MOVE: An End-to-End Solution To Network Denial of Service

Angelos Stavrou Angelos D. Keromytis Jason Nieh Vishal Misra Dan Rubenstein
Columbia University in the City of New York
{*angel,angelos,nieh,misra,danr*}*@cs.columbia.edu*

Abstract

We present a solution to the denial of service (DoS) problem that does not rely on network infrastructure support, conforming to the end-to-end (e2e) design principle. Our approach is to combine an overlay network, which allows us to treat authorized traffic preferentially, with a lightweight process-migration environment that allows us to move services easily between different parts of a distributed system. Functionality residing on a part of the system that is subjected to a DoS attack migrates to an unaffected location. The overlay network ensures that traffic from legitimate users, who are authenticated before they are allowed to access the service, is routed to the new location. We demonstrate the feasibility and effectiveness of our approach by measuring the performance of an experimental prototype against a series of attacks using PlanetLab, a distributed experimental testbed. Our preliminary results show that the end-to-end latency remains at acceptable levels during regular operation, increasing only by a factor of 2 to 3, even for large overlays. When a process migrates due to a DoS attack, the disruption of service for the end user is in the order of a few seconds, depending on the network proximity of the servers involved in the migration.

1 Introduction

One of the fundamental tenets of Internet design is the end-to-end (*e2e*) principle [40], which states that functionality should be placed as close to the network edges as possible, keeping the network core focused on the task of routing packets. While this has arguably contributed to the success of the Internet, it has created an ideology that resists mechanisms that require deployment in the interior (core) of the network. Examples of such technologies include some forms of QoS [4] and active networking [52].

Recent events have elevated network denial of service (DoS) attacks to a first-order security threat. While several mechanisms to suppress or counter their effects have been proposed [16, 44, 47, 51, 38, 35, 21, 53], so far none has been widely adopted. One characteristic these mechanisms share is their dependence on elements of the network infrastructure. Furthermore, it has been argued recently [12] that the network DoS problem is inherently impossible to solve without infrastructure support. This may provide some insight as to the lack of deployment of any of these mechanisms. *The question we examine in this paper is whether it is possible to provide a practical solution to the network DoS problem that does not require significant (or any) cooperation by network providers (ISPs).*

We present **M**igrating **OVE**rlay (MOVE), a system that aims to provide an *e2e*-compatible anti-DoS mechanism. Our approach is to separate "good" traffic from unknown traffic, and treat the former preferentially, using an overlay network in a manner similar to SOS [21, 22, 7, 30] but without using packet filtering. MOVE nodes are located at edge networks, requiring no infrastructure support. Traffic is differentiated on a per-session basis, using cryptographic authentication and/or Graphic Turing Tests (GTTs) to determine valid users (which may simply mean "humans"). The overlay routes traffic from legitimate users to the current network location of the protected service. When an attack against the hosting site is detected, we use a lightweight process-migration mechanism to re-locate the service to an unaffected site. Legitimate traffic is routed transparently to the new location, while malicious (or simply unknown) traffic will continue flowing to the old location. Our difference from SOS [21, 22], WebSOS [7, 30, 48], and Mayday [1], is that MOVE does not require a filtering perimeter to be constructed around the target site; instead, we use process migration to move (and obscure) the current location of the attacked service, and "stepping stone" hosts to maintain connectivity between the original site and the new location of the service. Architecturally, SOS introduced the general idea of using

an overlay and filtering to protect against some classes of DDoS attacks. WebSOS enhanced the front-end of the overlay (its interface with the remote clients) to enable more *ad hoc* interactions than SOS allowed. MOVE concerns itself with the back-end of the overlay (its interface with the protected sites), removing the dependency on network filtering. Our approach is similar to the concept of "hidden servers" in anonymity systems such as Tor [37, 11], although our use of server migration in MOVE allows us to protect against a larger class of attackers.

No aspect of MOVE depends on the network infrastructure itself, although it makes certain assumptions about the threat model. In particular, (a) there is a notion of *legitimate users*, (b) the attackers cannot take over arbitrary routers or eavesdrop at will on arbitrary network links, and (c) there exists a relatively large number of potential hosting sites. We discuss these assumptions further in Section 2.

Where these assumptions hold, we believe MOVE to be the first anti-DoS mechanism that does not require any additional functionality from the network. We hope to demonstrate that by making careful assumptions and relaxing the threat model in realistic ways, it is possible to design *efficient* and *effective* protection mechanisms that do not violate prevalent system and network design principles. To that effect, we test our experimental prototype on PlanetLab [36], a testbed for experimentation with network overlays. As we show, the overlay mechanism increases end-to-end latency by a factor of 2 to 3. Migrating a web server and its associated state causes less than 10 seconds of service disruption for the end user, and connectivity resumes transparently to the end applications; in the case of a VNC server with substantially more state, the service disruption time ranges between 17 and 22 seconds. The attacker is left with no indication as to the new location of the service, thus having to either distribute the attack traffic among various potential targets or try to guess the correct hosting site. An attacker that cannot guess the new location faster than 10 seconds (for the web server case) cannot permanently disrupt access to the service. Similar results are obtained when migrating a remote display application, VNC. Furthermore, clients need to use MOVE only when their connectivity to a service is disrupted; under regular network conditions, direct access to the servers would typically be used, minimizing the performance impact of MOVE.

The remainder of this paper is organized as follows. Section 2 gives an overview of the MOVE system architecture after describing its components. Section 3 describes our prototype implementation, and Section 4 presents some preliminary experimental results. We discuss other mechanisms that address the DoS problem in Section 5, and conclude the paper with Section 6.

2 System Architecture

We first describe the threat and application models under which our system operates. We then present the MOVE architecture, which uses elements of overlay networking and process migration.

2.1 Threat and Application Model

DoS attacks can take many forms, depending on the resource the attacker is trying to exhaust. For example, an attacker can try to cause the web server to perform excessive computation, or exhaust all available bandwidth to and from the server. In all forms, the attacker's goal is to deny use of the service to other users. Apart from the annoyance factor, such an attack can prove particularly damaging for time- or life-critical services (*e.g.,* tracking the spread of an real-world epidemic), or when the attack persists over several days. Of particular interest are *link congestion* attacks, whereby attackers identify pinch-points in the communications substrate and render them inoperable by flooding them with large volumes of traffic. An example of an obvious attack point is the location (IP address) of the destination that is to be secured, or the routers in its immediate network vicinity; sending enough attack traffic will cause the links close to the destination to be congested and drop all other traffic.

We assume that attackers are smart enough to exploit features of the architecture that are made publicly available. We do not specifically consider how to protect the architecture against attackers who can infiltrate the security mechanism that distinguishes legitimate traffic from (illegitimate) attack traffic: we assume that communications between overlay nodes remain secure so that an attacker cannot send illegitimate communications, masking them as legitimate. In addition, it is conceivable that more intelligent attackers could monitor communications between nodes in the overlay and, based on observed traffic statistics, determine additional information about the current configuration. Such attackers would have the ability to subvert arbitrary routers and/or eavesdrop at will on network links. As such attacks are considerably more difficult than denial of service, we consider them outside our scope. In [56], the authors analyze our overlay architecture under a model allowing for attackers that can compromise arbitrary nodes in the overlay, toward determining the identity of the beacon and/or secret servlets. They conclude that by layering

multiple overlays on top of each other, one can trade off increased resistance to such attackers with end-to-end performance.

Our prototype implementation is focused on two applications (although not limited to these): a web server and a remote display access application (VNC) [39]. We chose the Web as an implementation mechanism due to the facilities that common servers and browsers provide and the ease with which we could develop a prototype implementation. VNC is a good example of an application that maintains considerable state that cannot be easily replicated, without being "storage-heavy". In general, the applications we are most interested in are real-time, server-assisted applications, and other applications that require some state to be maintained by the server but are not fundamentally storage-oriented (*i.e.,* their processing component dominates the system performance costs). Our system supports applications that require access to a storage back-end, by maintaining a "lifeline" between the new and the old location of the service. The service can access the storage back-end over this lifeline, with some loss of performance, which we measure in Section 4. We discuss the lifeline in more detail in Section 2.2.1.

