2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks

Catch Me if You Can: A Cloud-Enabled DDoS Defense

Quan Jia, Huangxin Wang, Dan Fleck, Fei Li, Angelos Stavrou, Walter Powell
George Mason University
Fairfax, VA 22030
{*qjia, hwang14,dfleck, fli4, astavrou,wpowell*}*@gmu.edu*

***Abstract*—We introduce a cloud-enabled defense mechanism for Internet services against network and computational Distributed Denial-of-Service (DDoS) attacks. Our approach performs selective server replication and intelligent client reassignment, turning victim servers into moving targets for attack isolation. We introduce a novel system architecture that leverages a "shuffling" mechanism to compute the optimal reassignment strategy for clients on attacked servers, effectively separating benign clients from even sophisticated adversaries that persistently follow the moving targets.**

We introduce a family of algorithms to optimize the runtime client-to-server re-assignment plans and minimize the number of shuffles to achieve attack mitigation. The proposed shuffling-based moving target mechanism enables effective attack containment using fewer resources than attack dilution strategies using pure server expansion. Our simulations and proof-of-concept prototype using Amazon EC2 [1] demonstrate that we can successfully mitigate large-scale DDoS attacks in a small number of shuffles, each of which incurs a few seconds of user-perceived latency.

***Keywords*-DDoS, Moving Target Defense, Shuffling, Cloud**

I. Introduction

Distributed Denial-of-Service (DDoS) attacks pose one of the severest security threats to today's Internet services. A recent worldwide survey [2] reports both increased DDoS attack intensity and sophistication over the past few years. Popular open web services such as e-commerce, and security-sensitive web services including financial and banking websites, are among the top targets of the aggravating DDoS attacks [2], [3]. However, practical DDoS mitigation still primarily depends on packet filtering using access control lists (ACLs) and traffic signatures [2], despite their functional and operational limitations [4].

This paper introduces a cloud-enabled, shuffling-based moving target defense mechanism that confuses, evades, and isolates attackers that attempt to blend with benign users and mount DDoS attacks. Unlike previous moving target solutions [5], [6] that were designed for restricted applications and threat models, this work provides a comprehensive mechanism that protects generic Internet services from both network and computational DDoS attacks alike. By turning the protected servers into moving targets through carefully planned "shuffling" operations, the proposed architecture not only can evade naive attackers that only target static and invariable server locations, but is also able to segregate sophisticated attackers that can persist their attack to the moving server replicas.

To offer protection for Internet services, our defense takes advantage of the resource elasticity in modern cloud computing environments. When bombarded by a DDoS attack, we perform quick and selective server replication in the cloud. Instead of blindly expanding system capacity on demand, we replicate the attacked server instances and replace them with the newly instantiated replica servers at different network locations. The affected client sessions are migrated to the replacement replica servers. The attacked servers are taken offline and recycled after client migration is completed. The network locations of the new replica servers are not published and are only known to the subset of clients that are assigned to them. Thus, we establish a moving target environment that automatically evades the naive attackers that target static and invariable server locations.

However, sophisticated attackers may follow the "moved" replica servers as benign clients do, and re-target their attack to the new server locations. To segregate those sophisticated and persistent attackers from benign clients, we devised a shuffling mechanism that intelligently assigns clients to the new replica servers. By keeping track of the assignments and whether a new replica is also attacked, we can quickly identify and separate benign client sessions from potentially malicious ones. Through multiple rounds of shuffling, we can progressively quarantine the persistent attackers onto a gradually decreasing set of replica servers while separating benign clients away from them. The shuffling mechanism is effective even against on-and-off attacker strategies that attempt to disrupt our defense.

To quickly segregate the persistent attackers and mitigate the on-going attack, we aim to isolate the persistent attackers onto a small number of replica servers through client-to-server shuffling, while grouping as many benign clients as possible onto the replica servers that are attacker-free. To achieve this, we designed an optimal dynamic programming algorithm to guide client-to-server re-assignments throughout the shuffling operations. On average, this algorithm can separate out the maximum (i.e. optimal) number of benign clients in each shuffle. We also show that a faster greedy algorithm, discussed in [6], is able to produce near-optimal results. Our simulation results show that we can effectively mitigate strong DDoS attacks (100K persistent attackers)

978-1-4799-2233-8/14 $31.00 © 2014 IEEE
DOI 10.1109/DSN.2014.35

by saving 80% of 50K benign clients in approximately 60 shuffles, each of which takes only a few seconds to complete.

We envision the shuffling-based moving target defense as an on-demand service that can be readily deployed to protect a wide-range of Internet services. Unlike [6] that requires client authentication, our mechanism also offers DDoS resilience to open Internet services for anonymous users. By using replica application servers as moving targets to contain the impact of attackers, we are able to quarantine attacks targeting both network and computational resources of the victims. Our approach is especially suitable for mitigating DDoS attacks on web services, for the ease of accomplishing server replication and client migration. To verify this in practice, we implemented a proof-of-concept prototype that places replicated web servers in Amazon EC2 [1]. Experiments with the prototype revealed low user-perceived latencies introduced by client migration between two replica web servers, demonstrating the feasibility and agility of the proposed solution.

In summary, we make the following contributions:

- We introduce a novel shuffling-based moving target defense mechanism that can mitigate large-scale DDoS attacks on generic Internet services. By judiciously replacing attacked servers with newly instantiated replica servers and optimally shuffling client-to-server assignments, our solution can gradually isolate DDoS attacks on network and computation resources, restoring quality of service for benign-but-affected clients.
- We designed and employed a family of algorithms that generates optimal shuffling strategies according to different attack scenarios and scales. Through theoretical analysis and simulation-based evaluations, we demonstrated that our algorithms can direct runtime shuffling operations to achieve maximal attack segregation.
- Our solution is agnostic to the underlying network infrastructure and do not require ISP collaboration. Taking advantage of the resource elasticity and low cost of modern cloud computing services, our mechanism can be easily deployed in the cloud as a scalable and resilient end system against DDoS attacks.
- We implemented a proof-of-concept prototype of the proposed solution using Amazon EC2 [1], and tested it against off-the-shelf browsers and the content of popular websites. Experiments with our prototype indicate that we can quickly mitigate DDoS attacks with minimal user-perceived latency.

II. Design Space

A. Motivation & Related Work

Despite research efforts to address DDoS attacks, they remain potent threats to Internet services. The Internet is comprised of a network core that does little more than forwarding traffic, and sophisticated end systems that handle reliability, quality of service, and security [7]. To effectively stop distributed flooding attacks on a small group of end systems, filtering-based approaches [8], [9], [10] and capability-oriented mechanisms [11], [12], [13], [14] were proposed to filter out malicious traffic and prioritize legitimate traffic in the Internet core. However, implementing these solutions requires the universal upgrade of the legacy Internet infrastructure with corresponding security functionalities. Given the size of today's Internet and the millions of applications depending on it, such an extensive upgrade is unlikely. To overcome the constraints imposed by the physical Internet infrastructure, overlay-based defenses [15], [16], [17], [18], [19], [20] employ static and exposed overlay networks to dilute and absorb attack traffic. However, to withstand strong DDoS attacks, these overlay networks need to own abundant resources, making them quite expensive to build and maintain.

