed by the server, a SYN-ACK packet is returned to the client and connection establishment and object retrieval proceeds normally.

There are a few things to note in this procedure. First, the keyed hash function (Step 2) can dynamically change its behavior by simply changing the key. We discuss how this feature can further improve the robustness of the scheme in Section IV-B. Second, the attacker cannot discover the mapping for arbitrary IP addresses by permutation because it cannot always receive the

responses for them from the request router (Step 3). Third, the proposed scheme does not generate reverse traffic, e.g., TCP SYN-ACK, in response to the attack traffic. Therefore, its behavior to the rest of the network is much ‘nicer’ than the other SYN flood countermeasures such as SYN cookies. Finally, as we will see in Section IV-C, the resilience of the CDN improves even with a few CDN servers in a region, i.e., n can be small.

Resilience of hash-based request routing: Recall that the attacker will spoof the source IP address to hide the true origin of attack and to mimic traffic from real users. When spoofing the IP address, the attacker will try to guess an address that will pass the hashing test at a CDN server. However, the pseudo-randomness of the hash function ensures that the attacker’s guess is no better than a random selection.

Suppose the attacker spoofs the source address by randomly selecting an IP address. From our assumption of a uniform hash function H , statistically only $\frac{1}{n}$ fraction of the total attack traffic, for n CDN servers in a region, will pass the test at the server and enter the normal service queue. The other $\frac{n-1}{n}$ fraction of the attack traffic will fail the test and be placed in the low priority queue. Since the low priority queue has only a limited amount of buffer space as described above, most of the attack traffic will be silently dropped. Therefore, valuable resources at the server will not be wasted by the attack traffic.

From the attacker’s perspective, this scheme requires significantly more attack traffic to bring down a server. In a conventional CDN, the attacker must bring down all n CDN servers before the site is unavailable. With the addition of hash-based request routing, each CDN server will accept only $\frac{1}{n}$ of the attack traffic, thereby increasing by a factor of n the amount of traffic necessary to bring down each server. Thus, with this simple scheme in place, the attacker must generate $O(n^2)$ attack traffic in aggregate to victimize all of the CDN servers in the region.

The proposed scheme effectively addresses the problem of distinguishing attack traffic from a flash crowd [14]. In the case of an attack, ideally, the server should ignore the incoming requests without taking any action. On the other hand, in the case of a flash crowd, the server should try to service all user requests in the order of arrival, perhaps at the cost of increased response time. Our scheme roughly follows this ideal behavior by dropping most of the attack packets.

B. Practical considerations

Updating the shared secret: While it is expected that attackers can discover the address-to-server mappings only

for the hosts they infiltrate, they may learn the mapping for other addresses through various channels, for example, by eavesdropping on responses on the network. Although the attacker may not have the resources or time to learn all the mappings, the attacker may still manage to discover a large portion of it.

We can further impede the attacker by periodically changing the secret key K , thus invalidating any mappings learned by the attacker in the previous periods. Note that the CDSP can distribute a set of shared keys to request routers and servers in advance, allowing CDN servers and request routers to independently update the hash function based on the time of day, for example. To allow for temporary inconsistencies, the CDN server maintains a “hand-off” period, during which both old and new mappings are honored. With this added level of protection, it becomes significantly harder for the attacker to discover the mapping.

Overhead of the proposed scheme: The hash-based request routing scheme does add extra overhead to the CDN infrastructure in terms of key distribution and hash computation. Distribution overhead can be mitigated, however, by piggybacking the secret keys on control traffic from the administration site of CDSP. The hash computation at the request routers and the CDN servers must be optimized since they are on the critical path of object retrieval. In general, we can employ any uniform hash function, such as MD5, that generates a pseudo random sequence with a reasonable overhead.

To avoid the situation where a high-overhead hash computation is itself exploited by an attacker, the hashing algorithm may be designed for efficient implementation in hardware [22]. Snoeren *et al.* have demonstrated the feasibility of hardware-based fast hashing in the context of high speed routers, which compute multiple hashes for every forwarded packet [10]. A similar technique can be employed at the request routers and the CDN servers for fast hashing.

C. Evaluation

In this section, we present a few performance numbers as a proof of concept. We note that SYN cookies are widely used to protect servers from SYN flood attacks in real world. Hence, in this paper, we try to quantify the benefit of the hash-based request routing scheme *in addition to* SYN cookies.

