A Moving Target DDoS Defense Mechanism

Huangxin Wang*, Quan Jia, Dan Fleck, Walter Powell, Fei Li, Angelos Stavrou

Department of Computer Science,George Mason University,Fairfax,VA 22030, USA

Abstract

In this paper, we introduce a moving target defense mechanism that defends authenticated clients against Internet service DDoS attacks. Our mechanism employs a group of dynamic, hidden proxies to relay traffic between authenticated clients and servers. By continuously replacing attacked proxies with backup proxies and reassigning (*shuffling*) the attacked clients onto the new proxies, innocent clients are segregated from malicious insiders through a series of shuffles. To accelerate the process of insider segregation, we designed an efficient greedy algorithm which is proven to have near optimal empirical performance. In addition, the insider quarantine capability of this greedy algorithm is studied and quantified to enable defenders to estimate the resource required to defend against DDoS attacks and meet defined QoS levels under various attack scenarios. Simulations were then performed which confirmed the theoretical results and showed that our mechanism is effective in mitigating the effects of a DDoS attack. The simulations also demonstrated that the overhead introduced by the shuffling procedure is low.

Keywords: DDoS; Moving Target Defense; Secret Proxy; Insider; Shuffling

1. Introduction

Distributed Denial-of-Service (DDoS) attacks are a rapidly growing problem which poses an immense threat to the Internet. Arbor Networks reported a significant increase in the prevalence of large-scale distributed denial-of-service (DDoS) attacks in recent years [1]. In 2010, the largest reported bandwidth achieved by a flood-based DDoS attack reached 100 Gbps. Even as the bandwidth of attacks has increased, the cost of performing a DDoS attack has turned out to be surprisingly low. A Trend Micro white paper [2] reported that the price for a 1-week DDoS attack could be as low as $150 on the Russian underground market.

A number of mechanisms have been proposed in the past to prevent or mitigate the impact of DDoS attacks. Filtering-based approaches [3, 4, 5] use ubiquitously deployed filters that block unwanted traffic sent to the protected nodes. Capability-based defense mechanisms [6, 7, 8, 9] endeavor to constrain resource usage by senders to beneath a threshold defined by the defended system. Secure overlay solutions [10, 11, 12, 13, 14, 15] interpose a network of proxy nodes that redirect packets between clients and the protected nodes and are designed to absorb and filter out attack traffic. All these mechanisms are effective to varying degrees; but these static defense mechanisms either rely on the global deployment of additional functionalities on Internet routers or require large, robust, virtual networks designed to withstand the ever-larger attacks. Due to the large investment required and the vulnerability to sophisticated attacks such as sweeping [11] and adaptive flooding attacks [12], the development of novel, effective, efficient, and low cost defense mechanisms continues to be a high priority, but elusive goal.

Motivated by the aforementioned elusive goal, we propose *MOTAG* [16], a MOving Target defense mechanism AGainst Internet DDoS attacks. This dynamic DDoS defense mechanism implements a scheme of moving proxy nodes to protect centralized online services. In particular, *MOTAG* offers DDoS resilience for authenticated clients of security sensitive services such as online banking and e-commerce. *MOTAG* employs a layer of secret moving proxy nodes to relay communications between clients and the protected application servers.

The proxy nodes in *MOTAG* have two important characteristics. First, the proxy nodes are "secret" in that their IP addresses are concealed from the general public and are exclusively known only to legitimate clients and only after successful authentication. In order to avoid unnecessary information leakage, each authenticated client is provided with the IP address of only a single proxy node at any given time. Existing Proof-of-Work (PoW) schemes [17, 18, 19, 20] are employed to protect the client authentication channel. Second, the proxy nodes are "moving". As soon as an active proxy node is attacked, it is replaced by a set of alternate proxy nodes instantiated at a different IP address; and the clients associated with the attacked proxy node are migrated to alternative proxy node(s). We show that this migration to "secret" proxy nodes not only enables the *MOTAG* mechanism to mitigate brute-force DDoS attacks, but also provides a means to discover and isolate malicious insiders designed to divulge the location of the secret proxy nodes to external attackers. The malicious insiders are isolated via a

*Corresponding author at: Department of Computer Science, George Mason University, Fairfax, VA 22030, USA. Tel: +1 7039019768

Email addresses: hwang14@gmu.edu (Huangxin Wang), qjia@gmu.edu (Quan Jia), dfleck@gmu.edu (Dan Fleck), wpowell@gmu.edu (Walter Powell), fli4@gmu.edu (Fei Li), astavrou@gmu.edu (Angelos Stavrou)

shuffling process that reassigns and migrates clients through sequential sets of instantiated of proxy nodes. This paper presents the algorithms developed to (1) accurately estimate the number of insiders and (2) to dynamically determine client-to-proxy assignment that will "save" the largest number of legitimate clients after each shuffle.

Unlike previously proposed DDoS defense mechanisms, *MOTAG* does not rely on global adoption on Internet routers or collaboration across different ISPs to function. Also *MOTAG* neither depends on resource-abundant overlay networks to outmuscle high bandwidth attacks nor uses filters to provide fault tolerance. Instead, we take advantage of our proxy nodes' secrecy and mobility to fend off powerful DDoS attacks including sweeping and adaptive flooding attacks. Employing the *MOTAG* DDoS defense mechanism requires lower deployment costs while offering substantial defensive agility which results in effective and cost-efficient DDoS protection.

This paper is an extension of the work we presented in [16]. The main contributions of this paper are:

- A discussion of the reduction of the computational complexity of greedy algorithm from $O(N \cdot N_i)$ to $O(1)$ where N is the number of total clients and N_i is the number of insiders.
- A theoretical and empirical analysis of the insider quarantine capability of the greedy algorithm.
- A discussion of a special DDoS attack case for which a simple and elegant shuffling mechanism is designed to segregate innocent clients from insiders.

2. Threat Model and Assumptions

Instead of targeting open and general-purpose web services, we focused on protecting security sensitive online services against *network flooding attacks*. We assume that legitimate clients of the protected services are pre-authorized and their identities can be authenticated before they are served. We assume the availability of a cloud environment with sufficient computing power and bandwidth to instantiate numerous backup proxy nodes. Since only a small group of proxy nodes are active at any time, a cloud environment in which customers are charged only for running instances would be ideal to avoid extensive operational costs. We further assume that although powerful attackers with a high aggregate bandwidth are capable of simultaneously overwhelming many stand-alone machines on the Internet, attackers cannot saturate the well-provisioned Internet backbone links of ISPs, data centers, and cloud service providers.

We also assume that attackers, in case of uncertainty, can first perform reconnaissance attacks (e.g., IP and port scanning) to pinpoint targets for the subsequent flooding attacks. With knowledge of the *MOTAG* mechanism, attackers could attempt to flood the authentication channel through which the legitimate clients are admitted. Successful attacks against the authentication server are considered unlikely because it employs proof-of-work(PoW) schemes to prevent both computational and network flooding attack. *MOTAG* takes advantage of PoW schemes that are designed to prevent computational attacks. In a cloud environment, the ability of lightweight PoW schemes to quickly reject non-authentication requests makes it resistant to flooding attacks. Since it is significantly harder for attackers to pass strong authentication by brute force and reach the proxy nodes as legitimate clients, some attackers will attempt to uncover the network locations of proxy nodes and may plant "insiders" by compromising legitimate clients or eavesdropping on legitimate clients' network connections. However, the number of such insiders in a protected system is assumed to be limited.

3. MOTAG Architecture

The proposed *MOTAG* mechanism employs a group of dynamic proxy nodes that relay traffic between servers and authenticated clients and that provide a moving target defense mechanism that mitigates Internet service DDoS attacks. The IP address of the proxy nodes are hidden from clients (and potential attackers), and each client can only see the IP address of the proxy node to which he is randomly assigned. Therefore, insiders will only be able to attack the proxy node(s) to which they are assigned, and the innocent clients who are impacted by the attack will be only those who share the proxy node(s) with the insiders. In order to separate affected innocent clients from insiders, *MOTAG* instantiates proxy nodes and performs client-to-proxy reassignment under the guidance of the shuffling algorithm discussed in Section 4 and 5. In the following sections, we first give an overview of *MOTAG*, and then introduce the main components in greater detail.

3.1. MOTAG Overview

Figure 1 shows the overall architecture of *MOTAG* which consists of four inter-connected components: the authentication server, the proxy nodes, the filter ring, and the application server. The application server provides the online services (e.g., banking or e-commerce services) that we want to protect and make accessible to authenticated clients. The IP address of the application server is concealed from all clients and all traffic is relayed to the application server via the proxy nodes. The filter ring, similar to what was described in [12], is comprised of a number of high speed routers placed around the application server which allows inbound traffic only from valid proxy nodes. The proxy nodes are a group of dynamic and distributed cloud instantiations that relay communications between clients and the application server. The authentication server is responsible for authenticating clients, assigning legitimate ones to individual proxy nodes, and coordinating the shuffling of clients.

