Thwarting Web Censorship with Untrusted Messenger Discovery

Nick Feamster, Magdalena Balazinska, Winston Wang, Hari Balakrishnan, and David Karger

MIT Laboratory for Computer Science
200 Technology Square, Cambridge, MA 02139
{feamster,mbalazin,wwww,hari,karger}@lcs.mit.edu

Abstract. All existing anti-censorship systems for the Web rely on proxies to grant clients access to censored information. Therefore, they face the *proxy discovery problem*: how can clients discover the proxies without having the censor discover and block these proxies? To avoid widespread discovery and blocking, proxies must not be widely published and should be discovered in-band. In this paper, we present a proxy discovery mechanism called *keyspace hopping* that meets this goal. Similar in spirit to frequency hopping in wireless networks, keyspace hopping ensures that each client discovers only a small fraction of the total number of proxies. However, requiring clients to independently discover proxies from a large set makes it practically impossible to verify the trustworthiness of every proxy and creates the possibility of having untrusted proxies. To address this, we propose separating the proxy into two distinct components—the *messenger*, which the client discovers using keyspace hopping and which simply acts as a gateway to the Internet; and the *portal*, whose identity is widely-published and whose responsibility it is to interpret and serve the client's requests for censored content. We show how this separation, as well as in-band proxy discovery, can be applied to a variety of anti-censorship systems.

1 Introduction

Many political regimes and corporations actively restrict or monitor their employees' or citizens' access to information on the Web. Many systems try to circumvent these censorship efforts by using cooperative proxies. Anonymizer [1] is one of the oldest such systems. Peekabooty [15], Safeweb [11], and Zero Knowledge's WebSecure [13] use an SSL-encrypted channel to communicate requests to proxies outside of the censored domain, which then return the censored content over this encrypted channel. In Infranet [3], clients communicate with cooperating proxies by constructing a covert and confidential channel within an HTTP request and response stream, without engendering the suspicion that a visibly encrypted channel might raise.

These systems require a client within the censored domain to discover and communicate with a cooperating proxy outside of the domain, as shown in Figure 1. Each of these systems assumes that a censor blocks access to a Web server

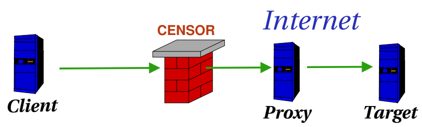

Fig. 1. Current censorship circumvention schemes rely on access to trusted proxies that serve clients' requests for censored content.

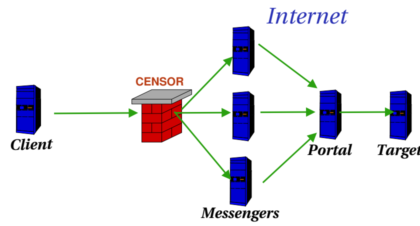

Fig. 2. Forwarding a message and decoding that request can be decomposed into two separate operations.

based on its identity (i.e., IP address or DNS name) and that the censor allows access to any host that does not appear to be delivering objectionable content. Thus, the livelihood of these systems depends on the existence of proxies that the censor does not know about.

All proxy-based censorship avoidance systems face the troubling *proxy discovery problem*. To gain access to censored content, clients must have access to cooperating proxies. However, if the censor can operate under the guise of a legitimate client, it can discover these proxies and block access to them. For example, China's firewall previously blocked access to the Safeweb proxy. An effective proxy discovery technique must allow a client to easily discover a few participating proxies but make it extremely difficult for a censor to discover *all* of these proxies. Any reasonable solution to the problem must defend against both *out-of-band* discovery techniques (e.g., actively scanning or watching traffic patterns) and *in-band* ones (e.g., where the censor itself becomes a client).