We set up a testbed consisting of one Web server and four clients, all running Linux 2.4.7 on Pentium III 500 MHz PCs (Figure 3). Among the clients, one is assumed to be a legitimate user and the other three serve

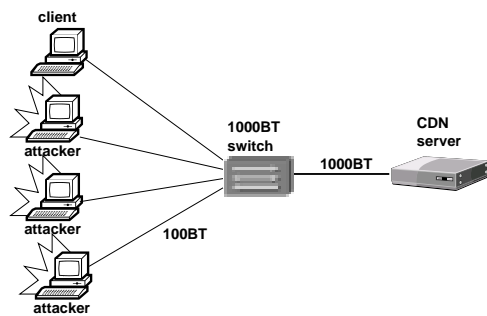


Fig. 3. Testbed configuration

the role of attackers. The machines are connected via an Alteon ACESwitch 180 Gigabit Ethernet switch. The clients are connected to the switch via fast Ethernet while the server is connected via Gigabit Ethernet to ensure that network does not become the bottleneck. The server runs Apache 1.3, and serves local copies of the CNN.com.

In this setting, we examine the performance of a *single* CDN server as if the server is a part of a CDN that employs hash-based request routing. We emulate the existence of other CDN servers in the region by controlling the hash parameter. Recall that the proposed hash-based algorithm can filter out $\frac{n-1}{n}$ fraction of attack packets at each server if there are n CDN servers collaborating in the same region. To simplify the test, we assume that the client and the attackers have cached the response from the request router and know the IP address of the server.

The attackers generate spoofed TCP packets at a specified rate using raw IP sockets. We use `httperf` [23] to generate request traffic from the legitimate client. For each scenario, five sets of data were collected, where each data consists of results from 1,000 accesses to the Web site. At the server, the Linux SYN cookies implementation is turned on as the basic protection against SYN flood attack. We note that, without SYN cookies, the availability of the server is compromised at a much lower attack rate.

Figure 4 presents the number of timed out requests in the 1,000 accesses from the legitimate user, where timeout is set to 15 seconds. The x-axis represents the attack rate (in packets/sec) on each server. The figure plots the results from the ‘SYN cookies only’ case in comparison with the ‘SYN cookies+hashing’ case, where the number of servers in the region is assumed to be 2, 3, 4, and 5. From the figure, we observe that the proposed hashing scheme reduces timeout events, thereby providing substantially better protection than the case when only SYN cookies used. For example, with the hashing scheme, the legitimate user does not experience timeouts at the attack rate of 3,500 packets/sec, whereas 50% of the requests timeout when only SYN cookies were used. We also observe that the level of

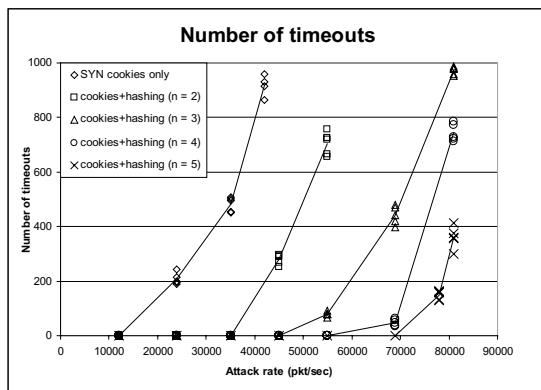


Fig. 4. Number of timeouts vs. attack traffic rate (packets/sec)

protection increases as the number of CDN servers in the region increases.

A similar trend can be found from Figure 5 for the average response time of the HTTP requests that did not timeout. As in the previous case, the average response time is significantly lower when the hashing scheme is used than the case without. For the same attack rate, the response time decreases with the number of CDN servers in the same region.

The improved performance in the hashing case results from the fact that the hash-based filtering reduces both inbound and outbound traffic at the server. Figure 6 presents the aggregate packet rate (attack traffic + user requests) into the SYN queue. The figure shows the inbound traffic is greatly reduced with the hashing scheme as compared to the ‘SYN cookies only’ case. Similarly, Figure 7 plots the outbound traffic from the server. From the figure, we observe that the hashing scheme reduces bogus SYN-ACK traffic sent from the server out into the network, which is an added benefit of our hashing scheme. Recall that SYN packets contain spoofed addresses; therefore, outbound SYN-ACKs are destined to random hosts all over the network [7].

