Using Graphic Turing Tests To Counter Automated DDoS Attacks Against Web Servers*

William G. Morein* Angelos Stavrou † Debra L. Cook*
Angelos D. Keromytis* Vishal Misra* Dan Rubenstein†

*Department of Computer Science †Department of Electrical Engineering
Columbia University
New York, NY
{wgm2001,angel,dcook,angelos,misra,danr}@cs.columbia.edu

ABSTRACT

We present WebSOS, a novel overlay-based architecture that provides guaranteed access to a web server that is targeted by a denial of service (DoS) attack. Our approach exploits two key characteristics of the web environment: its design around a human-centric interface, and the extensibility inherent in many browsers through downloadable "applets." We guarantee access to a web server for a large number of *previously unknown* users, without requiring pre-existing trust relationships between users and the system.

Our prototype requires *no modifications* to either servers or browsers, and makes use of graphical Turing tests, web proxies, and client authentication using the SSL/TLS protocol, all readily supported by modern browsers. We use the WebSOS prototype to conduct a performance evaluation over the Internet using PlanetLab, a testbed for experimentation with network overlays. We determine the end-to-end latency using both a Chord-based approach and our *shortcut* extension. Our evaluation shows the latency increase by a factor of 7 and 2 respectively, confirming our simulation results.

Categories and Subject Descriptors

C.2.0 [**Security and Protection**]: Denial of Service; C.2.1 [**Network Topology**]: Overlay Networks

General Terms

Security, Reliability.

*This work is supported in part by DARPA contract No. F30602-02-2-0125 (FTN program) and by the National Science Foundation under grant No. ANI-0117738 and CAREER Award No. 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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CCS'03, October 27–30, 2003, Washington, DC, USA.
Copyright 2003 ACM 1-58113-738-9/03/0010 ...$5.00.

Keywords

Graphic Turing Tests, Web Proxies, Java.

1. INTRODUCTION

The Web is increasingly being used for different kinds of services and interactions with, and between humans. Beyond displaying static content such as home pages or academic papers, the web is actively used for such diverse tasks as e-mail, banking, consumer purchasing, marketing, stock-quote dissemination and trading, and real-time communication. The wide availability of high-quality browsers and servers, as well as programmers' and users' familiarity with the tools and concepts behind web browsing ensure that ongoing creation of additional services.

Such an environment provides a rich set of targets for motivated attackers. This has been demonstrated by the large number of vulnerabilities and exploits against web servers, browsers, and applications. Traditional security considerations revolve around protecting the network connection's confidentiality and integrity, protecting the server from break-in, and protecting the client's private information from unintended disclosure. To that end, several protocols and mechanisms have been developed, addressing these issues individually [10, 27, 30, 14, 38, 29, 13, 8]. However, one area that has long been neglected is that of service availability in the presence of denial of service (DoS) attacks, and their distributed variants (DDoS).

Previous approaches that address the general network DoS problem [17, 9, 32] are *reactive*: they monitor traffic at a target location, waiting for an attack to occur. Once the attack is identified, typically via analysis of traffic patterns and packet headers, filters may be established in an attempt to block the offenders. The two main problems with this approach are the accuracy with which legitimate traffic can be distinguished from the DoS traffic, and the robustness of the mechanism for establishing filters deep enough in the network so that the effects of the attack are minimized.

We introduce *WebSOS,* an adaptation of the *Secure Overlay Services (SOS)* architecture [22]. Our intent is to prevent congestion-based DDoS attacks from denying *any* user's access to web servers targeted by those attacks. The novel aspects of WebSOS are *(a)* its use of *graphic Turing tests* in *lieu* of strong client authentication (as was proposed in SOS) to distinguish between human users and automated attack *zombies*, and (b) its transparency to browsers and servers, by taking advantage of browser extensibility.

In WebSOS, the portion of the network immediately surrounding attack targets (*i.e.,* the web servers) to be protected is protected

by high-performance routers that aggressively filter and block all incoming connections from hosts that are not approved, as shown in Figure 1. These routers are "deep" enough in the network (typically in an ISP's POP, as we discuss in Section 3) that the attack traffic does not adversely impact innocuous traffic. The identities of the small set of nodes that are approved at any particular time is kept secret so that attackers cannot try to impersonate them to pass through the filter. These nodes are picked from a set of nodes that are distributed throughout the wide area network. This superset forms a *secure overlay*: any transmissions that wish to traverse the overlay must first be validated at any of the entry points of the overlay using a Graphic Turing test to distinguish humans from attack scripts. Once inside the overlay, the traffic is tunneled securely to one of the approved (and secret from attackers) locations that can then forward the validated traffic through the filtering routers to the target. Thus, there are two main principles behind our design. The first principle is the elimination of communication pinch-points, which constitute attractive DoS targets, via a combination of filtering and overlay routing to obscure the identities of the sites whose traffic is permitted to pass through the filter. The second is the ability to recover from random or induced failures within the forwarding infrastructure or within the secure overlay nodes.

WebSOS is the first instantiation of the SOS architecture. We use this instantiation to evaluate the performance of the underlying overlay routing mechanism both in a local area scenario and over the Internet using the PlanetLab testbed [26]. The results show that the average increase in end-to-end latency is a factor of **2** beyond what is achieved using the standard web infrastructure. We believe this modest increase is an acceptable alternative to providing no service. Such a service can be used on an as-needed basis, and hence need not impact performance when no attack is in progress. These results validate our simulation analyses, where we used real ISP topologies to determine the added average latency imposed by the WebSOS mechanism.

1.1 WebSOS Architectural Scope

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[1]. 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. *It is such attacks that WebSOS was designed to address. Solving the much harder general denial-of-service problem where attackers could potentially have enough resources to physically partition a network is not addressed in this paper.*

We assume that attackers are smart enough to exploit features of the architecture that are made publicly available, such as the set of nodes that form the overlay. However, 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. We leave it as future work to explore how WebSOS can be used to protect against attacks by such highly specialized and sophisticated attackers.

[1]In one instance of a persistent DoS attack, a British ISP was forced out of business because it could not provide service to its customers.

1.2 Paper Organization

The remainder of this paper is organized as follows. Section 2 gives an overview of Secure Overlay Services (SOS) and graphic Turing tests, and discusses the specifics of the WebSOS architecture. In Section 3 we present our simulation results, using real ISP topologies. Section 4 presents details of our prototype implementation, while Section 5 contains our performance evaluation. Section 6 discusses other work in DoS detection, prevention, and mitigation. Finally, Section 7 concludes the paper.