MOTAG allows a client to access the application server via a proxy node only if the client can be successfully authenticated. One simple solution is to associate the application domain name with the IP address of the authentication server during DNS registration. Each successfully authenticated client is then randomly assigned to one of the active proxy nodes whose identities are not publicly known. The authentication server will inform each client of the IP address of a designated

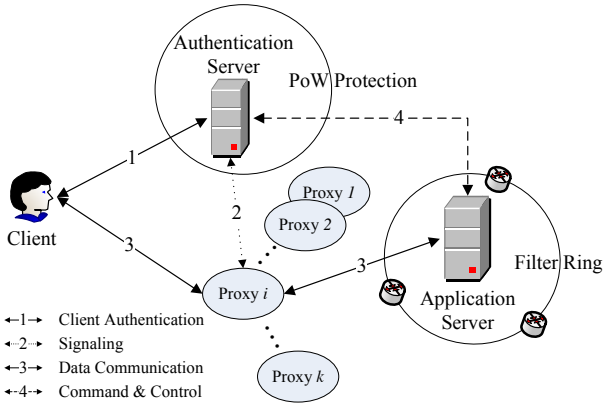


Figure 1: Overview of the MOTAG Architecture.

proxy node, and simultaneously notify the proxy node of the forthcoming connection from the client. This communication between the authentication server and the proxy nodes is via a dedicated signaling channel. Through this signaling channel, proxy nodes report to the authentication server if they are under attack. The authentication server also uses this channel to inform proxy nodes of client assignments and to support coordination of their actions against DDoS attacks.

The authentication server also assigns a *capability token* for each client-to-proxy session. This token is used to limit a client's throughput by specifying the number of packets (or, the number of bytes) allowed for the session in the next time window (t seconds). In order to validate a client connection, a proxy node should receive two identical copies of a capability token; one from the authentication server as notification of a new authenticated client assignment, and one from the client as a proof of identity. Every proxy node maintains a per-session counter and regulates traffic according to individual capability. Such capability-based policing is key to detecting external, brute-force flooding attacks in that it allows proxy nodes to distinguish between authorized packets and illegal ones. Furthermore, the use of a *capability token* helps frustrate an internal attempt to abuse the assigned capability.

For communications between proxy nodes and the application server, a lightweight authenticator as described in Mayday [12] can be employed for proxy identity validation. The filter ring routers can perform fast look-ups to verify such lightweight authenticators in proxy-to-application packets. These authenticators can be dynamically altered, and active proxy nodes will receive timely updates via the signaling channel. To prevent the authentication server from being flooded by botnets, we employ proof-of-work (PoW) schemes [17, 18, 19, 20] to ensure its accessibility for legitimate clients.

Overall, the procedure that a client needs to go through in order to get access to the application server is: First, the client resolves the domain name of the target web service via a DNS and is redirected to the authentication sever (step 1). Upon authentication, the authentication server assigns the client randomly to a proxy node. Both the client and proxy will receive a capability token (step 2) that serves to notify each of the client-to-proxy node assignment (including IP addresses) and provides a means of validating the client to the proxy node. The client can then communicate with application server via the proxy node (step 3). The specifics of the implementation of the proxy nodes and authentication server are discussed below.

3.2. Secret Moving Proxy nodes

MOTAG is designed to take advantage of the unique capabilities of a cloud environment. The *MOTAG* builds an intermediate layer consisting of a pool of geographically distributed proxy nodes designed to diffuse attackers' traffic and insulate the application server. When not under DDoS attack, only a small subset of the available proxy nodes are active. These active proxy nodes provide sufficient capability to relay normal traffic to the application server. When under DDoS attack, dynamically instantiated proxy nodes are substituted for the attacked proxy nodes in real time, confusing attackers with "moving" target proxy nodes. The "moving" proxy nodes are resilient to scanning attacks because they only respond to IP addresses of the authenticated clients.

When under attack, a proxy node informs the authentication server, and the authentication server coordinates the instantiation of new proxy node(s) at different network locations. Clients (both innocent and insiders) on the attacked proxy node(s) are migrated to the newly instantiated proxy nodes, and the proxy node(s) under attack are shut down. Proxy instantiation and client migration is a fast, lightweight operation because all session information is centrally stored at the application server; and all proxy nodes run the same, simple traffic redirection logic and maintain no client state. The clients connected to the under attack proxy node(s) are re-assigned across the entire set of active proxy nodes. The new client-to-proxy node assignments can be pushed to the affected clients by the authentication server, or the clients can be made to re-authenticated for security assurance. The overall process of proxy replacement and client re-assignment is called client-to-proxy *shuffling*, and details of the shuffling algorithm are presented in Section 4. No shuffling is performed if there is no attack, and only a small set of proxy nodes with constant IP addresses are required to serve all legitimate clients.

MOTAG is different from existing overlay network solutions [10, 11, 12, 15], which rely on a fairly static, high-capacity, network composition of overlay nodes to tolerate and filter out the attack traffic. Building and maintaining such an overlay entails extensive and continuous investment to acquire more nodes and bandwidth. As currently implemented, existing overlay networks may be subject to sever service disruptions due to sweeping [11] and adaptive [12] flooding attacks. In contrast, *MOTAG* proxy nodes are dynamic and their IP addresses are kept confidential. The combination of these two factors serves to enhance the agility of the *MOTAG* defense mechanism against massive, sophisticated attacks while reducing dependence on the capacity of individual proxy resources.

3.3. Authentication with Proof-of-Work Protection

An authentication server with assured accessibility is essential to our moving target defense. It acts as the initial checkpoint

to separate legitimate clients from external attackers. *MOTAG* uses established authentication procedures as a mechanism to bind a client to a specific network flow. Only with such a unique binding is the authentication server able to keep track of the proxy to which each client is assigned throughout the shuffling process. Since each client has to pass authentication before being assigned to a proxy node, the IP addresses of authenticated clients are recorded and sent to proxy nodes and used by the proxy nodes to enforce IP-based filtering. The authentication server is also responsible for coordinating subsequent client-to-proxy assignments during shuffling. *MOTAG* is agnostic to the specific authentication mechanism employed.

The authentication server is the only part of the *MOTAG* architecture that can be publicly addressed. Therefore, only it can be a target of non-insider assisted, distributed flooding attacks. To mitigate this type of attack, the authentication server takes advantage of existing proof-of-work (PoW) schemes [17, 18, 19, 20], which force clients to solve cryptographic puzzles before allowing them to consume resources on the application server. Using the PoW schemes, the authentication server can realize per-computation fairness regarding bandwidth usage among all clients [20], prevent connection depletion attacks [19], and mitigate DDoS attacks on application-level authentication protocols [17, 18]. Although mandating the extra computational tasks involved with PoW schemes can help reduce attackers' throughput, it also imposes considerable burden on legitimate clients. Therefore, PoW approaches are suitable for client authentication mechanisms that require authentication packets to be sent infrequently and which are delay-tolerant. However, PoW schemes are not preferred for securing application data communication due to their high overhead.

4. Client-to-Proxy Shuffling Modeling

While using hidden proxy nodes and enforcing client authentication can effectively prevent external attackers from reaching *MOTAG*'s packet delivery system, implementing a shuffling process that employs mobile proxies and intelligent shuffling of client-to-proxy assignments enables *MOTAG* to also mitigate insider attacks designed to expose the hidden proxy nodes to flooding attacks.

Under our assumptions, the use of effective authentication prevents DDoS attacks from external attackers. Therefore *MOTAG* is designed to combat attacks from authenticated users (insiders). Attackers can gain access to the application server and implant malicious insiders in the targeted system via social engineering, compromising legitimate clients, stealing clients' identities for authentication, and eavesdropping on clients' network connections. Insider attacks within a protected system are the results of targeted attacks with relatively high technical sophistication. Thus, the number of functioning insiders is expected to be small (maybe hundreds) compared to a typical external-only DDoS attack. Nevertheless, the damage they can cause is still significant. Once insiders uncover the IP addresses of some proxy nodes, they will notify external attackers who will carry out DDoS attacks against these exposed proxies. *MOTAG* is designed to combat such insider-assisted DDoS attacks, or simply insider attacks. Although insider attacks cannot be fully prevented, *MOTAG* aims to minimize their impact on innocent clients. In this portion of the paper, we discuss a client-to-proxy shuffling mechanism designed to quarantine insider attacks over time and ensure service accessibility for as many innocent clients as possible.

4.1. Shuffling Strategy Overview

In *MOTAG*, cloud computing capacity and bandwidth is reserved to meet the needs of the proxy nodes used in normal operations as well as providing sufficient capacity to instantiate a large number of additional proxy nodes. In the event of a DDoS attack, a small number of additional proxy nodes are instantiated. The total set of instantiated proxy nodes can be logically classified into two groups: namely *serving proxy nodes* and *shuffling proxy nodes*. The relatively static serving proxy nodes provide more reliable connection services for known innocent clients, while shuffling proxy nodes are responsible for the dynamic shuffling operations and are designed to provide intermittent connections to suspicious clients. During the course of an attack, sets of shuffling proxy nodes will be replaced and the associated clients reassigned to new shuffling proxy nodes. This shuffling of clients provides a continually moving target for insider-assisted attacks and allows *MOTAG* to isolate malicious insiders.