In summary, we conclude that, with our proposed hash-based request routing scheme, the attack rate required to bring down a single CDN server increases proportionally to the number of servers in the same region. Consequently, the aggregate attack traffic required to compromise all the servers in a region increases quadratically as the number of servers. In addition to the robustness against SYN flood attacks, our hashing scheme reduces the unwanted outbound SYN-ACK traffic.

V. ISOLATING THE IMPACT OF THE ATTACK

In this section, we outline strategies to allocate Web sites to different CDN servers in order to isolate the impact of an attack on any individual site. Ideally, we want

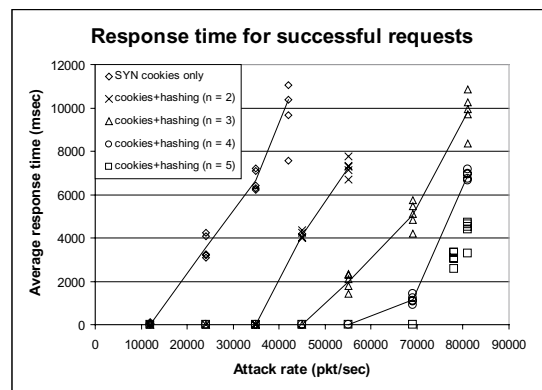


Fig. 5. Response time vs. attack traffic rate (packets/sec)

to allocate Web sites to as many servers as possible while ensuring that response time and throughput of the site deteriorate as little as possible when other CDN-hosted sites are under attack. In particular, we want to be able to *guarantee* at least a specified minimum degree of isolation between any two sites in the CDN, while providing good performance.

Intuitively, however, allocating a large fraction of the available CDN servers to each of two different sites results in significant overlap in the set of servers hosting both of the sites. Thus, an attack which brings down the servers hosting one site also collaterally causes a large loss of service for the other site. Therefore, we have the following two conflicting goals:

- 1) For each Web site, we wish to serve the site from a large number of CDN servers in each region.
- 2) For any pair of Web sites, if one is the target of a DDoS attack then the other should experience minimal service degradation.

The first goal maximizes the throughput of a site hosted by the CDN, and the second protects a Web site from attacks on any other site.

Our contributions in this paper are to relate the problem of site allocation to a coding theoretic framework and obtain good allocation strategies by adapting carefully chosen codes. We consider the case where there is one level of service, where each Web site is hosted on the same number of CDN servers in each region. In ongoing work we are investigating similar allocation strategies to offer multiple levels of service, for example to allocate more servers to more popular sites.

We note that allocating Web sites to a *subset* of CDN servers in the region requires modification of the hash-based request routing scheme. CDN servers now must use both the client IP address and the requested site in determining whether an arriving packet is legitimate. Without establishing the TCP connection, however, the server can-

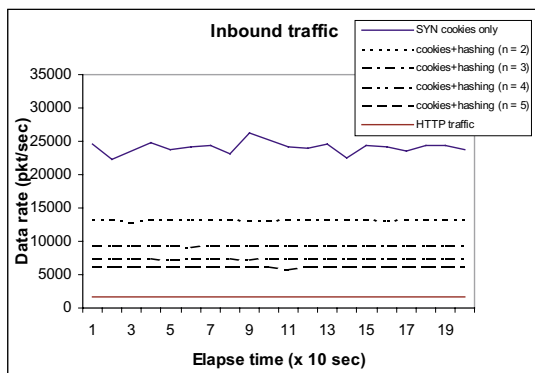


Fig. 6. Inbound traffic to the server's SYN queue

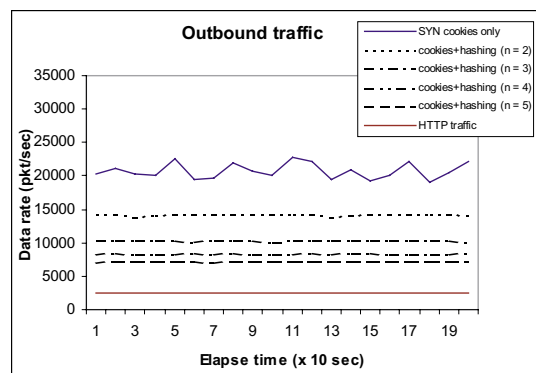


Fig. 7. Outbound traffic from the server

not automatically determine which site is being requested. Several techniques can be used to convey the site information and we outline one simple scheme in Section V-E.