Note that applications that do not require any state to be maintained can simply be load-balanced across several sites and contacted using Anycast, RR-DNS, *etc.*. Likewise, content-delivery applications can use data replication services such as Akamai to increase availability.

Finally, we assume that there exists a number of hosting sites that can accept the migrating service. These can be statically provisioned, *e.g.,* through a co-operative agreement among various service providers, or allocated on demand from a commercial entity selling (or renting) CPU time. In either case, we assume that there exist enough such hosting sites that an attacker cannot simply overwhelm all of them with a coordinated DoS attack. Instead, the attacker can successfully attack some (small) percentage of such sites. Note that these hosting sites *are not* part of the overlay itself. They only host migrated services, presumably under some contractual agreement with the owners of such services.

2.2 MOVE Architecture

The overall architecture of our system is shown in Figure 1, and shares several similarities with the architecture of WebSOS [30, 7]. For the remainder of the discussion, we will focus on web clients and servers, although our approach can easily be generalized to other services.

2.2.1 Servers In MOVE

Servers that are to be protected inform the overlay of their current location. Such servers are also authenticated using a standard security protocol such as SSL/TLS [10] with client certificates, or IPsec [20]. When a server migrates to a new location, it simply informs the overlay of its new location. Note that multiple servers, belonging to different organizations, may be using the system at the same time; the certificates they hold allow them to change the location status only for themselves, *i.e.,* they cannot cause the overlay to redirect and capture a competitor's traffic. We assume the existence of enough locations to choose as the new destination that it is impractical for an attacker to simply attack each site and determine (based on service response latency) which one is hosting a particular service; that is, it will take too long for an attacker to locate a service using that (or a similar) approach, compared to how quickly the system can migrate to a new location. We experimentally quantify this delay in Section 4.

However, notice that the process migration mechanism itself uses the network, which is presumed to be under a DoS attack. In keeping with the spirit of the *e2e* design, we preclude use of any form of QoS provisioning (although such arrangements can be very effective and do not require a lot of overhead, since the endpoints are fixed and known *a priori*). Instead, we assume that each hosting site has a secondary, potentially low-bandwidth connection to the Internet with a different IP address (either with the same or, better yet, with another ISP), which is not advertised through BGP. Thus, attack traffic from outside the home ISP cannot reach that interface, even if the attacker knows this alternate address.

We also require "stepping stones" in the same home ISP (but not necessarily operated by the ISP, *i.e.,* they can be located in end-networks), which allow the migrating process to reach a host that can communicate with other nodes outside the home ISP. During process migration, the server binary and state are saved (as we shall describe in Section 2.3), transfered to one of the stepping stones, and thence to a "random" hosting site, where the service is restarted. The service will then notify the overlay of its new location. The overlay will then redirect traffic from legitimate clients to this new location. The stepping stones can be part of the overlay (admitting new clients and routing their traffic), dedicated nodes, hosting sites that use the same home ISP, or any combination of these. The only important characteristic is that they can communicate beyond the local ISP, and with the ISP-specific address of the attacked site. For example, the ISP may be using internally a net-10 address

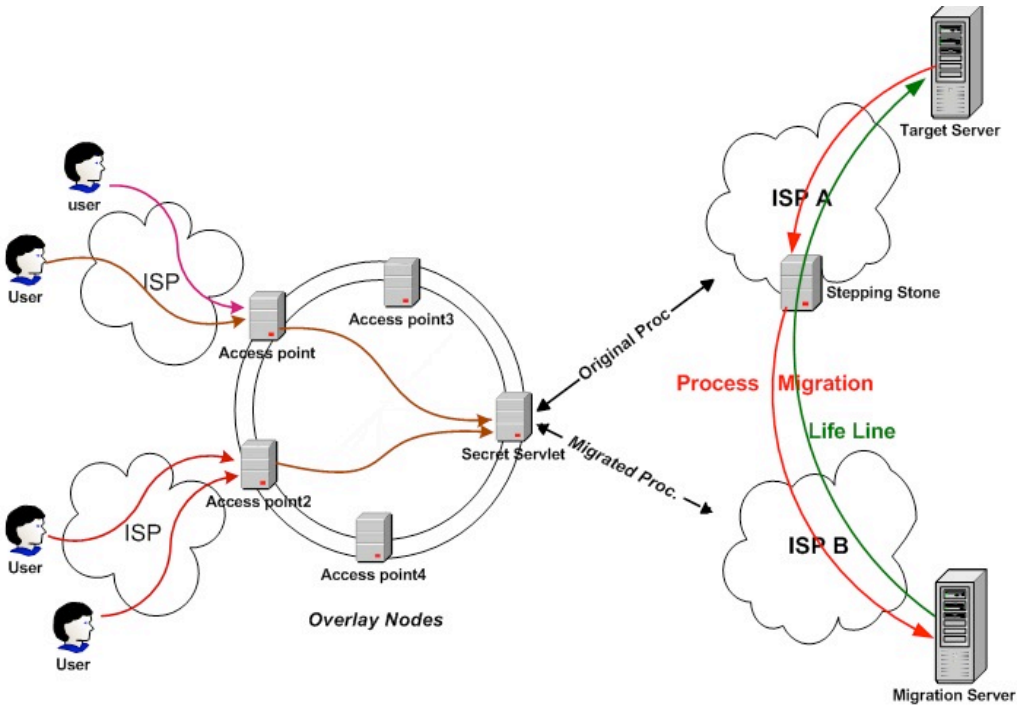

Figure 1. **Migrating Overlay system architecture.**

(*i.e.*, an address in the private `10.0.0.0/8` prefix) for the secondary address of stepping stones and protected sites. The stepping stone node also uses a temporary address that belongs to the ISP, that *is* globally routable. This address is changed periodically, such that an attacker cannot target a stepping stone with attack traffic. Thus, we have created an one-way communication channel through the stepping stone (outside connections through it are possible, incoming ones are not).

Finally, a "lifeline" connection is maintained between the new location and the old location *via the stepping stone,* such that the migrated service can access any storage that is attached to the old location. For example, consider a typical web server such as the one shown in Figure 2. In MOVE, we will migrate the front-end web server component, and use the lifeline connection to communicate to the business logic and database components. Although the stepping stone's address may change over time, we can use tunneling to maintain the connection with the new location. We eventually intend to use SCTP for this connection, since this allows us to easily do live-connection migration to a new IP address. Since the secondary link connects to another ISP (or we ensured that it uses a different set of links than the main Internet attachment point), the DDoS attack will not affect the lifeline. To avoid adding up to the lifeline overheads when a server is migrated several times, we collapse the lifeline by instructing the previous (original) stepping stone to connect to the new location.

2.2.2 Clients in MOVE

When the service is not under attack, clients can contact it directly. Once an attack is detected or suspected (*e.g.,* through loss of connectivity), traffic is diverted through MOVE. Clients that want to access the attacked service, contact any overlay node (an updated list of which can be published periodically) and authenticate using one of several techniques. The most straightforward approach, used in the initial version of MOVE, is to use SSL with client certificates that are signed by the entity operating the overlay, and authorize the holders to access either specific or any applications or servers using the overlay.