Prior to an attack, all the active proxy nodes are unmarked, and clients are randomly assigned to proxy nodes by the authentication server. Each client is assigned to only one proxy node. Proxy nodes that are subsequently attacked will be marked as shuffling proxy nodes while the unattacked proxy nodes remain unmarked and are still considered serving proxy nodes. Once a proxy node or nodes comes under attack, the authentication server employs the greedy algorithm described in Section 5.1 to repeatedly shuffle the client-to-proxy assignments within the shuffling proxy group to distinguish between, and eventually segregate, insiders and innocent clients.

After each shuffle, some shuffling proxy nodes will still be under attack, and some will not. The shuffling proxy nodes that are no longer under attack are unmarked and become serving proxy nodes and the associated clients are marked as trusted and considered as saved from the on-going attack (i.e. no longer affected by the attack). Clients connected to the attacked proxy nodes are considered untrusted since we cannot determine which clients are insiders. To save the innocent clients from among the untrusted group, the authentication server randomly re-distributes all the untrusted clients across the shuffling proxy nodes. Given an estimated number of suspicious clients and the available proxy nodes, new proxy nodes can be instantiated as shuffling proxy nodes to accelerate shuffling operations. As will be shown later, the more nodes used as shuffling proxies, the faster insiders will be quarantined and innocent clients saved. By repeating the client-to-proxy shuffling for multiple rounds and keeping record of the suspicious proxies/clients, most of the innocent clients can be gradually identified. Based on the estimated number of insiders and given a desired percentage of clients to be saved, the greedy algorithm initiates a limited number of shuffling rounds after which the (expected) percentage of innocent

clients will be saved, the remaining clients will be quarantined, and the attack damage will be minimized.

In order to reduce overhead, the shuffling process is stateless, meaning each shuffle is considered independent. The tags (trusted/untrusted) that are placed on clients are reset after each shuffle. These tags do not necessarily reflect the true identity of the clients, but instead characterize the proxy node to which they are assigned. Also, the roles of proxy nodes (shuffling/serving) are interchangeable across shuffles, depending on the behavior of attackers i.e previously unmarked proxy nodes can become shuffling proxy nodes if attacked during the shuffling process. The goal of shuffling operations is to separate innocent but attacked/suspected clients from true insiders. Because of the iterative nature of the sequential shuffles and since the characteristics of the individual clients are not used to determine the maliciousness of the client, insider attempts to mimic innocent clients will not affect *MOTAG*'s ability to detect and isolate attackers.

4.2. A Simple Shuffling Example

Figure 2 illustrates how client-to-proxy shuffling works. Initially, seven clients (including insiders, denoted by $C_1, \ldots, C_7$) are randomly assigned to three proxy nodes (denoted by K_1, K_2, and K_3), as suggested by the dotted lines: Client C_1, C_2, and C_3 are assigned to proxy node K_1, clients C_4 and C_5 are assigned to proxy node K_2, and clients C_6 and C_7 are assigned to proxy node K_3. In this example, clients C_3 and C_5 are insiders; they will divulge the location of the proxies in which they reside and bring an attack to proxy nodes K_1 and K_2. In this case proxy nodes K_1 and K_2 are marked as shuffling proxy nodes, and since the identity of specific clients cannot be determined, clients C_{1-5} are equally suspicious. As a result, the *MOTAG* reacts by replacing shuffling proxy nodes K_1 and K_2 with new shuffling proxy nodes K_4 and K_5. Clients and insiders previously assigned to K_1 and K_2 are re-assigned to K_4 and K_5. One possible reassignment scheme assigns C_1, C_3, and C_5 to K_4, while assigning C_2 and C_4 to K_5, as indicated by the solid lines. In this case, K_5 was not under attack and the clients on K_5 are saved. Conversely, K_4 is still be under attack and the clients on it are suspect. The assignment of clients C_6, C_7 and K_3 remain unchanged because they are not affected by the attack. As a result of this shuffle, only K_4 remains marked as a shuffling proxy node, and it's associated clients marked as suspect. Only K_4 will be involved in the next round of shuffling. Whereas proxy nodes K_3 and K_5 are unmarked as serving proxy nodes and will not be involved in the next round of shuffling.

4.3. Problem Modeling

To mitigate insider attacks as quickly as possible, and also to adapt to system dynamics such as client mobility, we need a shuffling algorithm that can identify and separate as many innocent clients as possible per shuffle. To that end, we first analyzed the effect of different client-to-proxy assignment schemes on the desired number of innocent clients saved. The main notations used in this model are summarized in Table 1.

Specifically, among a total number of N clients to be shuffled, the number of insiders is N_i, and the number of innocent

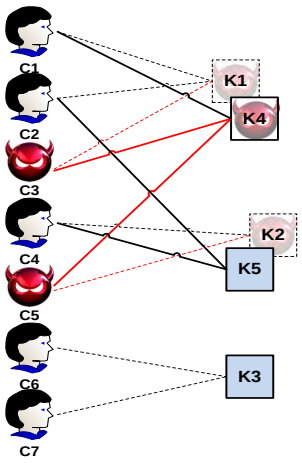


Figure 2: Client-to-Proxy Shuffling

Table 1: Notations used in this paper and their meanings

Notation	Meaning
N	number of clients (including insiders)
N_i	number of insiders
K	number of shuffling proxies
N_{cu}	number of innocent clients to be saved
N_{ca}	number of innocent clients that are under attack
A_j	number of clients assign to the j-th proxy
p_j	probability that the j-th shuffling proxy is not under attack

clients is N_c; so we have $N_i + N_c = N$ [1]. After one round of shuffling, N_{ca} innocent clients are still being attacked, and N_{cu} of them are not ($N_{ca} + N_{cu} = N_c$). Our goal is to mathematically compute the expected value of N_{cu} (denoted as $E(N_{cu})$) under different circumstances and find a method that maximize it given K available shuffling proxies. We use A_j to represent the number of clients appointed to proxy j.

Given that $E(N_{cu}) = \sum_{j=1}^{K} p_j A_j$, where p_j is the probability that proxy j is not under attack, an arbitrary proxy j, it is not under attack only when none of the insiders are connecting to it. Hence, p_j is also the probability that all insiders are assigned to proxy nodes other than j. According to simple combinatorics, $p_j = \binom{N-A_j}{N_i} / \binom{N}{N_i}$, where $\binom{N}{N_i}$ is the total number of ways to distribute the N_i insiders in the population N, and $\binom{N-A_j}{N_i}$ is the number of combinations in which all insiders are in the $N - A_j$ clients not connecting to proxy j. Therefore, the expected value of N_{cu} can be calculated by Equation (4.1).

$$E(N_{cu}) = \sum_{j=1}^{K} p_j A_j = \frac{\sum_{j=1}^{K} \binom{N-A_j}{N_i} A_j}{\binom{N}{N_i}} \tag{4.1}$$

[1] We provide a method to estimate the number of insiders in Section 5.4.

From the above we also have $E(N_{ca}) = N_c - E(N_{cu})$.

Given the total number of clients N, the number of insiders N_i, the number of shuffling proxies K, and the client-to-proxy assignment vector $\mathbf{A}$, we want to maximize $E(N_{cu})$. As a result of analyzing Equation (4.1), increasing the number of shuffling proxy nodes also increases the number of clients that are expected to be saved in each shuffle. In the extreme case where $K \geq N$, each client can be allocated to an exclusive proxy node ($A_j = 1, \forall j \in (1,K)$). $E(N_{cu}) = N_c$ implies that, in this case, no innocent client will be attacked. This is the ideal scenario where all insiders are quarantined within their own proxy nodes in a single round of shuffling. However, due to resource constraints, it will typically not be cost-effective to provide a dedicated proxy node for each client. In most cases, the client population would greatly outnumber the shuffling proxies ($K \ll N$). Consequently, the method of distributing clients across proxy nodes becomes vitally important to the number of clients saved.

Assuming we have a constant number of K shuffling proxy nodes, we are facing an optimization/maximization problem with Equation (4.1) as the objective function. The variables are summarized into the vector $\mathbf{A}$ of natural numbers that defines the client-to-proxy assignment scheme, with the constraint being

$$\sum_{j=1}^{K} A_j = N, \ \textit{where} \ \mathbf{A} \in \mathbb{N}^K \tag{4.2}$$

Although recursive algorithms such as dynamic programming can be employed to compute the optimal solution, we adopt a greedy approach described in Section 5.1 to produce a quick and near-optimal solution. Our simulations of various configurations showed that the results produced by the greedy algorithm approached very closely the theoretical upper bound of $E(N_{cu})$.

4.4. A Special Case

Before presenting the greedy algorithm solution to the client-to-proxy shuffling optimization problem in Section 5.1, we will first introduce a simple, elegant, and straightforward optimal solution to a special case of the general problem. We note that when the number of insiders (N_i) is less than or equal to the number of proxies (K), then evenly distributing all clients among all shuffling proxies will be optimal in saving maximum number of clients.