A. Relating site allocation to codes

Let \mathcal{S} be the set of CDN servers and \mathcal{W} the set of Web sites to allocate to the servers in \mathcal{S} . For each site $w \in \mathcal{W}$ form the bit vector of length $|\mathcal{S}|$ with bit i set if w is allocated to server i . This bit vector is called the *allocation vector* for the site.

Following standard terminology, the *Hamming weight* of a binary vector is defined to be the number of 1s in the vector. The Hamming weight of the allocation vector of a site represents the number of CDN servers that serve content for this Web site. Hence, our goals may be restated as:

- Requirement (1) states that allocation vectors of each site have the largest Hamming weight possible.
- Requirement (2) is that for every pair of sites w_1 and w_2 , the number of servers which serve w_1 but not w_2 (and vice versa) is as large as possible, i.e. if s_1 and s_2 are the allocation vectors of w_1 and w_2 respectively, then the Hamming weights of the bit vectors $(s_1 \wedge \overline{s_2})$ and $(s_2 \wedge \overline{s_1})$ should be as large as possible.¹

When all Web sites are treated equally, i.e. when all allocation vectors have equal Hamming weight, we have

$$\begin{aligned} & \text{Hamming weight}(s_1 \oplus s_2) \\ &= 2 \times \text{Hamming weight}(s_1 \wedge \overline{s_2}) \\ &= 2 \times \text{Hamming weight}(s_2 \wedge \overline{s_1}), \end{aligned} \quad (1)$$

where $a \oplus b$ denotes XOR of vectors a and b . The Hamming weight of $(s_1 \oplus s_2)$ is called the *Hamming distance* between s_1 and s_2 . Thus, in this restricted case, our problem is to find allocation vectors with large Hamming weight under the constraint that the Hamming

¹ \overline{x} denotes the ones-complement of the binary vector x .

distance between the vectors is as large as possible. Thus the problem of site allocation can be stated as follows:

Allocation problem: Given n , the number of CDN servers in a region, find an efficient algorithm to enumerate a large number of binary vectors each of length n , each vector having Hamming weight exactly h (as large as possible) and the minimum pairwise Hamming distance d between vectors being as large as possible.

Given such an algorithm, we can sequentially generate such n bit vectors and assign them as the allocation vectors for each Web site. Under such an allocation each Web site is served by h CDN servers out of n . If the servers hosting a particular site are all rendered inoperative due to a DDoS attack, then any other site is *guaranteed* to be served by at least $\lfloor \frac{d}{2} \rfloor$ servers. Each Web site thus utilizes $\frac{h}{n}$ of the available capacity and the resulting loss of service when any one Web site is taken down is at most $(1 - \frac{d}{2h}) * 100$ percent.

B. Allocation strategies from binary codes

In this section, we outline general allocation methods where we try to maximize h and d along with maximizing m , the total number of Web sites which we can accommodate. Later, in Section V-D we consider other possibilities such as the case when the CDSP has a target number of customers and wishes to find the best allocation (i.e., given m , optimize h and d). Our allocation strategies are adaptations of results from coding theory and we outline general constructions without details of actual codes. We refer the interested reader to [24] for comprehensive details about codes and their constructions.

In general, defining codes with many vectors where all codewords have exactly a fixed Hamming weight, is a very difficult problem in coding theory. The few codes

that exist to generate constant Hamming weight codewords, called *constant weight codes*, generally yield only a small (polynomial in the length of the code) number of codewords. To accommodate a larger number of Web sites, we take arbitrary codes and prune them to yield binary vectors fitting our specification. Our first cut at an allocation strategy is the following naive algorithm:

Algorithm 1: Fix a code of length n with a large minimum distance d . Choose parameter h so that there are enough codewords of Hamming weight h . Then generate all binary vectors with Hamming weight exactly h and output only those vectors which belong to the code.

Note that in the algorithm we have not fixed particular values for d or h . To do this we first fix a code from a family of codes which fixes the parameter d . Once we fix a code, the distribution on the Hamming weight of the vectors is defined. The parameter h is then chosen to have an allocation for at least m Web sites based on this distribution of Hamming weights.