The drawback of using an authentication protocol is that we must know in advance who the legitimate users are, so they can be provisioned with the appropriate authentication credentials. This is not a problem for applications such as VNC, where we only want the authorized user (owner) to access the service. In countering DoS attacks, however, we are often more interested in whether a particular request (*e.g.,* HTTP connection) originated

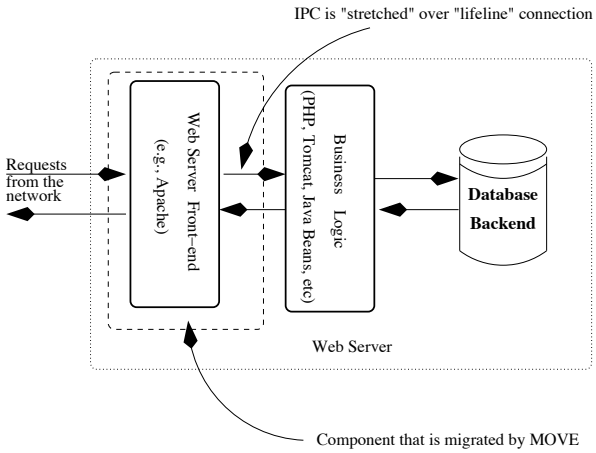

Figure 2. **Typical model of a web server.**

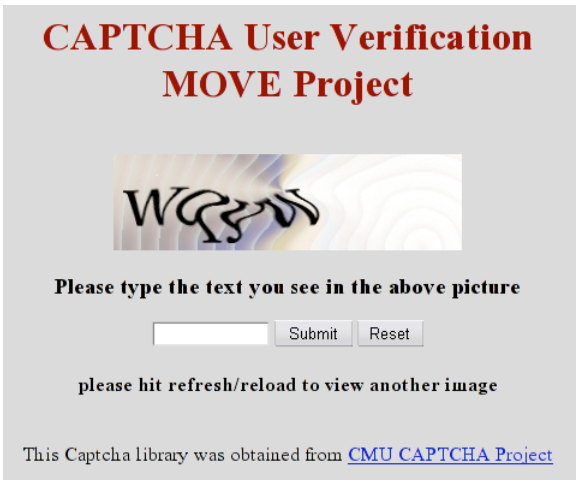

Figure 3. **Web-based Graphic Turing Test using GIMPY. The challenge in this case is "wqyw".**

from a human user or from a "DoS zombie" process, that is under the control of the attacker. Fortunately, it is possible to do so using *Graphic Turing Tests* (GTTs) [54], an example of which is shown in Figure 3. For vision-impaired users, it is possible to use audible tests of a similar nature (*e.g.,* type the word spelled in this audio clip); we do not consider such tests further in this paper. MOVE can support both authenticated and GTT-admitted clients simultaneously, as we describe in Section 3.2.

The particular GTT realization we use is GIMPY, which concatenates an arbitrary sequence of letters to form a word and renders a distorted image of the word as shown in Figure 3. GIMPY relies on the fact that humans can read the words within the distorted image and current automated tools cannot. The human authenticates herself by entering as ASCII text the same sequence of letters as what appears in the image. Although recent advances in visual pattern recognition [31] can defeat GIMPY, there is no solution to date that can recognize complicated images or relation between images like Animal-PIX. Although for demonstration purposes in our prototype, described in Section 3, we use GIMPY, we can easily substitute it with any other instance of a GTT. Once a client has passed the GTT, they are provided with a short-expiration certificate that can be used to access the overlay with TLS. We describe the implementation details in Section 3. Note that it is possible to provision users with longer-lived credentials, or just provide them with long-lived certificates, skipping GTT authentication altogether; the drawback of that approach is that the users must be known and provisioned *a priori*. We believe that with these two authentication methods (certificate-only and GTT), we can cover the majority of usage scenarios. Although we are aware of social-engineering attacks against CAPTCHAs, note that we are not limited to these as the only form of authentication; as we already mentioned, we can use certificate (and even password-based) authentication. We can integrate additional authentication mechanisms in the infrastructure as they are developed.

Authenticated clients then route all their traffic over the overlay, which redirects it to the current location of the protected service. Note that the actual location of a server is no longer implied by the host-name component of the URL, or the IP address this resolves to. An attacker that has no access to the overlay can try to attack a hosting site at random, or the IP address the URL resolves to. In that case, the server and all associated state is migrated to a new location. Alternatively, the attacker can target the overlay itself. However, as has been shown by other work on which we base our approach [21, 1], doing so is more difficult as the overlay size increases.

2.3 Process Migration

Process migration is the ability to transfer a process from one machine to another. It is a useful facility in distributed computing environments, especially as computing devices become more pervasive and Internet access becomes more ubiquitous. Although many approaches have been proposed [27], achieving process migration

functionality has been difficult in practice.

For our system, we use an approach based on [33], by effectively providing a thin virtualization layer, called a *POD,* on top of the operating system that provides a group of processes with a private namespace. The sandboxed process group always sees the same virtualized view of the system, which associates virtual identifiers with operating system resources such as process identifiers and network addresses. This decouples sandboxed processes from dependencies on the host operating system and from other processes in the system. This virtualization is integrated with a checkpoint-restart mechanism that enables the sandboxed processes to be migrated as a unit to another machine. These process groups are independent and self-contained, and can thus be migrated freely without leaving behind any residual state after migration, even when migrating network applications while preserving their network connections. We can therefore allow applications to continue executing after migration even if the machine on which they previously executed is no longer available. To support transparent network-connection migration, client-side support is required, which is straightforward to implement on the overlay nodes (*i.e.,* no changes to client (user) software are necessary). For our prototype implementation, this was not required, since the web model allows for temporary TCP connection failures. For application domains where this is not an option, we can augment the overlay nodes accordingly. We do not explore this option further in this paper.

The process migration system is designed to support migration of unmodified legacy applications while minimizing changes to existing operating systems. This is done by leveraging loadable kernel module functionality in commodity operating systems that allows us to intercept system calls as needed for virtualization and save and restore kernel state as needed for migration. The system's compatibility with existing applications and operating systems makes it simple to deploy and use. The system is implemented as a kernel module, allowing transparent migration among separate machines running independent versions of Linux (with unmodified kernels). Note that these systems do not need to share a single-system image.

In our web server experiments, all the server processes were contained in the same POD. When migrating the VNC environment, a number of different processes were contained inside the same POD: the VNC server itself, X11 terminals, a web browser, and a few other applications. We describe the configuration in more detail in Section 4.

2.4 Example of System Operation

To illustrate the use of our architecture by servers and clients, we describe the steps both sides must undertake to protect their communication channel:

- A server contacts any overlay node and informs it of its location. The connection is protected by SSL/TLS, and the server presents a certificate that proves to the overlay its right to specify a location for a particular hostname/URL. The overlay node confirms the validity of the certificate and informs other nodes in the overlay of the new location of the service. We will discuss in Section 3 the mechanism used in our prototype. The server periodically re-affirms its current location.

- A client that wants to communicate with the service contacts a random overlay node. After authenticating and authorizing the request via the CAPTCHA test, the overlay node securely proxies all traffic from the source to the target. Alternatively, a pre-authorized client that possesses a valid certificate can connect to the overlay without requiring any user interaction. As explained in [48], this step may also involve some type of payment (by the end user or the service owner) to the entity managing the overlay infrastructure. Following the discussion of [1], a number of overlay nodes may be involved in the routing, depending on the precise threat model and performance requirements. For example, to avoid the situation where an attacker can eavesdrop on an overlay node and determine the location of the service, we may want to use a two-hop overlay routing approach; if this is not a concern, we may use one-hop redirection instead. Alternatively, we can use full Chord-like routing [50] as in SOS [21], obscuring traffic patterns.

- When an attack is detected, the server process is suspended and migrated, using the system described in Section 2.3. A random hosting site is selected and, after querying its current status with respect to DoS attacks and other suspicious activity, the server is migrated there. To perform the migration, a "stepping stone" host that resides in the same ISP as the source hosting site is used, to achieve routability from an unpredictable source address (one that cannot be attacked from outside the ISP).