2. THE WEBSOS ARCHITECTURE

Because our approach is based on the Secure Overlay Services (SOS) [22] architecture, we first highlight its important aspects. We also briefly describe Graphic Turing tests, which implement human-to-overlay authentication. We close this section with a description of WebSOS.

2.1 Overview of SOS

Fundamentally, the goal of the SOS infrastructure is to distinguish between authorized and unauthorized traffic. The former is allowed to reach the destination, while the latter is dropped or is rate-limited. Thus, at a very basic level, SOS requires the functionality of a firewall "deep" enough in the network that the access link to the target is not congested. This imaginary firewall performs access control by using protocols such as IPsec [21]. This generally pre-supposes the presence of authentication credentials (*e.g.,* X.509 [5] certificates) that a user can use to gain access to the overlay. *We consider this one of the the largest drawbacks to SOS, as it precludes casual access to a web server by anonymous, yet benign users.*

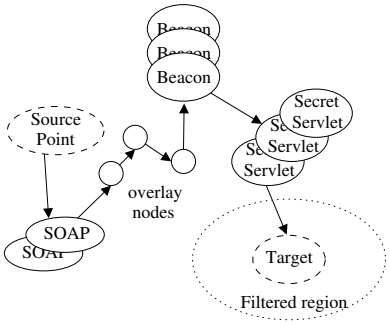

Figure 1: Basic SOS architecture. SOAP stands for Secure Overlay Access Point, and represents an entry point to the SOS overlay. SOS nodes can serve any of the roles of SOAP, Beacon, or Secret Servlet.

Since traditional firewalls themselves are susceptible to DoS attacks, what is really needed is a distributed firewall [4, 18]. To avoid the effects of a DoS attack against the firewall connectivity,

instances of the firewall are distributed across the network. Expensive processing, such as cryptographic protocol handling, is farmed out to a large number of nodes. However, firewalls depend on topological restrictions in the network to enforce access-control policies. In what we have described so far, an attacker can launch a DoS attack with spoofed traffic purporting to originate from one of these firewalls, whose identity cannot be assumed to remain forever secret. The insight of SOS is that, given a sufficiently large group of such firewalls, one can select a very small number of these as the designated authorized forwarding stations: only traffic forwarded from these will be allowed through the filtering router. In SOS, these nodes are called *secret servlets.* All other firewalls must forward traffic for the protected site to these servlets. Figure 1 gives a high-level overview of a SOS infrastructure that protects a target node or site so that it only receives legitimate transmissions. Note that the secret servlets can change over time, as long as the filtering router is also updated, and that multiple sites can use the same SOS infrastructure.

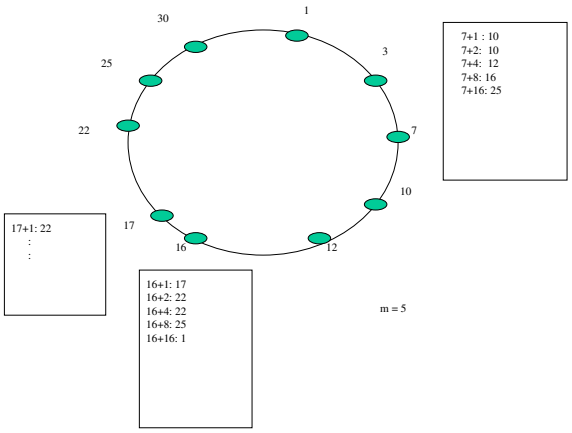

Figure 2: Chord-based overlay routing.

To route traffic inside the overlay, SOS uses Chord [34], which 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 [19] 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 pre-determined 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 $x + 2^{i-1}(\pmod{2^m})$, as shown in Figure 2. 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. In the example, if node 7 receives a packet whose destination is the identifier 20, the packet will route from 7 to 16 to 17. When the packet reaches node 17, the next node in the overlay is 22, and hence node 17 knows that 22 is responsible for identifier 20. 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 stated properties continue to hold.

SOS uses the IP address 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 IP address. This node, to which Chord delivers the packet, is not the target, nor is it necessarily the secret servlet. It is simply a unique node that will be eventually be reached, after up to $m = \log N$ overlay hops using Chord, regardless of the entry point. This node is called the *beacon*, since it is to this node that packets destined for the target are first guided. Chord therefore provides a robust and reliable, while relatively unpredictable for an adversary, means of routing packets from an overlay access point to one of several beacons.

Finally, the secret servlet uses Chord to periodically inform the beacon of the secret servlet's identity. Should the servlet for a target change, the beacon will find out as soon as the new servlet sends an advertisement. If the old beacon for a target drops out of the overlay, Chord will route the advertisements to a node closest to the hash of the target's identifier. Such a node will know that it is the new beacon because Chord will not be able to further forward the advertisement. By providing only the beacon with the identity of the secret servlet, traffic can be delivered from any firewall to the target by traveling across the overlay to the beacon, then from the beacon to the secret servlet, and finally from the secret servlet, through the filtering router, to the target. This allows the overlay to scale for arbitrarily large numbers of overlay nodes and target sites. *Unfortunately, this also increases the communication latency, since traffic to the target must be redirected several times across the Internet.* If the overlay only serves a small number of target sites, regular routing protocols or RON-like routing [2] may be sufficient.

2.2 Graphic Turing Tests

In order to prevent automated attacks from breaching the overlay, a CAPTCHA [36][37] visual test is implemented at the entry point of the overlay to verify the presence of a human user. CAPTCHA (Completely Automated Public Turing test to Tell Computers and Humans Apart) is a program that can generate and grade tests that most humans can pass, but automated programs cannot.

The particular CAPTCHA realization we use is GIMPY [37], 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 himself/herself by entering as ASCII text the same sequence of letters as what appears in the image. Updating the GIMPY interface to WebSOS can be performed without modifying the other architectural components.

Although recent advances in visual pattern recognition [25] can defeat GIMPY, there is no solution to date that can recognize complicated images or relation between images like Animal-PIX [35]. Although for demonstration purposes in our prototype, described in Section 4, we use GIMPY, we can easily substitute it with any other instance of graphic turing test.

Figure 3: WebSOS implementation of user Web Challenge using CAPTCHA. The challenge in this case is "fwst".

2.3 Sequence of Operations in WebSOS

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

- A site (target) installs a filter on a router in its immediate vicinity and then selects a number of WebSOS nodes to act as "secret servlets" that are allowed to forward traffic through the filter to the target site. Routers at the perimeter of the site are instructed to only allow traffic from these servlets to reach the internal of the site's network. These routers are powerful enough to do filtering using only a small number of rules on incoming traffic without adversely impacting their performance. In order to make guessing the identity of a secret servlet for a particular target harder for the attacker, the filtering mechanism uses packet fields with potentially high entropy. For example, only GRE [12] packets from a particular source (the secret servlet) containing a specific 32-bit value in the GRE Key field [11]. An attacker trying to slip attack traffic through the filter must guess not only the current servlet's IP address, but the correct 32-bit key as well. Although we expect 32 bits to be sufficient for this application, we can easily use larger keys with GRE to avoid brute-force attacks.