Unlike most traditional security mechanisms that are static in nature, moving target defense [21] advocates controlled changes in network, software, and other system dimensions over time to increase uncertainty and complexity for attackers. For instance, network address randomization [22] dynamically changes the IP addresses of Internet hosts to hamper the propagation of hit-list worms. The fast flux [23] technique employs a fast-changing proxy network to sustain the availability of phishing websites against lawful blocking. To mitigate DDoS attacks, MOVE [5] migrates a protected web server to a new network location when under attack. However, MOVE fails to account for the extended attacks inflicted by persistent attackers that continue to follow a migrated server. To address such advanced threats, MOTAG [6] proposed a lightweight, proxy-based shuffling approach to segregate the following attackers from benign clients. Nevertheless, MOTAG is only capable of mitigating network flooding attacks on Internet services that mandate client authentication. It is not suitable for securing open Internet services designed for anonymous users. Furthermore, neither MOVE or MOTAG offers protection against application-level flooding or other computational DDoS attacks.

This research improves on the concepts implemented in the MOTAG system to defend generic Internet services that support both authenticated and anonymous clients. Examples of such services include but are not limited to web portals, news sites, and e-commerce websites. We propose a shuffling-based moving target defense to protect Internet services against both network and computational DDoS attacks. The proposed solution is deployable as an end system by non-ISP organizations, without the need for upgrading legacy Internet infrastructure or changing client software. In addition, our mechanism is reactive and is triggered only when an attack is detected, incurring minimum maintenance costs under normal conditions.

B. Threat Model and Assumptions

We aim to mitigate network DDoS attacks that employ voluminous junk packets to deplete a victim's bandwidth or connections, and computational DDoS attacks that use maliciously crafted traffic to exhaust a victim's limited computation resources (CPU, memory, etc). These attacks are usually performed by attacker-controlled botnets that are composed of large numbers of infected machines, or bots. We assume botnets are capable of bombarding standalone physical servers on the Internet. To combat that, our solution proposes to deploy the protected services in the cloud that enables dynamic server replication and substitution. We assume botnets can swamp individual cloud servers or congest the Internet links of certain cloud domains. However, we also assume that botnets are not able to overwhelm the aggregate computation power or Internet bandwidth of all cloud data centers that host our servers.

We use the DNS service to redirect incoming clients to the cloud domains where the protected servers are deployed. We assume the DNS servers are well-provisioned and DDoS attacks against the DNS service are beyond the scope of this paper. Within each cloud domain, clients are redirected to the protected servers by cloud load balancers. To bypass these redirection steps, botnets may use reconnaissance attacks to locate and gain information about our servers.

We realize a cloud-based moving target defense using controlled server replication and substitution, turning the in-cloud replica server instances into moving targets. For that purpose, we consider two types of bots in any botnet, namely naive bots and persistent bots. Naive bots do not have intelligence, and cannot follow moving targets automatically. Instead, they can only attack static IP addresses or DNS names on a hit-list provided by persistent bots. Conversely, persistent bots are able to interact with the environment and adapt to changes. They can follow the moving replica servers autonomously as benign clients do. These persistent bots can act as insiders to launch computational attacks, or instruct naive bots to attack the new network locations of the "moved" replica servers. In practice, persistent bots can be human-controlled bots (botmasters) or bots with advanced programs such as browser-like software that can automatically follow redirects. For simplicity without loss of generality, any replica server assigned with one or more persistent bots is considered under attack in our modeling.

We assume the occurrence of a DDoS attack can be easily detected from indicators including sudden network congestion and abrupt surge of application-level traffic. Advanced traffic analysis techniques, such as [24], [25], can also be used for early detection and the detection of sophisticated attacks. Once an attack is detected, our shuffling-based moving target solution can be applied for attack mitigation and isolation.

III. System Overview

To mitigate DDoS attacks initiated by powerful botnets, we propose a cloud-enabled moving target defense mechanism to isolate the attack. Our goal is to save benign clients by separating them from both naive and persistent bots that attempt to drain the network and computational resources of the protected servers. This goal is accomplished by constructing in-cloud replica servers as moving targets while conducting client-to-server shuffling: when an attack occurs, our defense dynamically instantiates new replica servers to replace the attacked server instances, and intelligently re-assigns the affected client to the new replica servers. Our solution leverages the resource elasticity and scale offered by cloud computing to instantiate and hide new replica servers. Each client, identified by the IP address, is randomly assigned to an active server instance when entering the system. That assignment will be shuffled to one of the replacement replica servers when under attack. Turning in-cloud replica servers into moving targets via server replication and substitution enables us to evade naive bots that can only attack hit-list server locations. Further, performing client-to-server shuffling at server replacement, as guided by the algorithms to be discussed in Section IV, is a powerful technique that separates real benign clients from persistent bots that can follow the moving targets. Through multiple rounds of shuffling, we can progressively quarantine the persistent bots onto a gradually decreasing set of replica servers while separating them from benign clients. When there is no attack, only a small number of static servers are maintained to meet the requirement of the regular workload. A detailed description of the system architecture and components that enable our shuffling-based moving target defense is provided below.

A. System Architecture

The overall architecture of our DDoS mitigation mechanism is shown in figure 1. This architecture presents a distributed connection redirection system deployed across multiple cloud computing domains. To reach the protected Internet service, a client has to first resolve its domain name via DNS (step 1). The DNS server will send the client to a cloud domain where our DDoS defense is in place (step 2). One or more load balancers are established in each cloud domain to redirect newly arrived clients (step 3). A load balancer keeps records about the active replica servers within the same domain and assigns (redirects) new clients to these replicas according to selected load balancing algorithms. To achieve redirection-based load balancing, the load balancer replies to each client's requests with the unique network location (public DNS name or IP address) of the replica server to which the client is assigned. The load balancer informs both the client and the corresponding replica server of each new assignment (step 4). As a result, the replica server will add the client's IP address to its

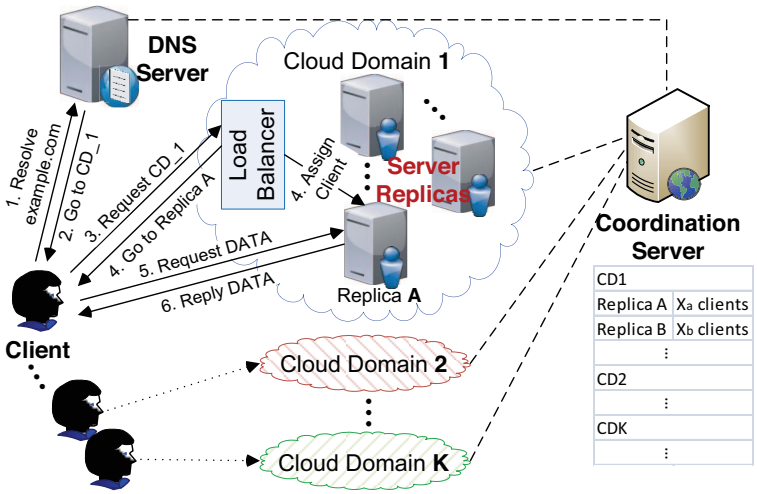

Figure 1. Architecture and Components

whitelist. Once the client contacts the replica server, the replica server will provide the client with the requested service (step 5, 6). The coordination server is the central controller that maintains global client-to-server bindings and directs defensive reactions to DDoS attacks. Although the mechanism is centrally controlled at the backend, all client-facing components are fully distributed, forming a robust layer of defense against DDoS attacks. Next, we describe each key component in greater detail.