Theorem 1 *Evenly distributing clients to all shuffling proxies maximizes $E(N_{cu})$ when $N_i \leq K$ and client-to-proxy distribution is uniform.*

PROOF. We prove Theorem 1 using the exchange argument. For simplicity without loss of generality, we consider an arbitrary pair of proxy nodes among all shuffling proxy nodes. Assuming these two proxy nodes are the i-th and j-th proxy nodes, and there are A_i and A_j clients on them respectively, where $A_i + A_j = T$. Given that client-to-proxy distribution is uniform and $N_i \leq K (K = 2)$, it is expected that there are 1 or 2 insiders here, i.e. $N_i = 1$ or 2. Using Equation (4.1), when $N_i = 1$, we have

$$E(N_{cu}) = \frac{\binom{T-A_i}{1} \cdot A_i}{\binom{T}{1}} + \frac{\binom{T-A_j}{1} \cdot A_j}{\binom{T}{1}} = \frac{2 \cdot A_i \cdot A_j}{T}$$

When $N_i = 2$, we have

$$E(N_{cu}) = \frac{\binom{T-A_i}{2} \cdot A_i}{\binom{T}{2}} + \frac{\binom{T-A_j}{2} \cdot A_j}{\binom{T}{2}} = \frac{A_i \cdot A_j \cdot (T-2)}{T \cdot (T-1)}.$$

In both cases, $A_i = A_j$ maximizes $E(N_{cu})$. Since this pair of shuffling proxy nodes is randomly chosen, we can iteratively apply the exchange argument to any pair of shuffling proxy nodes, and eventually every proxy node will have the same number of clients. From this we can conclude that $E(N_{cu})$ is maximized when all proxy nodes are assigned an equal number of clients.

As a result, in the special case where the number of insiders is less than or equal to the number of shuffling proxies, we can obtain an optimal client-to-proxy assignment in $O(1)$ by randomly assigning N_i/K clients to each of the shuffling proxy nodes. Next we will discuss the general case solved by a greedy algorithm.

5. Greedy Algorithm

In this section, we present a greedy algorithm for guiding client-to-proxy shuffling. First, we show a naive version of the greedy algorithm which applies an enumeration approach in finding the local optimal. Then by applying approximation techniques, we improve the greedy algorithm and dramatically reduce its computational complexity. Next, we theoretically investigate the insider quarantine capability of the greedy algorithm under various attack scenarios. Finally, to support the greedy algorithm, we present maximum-likelihood estimation(MLE) approach for estimating the number of insiders.

5.1. The Greedy Shuffling Algorithm

As determined from the previous discussion, evenly distributing all clients to all proxy nodes is optimal for saving clients when the number of insiders is less than or equal to the number of clients. However, in reality, the number of insiders can be far greater than the number of proxy nodes. In this more likely case, evenly distributing clients may result in all proxy nodes being under attack and thus shuffling may be unable to save clients efficiently. Therefore, the strategy of evenly distributing clients among proxy nodes may not be applicable to the case when the number of insider is far more than the proxy number. This motivated us to design a shuffling algorithm that can be applicable to general case.

Intuitively, assigning fewer clients to a given proxy node should reduce the probability of introducing insiders to this proxy. However, if the proxy node is not under attack, assigning fewer clients to it also results in fewer clients being saved.

Therefore, an efficient shuffling algorithm was needed to determine how many clients to assign to each proxy node in order to maximize the expected number of clients saved among all proxy nodes.

Based on this intuition, the greedy algorithm decides how many clients to assign to a given proxy node, one proxy node at a time, based on the goal of maximizing the overall expected number of clients to be saved in a given shuffle. For an arbitrary proxy j, the number of clients assigned to it and the number of innocent clients that are expectedly to be saved are denoted as A_j and $E(A_j)$, respectively. The greedy algorithm computes $E(A_j)$ as follows.

$$E(A_j) = \frac{\binom{N-N_j}{A_j}}{\binom{N}{A_j}} \cdot A_j \tag{5.1}$$

The detailed procedure of greedy algorithm used in computing the client-to-proxy assignment is shown in Algorithm 1. The main function is called *GreedyAssign*. Since in Equation (4.1) $E(N_{cu})$ is the sum of expected number of clients saved for each proxy node (i.e. $p_j \cdot A_j$) for all the active shuffling proxy nodes, an optimality analysis was first performed for an individual shuffling proxy node. For an arbitrary proxy node j, A_j can be any value within $[0, N-1]$. A_j cannot be N; otherwise, all clients would be under attack if there is a single insider present.

Since the value of N_i will affect the optimal choice of A_j, for a particular N_i, all possible values of A_j will be enumerated and the parameter ω that maximized $p_j \cdot A_j$ will then be selected. This subroutine is described in procedure *MaxProxy* of Algorithm 1. Under our greedy approach, we assign ω clients to as many proxy nodes as possible.

Function *GreedyAssign* is called recursively to assign the remaining clients to the rest of the proxy nodes. The computation will terminate when any one of three conditions is met: first, when there are more proxy nodes left than clients, i.e. when each client will be assigned to an exclusive proxy node; second, when there is only one proxy node left, i.e. all remaining clients will be appointed to a single proxy node; and, third, when the expected number of remaining insiders is rounded to 0, i.e. there are no insiders, and all remaining clients will be evenly distributed for load balancing. The overall computational complexity of the greedy algorithm is $O(N \cdot N_i)$. To further reduce the computational overhead throughout the shuffling procedures, the client-to-proxy assignment vectors for different N, K, N_i combinations can be pre-computed and stored in lookup tables for runtime reference.

To evaluate the optimality of the greedy algorithm, the results of experimental implementations were compared with the theoretical upper bound of $E(N_{cu})$. Since Equation (4.1) is a summation of $p_j \cdot A_j$ for each individual shuffling proxy j, the maximal value of (4.1) cannot be greater than the sum of the max of each $p_j \cdot A_j$ when relaxing Constraint (4.2), i.e. $Max(E(N_{cu})) \leq K \cdot Max(p_j \cdot A_j)$. Here, $Max(p_j \cdot A_j)$ can be obtained by running subroutine *MaxProxy*$(N, 0, N-1, N_i)$. The comparison between the greedy algorithm and the theoretical upper bound is done via simulations under various configurations on MATLAB. The results are presented in Section 6.

Algorithm 1 Greedy algorithm for computing client-to-proxy assignment

```
function GreedyAssign(Client, Insider, Prox)
    if Client ≤ Prox then
        Assign 1 exclusive proxy to each client
    else if Prox = 1 then
        Assign all clients to the proxy
    else if Insider = 0 then
        Evenly distribute Client over Prox
    else
        ω = MaxProxy(Client, 0, Client − 1, Insider)
        ProxToFill = ⌊(Client/ω)⌋
        if ProxToFill ≥ Prox then
            ProxToFill = Prox − 1
        RemC = Client − ProxToFill · ω
        RemP = Prox − ProxToFill
        RemA = Round((Insider·RemC)/Client)
        Fill ProxToFill proxy nodes with ω clients each
        Fill the rest proxy nodes according to
        GreedyAssign(RemC, RemA, RemP)

procedure MaxProxy(Client, Lbnd, Ubnd, Insider)
    Max = 0, MaxAssign = 0
    for i = Lbnd → Ubnd do
        Save = C(Client−i, Insider) · i / C(Client, Insider)
        if Save > Max then
            Max = Save, MaxAssign = i
    return MaxAssign
```

5.2. Improving The Greedy Algorithm

As discussed above, the greedy algorithm aims to maximize the overall number of innocent clients expected to be saved by summing up the expected number of clients saved in each single proxy. The subroutine *MaxProxy* of Algorithm 1 finds the value of A_i that maximizes $E(A_i)$ in Equation (5.1) by enumerating all possible choices. For the purpose of the discussion, we name this approach the *enumeration approach*. The enumeration approach can find the optimal value of A_i in $O(N \cdot N_i)$ time complexity, where N denotes the total number of all clients and N_i denotes the number of insiders among the clients. Therefore, as N and N_i gets larger, the running time of the subroutine *MaxProxy* will become notably longer.