To achieve these goals, a proxy-based censorship avoidance system should have the following characteristics:

– *The system should have a large number of proxies.* A system with no more than a few proxies is useless once those proxies are blocked. A system with more proxies makes it more difficult for a censor to block all of them.
– *Clients must discover proxies independently of one another.* If every client discovers the same few proxies, a censor could block access to these popular proxies and render the system useless.
– *The client must incur some cost to discover a proxy.* Because the censor can assume the identity (i.e., IP address) of any client behind its firewall, it is relatively easy for a censor to operate a large number of clients solely to discover proxies. As such, discovering a proxy should require a non-trivial investment of resources, such as solving a client puzzle [6].
– *Brute-force scanning techniques must not expose proxies.* A censor may suspect that a host is a proxy and try to verify this in some fashion (e.g., by acting as a client and seeing if it acts as a proxy, etc.). Thus, to an arbitrary end-host, a proxy should look innocuous.

We propose a proxy discovery technique called *keyspace hopping* that limits in-band discovery of proxies by ensuring that no client knows more than a small random subset of the total set of proxies. The technique also prevents out-of-band discovery by distributing client requests across the set of proxies and ensuring that each cooperating end-host only assumes the role of a proxy for a small set of clients at any given time.

The requirement that clients discover proxies independently implies that clients will utilize arbitrary proxies that they may not trust. This introduces a fundamental tradeoff: while having a large number of independently discoverable proxies makes the system more robust to being blocked, it also makes it increasingly difficult to ensure that all proxies are trustworthy. An ideal proxy discovery system should be resistant to blocking and ensure that the client only exposes its requests for censored content to trusted parties.

We propose a solution that achieves this goal by recognizing that the proxy actually serves two functions: *providing access* to content outside the firewall, and *serving requests* for that content. Our solution, summarized in Figure 2, employs a large number of untrusted *messengers*, which carry information to and from the uncensored Internet, without understanding that information; and a smaller number of *portals*, which a client trusts to faithfully serve requests for censored content without exposing its identity.

2 Proxy Discovery using Keyspace Hopping

Proxy-based anti-censorship systems must enable clients to discover proxies without enabling the censor to discover and block access to all of the proxies. Existing systems assume that there is some way to enable this discovery, but the problem has no obvious solution when the censor can become a client. Because of this possibility, *no single client (or small group of clients) should ever discover all proxies*. Proxies must come into existence more quickly than the censor can block them, and proxy discovery must be based on some client-specific property like IP address to raise the cost of impersonating many clients. In this section, we explore the design space for proxy discovery and describe our proposed mechanism, called *keyspace hopping*, that controls the rate at which any one client can discover proxies. In this section, we assume that the censor cannot operate a proxy, except for our analysis of in-band discovery in Section 2.3. We discuss how to completely relax this assumption in Section 3.

2.1 Design Considerations for Proxy Discovery

Anti-censorship systems should ensure that almost every client can always contact at least one proxy, even if the censor is able to block some of these proxies. The set of proxies should be difficult enough to discover that the only reasonable response by the censor would be to block access to the entire Internet.

A censor can discover proxies in two ways: *in-band*, by acting as a client of the anti-censorship system itself, and discovering proxies in the same manner as any other client; and *out-of-band*, by actively scanning Internet hosts to determine whether any of them behaves like a proxy (we have previously explained the

Technique	Description	Design principles
In-band	Censor becomes a client and attempts to discover proxies in the same way a client would.	– Use client-specific properties for proxy discovery. – Ensure no client can discover more than a small set of all proxies at any time.
Out-of-band	Censor uses traffic anomalies or active scanning techniques to discover proxies.	– Distribute clients evenly among available proxies. – Ensure a host only acts as a proxy for a small subset of clients at any time.

Table 1. A censor can discover and block proxies using either in-band or out-of-band discovery.

importance of maintaining proxy covertness for this reason [3]). Additionally, a censor can notice traffic anomalies that expose a proxy or a client, such as a sudden increase in traffic to a particular Web site or a group of clients that have very similar browsing patterns. Table 1 summarizes these discovery techniques and the corresponding design considerations.

Limiting In-Band Discovery If we assume that a censor can become a client, the censor can use the same discovery mechanisms that a client uses to discover proxies. Thus, the set of proxies that any one client can discover should be small and relatively independent from the sets that other clients discover. This client-specificity implies that clients should discover proxies through some in-band mechanism (note that this is a departure from our previous thoughts about proxy discovery [3]).