- When a WebSOS node is informed that it will act as a secret servlet for a site (and after verifying the authenticity of the request, by verifying the certificate received during the SSL exchange), it will compute the key k for each of a number of well-known consistent hash functions, based on the target site's network address. Each of these keys will identify a number of overlay nodes that will act as beacons for that web server.

- Having identified the beacons, the servlets or the target will contact them, notifying them of the servlets' association with a particular target. Beacons will store this information and use it for traffic-forwarding purposes.

- A source that wants to communicate with the target contacts a random overlay node, the Secure Overlay Access Point (SOAP). After authenticating and authorizing the request via the CAPTCHA test, the overlay node securely proxies all traffic from the source to the target via one of the beacons. The SOAP (and all subsequent hops on the overlay) can proxy the HTTP request to an appropriate beacon in a distributed fashion using Chord, by applying the appropriate hash function(s) to the target's IP address to identify the next hop on the overlay. To minimize delays in future requests, the client is issued a short-duration X.509 certificate, bound to the SOAP and the client's IP address, that can be used to directly contact the proxy-server component of the SOAP without requiring another CAPTCHA test.

This scheme is robust against DoS attacks because if an access point is attacked, the confirmed source point can simply choose an alternate access point to enter the overlay. Any overlay node can provide all different required functionalities (SOAP, Chord routing, beacon, secret servlet). If a node within the overlay is attacked, the node simply exits the overlay and the Chord service self-heals, providing new paths over the re-formed overlay to (potentially new sets of) beacons. Furthermore, no node is more important or sensitive than others — even beacons can be attacked and are allowed to fail. Finally, if a secret servlet's identity is discovered and the servlet is targeted as an attack point, or attacks arrive at the target with the source IP address of some secret servlet, the target can choose an alternate set of secret servlets.

Use of GRE for encapsulating the traffic between the secret servlet and the filtering router can offer an additional benefit, if we also use transparent proxies and IPsec for packet encapsulation between the proxies (replacing SSL). In that implementation scenario, as far as the target web server is concerned the HTTP/HTTPS connection from the browser was received directly. Thus, any return TCP traffic will be sent directly to the browser's IP address. Following our discussion in Section 2.4, this asymmetric connection routing will considerably improve the end-to-end latency and reduce the load on the overlay network (less traffic to proxy). While asymmetric routing was once considered potentially harmful, empirical studies show that most of the long-haul traffic (*e.g.,* non-local traffic) over the Internet exhibits high asymmetry [1]. Most of the arguments against this asymmetry arise from the difficulty of configuring packet classification mechanisms, which preclude stateful filtering and required synchronized configuration of multiple nodes (those the traffic may traverse). This would not be a problem in our case, as the asymmetry is exhibited far enough in the network (beyond the filtering router) that the local administrative tasks, such as configuring a firewall, remain unaffected. IPsec and transparent proxying techniques are well-known and (in the case of transparent proxies) widely used, thus we believe such an implementation is not unfeasible. For the purposes of this paper, we decided to implement the straight-forward version of WebSOS; development of the optimized version remains in our plans for future work.

In [22], the authors 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 the SOS system implements when it detects an attack. The authors 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. For more details on the analysis, see [22].

2.4 Forwarding Specifics

WebSOS uses SSL to provide two layers of encryption. First, messages are encrypted end-to-end, so that only the end-points of

the exchange (user and web-server) can view the data actually being transmitted. Additionally, WebSOS uses SSL over each hop of the overlay as a means of verifying the authenticity of the previous hop. No special functionality is required by the overlay nodes to perform these tasks; the user browser simply has to be supplied with the appropriate public key certificate(s) from the WebSOS administrator.

In the original SOS architecture, the path established from the user to the target through the overlay was unidirectional. Traffic in the reverse direction could also traverse the overlay, by reversing the roles of user and target. In that case, the path taken by requests and responses would be different. Alternatively, traffic from the target to the user could be sent directly (without using the overlay); this is usually not a problem, since most communication channels are full-duplex and, in the event of a DDoS attack, only the downstream portion (to the target) is congested. An additional benefit of this asymmetric approach is reduced latency, since most client/server traffic (especially in web environments) is highly asymmetric (*i.e.,* clients receive a lot more information than they transmit). This was possible because routing decisions in SOS are made on a per-packet basis.

In WebSOS, routing decisions are made on a per-connection basis. Any subsequent requests over the same connection (when using HTTP 1.1) and any responses from the web server can take the reverse path through the overlay. While this makes the implementation simpler, it also introduces increased latency, as the bulk of the traffic will also traverse the overlay. We give some thoughts on how to address this issue in Section 5.

3. SIMULATION

To understand the impact of the overlay network on the routing of packets between the source and target nodes, we have applied the SOS algorithm to two models of ISP networks [7].One model, indicative of a U.S. topology, is derived from AT&T's U.S. backbone network. The other, indicative of a European topology, is derived from Worldcom's (now MCI's) European backbone network. Remote access points were excluded from the AT&T model, as were connections from Worldcom's European POPs to points outside the geographical area. For each model, two algorithms for routing traffic through the overlay were tested, one based on Chord, which uses a random ordering of the overlay nodes, and a heuristic variation of CAN [28] that uses geographical ordering of the overlay nodes. In both cases, we tested variations on how the beacons and servlets were chosen in relation to each other, the target, and the source, *e.g.,* requiring some minimum distance between the servlet and target.

We first give a brief description of CAN, and then discuss the specifics of the simulation environment, such as ISP structure, the distribution of overlay nodes across ISP Points of Presence (POPs), and the selection strategies for beacons and secret servlets.

3.1 CAN

Like Chord, CAN uses a hash function to map overlay nodes to identifiers. However, a CAN identifier maps a node to a region within a d-dimensional space. Each overlay node contains a table of overlay nodes responsible for neighboring areas in the coordinate space. As shown in Figure 4, overlay node 7 would contain pointers to nodes 3, 6, 8, and 11. In its basic form, CAN does not assume any relationship between node positions of the coordinate space and their geographical positions in the real world. A variation suggested in [28] that assigns positions within the coordinate space being representative of the geography provided the basis for the heuristic used in the model.