To improve the efficiency of the greedy algorithm, we designed an approximation approach that can find a near-optimal value of A_i in $O(1)$ time complexity. This is done by solving Equation (5.1) using Stirling's Approximation $n! \approx \left(\frac{n}{e}\right)^n \sqrt{2\pi n}$, and a series of other approximations assuming $N_i \ll N$, as follows.

$$
\begin{aligned}
E(A_i) &= \frac{(N-N_i)!(N-A_i)!}{(N-N_i-A_i)!\,N!}\cdot A_i \\
&\approx \frac{(N-N_i)^{N-N_i}(N-A_i)^{N-A_i}\sqrt{(N-N_i)(N-A_i)}}{(N-N_i-A_i)^{N-N_i-A_i}N^N\sqrt{(N-N_i-A_i)N}}\cdot A_i \\
&\approx \frac{(N-N_i)^{N-N_i}(N-A_i)^{N-A_i}}{(N-N_i-A_i)^{N-N_i-A_i}N^N}\cdot A_i \\
&\approx \left(\frac{N-N_i}{N}\right)^{A_i}\cdot A_i
\end{aligned}
$$

Let $A_i = \frac{N\cdot x}{N_i}$, we have

$$
E(A_i) = \left(1-\frac{N_i}{N}\right)^{\frac{N\cdot x}{N_i}}\cdot\frac{N\cdot x}{N_i} = e^{-x}\frac{N\cdot x}{N_i} \tag{5.2}
$$

After applying approximation $\lim_{n\to\infty}(1-1/n)^n \approx e^{-1}$ on the derivation of Equation (5.2), we get

$$
\frac{\partial E(A_i)}{\partial x} = \frac{e^{-x}\cdot N}{N_i}\cdot(1-x) \tag{5.3}
$$

From Equation (5.3), the derivation of $E(A_i)$ equals 0 if and only if $x = 1$. In other words, $x = 1$ maximizes $E(A_i)$. As a result, assigning $A_i = \frac{N}{N_i}$ clients to proxy i will optimize the expected number of benign clients that can be saved, which is

$$
E\left(A_i = \frac{N}{N_i}\right) = \frac{N}{N_i\cdot e} \tag{5.4}
$$

With the analysis above, we can improve the greedy algorithm by using an approximation approach for the subroutine *MaxProxy*. Unlike the enumeration approach which enumerates all the possible values of A_i to find an optimal value that could maximize Equation (5.1), the approximation approach computes the value of A_i only based on the total client number and number of insiders, i.e. $A_i = \frac{N}{N_i}$. The enumeration approach has computational complexity $O(N\cdot N_i)$ while approximation approach has computational complexity $O(1)$.

To evaluate the accuracy and the optimality of the approximation approach, we compared it to the enumeration approach using a series of simulations in MATLAB. Figure 3 shows the number of clients that should be assigned to a given proxy node in order to save the expected maximum number of innocent clients from one proxy node under varying conditions for both the enumeration and the approximation approach. Since the enumeration approach enumerates and compares all possible choices, we know that its results are optimal. In Figure 3, the results produced by the approximation approach overlap with those given by the enumeration approach in almost all cases indicating that the approximation approach is nearly optimal.

For ease of presentation, we term the algorithm which adopts the *enumeration approach* to implement *MaxProxy* subroutine as *greedy algorithm*, and name the variant which replaces the *enumeration approach* with *approximation approach* as the *improved greedy algorithm*. Taking the analysis one step further, we implemented both the greedy algorithm and the improved greedy algorithm in MATLAB and ran simulations with varying numbers of insiders and clients. In Figure 4, the y-axis represents the deviation between the improved greedy algorithm and the greedy algorithm in the percentage of clients saved. The y-axis values were computed using $y = \frac{s_2-s_1}{s_1}$, where s_1 and s_2 were the percentage of clients saved under greedy and improved greedy algorithm respectively. From Figure 4, we can see that the deviation was consistently less than 1%, and from this we can conclude that the improved greedy algorithm has performance very close to the greedy algorithm.

Since the improved greedy algorithm has a lower time complexity in deciding client-to-proxy assignment(i.e. $O(1)$), is likely to be computationally faster, and generates near-optimal results, we believe the improved greedy approach is preferable for practical usage.

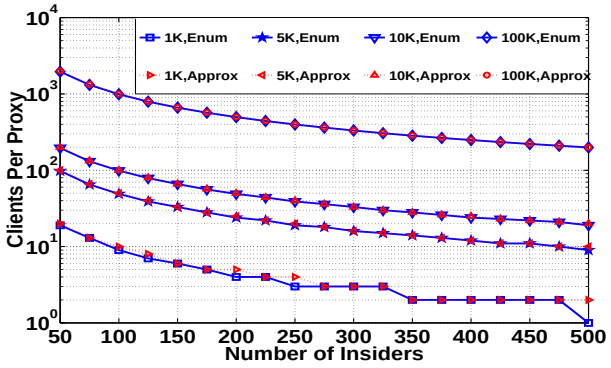


Figure 3: Number of clients assign to one proxy under enumeration approach and approximation approach, client numbers are 1K,5K,10K,100K respectively

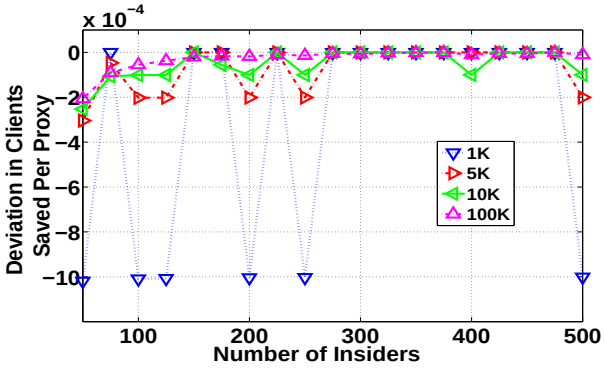


Figure 4: Deviation in percentage of innocent clients saved under greedy algorithm and improved greedy algorithm, proxy number is 100, client numbers are 1K,5K,10K,100K respectively

5.3. *Insider Quarantine Capability*

Since we are able to demonstrate that the greedy algorithm provides a near-optimal method for assigning clients to proxy nodes in order to save the maximum number of clients, we next approached the problem of estimating the insider quarantine capability of the greedy algorithm. This issue is considered very important since knowing the insider quarantine capability of the greedy algorithm would aid in estimating the number of proxy

nodes to instantiate as shuffling proxy nodes in order to meet quality of service (QoS) goals. Here, QoS refers to the specific percentage of clients saved.

The *insider quarantine capability* is defined as the percentage of clients can be quarantined (saved) from insiders given N clients (among which N_i are insiders), K proxy nodes and j rounds of shuffling. As is discussed in Section 5.2, the enumeration and the approximation approaches can both be used in the subroutine *MaxProxy* of greedy algorithm to achieve near-optimal assignment of clients to each shuffling proxy node. Since the approximation approach has performance close to that of the enumeration approach and is likely to perform faster, we used the approximation approach based greedy algorithm in analyzing the insider quarantine capability.

Equation (5.4) shows the number of clients that can be saved in one proxy node when there are N client and N_i insiders. From this equation it is evident that the number of clients saved is dependent on the number of shuffling proxy nodes used. Recall that the greedy algorithm tries to assign N/N_i clients to as many proxy nodes as possible, and put the rest of clients in the last proxy. Typically, the number of insiders is much larger than the number of shuffling proxy nodes, i.e. $N_i \gg K$, the last proxy is likely to be assigned far more clients than the other proxy nodes. Therefore, the last proxy is likely to have the highest probability of being attacked. For ease of analysis, we assume it will always be attacked. Thus the maximum expected number of clients saved will be the summation of the expected number of clients saved in the first $K-1$ proxy nodes, i.e. $\frac{(K-1)\cdot N}{N_i \cdot e}$. Let y be the percentage of clients saved among all N clients in one round of shuffle, then we have:

$$y = \frac{K-1}{N_i \cdot e} \tag{5.5}$$

As the number of insiders (N_i) will almost be the same each round and the number of shuffling proxy nodes (K) is a constant, we can regard y as the same number in each round of shuffling. If we define $E(U_j)$ as the percentage of clients not saved and $E(S_j)$ as the percentage of clients saved after j rounds of shuffle, then we have $E(U_j) = (1-y)^j$. Therefore, the percentage of clients saved in the first j rounds of shuffle is:

$$E(S_j) = 1 - E(U_j) = 1 - \left(1 - \frac{K-1}{N_i \cdot e}\right)^j \tag{5.6}$$

This result indicates that the percentage of clients saved with j rounds of shuffle only depends on the number of insiders,N_i, and the number of proxy nodes, K. The total number of clients,N, will not affect the percentage of clients saved.

In order to confirm the correctness of the achieved theoretical results, simulations were conducted using constant values for the number of shuffling proxy nodes and insiders (100 and 500 respectively), using $10K$, $50K$, $100K$ innocent clients and varying the number of rounds of shuffling. In Figure 5, the y-axis represents the accumulated percentage of clients saved for each value of the number of innocent clients. In all cases the experimental results very nearly overlap with the theoretical results. The small differences can probably be attributed to the use of the approximation approach in the greedy algorithm.