B. Load Balancer

A load balancer assigns each new client to an active replica server. Clients are identified by their IP addresses. Each client IP is only matched with one replica server (i.e. sticky sessions). The load balancer will inform the server and the client on every client-to-server assignment.

At least one load balancer is installed in each cloud domain to keep track of all co-located, active replica servers. Here, different domains refer to groups of separately managed cloud servers that do not share common bottleneck links. They can be datacenters deployed at different geo-locations by a same cloud provider, or different cloud systems operated by distinct providers. Deploying multiple load balancers per cloud domain and having more cloud domains can improve attack resiliency and fault tolerance. The DNS records of all load balancers can be registered under the domain name of the protected Internet service. Techniques including round-robin DNS [26] can be used to send clients to different load balancers.

Each load balancer maintains an up-to-date list of online replica servers in its local domain. Any load balancing algorithm can be applied to assign clients to replica servers. The load balancers redirect (instead of forwarding) each client to a running replica server. This can be achieved via DNS load balancing [27], or HTTP redirection messages (i.e. status code 301) [28] that are automatically handled by client browsers. The reasons for using redirecting load balancers are two-fold: first, traffic redirection resembles a two-way handshake, which can effectively stop junk packets with spoofed source IPs from hitting the replica servers; second, by not using forwarding and making load balancers the middleman in client-server communication, we lower the risk of having them as the bottleneck during an attack.

To ensure the load balancers' availability, many cloud providers, including Amazon [1], enable auto-scaling of a virtual load balancer to provide on-demand capacity. It is reported that the throughput of a dedicated load balancer can reach 40 Gbps [1]. To avoid becoming a system bottleneck, multiple distributed load balancers can be deployed at the backbone network link of each cloud domain to increase resiliency against strong flooding attacks.

C. Replica Servers

Compared to load balancers, application servers are more vulnerable to DDoS attacks. To combat the threat, we replicate the protected service in the cloud environment. Our defense makes sure that each replica server is bound to a unique public DNS name or IP address that can be addressed separately. When there is no attack, only a small number replica servers are maintained to meet the requirement of the regular workload. Each client is redirected to an active replica by a load balancer. Replica servers enforce whitelist-based filtering, only admitting clients whose IPs are confirmed by the referring load balancer.

Once some replica servers are bombarded by a DDoS attack, a number of substitute replica servers will be instantiated at different network locations in the cloud. All

[1]F5.com,"Big-ipsystem",http://www.f5.com/pdf/products/big-ip-platforms-datasheet.pdf

clients served by the attacked replica servers will be re-assigned across the entire set of replacement replica servers. This process is called **client-to-server shuffling**. To perform client-to-server shuffling, the attacked replica servers will redirect associated clients to the new server locations (i.e. public DNS names or IP addresses of the replacement replica servers). For web services, such notification can be simply implemented by HTTP redirection code (3xx) [28], or running a snippet of redirection code inside clients' browsers. The client redirection task is prioritized over any application logic on the attacked replica servers. Client redirection traffic is also treated preferentially in the cloud network. Therefore, client-to-server shuffling is still possible even if the replica servers and their local network are being overwhelmed by a DDoS attack. After sending out all notifications, the attacked replica server will be taken offline and recycled. To expedite the shuffling process, a few hot spare replica servers can be maintained at runtime. In the rest of this paper, replica servers that participate in a shuffling operation are called *shuffling replica servers*, or *shuffling replicas*. Replica servers that are not under attack do not participate in shuffling, and are thus called *non-shuffling replica servers*. Clients on the non-shuffling replica servers are considered benign and are regarded as saved from the on-going attack.

Since we use individual replica servers to compartmentalize the potential impact of attackers on both network level and application level, our solution is able to isolate DDoS attacks targeting both network and computational resources. By dynamically replacing the attacked replica servers, we turn the in-cloud server instances into moving targets that can effectively evade the naive bots attacking invariable hit-list server locations. To inflict longstanding attacks, botnets must continually chase the "moving" replica servers. Assuming botnets are incapable of overwhelming the underlying cloud infrastructure, they will have to rely on the persistent bots simulating normal clients to follow the moving replica servers. However, as is going to be discussed in Section IV, our shuffling mechanism can expose and isolate such persistent bots, and save the affected benign clients over time.

D. Coordination Server

The coordination server maintains the global state of the defense system and directs realtime actions against DDoS attacks. In particular, it tracks the number of clients bound to each replica server, and records which replicas are currently under attack. In response to a DDoS attack, the coordination server runs the shuffling algorithms to separate benign clients from bots. Based on the number of attacked replica servers and the current client distribution, the coordination server computes an optimal shuffling plan that maximizes the probability of attack segregation. To ensure efficient shuffling operations, the coordination server decides the number of clients to be re-assigned from an attacked replica to a newly instantiated replica. It does not control the specific assignments of individual clients. The coordination server communicates among cloud domains via a dedicated command & control channel that is not accessible by any client. Section IV provides detailed analysis about the shuffling procedure and the related algorithms.

IV. Shuffling-based Segregation

Table I
Notations used in this paper and their meanings

Notation	Meaning
N	number of clients (including benign clients and bots)
M	number of persistent bots
P	number of shuffling replicas
S	number of clients to be saved
p_i	probability that the i-th shuffling replica is not under attack
x_i	number of clients assigned to the i-th shuffling replica

This section presents an in-depth analysis of the development of the proof-of-concept client-to-server shuffling mechanism. The goal is to separate benign clients from persistent bots that follow the moving replica servers to continue a DDoS attack. This is hard to achieve using an engineering method because such persistent bots may behave exactly like benign clients. With our solution, new replica servers are dynamically instantiated during a DDoS attack to replace the ones bombarded by bots. The shuffling operation refers to a structured method of re-assigning the affected clients from the attacked replicas to the new shuffling replica servers. Replica servers that are currently not under attack do not participate in a shuffling operation, although they may start to if they are bombarded later.