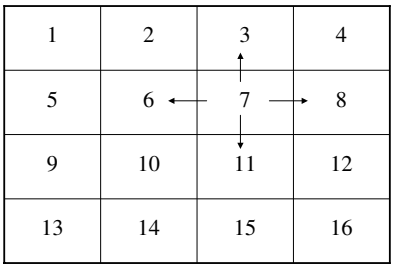

Figure 4: Overlay nodes serving regions of a coordinate-space.

3.2 Network Layout

A POP-level representation of the ISP was used, where each POP is assumed to consist of a hierarchy of routers as shown in Figure 5. At the top level are routers with links to other POPs. At the lowest level are links to client networks.

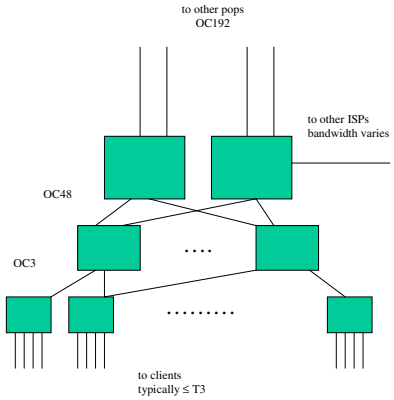

Figure 5: ISP POP structure used in the simulation.

Latencies between POPs were estimated from a subset of known latencies. Distances between POPs were estimated using airline miles. Three routers were included at the second level and twelve at the lowest level of each POP; however, for the statistics computed, the exact number of routers within a POP was not relevant, only the latency from the time a packet entered a router within a POP to the time it left the POP was needed.

The model assumes that there is ample bandwidth between POPs and that the choke points would be the links to clients. All latencies and distances to clients to their local POP are assigned the same value.

There were 19 POPs in the US model and 18 in the Europe model. Overlay nodes participating in the overlay were evenly distributed across POPs, meaning each POP served the same number

of client nodes eligible to be overlay nodes. In the cases where servlets and beacons were randomly chosen, this allowed each POP to be equally likely to have a client site that was a servlet or beacon. In the cases where the servlet and beacon nodes were not randomly chosen, there were more eligible nodes per POP than utilized and the even distribution did not impact selection. A node was not allowed to serve more than one purpose for a specific source-target pair, for example, a node could not be both a beacon and a servlet for the same target. Removing the restriction would result in shorter routes on average because some scenarios tested would pick the same node for both the servlet and beacon.

In each case, two client nodes served by each POP were included in the overlay. Since each source / target pair was tested individually, at most two nodes per POP would be selected to serve the functions of beacon and servlet. When ordering the overlay nodes according to the geographic heuristic described below, designating more than two nodes per POP could only change a route between a source and target by possibly passing through a different client on a given POP. When ordering the overlay nodes randomly and using Chord as the routing algorithm for the overlay, the probability that a client on a specific POP was picked as a beacon or servlet, or was at a certain position in the overlay impacted the route. Since it was assumed overlay eligible nodes were evenly distributed across all POPs, having 2 versus 100 overlay nodes per POP would not impact the probabilities and thus would not affect the results. The node for the source was chosen to be a client on the same POP as the source. The impact due to it being served by a different POP than the source would be to add the cost of the normal route between the source and SOAP to the cost of the route between the SOAP and target.

3.3 Routing Algorithms

In SOS, traffic from a source to a target utilizes a route which contains the following sequence of nodes in order: source, access point, beacon, servlet and target. Normal routing is used to reach the SOAP. Also, since the beacon knows the specific servlet for the target, and the servlet knows the location of the target, normal routing is used between the beacon and servlet, and between the servlet and target. An overlay route is used between the SOAP and beacon. The increase in the route length over that of the normal route between the source and target is due not only to the requirement that the route pass through specific nodes, but also due to the need to route through an overlay network between the SOAP and beacon as opposed to using the normal route between the two nodes. For normal routing, each node in the model contained a routing table populated via Dijkstra's algorithm, using minimum hops as the criteria for shortest path. Each node in the overlay network also contained a table with the destination address and overlay node id of a subset of overlay nodes. The table was populated based on the routing algorithms described below.

A routing algorithm for use in overlays is required to send traffic between the SOAP and beacon. The Chord algorithm was utilized in the first set of experiments. The overlay nodes where randomly ordered. The tables within each overlay node were populated using the method described previously involving powers of 2. The size of a node's table is $O(\log n)$, where n is the size of the overlay.

The second set of experiments used a heuristic which divided the POPs into geographical areas. This method is based on modifications suggested to the basic algorithm for CAN. For a specific area, A, a node n_A was chosen as the area's representative. Each n_A was an entry in each overlay node's table. In addition, if n_i is an overlay node in area A, n_i's table would include entries for each n_j in A, $i \neq j$. Thus an overlay node maintained pointers to every other overlay node in the same geographical area and to one overlay node in each other geographical area. For an overlay of size n, the size of a node's table is $O(n/5) + \#(\text{areas})$, which is $O(n/5)$ when n is large compared to the number of areas. The US model involved 6 areas, one contained 2 POPs and the other contained 3 or 4 POPs each. The Europe model contained 4 areas with 4 to 5 POPs each.

3.4 Beacon/Servlet Selection Scenarios

Seven source-target pairs were chosen in each of the two models. They were selected to represent a variation in source-target relations. Factors considered when selecting the pairs included the distance between cities, whether they were served by neighboring POPs and the level of connectivity for the POP. In all cases a servlet and beacon for a specific target were not permitted to be the same node and neither could serve as a SOAP .

For each model and each routing algorithm, the normal route between each source-target pair was computed then the following eight scenarios were tested on each pair. In the scenarios, minimizing the number of hops refers to the number of hops as calculated by normal routing.

1. Randomly select the servlet and beacon (100 trials per source-target pair were run).
2. Select the servlet to minimize the number of hops between the servlet and target, then select the beacon to minimize the number of hops between the beacon and servlet, with the restriction that the servlet and beacon not be served by the same POP.
3. Select the servlet to minimize the number of hops between the servlet and target, then select the beacon to minimize the number of hops between the beacon and source.
4. Select a servlet randomly from those approximately X miles from the target then select a beacon randomly from those approximately X miles from the servlet, where X was 1000 in the US model and 500 in the Europe model. In the case of the Europe model, a few POPs did not have neighbors within this distance, in which case the next closest available overlay node was used.

The first scenario was used to obtain an understanding of the impact when no selection criteria was utilized for the servlet and beacon. This would be the simplest version to implement. The second and third scenarios were aimed at keeping the intermediate nodes in the route near the end points to determine if the route between the source and target would then be similar to the normal route. These two scenarios using minimum distance instead of hops were tested on the US version, but the results were not noticeably different from the scenarios using hops. The fourth scenario was used to understand the impact of selecting the servlet and beacon so they would be served by different POPs than the target, which may be desired for diversity, but at the same time guaranteeing they would be relatively close in an attempt to avoid an unnecessarily long route.