Following the analyses of [21] and [1], the scheme is robust against DoS attacks because there exists no dependency on any individual link, router, overlay node,

or hosting site. If a node within the overlay is attacked, the node simply exits the overlay and clients switch to a new node. No node is more important or sensitive than others. Given "enough" redundancy, an attacker is left with the options of splitting their attack among all possible hosting sites and stepping stones, or trying to guess the currently used location and focus their attack there. How much constitutes "enough" depends on the expected severity of attacks; we intend to quantify this in future work. Intuitively, and given the attacks that have been seen on the Internet so far, we expect 15 to 20 distinct and well connected hosting sites plus a small number of stepping stones for each to be sufficient in making even large attacks infeasible. We intend to extend our preliminary analysis from [21] for this scenario.

In [21], we performed a preliminary analysis using simple networking models to evaluate the likelihood that an attacker is able to prevent communications to a particular target. This likelihood was determined as a function of the aggregate bandwidth obtained by an attacker through the exploitation of compromised systems. The analysis included an examination of the capabilities of static attackers who focus all their attack resources on a fixed set of nodes, as well as attackers who adjust their attacks to "chase after" the repairs that their system implements when it detects an attack. We demonstrated that even attackers that are able to launch massive attacks are very unlikely to prevent successful communication. For instance, attackers capable of launching debilitating attacks against 50% of the nodes in the overlay have roughly one chance in one thousand of stopping a given communication from a client who can access the overlay through a small subset of overlay nodes. The same security analysis applies to MOVE, by reducing it to an instance of SOS: let the hosting site be equivalent to the SOS "secret servlet" nodes; in both SOS and MOVE, an attacker must correctly guess the identity of the location/servlet, both of which lie in the same name space (IP address). Keeping the details of the overlay itself the same between the two approaches, it is easy to see the equivalence of the two from an analysis viewpoint. However, unlike SOS and Mayday, MOVE achieves its properties without any support from the network infrastructure.

3 Implementation

Since our MOVE prototype uses Chord [50] as the underlying overlay network, we first briefly describe Chord and then expand on the implementation of MOVE.

3.1 Chord

Chord can be viewed as a routing service that can be implemented atop the existing IP network fabric, *i.e.,* as a network overlay. Consistent hashing [18] is used to map an arbitrary identifier to a unique destination node that is an active member of the overlay. In Chord, each node is assigned a numerical identifier (ID) via a hash function in the range $[0, 2^m]$ for some predetermined value of m. The nodes in the overlay are ordered by these identifiers. The ordering is cyclic (*i.e.,* wraps around) and can be viewed conceptually as a circle, where the next node in the ordering is the next node along the circle in the clockwise direction.

Each overlay node maintains a table that stores the identities of m other overlay nodes. The i^{th} entry in the table is the node whose identifier x equals or, in relation to all other nodes in the overlay, most immediately follows $x + 2^{i-1} \pmod{2^m}$. When overlay node x receives a packet destined for ID y, it forwards the packet to the overlay node in its table whose ID precedes y by the smallest amount. The Chord algorithm routes packets around the overlay "circle", progressively getting closer to the desired overlay node. $O(m)$ overlay nodes are visited. Typically, the hash functions used to map nodes to identifiers do not attempt to map two geographically close nodes to nearby identifiers. Hence, it is often the case that two nodes with consecutive identifiers are geographically distant from one another within the network. The Chord service is robust to changes in overlay membership, and each node's list is adjusted to account for nodes leaving and joining the overlay such that the above properties continue to hold.

MOVE uses the hostname of the target (*i.e.,* web server) as the identifier to which the hash function is applied. Thus, Chord can direct traffic from any node in the overlay to the node that the identifier is mapped to, by applying the hash function to the target's host name. This node is simply a unique node that will be eventually be reached, after up to $m = \log N$ overlay hops, regardless of the entry point. For any particular service, this node will always know its current location. If this node is for some reason dropped from the overlay, a new node (the one with an address closest to the hash of the service's hostname) will subsume its role, and provide location-resolution services for that target. The new node will learn of the service's current location through the periodic re-confirmation message sent. Thus, location information does not need to be flooded to all nodes of the overlay network, which would make it difficult to support large numbers of services, without compromising reliability and robustness to attack.

3.2 MOVE Implementation

Our prototype system is based on WebSOS [30, 7]. Each overlay node is responsible for resolving the location of the requested service and creating a security communication tunnel with it. To that end, we use Chord to distribute the location information for each site: when a service informs MOVE of its current location, its hostname is hashed and the node thus indicated is informed of the location. In that sense, this node acts in a manner analogous to SOS *beacon* nodes. Similarly, when a MOVE node needs to forward a legitimate user's request to the service, it hashes the service hostname and sends a query to the Chord node whose address is closest to the hash result. Thus, in contrast to [7] and [21], rather than transporting the request and response through the Chord overlay, only routing information travels through it; data connections are proxied directly to the protected service's location. The information is cached and periodically refreshed by consulting the authoritative MOVE node for that target.

When a new request (in the form of a new TCP connection) is received, the MOVE node to which the client is connected (called an *access point*) first checks the local cache database for the current location of the requested service. If the lookup succeeds, the access point opens a new SSL connection to a random overlay node (to borrow from SOS terminology, a "secret servlet"), which allows us to avoid some of the eavesdropping attacks identified in [1]. Thus, a two-way communication channel is established between the client and the service, through the overlay. Authentication of the user by the overlay is accomplished through SSL. Authorized users are issued X.509 [5] certificates signed by the MOVE access point that administered the GTT. These certificates are only valid for a limited time (30 minutes), after which the user must pass another GTT. Furthermore, the certificates are bound to the IP address from which the GTT authentication came, and can only be used with the specific MOVE access point. Thus, an attacker cannot simply authenticate once and redistribute the same certificate to a large number of attack zombies. Each overlay node also communicates with other MOVE nodes over SSL connections. If the lookup fails, the access point queries the resolving node, as described previously.

When a request is issued by the client for a specific service, it is tunneled through a series of SSL-encrypted links to the target, allowing the entire transmission between the requester and target to be encrypted. The SSL connections between MOVE nodes are dynamically established, as new requests are routed. To accomplish this, we wrote a port forwarder that runs on the user's system, accepts plain-text proxy requests locally, and forwards them using SSL to the access point node. This is implemented as a Java applet that runs inside the browser that a user uses to authenticate himself. This Java applet is responsible for encrypting and forward to the access point requests from any service initiated by the client and can be configured to accept a proxy. This last requirement can be removed if we use interception of the socket communication at the operating system level.

Thus, to use MOVE, an authorized user simply has to access any access point, successfully respond to the Graphic Turing Test challenge, download the applet, and set the service proxy settings to the localhost, as shown in Figure 4. Java applets typically cannot communicate with any host other than the one they were downloaded from, but this is not a problem in our case. If the user replies successfully, the web server connects to a DBMS system (local or remote) and associates an RSA key and a certificate with the host. The key/certificate are unique per IP and have an expiration time that can be configured by the system administrator. The user is prompted to download a signed applet that runs locally using one browser window and contacts the Web Server via a temporary HTTPS connection to fetch the X.509 certificate.

The applet then starts listening for service connections on a local ports (*e.g.,* 8080) and establishes an SSL-tunnel connection with the server running on the access point (or elsewhere, since the signed applet has the ability to connect to any server by changing the Java Policy files on the users' machine). The proxy server matches the X.509 certificate and the IP from client to the private key obtained from the DBMS system and allows the connection to be proxied. The only imposition on the user is that he/she must change the Proxy settings of the local browser to point to the socket that listens for the applets. The same configuration is used when using VNC, with the proxy forwarding VNC rather than HTTP traffic.

The access point caches the server's location for use in future requests. That information is timed out after a period of time to allow changes to propagate correctly. The same basic mechanism is used by services to announce their presence to (and periodically update the information stored by) their corresponding resolving nodes.