To illustrate the idea of client-to-server shuffling, we show the steps for one round of shuffling through the example in Figure 2. The initial state of the system is displayed on the left, where six clients $\{C1, C2, \ldots, C6\}$ are randomly assigned to two replica servers $\{RS\_1, RS\_2\}$. Among these clients, $C3$ and $C4$ are persistent bots attempting to attack the protected service. Like benign clients, the persistent bots have gone through the redirection steps we described in Section III to reach the replica servers. After successfully locating the service, the persistent bots instruct the naive bots to bombard both RS_1 and RS_2. Our solution aims to mitigate such an attack by gradually segregating the persistent bots from the benign clients. For this purpose, we instantiate new replica servers $\{RS\_3, RS\_4\}$ in the cloud to replace the attacked server instances. Then, we shuffle the clients' assignments onto the new replica servers. One possible shuffle is to swap the assignments of $C2$ and $C6$, $C3$ and $C5$ from the previous allocation scheme. As

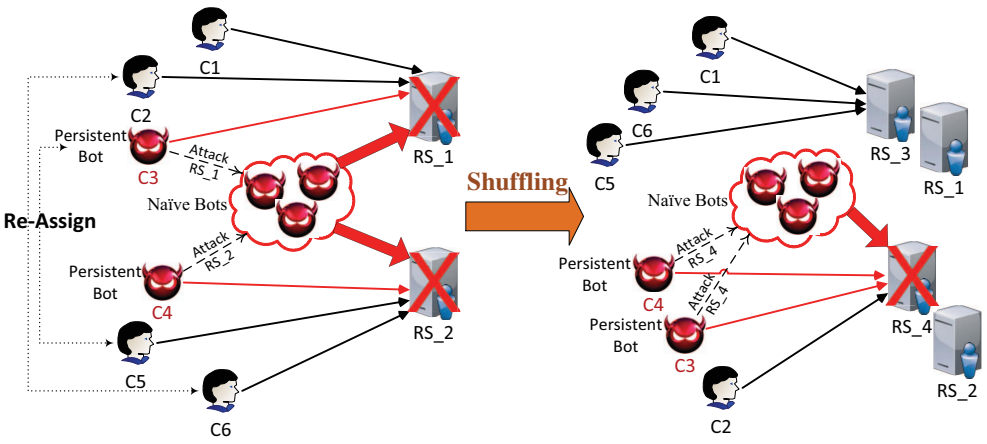

Figure 2. An example of client-to-server shuffling

is indicated by the right half of Figure 2, this client-to-server shuffling will segregate all persistent bots onto RS_4, leaving RS_3 out of the bots' vision. As a result, the benign clients assigned to RS_3 will be separated from the bots and saved from the ongoing DDoS attack. At this point, RS_3 and the saved clients will stop participating in the shuffling operation. New shuffling replica servers can be instantiated to shuffle the clients on RS_4.

To maximize the number of benign clients saved from each shuffle, the coordination server needs to make informed and optimal decisions based on the status quo. To that end, we developed algorithms to realize fast bot/attacker segregation. The controlled and optimized shuffling process will enable us to mitigate a strong DDoS attack in a few rounds of shuffling.

A. Theoretical Problem Modeling

We first constructed a mathematical model of the shuffling process. The notations used in this model are summarized in Table I. Consider a snapshot of the entire system before shuffling is performed: there are a total number of N clients, M of whom are persistent bots that act as malicious insiders ($M \leq N$). There are a constant number of P replica servers used for shuffling. Some replica servers may stop participating in the shuffling operations after certain rounds because they are no longer under attack (attacker-free). In that case, additional replica servers are activated to keep the number of shuffling replicas constant. We assume $P < N$; otherwise, each client can be allocated to an exclusive replica server to accomplish complete attacker isolation. In practice, many clients are assigned to each replica server.

For the purpose of fast bot segregation, we endeavor to save as many benign clients as possible from each round of shuffling. Since shuffling is a stochastic process, we can only calculate the probability that a given replica server is attacked. With that probability, we derived $E(S)$, the expected number of benign clients to be saved in one round. Therefore, solving the following optimization problem will maximize the number of benign clients to be saved.

$$\max E(S) = \sum_{i=1}^{P} p_i \cdot x_i = \frac{\sum_{i=1}^{P} \binom{N-x_i}{M} x_i}{\binom{N}{M}}$$
$$\text{subject to} \sum_{j=1}^{P} x_j = N \tag{1}$$

In the above equation, $x_1, x_2, \ldots, x_P$ denote the numbers of clients we assign to these P replica servers. p_i denotes the probability that the i-th replica is not under attack, which is also the probability that all M persistent bots are assigned to other replicas. Hence, $p_i = \frac{\binom{N-x_i}{M}}{\binom{N}{M}}$.

B. Optimal Solution

To solve the above optimization problem, we state the problem alternatively with regard to its key parameters. Let $S(N, M, P)$ denote the maximum expected number of benign clients that we can save in one shuffle, subject to a total number of N clients (including bots), M persistent bots, and P replica servers. To solve $S(N, M, P)$, we can instead solve $\max\{S(a, b, 1) + S(N - a, M - b, P - 1)\}$, where a is the number of clients assigned to the last replica and b is the number of persistent bots among a. Although we do not have direct control over b, there is a probability $\Pr(b)$ associated with each possible value of $b \in [0, \min\{a, M\}]$. These probabilities will vary as we change the value of a. For each different a, we can compute the expected result of the decomposed problem. In this way, the original optimization problem can be recursively broken down into a series of sub-problems, until all potential solutions for the sub-problems can be readily enumerated.

We use a dynamic programming approach to solve these sub-problems from bottom up, until eventually arriving at the optimal solution to the original problem. In particular, we incrementally build a look-up table, starting from using

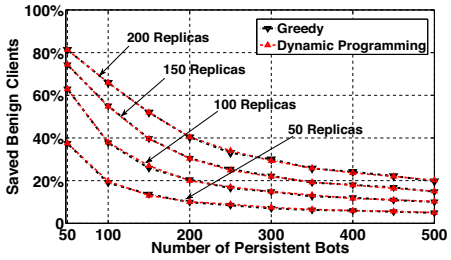

Figure 3. Compare the effectiveness of greedy algorithm and dynamic programming algorithm for one shuffle with 1000 clients. (*Curves are overlapping.*)

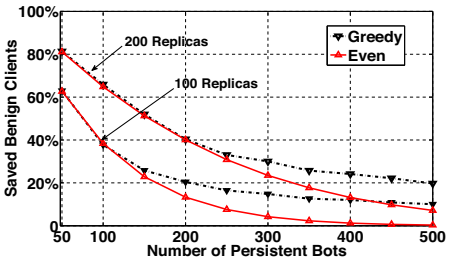

Figure 4. Compare the effectiveness of greedy algorithm and even distribution for one shuffle with 1000 clients.