Besides finding good values for h and d , we also wish to use codes which have explicit constructions and efficient algorithms to enumerate codewords. A particularly good class of codes which have easy algorithms to identify codewords are the class of linear codes. This includes a number of codes such as the Reed–Solomon codes [24].

Definition 3: An (n, k, d) code is a linear code of length n with minimum distance d and the dimension of the linear subspace is k . It is defined by a $k \times n$ binary matrix G called the generator matrix and the set of codewords is obtained by $x \times G$ where x ranges over all binary vectors of length k [24].

Note that linear codes produce codewords with arbitrary Hamming weight. The algorithm to generate codewords is straightforward: Sequentially enumerate vectors of length k and multiply by the generator matrix G . Alternately, a linear code is also defined by its *syndrome matrix* C [24], an $(n - k) \times n$ binary matrix: a n length word x belongs to the code if and only if $x \times C^T = 0$. Using properties of linear codes, our next refinement is the following:

Algorithm 2: Fix an (n, k, d) linear code with a large minimum distance d . Systematically generate all binary vectors of Hamming weight h and retain words x such that $x \times C^T = 0$. Alternately, enumerate vectors y of length k and generate codeword $y \times G$. Retain only those with Hamming weight h .

As before, the parameters d and h are chosen by first fixing a family of linear codes to define d . Once the code is fixed, h is chosen to maximize the number of

codewords with Hamming weight h in this code.

We describe another general scheme to obtain allocation vectors, which focuses on a particular value for h . Intuitively, if h is too large, then there are few codewords of Hamming weight h . Also, choosing too large a value for h makes the maximum distance (which can be at most $n - h$) small. On the other hand if h is small, then each Web site is served by at most h CDN servers and thus results in wasted capacity. A particularly good value for h is $\frac{n}{2}$: this is the weight at which we have the maximum number of binary vectors and hence potentially a large number of codewords. For $h = \frac{n}{2}$ we can use the following:

Algorithm 3: Fix a code \mathcal{C} of length n with minimum distance d . Define a modified code \mathcal{C}' such that for each codeword $c \in \mathcal{C}$, \mathcal{C}' contains the $2n$ length word $c' = c\bar{c}$.

In the modified code \mathcal{C}' , each codeword has length $2n$ and weight exactly n (half the length of the code). The minimum distance between words in \mathcal{C}' is at least $2d$. This is a quick way to use any code to produce words of constant Hamming weight with $h = \frac{n}{2}$. The number of codewords in \mathcal{C}' is the same as that of \mathcal{C} , but now each codeword can be used as an allocation vector.

These algorithms are general methods to convert codes into allocation strategies for Web sites to CDN servers. Plugging good codes into the constructions yields good allocation strategies. The equivalence holds in the other direction: any allocation strategy can be converted into a code. This equivalence is useful to verify if allocation strategies with certain parameters are possible: There are a number of tables [25] which list (for small values of n), given values for h and the distance d , the maximum number of codewords possible in such a code.

C. Example

As an example, suppose that the CDSP wishes to host 100 Web sites with the guarantee that if a Web site is attacked, all remaining sites have at least 3 functioning servers in the CDN region. Restated, the problem is: *given* $m = 100$ and minimum distance $d = 6$, find optimal values for n and h (See Table I). The first step is to find the minimum value of n for which a code with distance $d = 6$ and at least $m = 100$ codewords exists. From standard tables (see [26]), we see that the minimum possible value for n is 15. Our first cut is to use a very specialized non-linear code [26] which yields about 128 codewords with Hamming weight 8 and with length $n = 16$. This is a fairly optimal allocation strategy using an esoteric non-linear code.

TABLE I

EXAMPLE SITE ALLOCATION FOR 100 WEB SITES WITH DEGREE OF ISOLATION = 3 ($m = 100$ AND $d = 6$)

	# Total server	# Servers/site	Code
Algorithm 1	$n = 16$	$h = 8$	specialized non linear code [26]
Algorithm 2	$n = 21$	$h = 11$	Reed-Solomon code
Algorithm 3	$n = 30$	$h = 15$	Hamming code

Another allocation can be obtained using the Reed–Solomon code of length $n = 21$ with a distance of $d = 5$ which yields 512 codewords. Inspecting the distribution of the number of codewords for each Hamming weight, we find the number of codewords is maximized at weights 10 and 11. We choose $h = 11$ and select only codewords of weight 11 which yields 126 codewords. For these constant weight words, the distance (which must be even) is actually 6 which matches the parameters we require.