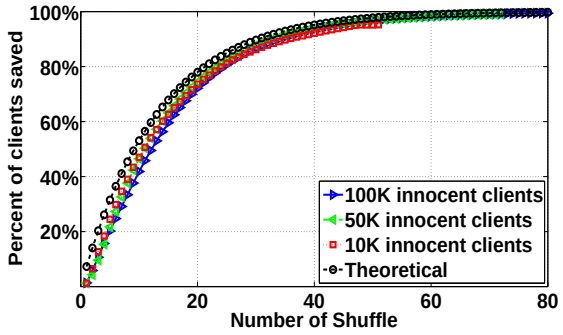


Figure 5: Percentage of clients saved among all clients under 100 proxy nodes, 500 insiders and various innocent client numbers

5.4. Estimating the Number of Insiders

In our earlier discussion, we assume the number of insiders (N_i) is fixed and given; however, in practice, we have no such prior knowledge. Since the value of N_i has direct influence on the client-to-proxy assignment, accurate estimation is important to the efficiency of the shuffling mechanism. We investigated the use of a maximum-likelihood estimation (MLE) as a source of an estimate of the number insiders.

In order to use the MLE, it was necessary to establish a connection between the number of insiders N_i and the number of proxy nodes that are not under attack (denoted as X). In a particular attack where $X = m$, we can calculate the probabilities $Pr(X = m)$ with regard to different N_i values, and use the N_i value that maximizes the probability as the estimated number of insiders. According to the inclusion-exclusion principle under balls-and-urns model [21], we can compose Equation (5.7) to calculate $Pr(X = m)$, where $Pr(X \geq M)$ stands for the probability that at least M ($M = m, m+1, \ldots, K$) proxy nodes are not attacked, K is the total number of all shuffling proxy nodes.

$$\begin{aligned} Pr(X=m) = {} & Pr(X \geq m) - \binom{m+1}{m} Pr(X \geq (m+1)) \\ & + \binom{m+2}{m} Pr(X \geq (m+2)) - \ldots \\ & + (-1)^{K-m} \binom{K}{m} Pr(X \geq K) \end{aligned} \tag{5.7}$$

In particular, these M not-under-attack proxy nodes constitute the set $\mathbf{U} = \{u_1, u_2, \ldots, u_M\}$, where u_j is the real ID of the jth available proxy node. Set $\mathbf{U}$ can be any M sized subset of the K shuffling proxy nodes.

The derivation of $Pr(X \geq M)$ is similar to the derivation of Equation (4.1). If a particular set $\mathbf{U}$ of proxy nodes are not attacked, the insiders must be among the clients assigned to the remaining proxy nodes (the complement of $\mathbf{U}$). Thus, we have Equation (5.8), in which $\sum_{\mathbf{U}}^{(M)}$ denotes the summation over all possible combinations of $\mathbf{U}$ (all M sized subsets of the K shuffling proxy nodes), and $N - \sum_{j=1}^{M} A_{u_j}$ gives the number of clients connecting to the proxy nodes not in $\mathbf{U}$. u_j is an arbitrary proxy node in the set, and A_{u_j} denotes the number of clients assigned to that node.

$$Pr(X \geq M) \quad = \quad \frac{\sum_{\mathbf{U}}^{(M)} \binom{N-\sum_{j=1}^{M} A_{u_j}}{N_i}}{\binom{N}{N_i}} \tag{5.8}$$

Under a certain client-to-proxy assignment scheme **A**, we can now derive $Pr(X = m)$ with N_i by combining Equation (5.7) and (5.8).

To evaluate the insider estimation algorithm, simulations were run in MATLAB varying the number of insiders. Based on the number of attacked proxy nodes, the algorithm uses the MLE to estimate the real number of insiders. The estimate of the number of insiders that maximizes the result of Equation (5.7) is used as the overall estimate. These estimations are plotted against the actual numbers of insiders in Figure 6. For each data point, we ran the simulation 30 times to compute the mean and 99% confidence intervals. The overlapping plots and the small error bound (relative to the number of insiders) indicate that the MLE estimate of the number of insiders is quite accurate.

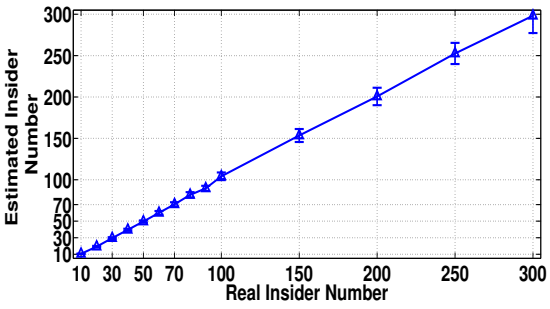


Figure 6: Insider estimation under 10K clients, 100 shuffling proxy nodes

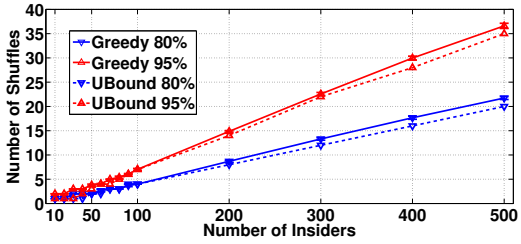
6. MOTAG Evaluation

In the previous discussions, we generated a theoretical basis for *MOTAG* and demonstrated via simulations, that our assumptions do not significantly degrade its expected performance. In this section, we assess the overall effectiveness of the *MOTAG* in mitigating the effects of a DDoS attack and evaluate the overhead introduced by the shuffling mechanism.

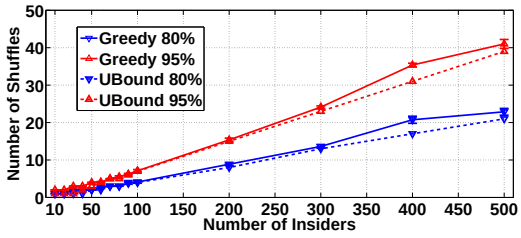
6.1. Insider Quarantine Capability Evaluation

In evaluating the effectiveness of *MOTAG*, we (1) compared the performance of an implementation of *MOTAG* using the greedy algorithm to the theoretical upper bound discussed in the previous sections, and (2) demonstrated that the greedy algorithm can segregate innocent clients from insiders in only a few shuffles.

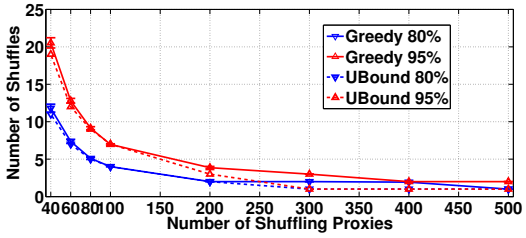
In the simulations, we implemented the algorithms of *MOTAG* on MATLAB and simulated various combinations of the number of clients, the number of insiders, the number of shuffling proxy nodes. Although *MOTAG* does not require any of these factors to remain constant from shuffle to shuffle, for ease of comparison, all these factor were held constant during each



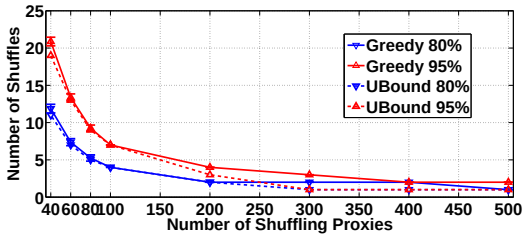
(a) Varying the number of insiders under 10K clients, 100 shuffling proxies



(b) Varying the number of insiders under 100K clients, 100 shuffling proxies



(c) Varying the number of shuffling proxies under 10K clients, 100 insiders



(d) Varying the number of shuffling proxies under 100K clients, 100 insiders

Figure 7: The number of shuffles needed to save 80% and 95% of innocent clients

simulation run (series of shuffles). In each simulation, the clients designated as insiders were randomly selected from the pool of clients using Mersenne twister [22] as our random number generator. The insiders were assumed to always generate attacks on the proxy nodes to which they were connected, and the attackers were assumed to have sufficient bandwidth to overwhelm an attacked proxy node. In practice, this means that a single insider assigned to a proxy node will ensure that node is attacked. However, as discussed earlier, due to the strong authentication enforced by the authentication server, we assumed only a limited number of insiders (hundreds) will be able to initiate attacks. In these simulations, *MOTAG* used the MLE method from Section 5.4 to estimate the number of insiders and uses

the greedy algorithm from Section 5.1 to determine the client-to-proxy assignments for the each shuffle.

Figure 7 shows the number of shuffles needed to save 80% and 95% of the innocent clients using the greedy algorithm (solid lines) and the theoretical upper bound from Section 5.1 (dotted lines). As was expected, saving a specific percentage of the innocent clients requires more shuffles as the number of insiders increases and fewer shuffles as the number of shuffling proxy nodes increases. In Figures 7a and 7b the total number of clients (10k for 7a and 100k for 7b) and the number of shuffling proxy nodes (100) are held constant while the number of insiders varies from 10 to 500. In Figures 7c and 7d the total number of clients are the same as in the previous two figures, but the number of insiders is held constant at 100 and the number of shuffling proxy nodes varies between 40 and 500. Each simulation was run 30 times to generate average data points and 99% confidence intervals.