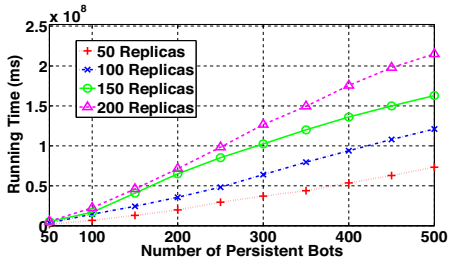

Figure 5. Running time of the dynamic programming algorithm with 1000 clients.

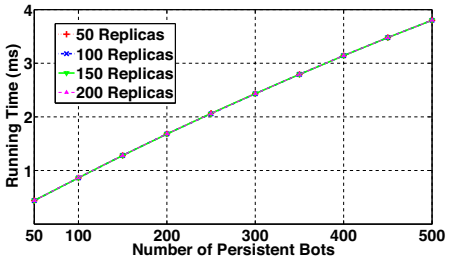

Figure 6. Running time of the greedy algorithm with 1000 clients.

only one replica server ($P = 1$). To compute the one replica case, we have

$$S(a, b, 1) = \begin{cases} a, & b = 0 \\ 0, & b > 0 \end{cases} \tag{2}$$

Solutions for larger replica numbers can be generated using

$$S(N, M, P) = \max_{1 \le a \le n-1} \{ \sum_{b} \Pr(b) \times [S(a, b, 1) + S(N - a, M - b, P - 1)] \}$$

where

$$\Pr(b) = \frac{\binom{M}{b}\binom{N-M}{a-b}}{\binom{N}{a}}, b \in [0, \min(a, M)] \tag{3}$$

Algorithm 1, below, describes the implementation of the dynamic programming procedure. Two tables, assign_no and save_no, are created to store the number of clients to assign to each replica server and the expected number of benign clients to be saved, respectively. The overall time complexity of the algorithm is $O(N^3 \cdot M^2 \cdot P)$, while the space complexity is $O(N \cdot M \cdot P)$.

C. A Fast Heuristic Greedy Algorithm

Although the dynamic programming algorithm is guaranteed to produce optimal client-to-server assignment, it is inadequate for making realtime decisions against on-going DDoS attacks due to its high time complexity. Therefore, we employed the greedy algorithm described in [6] for making

Algorithm 1 Optimal-Assign(N, M, P)

1: Initialize $save\_no[0, \cdots, N, 0 \cdots, M, 0 \cdots, P]$ and $assign\_no[0 \cdots, N, 0 \cdots, M, 0 \cdots P]$
2: **for** $i \leftarrow 1, N$ **do**
3: **for** $j \leftarrow 1, M$ **do**
4: **for** $k \leftarrow 1, P$ **do**
5: compute $S(i, j, k)$ using Equations 2 and 3, with $a \in [1, i-1]$ and $b \in [1, \min\{j, a\}]$;
6: select $a = \alpha$ that maximize $S(i, j, k)$;
7: update table entry $assign\_no[i, j, k] = \alpha$ and $save\_no[i, j, k] = S(i, j, k)$.

runtime shuffling decisions. Instead of targeting the global optimization problem, the greedy algorithm optimizes for one replica server at a time. For one server, we enumerate all possible values of x_i and select the one (ω) that maximizes Equation 1 with $P = 1$. We then attempt to assign ω clients to as many replica servers as possible, until we run out of clients or replica servers. If the remaining clients are less than ω, we update the problem setting (N', M', P') with the remaining unassigned clients and replica servers, and run recursively. When only one replica server is left, all remaining clients are assigned to it. The computational complexity of the greedy algorithm is $O(N \cdot M)$, while its space complexity is $O(P)$.

D. Algorithm Evaluations

To compare the greedy algorithm and the dynamic programming algorithm, we implemented both algorithms in

Matlab and compared them head-to-head through a series of simulations. It is worth mentioning that the high computational complexity of the dynamic programming algorithm only permitted experiments with relatively small parameters (up to 1000 clients and 200 replica servers).

Figure 3 shows the expected numbers of benign clients saved in one shuffle for each of the two algorithms. As we can see, the curves denoting respective algorithms almost overlap in all cases, indicating that the effectiveness of the greedy algorithm approaches closely to the optimal dynamic programming algorithm. The advantage of using the greedy algorithm is reflected by comparing Figure 5 and 6. Even though small client, persistent bot, and replica server numbers were used, the dynamic programming algorithm required tens of hours to compute the client-to-server assignments. In contrast, the greedy algorithm needed only a few milliseconds complete the same tasks. Therefore, the greedy algorithm is preferable as the runtime algorithm to guide shuffling operations against on-going DDoS attacks.

To demonstrate the importance of using optimized algorithms, we compared the greedy algorithm and the naive strategy that evenly distribute clients to all replica servers using the same numbers of clients and persistent bots as in Figure 3. According to simulation results displayed in Figure 4, the performance (i.e. the expected numbers of benign clients saved) of the naive approach was close to that of the greedy algorithm only when the number of persistent bots was smaller than that of replica servers. Otherwise, the naive approach performed much worse, saving almost no benign clients when the number of persistent bots was significantly higher than the number of replica servers.

V. Attack-Scale Estimation

When applying either the greedy algorithm or the dynamic programming algorithm, the number of persistent bots M is a key parameter in computing optimized client-to-server shuffle plans. In the earlier discussions, we assumed that M is known. However, in practice, no such prior knowledge is available. Since M has direct influence on the client-to-server assignments, accurately estimating M is critical. To estimate M, we adopt the Maximum Likelihood Estimation (MLE) algorithm proposed in [6]. However, we show that the MLE algorithm can accurately estimate the number of persistent bots under most, but not all conditions. Furthermore, we analyze and then prove the critical condition that has to be satisfied for the MLE algorithm to make accurate estimations.

The core idea of the MLE algorithm is to evaluate the likelihood of each possible value of M with regard to the number of observed replica servers that are under attack (denoted as X). The MLE algorithm is going to compute the probability that a particular value of M is going to cause X replica servers to be attacked. All possible values of M are enumerated and the one that corresponds to the highest probability becomes the final estimate of M. The lower bound of M is X, and the upper bound is the total number of clients assigned to all attacked replica servers. The time complexity of the MLE algorithm is $O(M^2 \cdot P)$.

To evaluate the accuracy of the MLE through example cases, we implemented the algorithm in Matlab and ran a series of simulations. The actual persistent bot numbers (X axis), the estimated persistent bot numbers (the solid line corresponding to the Y axis on the left), along with the percentage of attacked replica servers (the dotted line corresponding to the Y axis on the right), are plotted in Figure 7. In this figure, each data point is the mean of 40 repeated runs surrounded by 99% confidence interval. From the results, we notice that the MLE produces accurate estimation numbers, unless nearly all the shuffling replica servers are under attack.