A slightly less optimal, but straightforward, allocation is to use Algorithm 3 choosing the code \mathcal{C} to be the Hamming code of length 15 and distance 3. With our parameters the Hamming code has 2048 codewords. Plugging this code into the Algorithm 3 gives us an easily implementable allocation strategy where $n = 30$ and each Web site is assigned to at least 15 CDN servers. While not optimal, the code yields a large number of codewords which gives us the flexibility to expand to more Web sites.

We have chosen these codes from many possibilities, to illustrate the tradeoffs. Optimal codes generally tend to be non-linear with complex encoding algorithms. Straightforward choices for codes such as the Reed–Solomon code give us slightly less optimal values of n .

D. General Allocation Strategies

In this section we discuss a number of possible allocation strategies, using various codes to place different emphasis on the number of hosted Web sites (m), number of CDN servers per hosted site (h), and the degree of isolation between sites (d). Table II summarizes the trade-offs of site allocation strategies using these codes.

Allocations for a small number of Web sites: Our first case is when m , the number of Web sites, is small compared to n , the number of CDN servers. If $m \leq (2n - 2)$ we can use subsets of Hadamard codes [24], [25] and get very good guarantees on the Hamming distance. The Hadamard code is a $(n, \log(2n), \frac{n}{2})$ linear code with $2n$ codewords. In fact, using these codes one can construct $2n - 2$ binary vectors each with a Hamming weight $\frac{n}{2}$ with minimum pairwise distance $\frac{n}{2}$. With these as allocation vectors, we can assign each site to $\frac{n}{2}$ servers and guarantee that a site will always be served by $\frac{n}{4}$ servers,

even if all the servers hosting one Web site become unavailable. For small values of m , we can therefore get very good guarantees on resilience.

Codes with efficient algorithms: Besides optimization of parameters, we need efficient constructions of allocation vectors. A good class of codes with a large minimum distance and efficient algorithms are Reed–Solomon codes. Choosing parameters carefully, and using Algorithm 3 stated above, given n , we can use Reed–Solomon codes to enumerate an exponential number ($2^{c_1 n}$) of codewords with a minimum distance of at least $\frac{n}{c_2 \log(n)}$, where c_1 and c_2 are constants. Thus, we can guarantee that no Web site will suffer more than a $\log(n)$ factor drop in service under attack. Although this is high, these codes have the advantage that allocation algorithms are easily implemented.

Allocation strategies for a range of parameters: There are a number of advanced codes which can be converted to good allocation strategies. Care should be taken, however, while using more advanced coding methods since they typically have complex algorithms for encoding, and yield the best parameters only for large values of n . One such family of codes are Justesen codes [24]. Plugging these codes in Algorithm 3, gives us an algorithm yielding an exponential number of allocation vectors where each Web site is allocated to $\frac{n}{2}$ CDN servers and we can guarantee that a Web site which is not under attack will at most suffer a small constant factor loss of service.

Ruling out some allocation parameters: While the above are general example of some codes, in practice, the actual code depends entirely on the values of the parameters n , m and desired values for h and d . The first step to finding codes to convert to allocation vectors is to investigate the feasibility. For example, given a particular value for n , there are tables on upper bounds for the number of codewords for various values of d [25]. This directly gives us the maximum number of Web sites which we can accommodate given a value of d . Similarly, there are tables which list upper bounds on the number of constant weight codewords possible given n , d , and h . These can be used to rule out possible values for d and h based on m .

TABLE II
SUMMARY OF THE SITE ALLOCATION STRATEGIES USING CODES

Code	Properties	Comments
Hadamard code [24], [25]	$m = 2n - 2$, $h = \frac{n}{2}$, $d = \frac{n}{2}$	good isolation, small number of sites (m)
Reed-Solomon code [24]	$m = O(2^{c_1 n})$, $h = \frac{n}{2}$, $d = \frac{n}{c_2 \log n}$	balances isolation and number of sites, efficient construction
Justesen code [24]	$m = O(2^{c_1 n})$, $h = \frac{n}{2}$, $d = c_2 n$	good isolation, many sites, higher complexity

E. Conveying site information to CDN servers

The hash-based request routing algorithm, as described in Section IV differentiates bogus packets from legitimate packets based on a hash function that uniquely maps client IP addresses to CDN servers in the region. With the introduction of the site allocation algorithm, however, the request routing algorithm must map a client to a CDN server identifier based on the client address and also the Web site it wishes to access. This can be accomplished by introducing a hash function for each customer Web site. The effectiveness of hash-based request routing may be degraded, though, if a CDN server does not know which hash function to apply when it receives a packet that does not indicate the site being requested (e.g., a TCP SYN packet).