Overall, the results of the simulations presented above indicate that the performance of *MOTAG* is close to the theoretical optimum. This suggests that the reassignment of clients to shuffling proxy nodes is also nearly optimal. Specific conclusions can be drawn from the individual figures. Figures 7a and 7b show that the number of shuffles needed to save the same percentage of innocent clients grows approximately linearly with the increase in the number of insiders. Similarly, Figures 7c and 7d indicate that the number of shuffles required to save a given percentage of innocent clients increases when fewer proxy nodes are used in each shuffle. The number of shuffles rises slowly while the number of proxy nodes outnumbers the insiders, but that number begins to rise ever more steeply when the number of proxy nodes falls below the number of insiders (100). Moreover, the narrow confidence intervals of all the data points indicate that the performance of the *MOTAG* shuffling algorithm is reliable and predictable.

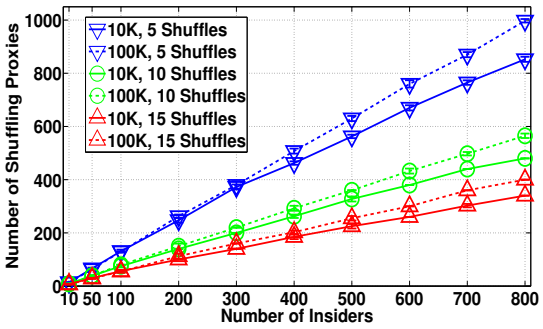


Figure 8: Number of proxy nodes needed to save 95% of innocent clients within 5, 10, and 15 shuffles, with 10K and 100K clients and a increasing number of insiders

Figure 8 shows the relationship between the number of insiders and the number of shuffling proxy nodes for 5, 10 , and 15 shuffles with the number of clients held constant at 10k (solid lines) and 100k (dotted lines). For both cases, when 80% and 95% of the innocent clients are saved, there seems to be a nearly linear relationship between the number of required shuffling proxy nodes and the number of insiders. Also, the 10-fold increase, from 10K to 100K, in the number of clients does not require a similar 10-fold increase in the number of shuffling proxy nodes. This seems to indicate that the greedy shuffling algorithm is robust with respect to the number of innocent clients. Overall, the simulation results show that *MOTAG* is effective in mitigating the effect of DDoS attack initiated by hundreds of insiders within a few shuffling.

6.2. Overhead

The experimental results presented above indicate that using *MOTAG* would be effective in defense, but implementing *MOTAG* would also increase the overhead in the communications between clients and the application server due to the proxy-based communication relay and the client-to-proxy shuffling.

Table 2: Latency overhead introduced by proxy indirection

	Direct	Indirect			
	RTT	Mean RTT	Overhead	Max RTT	Overhead
1	63ms	104ms	63.35%	143ms	125.41%
2	86ms	99ms	15.64%	128ms	49.45%
3	83ms	102ms	23.73%	133ms	60.47%
4	90ms	112ms	23.77%	131ms	45.18%
5	84ms	107ms	27.73%	120ms	42.48%

Table 3: Throughput overhead introduced by proxy indirection (Mb/s)

	1	2	3	4	5
Direct	90.66	83.46	86.24	123.30	121.20
Indirect	15.20	14.46	13.99	15.97	14.09

First, to assess the overhead introduced by proxy-based traffic relays, 10 geographically distinct U.S. nodes were selected from PlanetLab to form 5 end-to-end flows. Also, 24 other nodes that spread across the country were selected to serve as proxy nodes. The latency and throughput for both direct and relayed communications were measured for each of the 5 flows, and the results are shown in Tables 2 and 3, respectively. SSH tunneling through individual proxy nodes was employed to redirect traffic between end nodes. Mean round trip times (RTT) were obtained by bouncing short TCP messages back and forth between the end nodes of each flow 100 times. Throughput data was the average of 10 Iperf [23] sessions. From the tables, the impact of introducing proxy nodes on latency (usually less than 30%) is much less significant than its influence on throughput. The drop in throughput is not only due to traffic redirection in the proxy nodes, but is also a result of message encryption and decryption by SSH agents. Since *MOTAG* does not specify communication protocols, various cryptographic strategies, including no encryption, can be listed as options when implementing *MOTAG* based systems. Users can make informed decisions based on the nature of the protected application.

Table 4: Time to switch between two proxy nodes (seconds)

	1	2	3	4	5
MEAN	0.514	0.512	0.509	0.546	0.530
MAX	0.677	0.773	0.693	0.714	0.753
MIN	0.291	0.208	0.249	0.357	0.214

The overall agility, and therefore the usability, of *MOTAG* is dependent on the time needed to shuffle clients among different proxy nodes. Rapid shuffles will make it harder for attackers to "follow" the moving target (proxy) nodes as well as reducing the time needed to quarantine insiders. Likewise, reducing the time for individual shuffles will save clients more rapidly and improve the overall QoS by reducing the severity of service disruptions. Therefore, to quantify the impact of our mechanism on the end users, the time needed for a client to switch from one proxy node to another was determined. To that end, 5 geographically dispersed nodes were chosen from PlanetLab to be the destination servers. Another node was randomly selected to play the role of the authentication server. Finally, 8 PlanetLab nodes were selected as proxy nodes. The time between when the redirection message was sent by the authentication server to the client and when the client connected to the destination server via the new proxy node was recorded as the time to migrate clients between proxy nodes. Concurrent with client migration, the authentication server sends a session ticket to both the client and the new proxy node, the client is authenticated by the proxy node when this ticket is validated. Only upon authentication will the new proxy node begin relaying packets for the client. Table 4 presents the average, maximum, and minimum proxy switching times for each destination node. The numbers are fairly small yet consistent; the less than one second proxy switching time should not cause significant service disruption for most non-realtime applications.

7. Related Work

A number of research efforts have been devoted to defense against DDoS attacks over the past decade [24]. Filtering-based approaches [3, 4, 5] intend to use ubiquitously deployed filters to block unwanted traffic far away from the protected nodes. These filters assume that attack traffic can be differentiated from legitimate traffic. However, this is usually a difficult task because clever attackers can spoof IP addresses and mimic legitimate senders. Instead of trying to distinguish and then block malicious traffic, *MOTAG* uses client authentication to filter out unauthorized clients. Only authenticated clients will be assigned to the hidden moving Internet proxy noted that directly communicate with the protected application server.

Capability-based mechanisms employ a different philosophy from filter mechanisms. Capability-based mechanisms focus on controlling resource usage by the destination node [6, 7, 8, 9]. Senders have to obtain the receivers' explicit permission before transmitting packets to them. Traffic from authorized or privileged senders with valid capability permissions can be prioritized during an attack. Using capability-based mechanisms is a more proactive form of defense than filtering, but both solutions rely on the global adoption of protocols on Internet routers for adequate capability enforcement. This is unlikely to happen given limited financial incentives. *MOTAG* employs a dynamic capability-based mechanism to augment the moving target defense. *MOTAG* uses capability tokens to identify and rate-limit authenticated clients at the proxy nodes. The use of these capability tokens allows legitimate clients to use (but not abuse) the dynamically adjusted aggregate bandwidth of the proxies rather than depending on static defenses on Internet routers for traffic policing.

To eliminate the physical network constraints and administrative boundaries, secure overlay networks have been implemented to provide flow authentication, filtering, and redirection as well as attack tracking and tolerance enhancement [10, 11, 12, 13, 14, 15]. The goal of overlay networks, such as TOR [25], is to hide the protected nodes behind a well-provisioned, distributed overlay network that is capable of absorbing DDoS traffic. But, by using an exposed, relatively static, overlay network to withstand the ever-intensifying DDoS attacks inflicted by expanding botnets, the defenders will involve themselves in a never-ending bandwidth arms race with potential attackers. Even if a strong overlay network that can mitigate the effects of DDoS attacks is in place, advanced attackers can attack a small portion of the overlay network and sweep through the entire network step by step [11]. By repeating such sweeping attacks, attackers are guaranteed to hit the critical nodes and cause major service disruptions. Sophisticated attackers can even assess the impact of their attacks via recruited legitimate clients, and then adapt their attacks based this feedback and focus on the pinch points [12]. Moreover, overlay networks are vulnerable to insider attacks that can potentially expose the protected server to external attacks [26].

As an alternative to overlay networks, other mechanisms that hide the paths to selected services behind intermediate protections [27, 28] have been implemented. These mechanisms tend to employ a simpler, easier-to-deploy protection layer to filter out un-authorized traffic and are thus conceptually similar to *MOTAG*. Unfortunately, these protections layers fail to defend against attacks in which authorized clients act as malicious insiders to compromise their inter-layer protection. In this paper, we thoroughly analyzed insider threats and proposed a shuffling mechanism to quarantine insider attacks.

MOTAG takes advantage of the mobility in its packet relay proxy nodes in its shuffling mechanism that segregates insiders from innocent clients. The mobile relay proxy nodes resemble the earlier network address randomization technique against hitlist worms [29] and the fast-flux scheme to sustain accessibility to illegal commercial websites [30]. To the best of our knowledge, we are the first in using such dynamic method on defense against DDoS attacks.

8. Discussion

MOTAG is a dynamic traffic relay framework open only to

authenticated clients. It is designed to counter both brute force and sophisticated DDoS attacks against the protected application server as well as other *MOTAG* components.