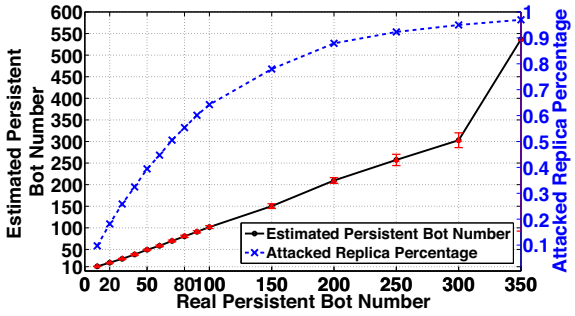

Figure 7. Evaluate MLE algorithm through examples (10000 clients, 100 shuffling replica servers)

As discussed above, MLE always chooses the value of M that maximizes the likelihood function. For the special case where all shuffling replicas are attacked, the likelihood is always greater with the higher value of M. Consequently, the largest possible M, i.e. the overall client number N, becomes the final estimate. Given this is an unrealistic overestimation, extra replica servers should be instantiated to make sure at least one replica is not under attack. Below, we theoretically determine the number of shuffling replicas P needed to meet this requirement under uniform client-to-server assignment.

We abstract an approximate model as follows: there are P shuffling replica servers and M persistent bots, and we assign M persistent bots sequentially onto the replica servers.

Theorem 1: If $M > \log_{1-\frac{1}{P}}\left(\frac{1}{P}\right)$, there is a high probability that all replica servers will be attacked.

Proof: Let X denote the number of replica servers that are not under attack. Let

$$X_i = \begin{cases} 1, & \text{if replica server } i \text{ is not under attack;} \\ 0, & \text{otherwise.} \end{cases}$$

Then the expected value of X is

$$E(X) = \sum_{i=1}^{P} E(X_i) = \sum_{i=1}^{P} \Pr[X_i = 1] = P\left(1 - \frac{1}{P}\right)^M$$

When all shuffling replicas are attacked with high probability, the expected number of un-attacked shuffling replicas $E(X) < 1$. We have

$$E(X) < 1 \Rightarrow P\left(1 - \frac{1}{P}\right)^M < 1 \Rightarrow M > \log_{1-\frac{1}{P}}\left(\frac{1}{P}\right)$$

■

According to Theorem 1, when all shuffling replica servers are under attack, P must be increased such that $M \leq \log_{1-\frac{1}{P}}\left(\frac{1}{P}\right)$. The resource elasticity permitted by the underlying cloud infrastructure allows sufficient space for us to increase the number of replica servers. Therefore, we can always be confident in the estimations made by the MLE algorithm.

VI. System Evaluation

In this section, we experimentally evaluated the effectiveness and overhead of the proposed solution using experiments with a proof-of concept prototype. First, we assessed the performance of our algorithms against simulated large-scale DDoS attacks. Next, we measured the delay incurred by the client re-assignment operations.

A. Simulation-based Evaluation

To quantitatively evaluate the efficacy of our approach, we implemented both the greedy shuffling algorithm and the MLE algorithm in Matlab, and ran simulated attacks against them. We varied the number of benign clients and persistent bots across individual simulation runs to study the performance of our algorithms under different conditions. We assumed both benign clients and persistent bots arrive in a Poisson process. On average, the arrival rate of persistent bots was 5000 per 3 shuffles while that of benign clients was 100 per 3 shuffles. The shuffling algorithm decided the total number of clients assigned to each replica server. Benign clients, together with persistent bots, were randomly assigned to replica servers. Replica servers with any number of persistent bots assigned were regarded as attacked. We experimented with different, fixed numbers of shuffling replica servers performing continuous shuffling operations to save affected benign clients. All simulations were repeated 30 times to generated data from which a mean and 99% confidence interval were calculated.

First, we ran simulations with 1000 shuffling replicas and varied numbers of benign clients and persistent bots. The results are plotted in Figure 8. One can see that the number of shuffles required to save 80% and 95% of benign clients rises slowly with the increase in the populations of persistent bots and clients. In the worst case, a ten-fold increase in the number of persistent bots results in less than three-fold increase in the number of shuffles; and for a given number of persistent bots, a five-fold increase in the benign clients only introduces less than 70% (40) more shuffles to save the designated percentage of clients.

Next, we changed the number of shuffling replica servers while keeping the client population (10^4, 5×10^4) and persistent bot population (10^5) constant. The curves in Figure 9 show that the number of shuffles needed to save the same percentage of benign clients drops steadily when more replica servers are added.

One interesting pattern consistent across both figures is that in most cases, the number of shuffles it takes to save 95% of benign clients is more than 40% higher than that to save 80% of benign clients. To explore this pattern in greater detail, we recorded the number of benign clients saved in each shuffle and plotted a cumulative percentage graph in Figure 10. The curves show that the early shuffles were able to separate more benign clients from bots than the latter shuffles. This effect is due to that as more benign clients were saved, persistent bots accounted for a greater percentage of the remaining population, making it harder to separate out the benign clients.

B. Prototype-based Evaluation

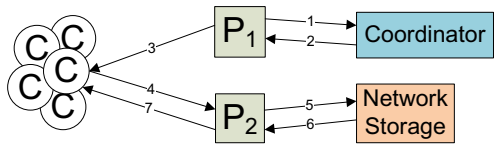

Figure 11. System prototype (C – Client, P – Replica Server)

To study the overhead of our approach, we built a proof-of-concept prototype as described by Figure 11. We implemented two replica servers and a coordination server/coordinator on separate Amazon EC2 [1] micro instances. We used 60 PlanetLab [2] nodes as clients. Each of the replica servers runs a simple web server that displays a static web page (246KB) fetched from a network mounted storage. The web server logic is written in Node.js [3].

Initially, all clients are served by replica P_1. When a simulated attack is triggered on P_1, P_1 consults the coordinator for next step (step 1). In general, the coordinator will make shuffling decisions either by running the greedy algorithm or by looking up the pre-computed client-to-server assignment tables generated by the dynamic programming algorithm. The decisions are sent back to the attacked replica to guide subsequent shuffling operations. For the purpose of overhead evaluation, the coordinator responded by asking P_1 to redirect all clients to replica P_2 (step 2). As a result, P_1 proactively sent redirection notifications to all clients (step

[2] "Planetlab", http://www.planet-lab.org
[3] "Node.js", http://www.nodejs.org

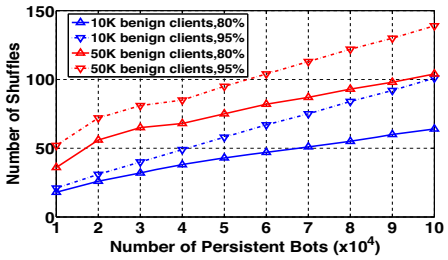

Figure 8. Number of shuffles to save 80% and 95% of 10^4 and 5×10^4 benign clients, with 1000 shuffling replica servers, and varying persistent bot numbers.

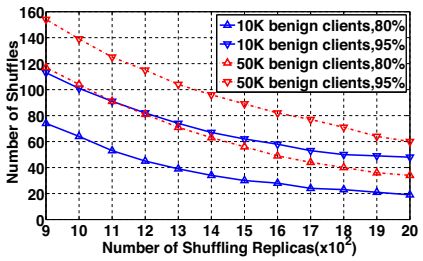

Figure 9. Number of shuffles to save 80% and 95% of 10^4, and 5×10^4 benign clients, with 10^5 persistent bots and varying shuffling replica server numbers.