3.5 Results

Results are presented in terms of the ratio of the measurement for the SOS route to that of the normal route between the source and target. The measurements are for one direction only, source to target. Table 1 shows the ratio of the latency using SOS to the latency expected when using normal routing. The scenario number corresponds to the previous list. These were averaged over all source-target pairs. The worst case from all source-target pairs is

Table 2: Worst-case ratio: latency with SOS *vs.* normal routing.

	US/Chord	US/CAN	Europe/Chord	Europe/CAN
scenario				
1 random selection — worst individual source-target average over 100 trials	8.76	6.05	8.05	5.81
2 min hops	7.57	3.76	4.74	3.26
3 min hops	10.9	2.14	11.29	2.14
4 diversity	10.57	6.24	8.1	3.57

Table 3: Numbers of POPs in SOS routing *vs.* normal routing.

	US/Chord	US/CAN	Europe/Chord	Europe/CAN
scenario				
1 random selection — worst individual source-target average over 100 trials	4	3	4	2.5
2 min hops	2	1.5	2	1.5
3 min hops	5	1	4.2	1
4 diversity	3.5	2.5	4.2	2

Table 1: Average ratio: latency with SOS *vs.* normal routing.

	US Chord	US CAN	Europe Chord	Europe CAN
scenario				
1 random selection	4.51	4.16	5.69	4.11
2 min hops	3.45	2.4	3.25	2.54
3 min hops	7.19	1.75	6.77	1.74
4 diversity	5.18	4.08	5.6	2.88

shown in Table 2. Table 3 indicates the increase in the number of ISP POPs involved in a route compared to that of the normal route.

When using scenario 3 with the geographic heuristic, the servlet was always selected from a node on the same POP as the target and the beacon was selected from a node on the same POP as the source and SOAP because there were eligible nodes at every POP. This resulted in the SOS route being identical to the normal route with the addition of a few detours to clients within the first and last POPs in the route, thus it was expected to produce the best results in terms of latency.

The results reported for random selection are averaged over 100 trials run per source-target pair. The actual increase in latency may be much higher depending on the specific servlet and beacon chosen. The greatest increase occurs when the source and target are close together. The overlay route may involve points geographically far from the source and target, turning a normally short route into one that may traverse every POP in the ISP at least once. Among all trials involving random selection, the worst case in the Europe model was an increase in latency 15 times that of the normal route between London and Paris when using Chord and 9.5 times when using the geographical heuristic. In the US model, the worst case also involved a latency 15 times normal between NY and Philadelphia when using Chord and 8.86 times when using the geographical heuristic. For NY to Philadelphia, the worst case increase using the geographical heuristic is approximately the same as the average (8.76) when using Chord. The worst cases from all trials involved latencies of 378*ms* using Chord and 230*ms* using the geographical heuristic.

The number of POPs serves as a measure of the complexity of the route but does not necessarily imply a physically long route because several POPs may be geographically close. In scenario 3, the beacon would be selected on the same POP as the SOAP. The ratio for scenario 3 using Chord is high due to a couple of source-target pairs in which the beacon's overlay id was just prior to that of the SOAP's id, resulting in routing through several overlay nodes in the path between the SOAP and beacon.

When using Chord, other variations for populating the overlay node's tables using powers of 3 and $i + x_j$, where x_j is the j^{th} number in the Fibonacci series, for j = 3,4,5.., were tested on a subset of source-target pairs but had no noticeable impact on the length of the route between the SOAP and beacon. A geographic ordering of the overlay nodes was also tested while maintaining the Chord routing. Nodes that were geographically close were assigned IDs placing them close together on the overlay network. While this shortened the route in cases where nodes X and Y were physically close, a packet was being routed from X to Y using the overlay and X was assigned a lower overlay id than Y; it resulted in a worst case scenario when Y was assigned the overlay id just prior to X's because the packet would route to $O(\log n)$ overlay nodes before reaching the one that knew about X.

3.6 Other Considerations

If the overlay nodes are placed within POPs, as opposed to being located in clients' networks, we eliminate the latency due to the connection between the POP and client, and it could be more difficult to attack. In contrast to a client's LAN which may receive traffic for multiple reasons and has a relatively low bandwidth connection to the POP, a server dedicated to SOS and attached to a router within a POP allows most invalid traffic to be filtered out in a high-capacity area. However, the use of special purpose servers would result in fewer potential overlay nodes. Furthermore, such servers would not remove the delay due to cross-country routes through the overlay.

Having the overlay network span multiple ISPs will increase the latency of the SOS route. There will be a larger number of POPs serving potential overlay nodes. Even if the overlay nodes are geographically distributed in the same manner as with one ISP, the route between any pair of overlay nodes will increase on average due to having to route between ISPs. When the overlay nodes are in the same city but are served by different ISPs, having to route from

one ISP POP to another ISP's POP, as opposed to routing between nodes within the same POP, will increase latency. Furthermore, if there is no peering point between the ISPs for that city, the route will require a path to a different city.

4. IMPLEMENTATION

While the simulation results are encouraging, we felt that experimentation in real networks was necessary to validate our approach. To that end, we developed a prototype of WebSOS, consisting of three main modules. The components are a communications module, a WebSOS routing module, and an overlay routing module running on each node in the WebSOS overlay. The interactions of these are shown in Figure 6.

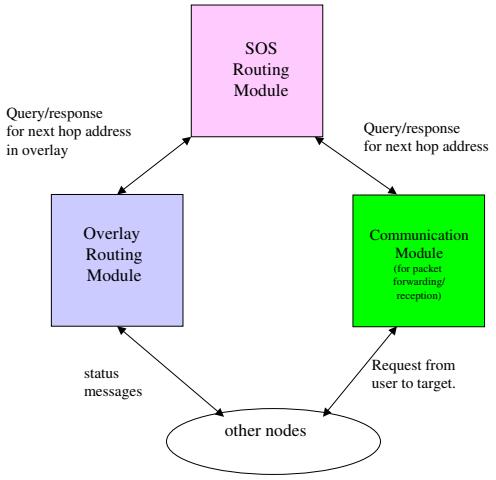

Figure 6: Software modules for the WebSOS implementation.

The communications module is responsible for forwarding HTTP requests and responses among the nodes in the WebSOS overlay. When a new proxy request (in the form of a new TCP connection) is received, the communications module calls the WebSOS routing module with the target's destination address to obtain the address of the next hop in the overlay. It then opens a new TCP connection to that node and relays the received HTTP request. Any traffic received in the opposite direction (*i.e.,* the HTTP responses and web content) are relayed back to the source. Authentication of the requesting node by the access point (SOAP) and by internal nodes is accomplished through SSL. Authorized users and WebSOS overlay nodes are issued X.509 [5] certificates signed by the SOAP, once the user has succeeded in the CAPTCHA authentication.