In a DoS attack, the target server migrates to a randomly-chosen location and, once there, notifies the overlay of its new location. The migration is done using the process migration mechanism we described in Section 2.3. System migration is largely implemented as a

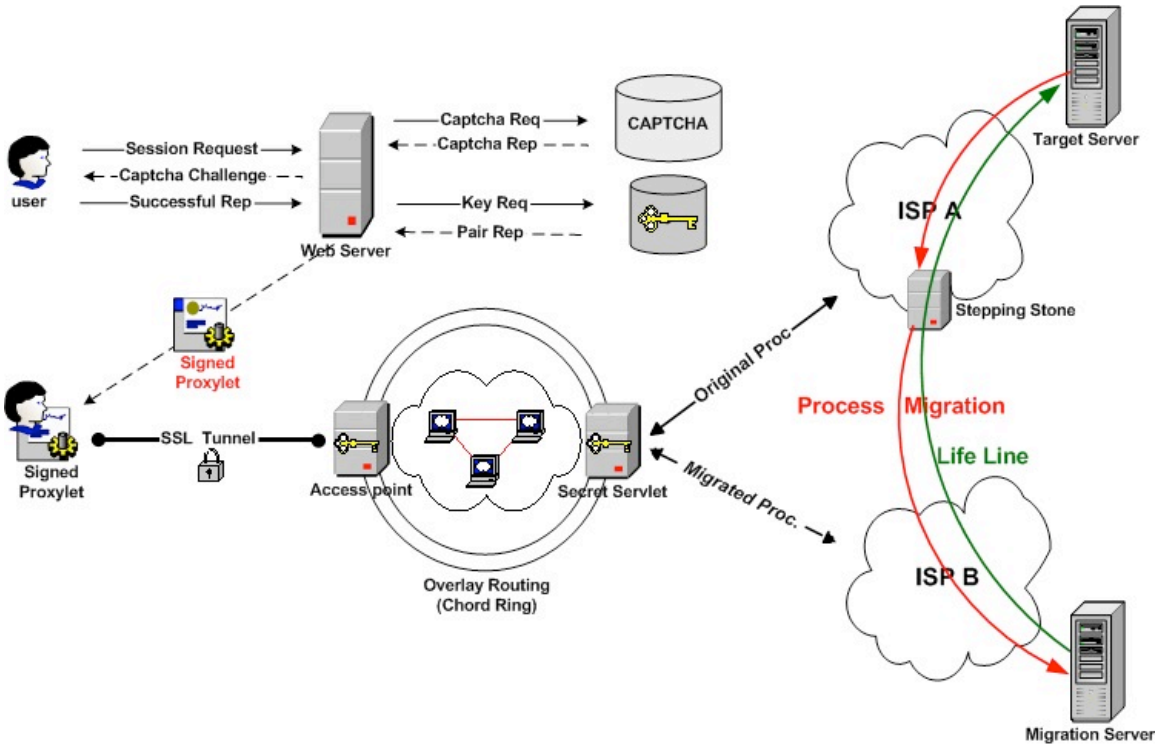

Figure 4. **MOVE client session initiation diagram.**

loadable Linux kernel module. In our system, this mechanism was used to load and checkpoint the Apache and VNC servers respectively, creating the necessary images of the running processes. These process images, which included the current process state, were then transfered to the remote server and restarted.

4 Experimental Results

To evaluate MOVE's impact on performance and availability, we deployed the prototype implementation on a number of PlanetLab nodes [36], distributed across the Internet. We measured (1) the impact of using MOVE to the client's end-to-end latency, (2) the delay in making a server available again at a new site once a DoS attack has been launched, and (3) the impact of the lifeline connection to the server's performance. In our experiments, we used the following entities (see Figure 4):

- A node acting as the client, using an off-the-shelf web browser.
- An http *target* server (Apache) and a VNC server, in two separate experiments.
- A set of PlanetLab nodes participating in the overlay network and providing the necessary traffic redirection facilities.
- A *migration* server, to which the server is migrated from its original location when attacked.

In our experiments, the legitimate client was located inside Columbia University's network and the traffic was redirected from various nodes inside the PlanetLab network toward the web and VNC servers, which were initially located on the local (Columbia) network. We deployed the MOVE implementation on 76 PlanetLab nodes, which formed a Chord ring, per our discussion in Section 3.

To determine the end-to-end latency experienced by a client using our system, we used the MOVE overlay to contact various web servers and download the initial page. The results are shown in Table 1. The difference in performance over previous work that used the same benchmark [7] is due to the fact that traffic in MOVE only traverses two overlay nodes, as opposed to the full Chord overlay, meaning fewer redirections between overlay nodes. As shown in [28], the increase in latency is typically dominated by the end-to-end communication overheads. An additional delay cost is the SSL-

Table 1. **Latency (in seconds) when contacting a number of web servers directly and while using MOVE; in all cases, we download the initial web page. The last column shows the factor increase in latency The testing was performed on a 76 node subset of the PlanetLab testbed using the Chord overlay. The numbers are averaged over 25 requests.**

Server	Direct	MOVE	Ratio
Yahoo!	1.32	3.67	2.78
VeriSign	3.41	6.77	1.98
BBC News	1.11	3.17	2.85
Microsoft	1.51	4.01	2.65
Slashdot	3.66	7.21	1.96
FreeBSD	1.49	3.81	2.55

Table 2. **Delay in re-establishing availability after disruption (due to DDoS) for an httpd server, migrated from the initial site to a co-located server (using NFS/UDP and SHFS/TCP), and to a remote site (using SHFS). The size of the server state was 9.8MB on average. We also include the round-trip latency between the target and migration servers in all cases.**

Migration Server	RTT Latency	Migration Time
Co-located (NFS)	1.02ms	0.761s
Co-located (SHFS)	1.02ms	1.162s
U. Penn (SHFS)	10.6ms	6.632s

processing overhead from the generation of the SSL tunnel and the encryption of the data from client to the overlay and inside the overlay; use of cryptographic accelerators may further improve performance in that area [23].

To measure the delay in re-establishing the server's availability, which is the time during which a client using MOVE experiences service disruption, we used an Apache httpd server running under the default configuration. The server, initially located inside our local network, was moved to the *migration* server. The state of the server processes amounted to 9.8 MBytes on average. We measured the migration time in the following cases: when the migration server was located inside our local network, and when the migration server was located at the University of Pennsylvania, approximately 11 hops or 10ms *ping* time away (over Internet2). In both scenarios, the state files were transfered using the (Secure) SHell FileSystem (SHFS) [46], which operates over the popular SSH/SCP protocol suite. We chose SHFS because of its ease of installation and use; other filesystems, such as LBFS [32] or CODA [41] can also be used instead SHFS. For the local-migration case, we used NFS (over UDP) as a second way of transferring state. We show the results in Table 2. The availability delay varies between 1 second to 6.6 seconds, as round-trip times increase from 1ms (for the local-migration case) to $10ms$. We believe that this level of disruption is acceptable in the presence of a DoS attack, when the alternative is total loss of service. One may observe that an attacker may be considered successful in some scenarios if they can cause service downtime of 7 seconds even few minutes, if only because the end users will be annoyed. One possibility we plan to examine in future work is the use of "hot spares" that are kept synchronized with the live service and to which we can divert traffic at sort notice through MOVE.

To better analyze the contribution of the lifeline connection to the overall latency for large interactive, non-caching applications like thin clients, we constructed the following experiment: initially, a VNC [39] server is located on the target server. The client connects via MOVE and creates a VNC session consisting of Mozilla browser, Gaim Instant Messenger, Kword, a PS/PDF viewer and two terminals connected to the target server. Upon detection of the attack, we checkpointed the POD containing the VNC session and server, transfered the state to the migration server and restarted it. The state transfered to the migrate server amounted to 55.9 MBytes. For the migration, we used a co-located server, *Site 1,* and two remote servers, *Site 2* and *Site 3,* with different network proximity to the target server. Figure 5 shows the average availability delay for this scenario, which is dominated by the transfer of the POD state; the relatively small checkpoint and restart overheads remain constant for all experiments.