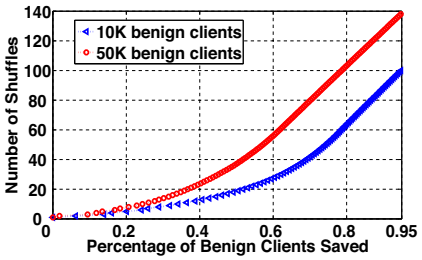

Figure 10. Cumulative percentage of saved benign clients vs. number of shuffles, with 10^5 persistent bots, 10^4, and 5×10^4 benign clients.

3). Upon reception, clients contacted and reloaded the same web page from P_2 (step 4, 5, 6, 7).

Unlike conventional HTTP-based communications that start with client requests, the redirection operation is always initiated by an attacked replica server. To send unsolicited messages to HTTP clients, we take advantage of the WebSocket [29] technology multiplexing HTTP(S) ports (80 and 443). WebSocket is well-supported by all major browsers. Therefore, the adoption of our mechanism does not depend on extra software being installed on the client side. For this prototype, our server injects a snippet of Javascript code (40 LOC) into the requested web page to establish a WebSocket between clients and the replica server.

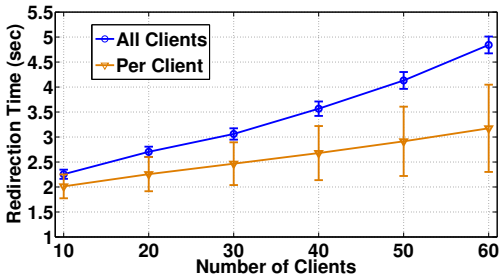

Figure 12. Client migration time between two replica servers

With this prototype system, we studied the time overhead of the redirection operations associated to client-to-server shuffling. This prototype ran Firefox browsers (v17.0) on up to 60 geographically distributed PlanetLab nodes as clients, who visited the same web page (246KB) served by replica server P_1 concurrently. The time for all clients to complete steps 1-7 (i.e. redirection time from P_1 to P_2) is shown in Figure 12. In this figure, the upper curve shows the time it took to successfully redirect all clients, while the lower curve reveals the average redirection time per client. The measurement for each data point was repeated 15 times to obtain the mean and 95% confidence interval. The results show that we can re-assign 60 clients in less than 5 seconds. Overall, the data indicates a low overhead for client re-assignment operations during shuffling, but we see room for additional improvement because our server program was single-threaded and not optimized at all. The prototype was mainly developed to show the feasibility of the approach and provide a lower-bound on performance. A practical DDoS defense system with more replica servers is not expected to slow down because all replica servers act independently and in parallel.

VII. Discussion

In this section, we discuss some practical aspects of the system including resiliency and deployment of the shuffling-based moving target defense. First, we analyze how our defense mechanism can effectively respond to different attacker strategies. Then, we compare our defense mechanism with existing solutions in terms of ease of deployment and cost.

Resiliency against network and computational attacks: Our solution is able to mitigate DDoS attacks targeting both network and computational resources. When under attack, our mechanism dynamically instantiates replica servers in the cloud to compartmentalize the impact of attackers. All clients, including attackers, are divided into smaller groups and mapped to different replica servers. The replica servers are distributed across the entire network space of the cloud environment to diffuse the network traffic generated by attackers. Furthermore, individual replica servers process incoming application traffic independently without interfering each other. Therefore, both the network and computational burden inflicted by attackers can only affect the replica servers to which attackers are assigned.

Effectiveness against naive and persistent attackers: The shuffling-based moving target defense can effectively mitigate DDoS attacks performed by a wide range of adversaries: naive attackers/bots focusing their attack on static server locations, and more sophisticated attackers/bots that persist their attack across newly instantiated replica servers.

Our system is able to evade naive bots by dynamically instantiating new replica servers in the cloud to replace the bombarded server instances. Since the naive bots are unable to autonomously re-target the new servers, the attack is easily curtailed. For sophisticated bots that can persist their

attack, we harness the re-assignment strategies of the client sessions to newly instantiated replacement replica servers. Indeed, our shuffling operations can gradually segregate the sessions generated by bots that follow the moving target from benign clients by intelligently re-assigning all client sessions across the set of replacement replica servers. The effectiveness of the shuffling operations can be measured via simulations. By carefully optimizing the shuffling plan, we show that our mechanism can mitigate strong DDoS attacks (up to $100K$ of sophisticated bots) on medium-scale Internet services (up to $50K$ online clients) in approximately 60 shuffles. In our experiments, we assumed the worst case scenario where bot-generated DDoS traffic can "catch" the moving replica servers instantly and without any coordination or re-targeting cost. However, in practice, it will take a non-trivial amount of effort and time before botnets can re-coordinate and re-focus their traffic onto the new replica servers. Therefore, the protected system is expected to be attack-free for a reasonably long period of time. On the contrary, our shuffling operations will kick off immediately once an attack is noticed, and can complete swiftly, causing only a brief "lag" to be perceived by the clients.

Resiliency against aggressive and non-aggressive attackers: As was shown in our analysis and simulations, persistent bots that keep following and aggressively attacking the moving replica servers will be gradually segregated from benign clients. To disrupt our defense, certain persistent bots may adopt a non-aggressive strategy to perform on-and-off attacks. These non-aggressive bots may seek to stay with benign clients longer to map out the defense system, or cause more damage later. However, our shuffling-based moving target defense is stateless, only focusing on the current state of the replica servers. Only the bombarded replica servers will participate in the shuffling operations. The replica servers that are not under attack will not be shuffled. Therefore, the non-aggressive bots will not be able to gain extra information about the system or blend with larger groups of benign clients. Instead, they will only lead to a reduced DDoS attack intensity on the protected service.

Resiliency against other Attacks: To avoid getting segregated, other persistent bots may stop their attack once they detect a shuffling operation and re-enter the system through the load balancers. To defeat such bots' attempts to re-blend with benign clients, we record the last server assignment of each client for a certain period of time after the client leaves the system. If these clients re-enter the system before their previous records expire, they will be assigned to the same recorded replica servers. Bots will be treated as new clients only when they re-enter the system using a new IP address. To resolve this problem, future work needs to find a better way to uniquely identify each bot.

Our client redirection mechanism can easily defeat IP spoofing attacks. If not using their real IP addresses, bots are unable to receive the redirection messages sent by servers or the load balancers, hence will be left behind our moving replica servers. To bypass client redirection and still uncover the network locations of the in-cloud replica servers, attackers may perform reconnaissance attacks such as IP and port scanning. However, since we constantly shift the network locations of the replica servers, it is difficult for attackers to pick the right target even if they have profiled the entire IP pool of the underlying cloud infrastructure.