Resistance to brute-force attacks. With *MOTAG* protection, external attackers will not be able to locate the hidden proxy nodes without planting insiders, i.e. compromising or eavesdropping on legitimate clients. Using hidden proxy nodes ensures that external attackers will not be able to discover the IP range of the entire proxy pool, and individual insiders will only be able to discover the IP addresses of a small subset of the active proxy nodes. The IP filtering employed by the *MOTAG* proxy nodes and the filtering ring ensures that a *MOTAG* implementation will be resistant to scanning attacks because all active proxies will only respond to IP addresses representing legitimate clients. The ability to distribute the individual components of *MOTAG* to one or more cloud providers means that the proxy pool (which can be the entire cloud domain) can potentially be so large that even powerful botnets will be unable to attack all the active proxy nodes.

The mobility of proxy nodes adds another layer of resiliency against brute-force attackers. If attackers discover a secret proxy node by chance, the attacked node will quickly "move away"; and without the ability to trace the shifting proxies, external attackers will not be able to retarget their attack.

The only exposed (static) component of the system is the authentication server. Existing PoW schemes provide protection from external attacks by isolating the *MOTAG* components from external threats through strong authentication protocols.

Resistance to insider attacks. Malicious insiders pose a more serious DDoS threat to a *MOTAG* protected server than external attackers because insiders will have access to the IP address(es) of one or more secret moving proxy nodes, and can expose the proxy nodes to potentially powerful external botnet attacks. As discussed in Section 4, by dynamically "moving" proxy nodes and shuffling client-to-proxy designation optimally, *MOTAG* can quarantine insiders in a few rounds, thus mitigating the effects of a DDoS attack and maintain the QoS of authenticated clients. *MOTAG* protects the precious bandwidth of the application server, and any bandwidth that is wasted due to attackers is at most proportional to the ratio of proxies under attack (rather than the number of attackers). Any wasted bandwidth will be recovered as multiple rounds of shuffling reduce the number of proxy nodes under attack. In addition, since the shuffling decisions are specific to each round, *MOTAG* can easily adjust to changing to system dynamics such as the varying arrival and departure rates of both clients and insiders. Some insiders may delay attacking for several shuffles, aiming to profile the IPs of the proxy nodes. However, since proxy nodes that are not under attack are not marked as shuffling proxy nodes, the associated clients are not shuffled, and silent insiders will only be able to discover the IP address of a single proxy node and thus fail in their purpose.

Resistance to compromised proxies. With the help of malicious insiders, attackers may compromise some proxy nodes. If successful, the application server and the authentication server will be directly exposed to attackers. However, lightweight authenticators, similar to what was used in [12], can be used to identify and filter proxy-to-server traffic in the filter ring. Therefore, traffic from any compromised proxy nodes that are exploited to attack the application server can be readily identified and their packets blocked by the high-speed filter ring deployed around the server.

9. 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.

10. Conclusion

In this paper, we presented *MOTAG*, a framework that employs dynamic, hidden proxy nodes as moving targets to mitigate insider-assisted network flooding DDoS attacks. The dynamic proxy nodes perform packet forwarding between the authenticated clients and a protected application server. When a DDoS attack is mounted against *MOTAG* proxy node(s), the authenticated clients connected to the attacked proxies are reassigned to alternative proxy nodes in realtime, enabling them to evade the ongoing attack and maintain access to the protected services. By continuously replacing the attacked proxies and reassigning (shuffling) the attacked clients onto the new proxies, *MOTAG* is able to separate innocent clients from insiders through a series of shuffles.

To optimize the segregation of innocent clients, we designed a novel, efficient greedy algorithm that guides the client-to-proxy shuffling. In addition, the insider quarantine capability of the greedy algorithm was studied and quantified to enable defenders to estimate the resources required to defend against DDoS attacks and meet defined QoS levels under various attack scenarios. The simulation results showed that *MOTAG* can efficiently mitigate a DDoS attack assisted by hundreds of insiders using a small number of shuffles. In the future work, we plan to study the economic cost of this defense scheme and extend the defense scheme to protect a system with anonymous clients.

References

[1] R. Dobbins, C. Morales, Worldwide infrastructure security report vii (2011).
URL http://www.arbornetworks.com/report

[2] T. Micro, Russian underground 101, http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-russian-underground-101.pdf (2012).

[3] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, S. Shenker, Controlling high bandwidth aggregates in the network, ACM Computer Communication Review 32 (2002) 62–73.

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

[5] X. Liu, X. Yang, 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, ACM, New York, NY, USA, 2008, pp. 195–206. doi:http://doi.acm.org/10.1145/1402958.1402981.

[6] T. Anderson, T. Roscoe, D. Wetherall, Preventing internet denial-of-service with capabilities, SIGCOMM Comput. Commun. Rev. 34 (1) (2004) 39–44. doi:http://doi.acm.org/10.1145/972374.972382.

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

[8] X. Yang, D. Wetherall, T. Anderson, Tva: a dos-limiting network architecture, IEEE/ACM Trans. Netw. 16 (6) (2008) 1267–1280. doi:http://dx.doi.org/10.1109/TNET.2007.914506.

[9] X. Liu, X. Yang, Y. Xia, Netfence: preventing internet denial of service from inside out, in: Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM, SIGCOMM '10, ACM, New York, NY, USA, 2010, pp. 255–266. doi:http://doi.acm.org/10.1145/1851182.1851214.
URL http://doi.acm.org/10.1145/1851182.1851214

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

[11] A. Stavrou, A. D. Keromytis, Countering dos attacks with stateless multipath overlays, in: Proceedings of the 12th ACM conference on Computer and communications security, CCS '05, ACM, New York, NY, USA, 2005, pp. 249–259. doi:10.1145/1102120.1102153.
URL http://doi.acm.org/10.1145/1102120.1102153

[12] 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, USENIX Association, Berkeley, CA, USA, 2003, pp. 3–3.

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

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

[15] C. Dixon, T. Anderson, A. Krishnamurthy, Phalanx: withstanding multimillion-node botnets, in: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, NSDI'08, USENIX Association, Berkeley, CA, USA, 2008, pp. 45–58.
URL http://dl.acm.org/citation.cfm?id=1387589.1387593

[16] Q. Jia, K. Sun, A. Stavrou, Motag: Moving target defense against internet denial of service attacks, in: Proceedings of the 22nd International Conference on Computer Communications and Networks (ICCCN), Nassau, Bahamas, 2013.

[17] T. Aura, P. Nikander, J. Leiwo, Dos-resistant authentication with client puzzles, in: Security Protocols Workshop, 2000, pp. 170–177.

[18] D. Dean, A. Stubblefield, Using client puzzles to protect tls, in: Proceedings of the 10th conference on USENIX Security Symposium - Volume 10, SSYM'01, USENIX Association, Berkeley, CA, USA, 2001, pp. 1–1.
URL http://dl.acm.org/citation.cfm?id=1251327.1251328

[19] B. Waters, A. Juels, J. A. Halderman, E. W. Felten, New client puzzle outsourcing techniques for dos resistance, in: Proceedings of the 11th ACM conference on Computer and communications security, CCS '04, ACM, New York, NY, USA, 2004, pp. 246–256. doi:10.1145/1030083.1030117.
URL http://doi.acm.org/10.1145/1030083.1030117

[20] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, Y.-C. Hu, Portcullis: Protecting connection setup from denial-of-capability attacks, in: Proceedings of the ACM SIGCOMM, 2007.

[21] N. Johnson, S. Kotz, Urn Models and Their Applications: An Approach to Modern Discrete Probability Theory, Wiley, New York, 1977, Ch. 1.3.2.

[22] M. Matsumoto, T. Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, ACM Trans. Model. Comput. Simul. 8 (1) (1998) 3–30. doi:10.1145/272991.272995.
URL http://doi.acm.org/10.1145/272991.272995

[23] Iperf, http://iperf.sourceforge.net.

[24] M. Abliz, Internet denial of service attacks and defense mechanisms, Tech. Rep. TR-11-178, University of Pittsburgh (Mar 2011).

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

[26] L. Overlier, P. Syverson, Locating hidden servers, in: Proceedings of the 2006 IEEE Symposium on Security and Privacy, SP '06, Washington, DC, USA, 2006, pp. 100–114.

[27] V. Kambhampati, C. Papadopoulos, D. Massey, Epiphany: A location hiding architecture for protecting critical services from ddos attacks, in: The 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012), 2012.

[28] X. Wang, M. K. Reiter, Wraps: Denial-of-service defense through web referrals, in: 25th IEEE Symposium on Reliable Distributed Systems, IEEE Computer Society, 2006, pp. 51–60.

[29] S. Antonatos, P. Akritidis, E. P. Markatos, K. G. Anagnostakis, Defending against hitlist worms using network address space randomization, in: Proceedings of the 2005 ACM workshop on Rapid malcode, WORM '05, ACM, New York, NY, USA, 2005, pp. 30–40. doi:10.1145/1103626.1103633.
URL http://doi.acm.org/10.1145/1103626.1103633

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