In addition, the underlying file-access mechanism seems to play a significant role, especially when we establish a low-bandwidth link between the target and the migration server. Accessing the target's database or file system via the lifeline can also increase the end-to-end latency. For servers that are either co-located or connected through a low-latency link (<5ms), we can use NFS/UDP since it is fast and reliable. However, if the established connection is of high latency (and low bandwidth), NFS/UDP becomes unresponsive. To measure the performance of our system when using a high-latency connection, we instead used SHFS. Figures 6 and 7 show the average round trip time (RTT) and effective throughput respectively for all the migration sites. We can see that as we increase the distance between

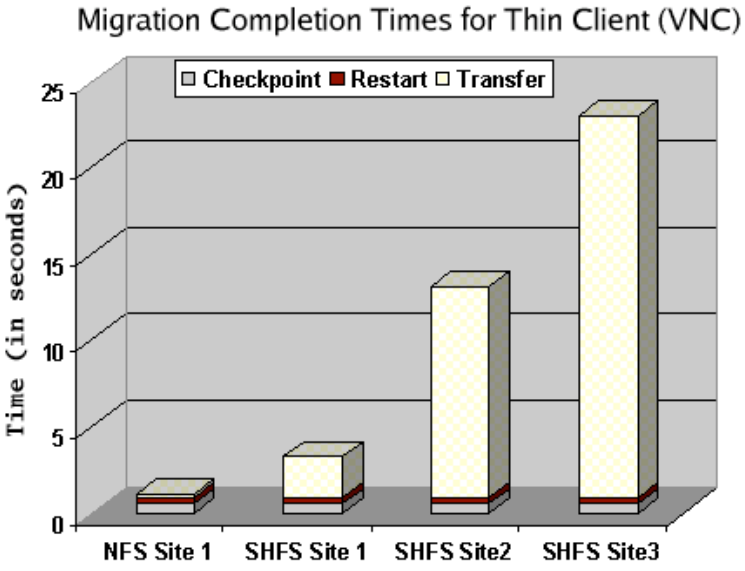

Figure 5. **Average migration time (in seconds) for a VNC server migrates to a co-located server (*Site 1*) and to remote servers (*Site 2* and *Site 3*) using NFS/UDP and SHFS/TCP using a tunnel through the lifeline. We observe that the total time is dominated by the transfer time. Prior to the migration the VNC server was running Mozilla browser, Gaim, Kword, PS/PDF viewer and two terminal applications.**

the target and the migration server, we have a proportional decrease in the average data throughput. Figure 7 shows that when we use NFS to connect the target and migration servers for *Site 1,* we achieve better throughput compared to using SHFS.

Choosing a file-access method to connect through the lifeline depends both on the network proximity of the migrating site and the maximum network latency allowed by the application we migrate. For thin-client applications like VNC, all the applications are already loaded prior to migration and the user communicates with the target server only to read or write data from within an application. Thus, the lifeline utilization is relatively low, allowing for relatively smooth operation under conditions of high network latency. For applications that access the filesystem frequently, such as HTTP servers, it is necessary to use a network file system with good caching characteristics. For database applications, we need to ensure that the lifeline can sustain the load from the migrated middleware server to the back-end database server.

5 Related Work

As a result of its increased popularity and usefulness, the Internet contains both interesting targets and enough malicious and ignorant users that DoS attacks are simply not going to disappear on their own; indeed, although the press has stopped reporting such incidents, recent studies have shown a surprisingly high number of DoS attacks occurring around the clock throughout the Internet [29]. Worse, the Internet is increasingly being used for time-critical applications. A further compounding factor is the susceptibility of the basic protocols (*i.e.,* IP and TCP) to denial of service attacks [45, 14, 42].

The need to protect against or mitigate the effects of DoS attacks has been recognized by both the commercial and research world. Some work has been done toward achieving these goals, *e.g.,* [16, 8, 44, 43, 13, 47, 51, 26]. These mechanisms focus on detecting the source of DoS attacks in progress and then countering them, typically by "pushing" some filtering rules on routers as far away from the target of the attack (and close to the sources) as possible. The motivation behind such approaches has been twofold: first, it is conceptually simple to introduce a protocol that will be used by a relatively small subset of the nodes on the Internet (*i.e.,* ISP routers), as opposed to requiring the introduction of new protocols that must be deployed and used by end-systems. Second, these mechanisms are fairly transparent to protocols, applications, and legitimate users. Unfortunately, these reactive approaches by themselves

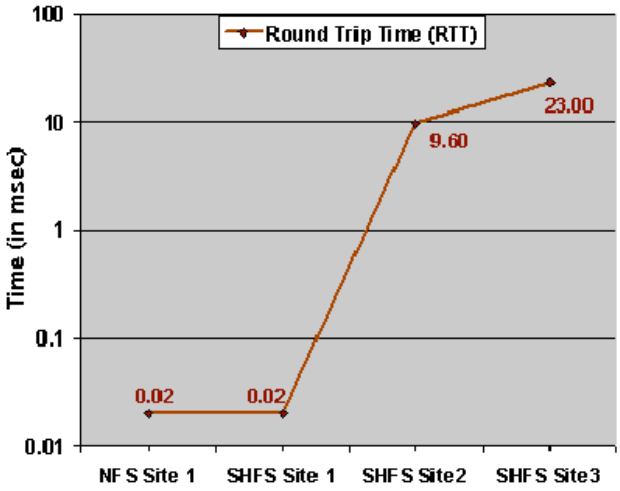

Figure 6. **Round Trip Time (RTT) between the target server and the migration sites when using the lifeline tunnel. Note that the Y axis uses logarithmic scale.**

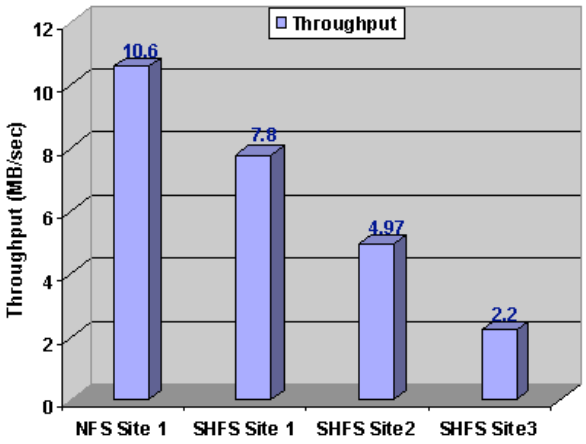

Figure 7. **Average data throughput in MB/sec when accessing target server files from the migration sites using the lifeline tunnel. Notice that as we increase the network distance, we have a proportional decrease in the average data throughput**

are not adequate, since large-scale coordination across multiple administrative domains is not always practical.

The D-WARD system [38] monitors outgoing traffic from a given source network and attempts to identify attack traffic by comparing against models of reasonable congestion control behavior. The amount of throttling on suspicious traffic is proportional to its deviation from the expected behavior, as specified by the model. In COSSACK [34], participating agents at edge networks exchange information about observed traffic and form multicast cliques to coordinate attack suppression. An interesting approach is that of [17], which proposes an IP hop-count-based filter to weed out spoofed packets. The rationale is that most such packets will not have a hop-count (TTL) field consistent with the IP addresses being spoofed. In [15], the authors use a combination of techniques that examine packet contents, transient ramp-up behavior and spectral analysis to determine whether an attack is single- or multi-sourced, which would help focus the efforts of a hypothetical anti-DoS mechanism.

A variant of the packet marking approaches creates probabilistically unique path-marks on packets without requiring router coordination; end-hosts or firewalls can then easily filter out packets belonging to a path that exhibits anomalous behavior [57]. Although this approach avoids many of the limitations of the pure marking schemes, it requires that core routers "touch" packets (rather than simply switch them), and assumes that the limited resource is the target's CPU cycles, rather than the available bandwidth (*i.e.*, preventing the DoS attack is "simply" a matter of quickly determining which packets the server should ignore). In our work, we assume that the scarce resource is bandwidth. Collins and Reiter [6] present an empirical analysis of several different anti-DoS techniques (including Pi [57] and Hop-Count Filtering [17]) that use filters near the target of an attack, using traces of real DDoS attacks to simulate the impact of the filters on the attack traffic.