The main *WebSOS routing module* receives requests from the communications module and responds with the IP address of the next node in the WebSOS overlay to which the request should be forwarded. The module first checks whether the current node serves a specific purpose (*i.e.,* whether is it a beacon or secret servlet for that target). If the node serves no such purpose, the module calls the overlay routing module to determine the next hop in the WebSOS overlay and passes the reply onto the communications module. Presently, the WebSOS routing module is initialized with configuration data at startup indicating which nodes serve specific purposes. We are working on an administrative module with increased flexibility to avoid this static provisioning.

The overlay routing module is a general routing algorithm for overlay networks. An implementation of Chord was written for the initial tests. However, this module can be replaced with any other routing algorithm, *e.g.,* CAN [28]. It receives queries containing a destination IP address (the web server's) and responds with the IP address of the next node in the overlay to which the request should be forwarded. For maintenance of its own routing algorithm, the Chord implementation also communicates with other overlay nodes to determine their status, as described in [34].

When a request is issued by the browser, 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 WebSOS nodes are dynamically established, as new requests are routed. One problem we ran into while developing the WebSOS prototype is that web browsers do not provide support for the actual proxy request to be encrypted. To solve this problem, we wrote a port forwarder that runs on the user's system, accepts plaintext 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 itself. The Java applet itself is not considered part of the WebSOS overlay and is not trusted to perform any access control decisions; it is simply a "helper" application.

Thus, to use WebSOS, an authorized user simply has to access any SOAP, successfully respond to the CAPTCHA challenge, download the applet, and set the browser's proxy settings to the localhost, as shown in Figure 7. 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 is successful in his/her reply, then the web server connects to a DBMS system (local or remote) and associates a pair of RSA keys (a private key and a certificate) with the host. This set of keys 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 HTTP/HTTPS connections on a local port (*e.g.,* 8080) and establishes an SSL-tunnel connection with the proxy server running on the SOAP (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.

Initial prototyping of the communications module used Apache, whose proxy module was modified to query the routing module for the next hop. This worked well when unencrypted HTTP requests were issued by the browser. However, when we encountered the requirement for end-to-end authentication and encryption, we changed the implementation to use a stand-alone proxy server instead of Apache.

We intend to expand the implementation to include additional modules addressing the administration and maintenance of the overlay. A centralized administration module will be used to set node characteristics in real time, including assigning specific roles (beacon, SOAP, secret servlet) to nodes, and changing the operational status of nodes. A maintenance module running on each node will check the status of all nodes in the WebSOS overlay and provide updates to both the main and overlay routing modules in order for routing to be adjusted. This module will also serve as the interface to centralized administration by receiving updates regarding a

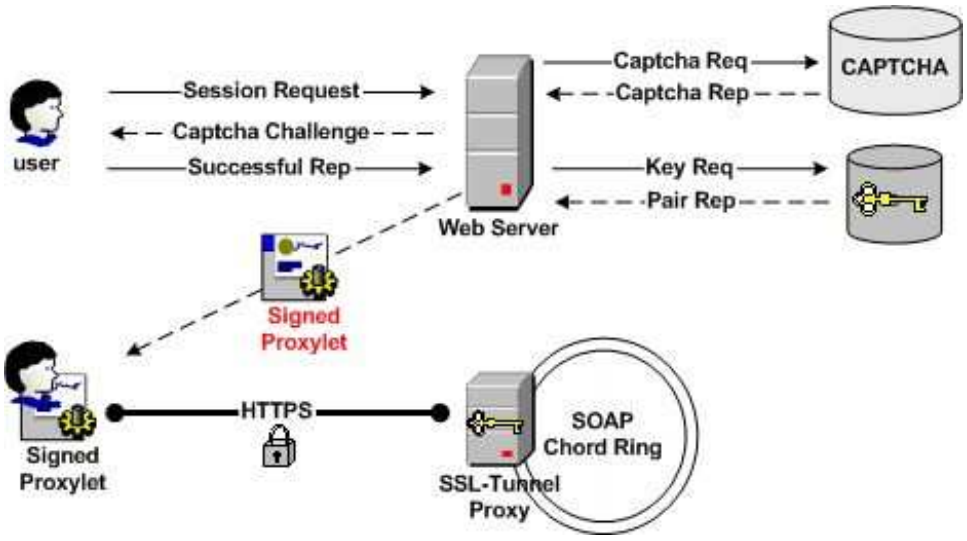

Figure 7: WebSOS client session initiation diagram.

node's function and status, and passing the updates to the appropriate routing module.

An adaptation of the initial implementation was created, to improve performance: rather than transporting the request and response through the full overlay network, only routing information travels through the overlay. As before, the requester makes a proxy request to the SOAP. At that point, the SOAP sends a UDP message into the overlay, specifying the target. The message is routed to the beacon, which responds directly to the SOAP with information on the secret servlet for that target. The SOAP then connects to the servlet, which proxies the request as before, in effect creating a *shortcut* through the overlay.

The SOAP caches the servlet information for use in future requests. That information is timed out after a period of time to allow for changes to propagate correctly. The same basic UDP protocol is used by servlets to announce their presence to (and periodically update) the beacons for the various targets.

5. EXPERIMENTAL EVALUATION

In order to quantify the overhead associated with use of WebSOS, we created a simple topology running on the local network (100 Mbit fully-switched Ethernet). For our local-area network overlay, we used 10 commodity PCs running Linux Redhat 7.3. We measured the time-to-completion of https requests. That is, we measured the elapsed time starting when the browser initiates the TCP connection to the destination or the first proxy, to the time all data from the remote web server have been received. We ran this test by contacting 3 different SSL-enabled sites: *login.yahoo.com, www.verisign.com,* and the Columbia course bulletin board web service (at `https://www1.columbia.edu/sec/bboard`). For each of these sites, we measured the time-to-completion for a different number of overlay nodes between the browser and the target (remote web server).

The browser was located on a separate ISP The reason for this configuration was to introduce some latency in the first-hop connection (from the browser to the SOAP), thus simulating (albeit using a real network) an environment where the browsers have slower access links to the SOAPs, relative to the links connecting the overlay nodes themselves (which may be co-located with core routers). By locating all the overlay nodes in the same location, we effectively measure the aggregate overhead of the WebSOS nodes in the optimal (from a performance point of view) case.