We can address this problem if we can force the client to inform the CDN server which site is being requested when sending the first SYN to establish a connection. One simple way is to encode the site information in the IP address returned by the request router. Specifically, the request router responds with an address whose network portion contains the true network address of the CDN server, but the host portion contains an encoding of $\langle \text{site}, \text{server id} \rangle$.

For example, if the CDN server to contact for site A is 192.19.1.13, the request router may respond with 192.19.212.9, where the network address, 192.19.0.0/16, is the same but the host portion of the address, 212.9, is encoded information about the site A and the server id. When the client sends an initial SYN packet, the packet will be routed to the 192.19.0.0/16 subnet where the CDN server cluster resides. At the entry point, the packet is switched to the correct server using a switch (e.g., a fast NAT box or layer-4 switch), which typically performs load balancing across the CDN servers. In this scheme, the switch forwards the packet to the CDN server based on the $\langle \text{site}, \text{server id} \rangle$ encoding, and the server can compute the hash function to decide if the packet is legitimate. Alternatively, the CDSP may co-locate the hashing functionality at the switch to filter out bogus packets earlier. This scheme is appealing in that it is transparent to the client, and is applicable to CDSPs who deploy a few

clusters of servers in each region.

VI. DISCUSSION

Security vs. performance trade-offs: In most systems security features come at the cost of degraded performance, and our proposed DDoS countermeasures for CDNs face a similar trade-off. As described in Section IV, the hash-based request routing scheme is unlike a standard CDN request routing algorithm that chooses the optimal server based on network or server load, or network proximity. Rather, we assume that each CDN server within a given region provides roughly equivalent performance for clients assigned to that region. To allow further optimization within a region, the approach could be modified to use weighted hash functions, for instance, where the weights are determined using conventional request routing metrics. However, if the request routing function exhibits some predictability based on performance-related information, it may increase the vulnerability of CDN servers to attack. Similarly, the site allocation strategy presented in Section V potentially reduces the performance of an individual Web site by assigning it to fewer CDN servers.

CDN server distribution and footprint: The size and distribution of the CDN influences the effectiveness of our DDoS countermeasures. The mechanisms we propose are directly applicable to large CDSPs which currently operate thousands of CDN servers distributed across many networks. On the other hand, if the CDN is composed of a few large regions, each containing a small number of CDN servers, it may be impossible to find a site allocation that provides sufficient Web site isolation. Similarly, the additional protection afforded by the hash-based request routing is significantly reduced with a small number of servers. The applicability to small CDNs may be improved, however, with the emergence of CDN peering in which multiple, administratively separate CDNs are combined to create a larger virtual CDN with increased reach and distribution.

VII. CONCLUSION

Recent work on countering DDoS attacks typically has focused primarily on attacks targeting a centralized server location or network resources. Increasingly, however, high-profile sites are distributed using CDNs. While CDNs, owing to their distributed structure, promise better resilience to DDoS attacks, the shared nature of the CDN infrastructure introduce unique challenges. In this paper, we proposed two mechanisms to significantly improve the resilience of CDN-hosted Web sites and CDN servers to DDoS attacks: (a) a hash-based request routing scheme that enables CDN servers to effectively distinguish attack traffic from legitimate requests, and (b) site allocation algorithms, based on coding theory, which guarantee a minimum level of availability of the sites that are not directly under attack. Together, these schemes improve the resilience of CDN-hosted Web sites, and complement existing techniques used to counter DDoS attacks. Several issues remain to be addressed in future work. For example, mechanisms are still needed to secure request routers from attack. Also, a CDSP may wish to support multiple levels of service, or to handle cases where some sites require more or fewer servers. Although the direct relation to codes is not valid, we are developing allocation strategies for multiple classes of service using codes.