[35] describes filtering out source-spoofed packets inside the Internet core, and discusses the effectiveness of this approach. The authors suggest piggy-backing on BGP to propagate the necessary information. DDoS attacks using real IP addresses are not affected by this scheme.

[19] proposes using Class-Based Queuing on a web load-balancer to identify misbehaving IP addresses and place them in lower priority queues. However, many of the DDoS attacks simply cause congestion to the web server's access link.

Another approach to mitigating DoS attacks against information carriers is to massively replicate the content being secured around the entire network, *e.g.*, [49]. To prevent access to the replicated information, an attacker must attack all replication points throughout the

entire network — a task that is considerably more difficult than attacking a small number of, often co-located, servers. Replication is a promising means to preserve information that is relatively static, such as news articles. However, there are several reasons why replication is not always an ideal solution. For instance, the information may require frequent updates complicating large-scale coherency (especially during DoS attacks), or may be dynamic by its very nature (*e.g.,* a live web-cast). Another concern is the security of the stored information: engineering a highly-replicated solution without leaks of information is a challenging endeavor.

An extension of the ideas of SOS [21, 7, 30] appears in [1]. There, the two main facets of the SOS architecture, filtering and overlay routing, are explored separately, and several alternative mechanisms are considered. It is observed that in some cases, the various security properties offered by SOS can still be maintained using mechanisms that are simpler and more predictable. However, some second-order properties, such as the ability to rapidly reconfigure the architecture in anticipation of or in reaction to a breach of the filtering identity (*e.g.,* identifying the secret servlet) are compromised. In most other respects, the two approaches are very similar.

A system similar to MOVE is proposed in [24], where servers are hidden among a large number of honepots. A continuously changing subset of the servers is active and providing service, switching to honeypot mode when an attack is detected. An overlay is responsible for routing legitimate clients to the currently active servers, and the system is evaluated via a set of simulations.

Gligor [12] proposes the use of a server that can produce tickets at line speeds. Clients must obtain a ticket from this server before they are allowed to access a protected service. The approach is geared towards application-level DoS protection, with some other mechanism, such as SOS or Pushback, used to address network-level DoS attacks. Anderson *et. al* [2] subsequently proposed a similar system for use at the network layer of an Internet-like architecture designed with a clean slate, assuming a distributed token server architecture and rate-limiting/filtering traffic on routers based on these tokens. Another similar idea appears in [25].

The NetBouncer project [53] considers the use of client-legitimacy tests for filtering attack traffic. Such tests include packet-validity tests (*e.g.,* source address validation), flow-behavior analysis, and application-specific tests, including Graphic Turing Tests. However, since their solution is end-point based, it is susceptible to large link-congestion attacks. [58] is the first system to create stateless flow filtering by having each router add "capabilities" to packets that traverse them; the receiver of these packets is then responsible for sending these capabilities to its peers, which will allow them to send traffic at higher rates (privileged traffic). Unprivileged traffic is limited to a fraction of the available bandwidth; thus, although a DoS attack can prevent new connections from being established (by overloading the control channel used to communicate these capabilities), existing connections will be unharmed.

[3] examines several different DDoS mitigation technologies and their interactions. Among their conclusions, they mention that requiring the clients to do some work, *e.g.,* [9], can be an effective countermeasure, provided the attacker does not have too many resources compared to the defender. Wang and Reiter [55] introduced the idea of a puzzle auction as a way to ease some of the practical deployment difficulties, *e.g.,* selecting the appropriate hardness for the puzzles. Their intuition is to let clients bid for the resources by tuning the difficulty of the puzzles they solve. When the server is attacked, legitimate clients gradually increase their bids (puzzle difficulty), eventually bringing the cost outside the adversary's capabilities. The authors envision combining their scheme with some anti-DoS mechanism that counteracts volume-based attacks [16, 57, 21].

6 Conclusions

We described MOVE, an architecture for protecting specific classes of software services, such as a web server, from network-based denial of service (DoS) attacks, without requiring any additional functionality to be placed inside the network. We believe our work to be the first to demonstrate that it is possible to counter network DoS attacks without requiring support from the infrastructure itself.

MOVE combines a network overlay with a lightweight process-migration mechanism. The overlay nodes accept connections from legitimate users, and route their traffic to the current location of the service. Our definition of legitimate is flexible, and can vary from users authenticated with cryptographic credentials to Graphic Turing Tests, which simply distinguish between humans and automated zombies. If a service is attacked, it is migrated to a new, randomly selected location. An attacker is left with the option of splitting the attack traffic to all potential targets, or trying to guess the current active location.

We use our prototype implementation on the PlanetLab testbed, which allows us to distribute the MOVE nodes across the Internet. In a series of experiments, we show that the end-to-end latency imposed by MOVE can

be as low as a factor of 2 to 3 higher than direct communication between client and server, which we believe is an acceptable cost when dealing with DoS attacks. The service disruption to the end user, that is the time to migrate an attacked service, ranges from 10 to 25 seconds, depending on the location of the migration server. Our plans for future work include eliminating (or minimizing) the need for stepping stones, and analyzing the security of the system against different adversarial models [56].

Acknowledgements

We would like to thank the anonymous reviewers for their comments and suggestions that helped improve this paper.

This work is supported in part by DARPA contract No. F30602-02-2-0125 (FTN program) and by the National Science Foundation under grants ANI-0117738 and ITR CNS-0426623, and CAREER Award ANI-0133829, with additional support from Cisco and Intel Corporation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] D. G. Andersen. Mayday: Distributed Filtering for Internet Services. In *4th USENIX Symposium on Internet Technologies and Systems USITS*, March 2003.

[2] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet Denial-of-Service with Capabilities. In *Proceedings of the* 2^{nd} *Workshop on Hot Topics in Networks (HotNets-II)*, November 2003.

[3] W. J. Blackert, D. M. Gregg, A. K. Castner, E. M. Kyle, R. L. Hom, and R. M. Jokerst. Analyzing Interaction Between Distributed Denial of Service Attacks and Mitigation Technologies. In *Proceedings of DISCEX III*, pages 26–36, April 2003.

[4] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specifi cation. Internet RFC 2208, 1997.

[5] CCITT. *X.509: The Directory Authentication Framework*. International Telecommunications Union, Geneva, 1989.

[6] M. Collins and M. Reiter. An empirical analysis of target-resident DoS fi lters. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.

[7] D. L. Cook, W. G. Morein, A. D. Keromytis, V. Misra, and D. Rubenstein. WebSOS: Protecting Web Servers From DDoS Attacks. In *Proceedings of the* 11^{th} *IEEE International Conference on Networks (ICON)*, pages 455–460, September 2003.

[8] D. Dean, M. Franklin, and A. Stubblefi eld. An Algebraic Approach to IP Traceback. In *Proceedings of the Symposium on Network and Distributed System Security (SNDSS)*, pages 3–12, February 2001.

[9] D. Dean and A. Stubblefi eld. Using Client Puzzles To Protect TLS. In *Proceedings of the* 10^{th} *USENIX Security Symposium*, August 2001.

[10] T. Dierks and C. Allen. The TLS protocol version 1.0. RFC 2246, January 1999.

[11] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the* 13^{th} *USENIX Security Symposium*, pages 303–319, August 2004.

[12] V. D. Gligor. Guaranteeing Access in Spite of Distributed Service-Flooding Attacks. In *Proceedings of the Security Protocols Workshop*, April 2003.