Table 4 shows the results for the case of 0 (browser contacts remote server directly), 1, 4, 7, and 10 overlay nodes. The times reported are in seconds, and are averaged over several HTTPS GET requests of the same page, which was not locally cached. For each GET request, a new TCP connection was initiated by the browser. The row labeled "Columbia BB (2nd)" shows the time-to-completion of an HTTPS GET request that uses an already-established connection through the overlay to the web server, using the HTTP 1.1 protocol.

As the figure shows, WebSOS increases the end-to-end latency between the browser and the server by a factor of 2 to 3. These results are consistent with our simulations of using SOS in an ISP topology, where the latency between the different overlay nodes would be small, as discussed in Section 3. The increase in latency can be primarily attributed to the network-stack processing overhead and proxy processing at each hop. It may be possible to use TCP splicing [6] or similar techniques to reduce connection handling overhead, since WebSOS performs routing on a per-request basis. Also, in the experiments we ran, we did not make use of the asymmetric routing option possible with the use of GRE as both a filtering and an encapsulation mechanism, as discussed in Section 2.3.

Furthermore, there is an SSL-processing overhead for the inter-overlay communications. A minor additional cryptographic overhead, relative to the direct access case, is the certificate validation that the SOAPs have to perform, to determine the client's authority to use the overlay, and the SSL connection between the proxy running on the user's machine and the SOAP. As shown in [23], such overheads are typically dominated by the end-to-end communication overheads. Use of cryptographic accelerators can further improve performance in that area. One further optimization is to maintain persistent SSL connections between the overlay nodes. However, this will make the task of the communication module harder, as it will have to parse HTTP requests and responses arriving over the same connection in order to make routing decisions.

Table 5 shows the same experiment using PlanetLab [26], a wide-

Table 4: Latency (in seconds) when contacting various SSL-enabled web servers directly and with different numbers of (intermediate) overlay nodes over the local-Ethernet network.

Server	Direct	1 node	4 nodes	7 nodes	10 nodes
Yahoo!	1.39	2.06	2.37	2.79	3.33
Verisign	3.43	4.22	5.95	6.41	9.01
Columbia BB	0.64	0.86	1.06	1.16	1.21
Columbia BB (2nd)	0.14	0.17	0.19	0.20	0.25

Table 5: Latency (in seconds) when contacting various SSL-enabled web servers directly and with different numbers of (intermediate) overlay nodes using the PlanetLab network.

Server	Direct	1 node	4 nodes	7 nodes	10 nodes
Yahoo!	1.39	3.15	5.53	10.65	14.36
Verisign	3.43	5.12	7.95	14.95	22.82
Columbia BB	0.64	1.01	1.45	3.14	5.07
Columbia BB (2nd)	0.14	0.23	0.28	0.57	0.72

area overlay network testbed. The PlanetLab nodes are distributed in academic institutions across the country, and are connected over the Internet. We deployed our WebSOS proxies PlanetLab and ran the exact same tests. Naturally, the direct-contact case remains the same. We see that the time-to-completion in this scenario increases by a factor of 2 to 10, depending on the number of nodes in the overlay. In each case, the increase in latency over the local-Ethernet configuration can be directly attributed to the delay in the links between the WebSOS nodes. While the PlanetLab configuration allowed us to conduct a much more realistic performance evaluation, it also represents a worst-case deployment scenario for WebSOS: typically, we would expect WebSOS to be offered as a service by an ISP, with the (majority of) WebSOS nodes located near the core of the network. Using PlanetLab, the nodes are distributed in (admittedly well-connected) end-sites. We would expect that a more commercial-oriented deployment of WebSOS would result in a corresponding decrease in the inter-overlay delay. On the other hand, it is easier to envision end-site deployment of WebSOS, since it does not require any participation from the ISPs.

Finally, while the additional overhead imposed by WebSOS can be significant, we have to consider the alternative: no web service while a DoS attack against the server is occurring. While an increase in end-to-end latency by a factor of 5 (or even 10, in the worst case) is considerable, we believe it is more than acceptable in certain environments and in the presence of a determined attack.

Table 6 shows the results when the shortcut implementation was tested on the PlanetLab testbed. This implementation provides significant performance improvements, particularly on subsequent requests for the same site because of the caching. To simulate the effects of an attack on individual nodes in the overlay, we simply brought down specific nodes. The system healed itself within 10 seconds.

Table 6: Latency (in seconds) when contacting various SSL-enabled web servers directly and while using the shortcut implementation of the WebSOS system. The testing was performed on a 76 node subset of the PlanetLab testbed using the Chord overlay. The hops to the beacon ranged from 4 to 8 and did not have a significant effect on latency. The *cached* column refers to subsequent requests using the same SOAP, whereupon the Secret Servlet information has been cached.

Server	Direct	Original Request	Cached Requests
Yahoo!	1.39	4.15	3.67
Verisign	3.43	7.33	6.77
Columbia BB	0.64	3.97	3.43
Columbia BB (2nd)	0.14	0.55	0.56

6. 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 [24]. Worse, the Internet is increasingly being used for time-critical applications (*e.g.,* electricity production monitoring and coordination between different generators). A further compounding factor is the susceptibility of the basic protocols (*i.e.,* IP and TCP) to denial of service attacks [33, 16, 31].

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.,* [17, 9, 32, 15]. However, 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. Thus, they fall into this class of approaches that are reactive. 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 are not always adequate solutions.

- Methods that filter traffic by looking for known attack patterns or statistical anomalies in traffic patterns can be de-

feated by changing the attack pattern and masking the anomalies that are sought by the filter. Furthermore, statistical approaches will likely filter out valid traffic as well. Since the Internet spans multiple administrative domains and (legal) jurisdictions, it is often very difficult, if not outright impossible, to shut down an attack by contacting the administrator or the authorities closest to the source. In any case, such action cannot be realistically delivered in a timely fashion (often taking several hours). Even if this were possible, it is often the case that the source of the attack is not the real culprit but simply a node that has been remotely subverted by a cracker. The attacker can just start using another compromised node.

- Using a "pushback"-like mechanism, such as the one described in [17], to counter a DoS attack makes close cooperation among different service providers necessary: since most attacks use random source IP addresses (and since ingress filtering is not widely used), the only reliable packet field that can be used for filtering is the destination IP address (of the target). If filters can only be pushed "halfway" through the network between the target and the sources of the attack, the target runs the risk of voluntarily cutting off or adversely impacting (*e.g.,* by rate-limiting) its communications with the rest of the Internet. The accuracy of such filtering mechanisms improves dramatically as the filters are "pushed" closer to the actual source(s) of the attack. Thus, it will be necessary for providers to allow other providers, or even end-network administrators, to install filters on their routers. Apart from the very realistic possibility of abuse, it is questionable whether such collaboration can be achieved to the degree necessary.