REFERENCES

- [1] R. Power, "2001 CSI/FBI computer crime and security survey," *Computer Security Issues and Trends*, vol. 7, no. 1, 2001. published by Computer Security Institute <http://www.gocsi.com>.
- [2] P. Ferguson and D. Senie, "Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing." Internet Request for Comments (RFC 2827), May 2000.
- [3] Kihong Park and Heejo Lee, "On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets," in *Proceedings of ACM SIGCOMM*, pp. 15–26, August 2001.
- [4] "Strategies to protect against distributed denial of service (DDoS) attacks." Cisco Systems White Paper, February 2000. <http://www.cisco.com/warp/public/707/newsflash.html>.
- [5] F. Kargl, J. Maier, and M. Weber, "Protecting Web servers from distributed denial of service attacks," in *Proceedings of International World Wide Web Conference*, vol. 10, May 2001.
- [6] D. J. Bernstein, "SYN cookies." <http://cr.yp.to/syncookies.html>, November 2001.
- [7] David Moore, Geoffrey M. Voelker, and Stefan Savage, "Inferring Internet Denial-of-Service Attack," in *Proc. USENIX Security Symposium*, August 2001.
- [8] H. Burch and B. Cheswick, "Tracing anonymous packets to their approximate source," in *Proc. USENIX Systems Administration Conference (LISA)*, (New Orleans, LA), December 2000.
- [9] G. Sager, "Security fun with OCxmon and cflowd." Presentation to Internet-2 Measurement Working Group, November 1998. <http://www.caida.org/projects/ngi/content/security/1198/>.
- [10] Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Stephen T. Kent, and W. Timothy Strayer, "Hash-based IP Traceback," in *Proceedings of ACM SIGCOMM*, pp. 3–14, August 2001.
- [11] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson, "Practical Network Support for IP Traceback," in *Proceedings of ACM SIGCOMM*, pp. 295–306, August 2000.
- [12] S. M. Bellovin, M. D. Leech, and T. Taylor. Internet draft (draft-ietf-itrac-01.txt), April 2002.
- [13] CERT Coordination Center, "TCP SYN flooding and IP spoofing attacks." CERT Advisory CA-1996-21, September 1996. <http://www.cert.org/advisories/CA-1996-21.html>.
- [14] J. Jung, B. Krishnamurthy, and M. Rabinovich, "Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites," in *Proceedings of 11th WWW Conference*, May 2002.
- [15] B. Krishnamurthy, C. Wills, and Y. Zhang, "On the use and performance of content distribution networks," in *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, November 2001.
- [16] A. Barbir *et al.*, "Known CN request-routing mechanisms." Internet Draft (draft-ietf-cdi-known-request-routing-00.txt), February 2002.
- [17] A. Shaikh, R. Tewari, and M. Agrawal, "On the effectiveness of DNS-based server selection," in *Proc. IEEE INFOCOM*, (Anchorage, AK), April 2001.
- [18] Z. M. Mao, C. D. Cranor, F. Douglis, M. Rabinovich, O. Spatscheck, and J. Wang, "A precise and efficient evaluation of the proximity between web clients and their local DNS servers," in *Proceedings of USENIX Annual Technical Conference*, June 2002.
- [19] CERT Coordination Center, "Denial of service attacks." CERT Tech Tips, June 2001. http://www.cert.org/tech_tips/denial_of_service.html.
- [20] CERT Coordination Center, "Code Red worm exploiting buffer overflow in IIS indexing service DLL." CERT Advisory CA-2001-19, July 2001. <http://www.cert.org/advisories/CA-2001-19.html>.
- [21] Mihir Bellare and Ran Canetti and Hugo Krawczyk, "Keyed Hash Functions for Message Authentication," in *Proceedings of CRYPTO*, pp. 1–15, August 1996.
- [22] Hugo Krawczyk, "LFSR-based hashing and authentication," in *Proceedings of CRYPTO*, pp. 129–139, August 1994.
- [23] D. Mosberger and T. Jin, "httpperf: A tool for measuring web server performance," *ACM Performance Evaluation Review*, vol. 26, pp. 31–37, December 1998.
- [24] R. E. Blahut, *Theory and practice of Error-Control Codes*. Addison Wesley, 1983.
- [25] V. Pless and W. Huffman, eds., *Handbook of Coding Theory*. Elsevier, Amsterdam, 1998.
- [26] M. Best, A. Brouwer, F. MacWilliams, A. Odlyzko, and N. Sloane, "Bounds for Binary Codes of Length less than 25," in *IEEE Trans. Information Theory*, pp. 81–93, January 1978.