[13] M. T. Goodrich. Effi cient Packet Marking for Large-Scale IP Traceback. In *Proceedings of the* 9^{th} *ACM Conference on Computer and Communications Security (CCS)*, pages 117–126, November 2002.

[14] L. Heberlein and M. Bishop. Attack Class: Address Spoofi ng. In *Proceedings of the* 19^{th} *National Information Systems Security Conference*, pages 371–377, October 1996.

[15] A. Hussain, J. Heidemann, and C. Papadopoulos. A Framework for Classifying Denial of Service Attacks. In *Proceedings of ACM SIGCOMM*, pages 99–110, August 2003.

[16] J. Ioannidis and S. M. Bellovin. Implementing Pushback: Router-Based Defense Against DDoS Attacks. In *Proceedings of the ISOC Symposium on Network and Distributed System Security (SNDSS)*, February 2002.

[17] C. Jin, H. Wang, and K. G. Shin. Hop-Count Filtering: An Effective Defense Against Spoofed DoS Traffi c. In *Proceedings of the* 10^{th} *ACM International Conference on Computer and Communications Security (CCS)*, pages 30–41, October 2003.

[18] D. Karger, E. Lehman, F. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 654–663, May 1997.

[19] F. Kargl, J. Maier, and M. Weber. Protecting Web Servers From Distributed Denial of Service Attacks. In *Proceedings of the W3C World Wide Web Conference (WWW)*, pages 514–524, 2001.

[20] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401, November 1998.

[21] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *Proceedings of ACM SIGCOMM*, pages 61–72, August 2002.

[22] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: An Architecture For Mitigating DDoS Attacks. *IEEE Journal on Selected Areas of Communications (JSAC)*, 33(3):413–426, January 2004.

[23] A. D. Keromytis, J. L. Wright, and T. de Raadt. The Design of the OpenBSD Cryptographic Framework. In *Proceedings of the USENIX Annual Technical Conference*, pages 181–196, June 2003.

[24] S. M. Khattab, C. Sangpachatanaruk, D. Moss, R. Melhem, and T. Znati. Roaming Honeypots for Mitigating Service-Level Denial-of-Service Attacks. In *Proceedings of the 24^{th} International Conference on Distributed Computing Systems (ICDCS)*, pages 238–337, March 2004.

[25] K. Lakshminarayanan, D. Adkins, A. Perrig, and I. Stoica. Taming IP Packet Flooding Attacks. In *Proceedings of the 2^{nd} Workshop on Hot Topics in Networks (HotNets-II)*, November 2003.

[26] J. Li, M. Sung, J. Xu, and L. Li. Large-Scale IP Traceback in High-Speed Internet: Practical Techniques and Theoretical Foundation. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.

[27] D. Milojicic, F. Douglis, and R. Wheeler. *Mobility: Processes, Computers, and Agents*. Addison Wesley Longman, February 1999.

[28] S. Miltchev, S. Ioannidis, and A. D. Keromytis. A Study of the Relative Costs of Network Security Protocols. In *Proceedings of the USENIX Annual Technical Conference, Freenix Track)*, pages 41–48, June 2002.

[29] D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial-of-Service Activity. In *Proceedings of the 10^{th} USENIX Security Symposium*, pages 9–22, August 2001.

[30] W. G. Morein, A. Stavrou, D. L. Cook, A. D. Keromytis, V. Misra, and D. Rubenstein. Using Graphic Turing Tests to Counter Automated DDoS Attacks Against Web Servers. In *Proceedings of the 10^{th} ACM International Conference on Computer and Communications Security (CCS)*, pages 8–19, October 2003.

[31] G. Mori and J. Malik. Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA. In *Computer Vision and Pattern Recognition (CVPR*.

[32] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network fi le system. In *Proceedings of the 18^{th} ACM Symposium on Operating Systems Principles (SOSP)*, pages 174–187, Chateau Lake Louise, Banff, Canada, October 2001.

[33] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The Design and Implementation of Zap: A System for Migrating Computing Environments. In *Proceedings of the 5^{th} Symposium on Operating Systems Design and Implementation (OSDI)*, pages 361–376, December 2002.

[34] C. Papadopoulos, R. Lindell, J. Mehringer, A. Hussain, and R. Govindan. COSSACK: Coordinated Suppression of Simultaneous Attacks. In *Proceedings of DISCEX III*, pages 2–13, April 2003.

[35] K. Park and H. Lee. On the Effectiveness of Route-based PAcket Filtering for Distributed DoS Attack Prevention in Power-law Internets. In *Proceedings of ACM SIGCOMM*, pages 15–26, August 2001.

[36] L. Peterson, D. Culler, T. Anderson, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proceedings of the 1^{st} Workshop on Hot Topics in Networks (HotNets-I)*, October 2002.

[37] M. Reed, P. Syverson, and D. Goldschlag. Anonymous Connections and Onion Routing. *IEEE Journal on Selected Areas in Communications (JSAC)*, 16(4):482–494, May 1998.

[38] P. Reiher, J. Mirkovic, and G. Prier. Attacking DDoS at the source. In *Proceedings of the 10^{th} IEEE International Conference on Network Protocols*, November 2002.

[39] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual Network Computing. In *Proceedings of IEEE Internet Computing*, volume 2, January/February 1998.

[40] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[41] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, and D. Streere. Coda: A Highly Available File System for Distributed Workstation Environments. *IEEE Transactions on Computers*, 39(4), April 1990.

[42] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP Congestion Control with a Misbehaving Receiver. *ACM Computer Communications Review*, 29(5):71–78, October 1999.

[43] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *Proceedings of ACM SIGCOMM*, pages 295–306, August 2000.

[44] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Network Support for IP Traceback. *ACM/IEEE Transactions on Networking*, 9(3):226–237, June 2001.

[45] C. Schuba, I. Krsul, M. Kuhn, E. Spafford, A. Sundaram, and D. Zamboni. Analysis of a Denial of Service Attack on TCP. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 208–223, May 1997.

[46] SHFS Development Team. SHell File System, SHFS. `http://shfs.sourceforge.net/`.

[47] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, S. Kent, and W. Strayer. Hash-Based IP Traceback. In *Proceedings of ACM SIGCOMM*, August 2001.

[48] A. Stavrou, J. Ioannidis, A. D. Keromytis, V. Misra, and D. Rubenstein. A Pay-per-Use DoS Protection Mechanism For The Web. In *Proceedings of the Applied Cryptography and Network Security (ACNS) Conference*, pages 120–134, June 2004.

[49] A. Stavrou, D. Rubenstein, and S. Sahu. A Lightweight, Robust P2P System to Handle Flash Crowds. *IEEE Journal on Selected Areas in Communications (JSAC)*, 22(1):6–17, January 2004.

[50] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Application. In *Proceedings of ACM SIGCOMM*, August 2001.

[51] R. Stone. CenterTrack: An IP Overlay Network for Tracking DoS Floods. In *Proceedings of the USENIX Security Symposium*, August 2000.

[52] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine*, pages 80–86, January 1997.

[53] R. Thomas, B. Mark, T. Johnson, and J. Croall. NetBouncer: Client-legitimacy-based High-performance DDoS Filtering. In *Proceedings of DISCEX III*, pages 14–25, April 2003.

[54] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. CAPTCHA: Using Hard AI Problems For Security. In *Proceedings of EUROCRYPT*, 2003.

[55] X. Wang and M. K. Reiter. Defending Against Denial-of-Service Attacks with Puzzle Auctions (Extended Abstract). In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2003.

[56] D. Xuan, S. Chellappan, and X. Wang. Analyzing the Secure Overlay Services Architecture under Intelligent DDoS Attacks. In *Proceedings of the* 24^{th} *International Conference on Distributed Computing Systems (ICDCS)*, pages 408–417, March 2004.

[57] A. Yaar, A. Perrig, and D. Song. Pi: A Path Identifi cation Mechanism to Defend against DDoS Attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2003.

[58] A. Yaar, A. Perrig, and D. Song. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.