The same concerns hold for the case of collaborative action by the ISPs: even easy to implement mechanisms such as ingress filtering, that could reduce or even eliminate spoofed-address DoS attacks, are still not in wide use. We believe it is rather unrealistic to expect that cooperative providers would even establish static filters to allow legitimate (paying) clients to tunnel through their infrastructure with any assurance of quality of service, and much less so for the case of mobile or remote clients (as may be the case for emergency teams).

[20] proposes using Class-Based Queuing on a web load-balancer to identify misbehaving IP addresses and place them in lower priority queues. However, most DDoS attacks use spoofed IP addresses that vary over time, thus defeating classification. Even if the same address is used, the amount of state that the load-balancer needs to keep may be prohibitive. Furthermore, many of the DDoS attacks simply cause congestion to the web server's access link. To combat that, the load-balancer would have to be placed closer to the network core. Not only would this further compound the state-explosion problem, but such detailed filtering and especially state-management on a per-source-IP address basis can have performance implications at such high speeds.

Another approach to mitigating DoS attacks against information carriers is to massively replicate the content being secured around the entire network. 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 and generalization of the ideas in SOS appears in [3]. 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.

7. CONCLUSIONS

We presented WebSOS, an architecture that allows legitimate users to access a web server in the presence of a denial of service attack. The architecture uses a combination of Graphic Turing tests, cryptographic protocols for data origin authentication, packet filtering, overlay networks, and consistent hashing to provide service to casual web-browsing users. We discussed our prototype implementation, which uses standard web proxying and authentication mechanisms built in all browsers. Our architecture requires no changes to web servers, browsers, or existing protocols.

We conducted a performance evaluation of WebSOS over both a local area network and over the Internet using PlanetLab, a testbed for experimentation with network overlays and similar technologies. Our experiments show that, in a realistic but worst-case deployment scenario, the end-to-end communication latency between browser and server increases on the average by a factor of 7, with a worst case of 10. We also implemented and evaluated a shortcut optimization, which reduced the latency to a factor of 2. These results are consistent with our simulations. We also discussed other optimizations. However, we believe that even at its current level, the overhead imposed is acceptable for many critical environments and applications.

Future work plans include completion and long-term deployment of the WebSOS prototype on PlanetLab, development of the IPsec-enabled prototype that allows for transparent proxying and asymmetric traffic routing for improved performance, and more comprehensive performance measurements, over a longer period of time and for a wider set of users and web sites.

8. ACKNOWLEDGEMENTS

Alexander Konstantinou's NetCallback was used as a basis for some of the forwarding code in the communications module. Abhinav Kamra wrote the Chord implementation used for overlay routing.

9. REFERENCES

[1] L. Amini, H. Schulzrinne, and A. Lazar. Observations from Router-level Internet Traces. In *DIMACS Workshop on Internet and WWW Measurement, Mapping and Modeling*, February 2002.

[2] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proceedings of the 18th Symposium on Operating Systems Principles (SOSP)*, October 2001.

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

[4] S. M. Bellovin. Distributed Firewalls. *;login: magazine, special issue on security*, pages 37–39, November 1999.

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

[6] A. Cohen, S. Rangarajan, and J. H. Slye. On the Performance of TCP Splicing for URL-Aware Redirection. In *USENIX Symposium on Internet Technologies and Systems*, 1999.

[7] D. Cook. Analysis of Routing Algorithms for Secure Overlay Service. Computer Science Department Technical Report CUCS-010-02, Columbia University, April 2002.

[8] L. Cranor, M. Langheinrich, M. Massimo, M. Presler-Marshall, and J. Reagle. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. Technical report, World Wide Web Consortium (W3C), 2002.

[9] D. Dean, M. Franklin, and A. Stubblefield. An Algebraic Approach to IP Traceback. In *Proceedings of the Network and Dsitributed System Security Symposium (NDSS)*, pages 3–12, February 2001.

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

[11] G. Dommety. Key and Sequence Number Extensions to GRE. RFC 2890, September 2000.

[12] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. Generic Routing Encapsulation (GRE). RFC 2784, March 2000.

[13] D. Goldschlag, M. Reed, and P. Syverson. Onion routing for anonymous and private internet connections. *Communications of the ACM (USA)*, 42(2):39–41, 1999.

[14] L. Gong and R. Schemers. Implementing Protection Domains in the Java Development Kit 1.2. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*, pages 125–134, March 1998.

[15] M. T. Goodrich. Efficient Packet Marking forLArg-Scale IP Traceback. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS)*, pages 117–126, November 2002.

[16] L. Heberlein and M. Bishop. Attack Class: Address Spoofing. In *Proceedings of the 19th National Information Systems Security Conference*, pages 371–377, October 1996.

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

[18] S. Ioannidis, A. Keromytis, S. Bellovin, and J. Smith. Implementing a Distributed Firewall. In *Proceedings of Computer and Communications Security (CCS)*, pages 190–199, November 2000.

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

[20] F. Kargl, J. Maier, and M. Weber. Protecting web servers from distributed denial of service attacks. In *World Wide Web*, pages 514–524, 2001.

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

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

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

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

[25] G. Mori and J. Malik. Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA. In *Computer Vision and Pattern Recognition CVPR'03*, June 2003.

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

[27] V. Prevelakis and D. Spinellis. Sandboxing applications. In *Proceedings of the USENIX Technical Annual Conference, Freenix Track*, pages 119–126, June 2001.

[28] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM*, August 2001.

[29] M. K. Reiter and A. D. Rubin. Anonymous Web transactions with crowds. *Communications of the ACM*, 42(2):32–48, 1999.

[30] R. S. Sandhu and J. S. Park. Decentralized user-role assignment for web-based intranets. In *ACM Workshop on Role-Based Access Control*, pages 1–12, 1998.

[31] 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.

[32] 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.

[33] C. Schuba, I. Krsul, M. Kuhn, E. Spafford, A. Sundaram, and D. Zamboni. Analysis of a denial of service attack on tcp. In *IEEE Security and Privacy Conference*, pages 208–223, May 1997.

[34] 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.

[35] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. The CAPTCHA Web Page: *http://www.captcha.net*. 2000.

[36] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. CAPTCHA: Using Hard AI Problems For Security. In *Proceedings of EUROCRYPT'03*, 2003.

[37] L. von Ahn, M. Blum, and J. Langford. Telling Humans and Computers Apart (Automatically). In *Communications of the ACM*, (to appear).

[38] D. S. Wallach, D. Balfanz, D. Dean, and E. W. Felten. Flexible Security Architecture for Java. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP)*, October 1997.