Scalability and cost: Our defense is scalable to withstand strong DDoS attacks. As long as the underlying cloud computing infrastructure is available, our solution will gradually segregate attackers from benign-but-affected clients. The characteristics of the cloud computing service enable us to quickly and easily expand the number of replica servers in order to accomplish shuffling-based attack segregation, and to then scale down to a small number of server instances when not under attack. Unlike existing filtering-based, capability-based, and overlay-based defenses that require static and expensive infrastructure support, resources in our solution are allocated elastically and dynamically, not permanently, which results in low setup and maintenance costs. Attacked server instances are recycled, freeing resources which can be used to instantiate new servers for future rounds of shuffling. A quantitative study on the cost of the shuffling-based moving target defense is part of our future work plans.

Deployment: Our DDoS defense mechanism is designed to be deployed as an end system in the cloud. As we have demonstrated in our proof-of-concept prototype, it can be independently implemented and managed by a non-ISP organization.

VIII. Acknowledgements

This work is supported by the United States Defense Advanced Research Projects Agency (DARPA) through Contract FA8650-11-C-7190. Opinions, findings, conclusions and recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the US Government, or DARPA.

IX. Conclusion

We proposed a cloud-enabled, shuffling-based, moving target mechanism to mitigate DDoS attacks against open Internet services. When under attack, our approach dynamically instantiates replica servers in the cloud and intelligently re-maps and migrates client sessions to new and un-advertised server locations. This defense strategy not only helps the protected service evade the naive attackers that bombard hit-list network locations, but also enables planned client-to-server shuffling operations to segregate the persistent attackers that continue to follow and attack the moving replica servers. We thoroughly studied the algorithms that guide and optimize the shuffling operations. Our simulation results show that the proposed mechanism

can effectively mitigate strong DDoS attacks (100K bots) by saving 80% of 50K benign clients in approximately 60 shuffles, with each shuffling taking only a few seconds to complete. In conclusion, our defense mechanism is effective, easy-to-deploy, and scalable against strong and sophisticated DDoS attacks, and is more cost-effective than existing static DDoS defense solutions. Therefore, our approach offers an increasingly appealing DDoS defense solution that can be offered as a service by existing cloud providers and enterprises with a large network footprint.

REFERENCES

[1] "Amazon web services," http://aws.amazon.com.

[2] D. Anstee and D. Bussiere, "Worldwide infrastructure security report viii," 2012. [Online]. Available: http://www.arbornetworks.com/report

[3] NBCNEWS, "Bank website attacks reach new high: 249 hours offline in past six weeks," http://www.nbcnews.com/technology/bank-website-attacks-reach-new-high-249-hours-offline-past-1C9195242, April 2013.

[4] "Layered intelligent ddos mitigation systems," 2011. [Online]. Available: http://www.arbornetworks.com/ddos/Layered%20Intelligent%20DDoS%20Mitigation%20Systems.pdf

[5] A. Stavrou, A. D. Keromytis, J. Nieh, V. Misra, and D. Rubenstein, "Move: An end-to-end solution to network denial of service," in *Proceedings of NDSS*. The Internet Society, 2005.

[6] Anonymous, "Motag: Moving target defense against internet denial of service attacks," in *Proceedings of 22nd International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2013.

[7] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comput. Syst.*, vol. 2, no. 4, pp. 277–288, Nov. 1984.

[8] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling high bandwidth aggregates in the network," *ACM Computer Communication Review*, vol. 32, pp. 62–73, 2002.

[9] P. Ferguson and D. Senie, "Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing," RFC 2827 (Best Current Practice), Internet Engineering Task Force, May 2000, updated by RFC 3704.

[10] X. Liu, X. Yang, and Y. Lu, "To filter or to authorize: network-layer dos defense against multimillion-node botnets," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*. New York, NY, USA: ACM, 2008, pp. 195–206.

[11] T. Anderson, T. Roscoe, and D. Wetherall, "Preventing internet denial-of-service with capabilities," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, pp. 39–44, 2004.

[12] A. Yaar, A. Perrig, and D. Song, "Siff: A stateless internet flow filter to mitigate ddos flooding attacks," in *IEEE Symposium on Security and Privacy*, 2004, pp. 130–143.

[13] X. Yang, D. Wetherall, and T. Anderson, "Tva: a dos-limiting network architecture," *IEEE/ACM Trans. Netw.*, vol. 16, no. 6, pp. 1267–1280, 2008.

[14] X. Liu, X. Yang, and Y. Xia, "Netfence: preventing internet denial of service from inside out," in *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 255–266.

[15] A. D. Keromytis, V. Misra, and D. Rubenstein, "Sos: Secure overlay services," in *Proceedings of ACM SIGCOMM*, 2002, pp. 61–72.

[16] D. G. Andersen, "Mayday: distributed filtering for internet services," in *USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*. Berkeley, CA, USA: USENIX Association, 2003, pp. 3–3.

[17] R. Stone, "Centertrack: an ip overlay network for tracking dos floods," in *SSYM'00: Proceedings of the 9th conference on USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2000, pp. 15–15.

[18] A. Mahimkar, J. Dange, V. Shmatikov, H. Vin, and Y. Zhang, "dfence: Transparent network-based denial of service mitigation," in *NSDI*, 2007.

[19] C. Dixon, T. Anderson, and A. Krishnamurthy, "Phalanx: withstanding multimillion-node botnets," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 45–58.

[20] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th Usenix Security Symposium*, 2004.

[21] "Moving target defense." [Online]. Available: http://www.cyber.st.dhs.gov/moving-target-defense/

[22] S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagnostakis, "Defending against hitlist worms using network address space randomization," in *Proceedings of the 2005 ACM workshop on Rapid malcode*, ser. WORM '05. New York, NY, USA: ACM, 2005, pp. 30–40.

[23] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling, "Measuring and detecting fast-flux service networks," in *NDSS*. The Internet Society, 2008.

[24] T. M. Gil and M. Poletto, "Multops: a data-structure for bandwidth attack detection," in *Proceedings of the 10th conference on USENIX Security Symposium - Volume 10*, ser. SSYM'01. Berkeley, CA, USA: USENIX Association, 2001, pp. 3–3.

[25] A. Hussain, J. Heidemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '03. New York, NY, USA: ACM, 2003, pp. 99–110.

[26] T. Brisco, "DNS Support for Load Balancing," RFC 1794 (Informational), Internet Engineering Task Force, Apr. 1995. [Online]. Available: http://www.ietf.org/rfc/rfc1794.txt

[27] P. Membrey, D. Hows, and E. Plugge, "Dns load balancing," in *Practical Load Balancing*. Springer, 2012, pp. 53–69.

[28] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616 (Draft Standard), Internet Engineering Task Force, Jun. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[29] I. Fette and A. Melnikov, "The WebSocket Protocol," RFC 6455 (Informational), Internet Engineering Task Force, Dec. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6455.txt