To slow in-band discovery, we impose the following constraints: the proxies that any client discovers should be a function of some characteristic that is 1) reasonably specific to that client, 2) not easily modified, and 3) requires significant resources to compute. Two obvious characteristics of a client that satisfy the first two constraints are the client's IP address and subnet. Unfortunately, a censor that operates a firewall can easily assume an IP address or subnet behind that firewall. Hence, we must also require some significant investment of resources *per-client*, such as client puzzles [6], that makes it reasonably expensive for one entity to assume many different identities.

Limiting Out-of-Band Discovery A censor might try to discover proxies using out-of-band discovery techniques. For example, all Web servers that run an Infranet responder might behave in a similar fashion (e.g., providing slower than normal Web response times, etc.). Alternatively, if many clients send requests to a single proxy within a small time period, a censor might notice a large increase in the number of connections to a host that does not ordinarily receive much traffic. It should be reasonably difficult for a censor to discover all proxies using these types of out-of-band discovery techniques.

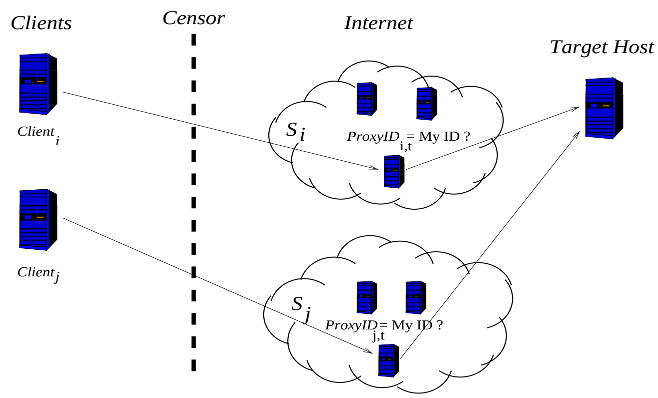

Fig. 3. In keyspace hopping, clients and proxies agree on which proxy forwards which client's request. Each client discovers a unique set of proxies.

To make out-of-band discovery more difficult, a host should only act as a proxy for a certain subset of clients at any time. This prevents one proxy from attracting traffic from an abnormally large number of clients. More importantly, it prevents a host from always appearing as a proxy to all clients, thus making it less likely that an out-of-band probe from an arbitrary host will expose the proxy. Furthermore, the set of clients that a proxy serves should change over time. This makes proxy discovery more difficult for the censor because the censor does not know which hosts are acting as proxies for which clients.

2.2 Keyspace Hopping

We apply the design principles from Section 2.1 to our proxy discovery system, called *keyspace hopping* because of its similarities to frequency hopping [9]. Frequency hopping is used in wireless communication; the basic idea is to modulate a signal on a carrier frequency that changes pseudorandomly over time. Wireless communication uses frequency hopping to resist jamming, since an adversary must either saturate the entire frequency band with noise or track the frequency hopper's choice of carriers.

We propose a similar idea, with the exception that the censor is attempting to jam communication channels by preventing a client from reaching any proxies. At any given time, a certain proxy (or set of proxies) agrees to serve requests for a client, and the client forwards its requests to that proxy, as shown in Figure 3. To block a client's communication with its proxies, the censor must block communication with all of the client's proxies.

Keyspace hopping must solve several problems. The first problem is *proxy assignment*: what is the appropriate mechanism for assigning clients to proxies? Next, clients must perform *lookup*: how do clients discover the IP addresses of their proxies while preventing the censor from performing arbitrary lookups to discover all proxies? Finally, the system must have a *bootstrapping* phase: how can the client, initially knowing nothing, obtain the necessary information to discover its set of proxies? The rest of this section addresses these problems.

Proxy Assignment To guarantee that no single client can ever discover a large fraction of the proxies, keyspace hopping assigns a small subset of all proxies to each client. To prevent proxies from being actively scanned, and to balance client requests across proxies, keyspace hopping dictates when a client can use a particular proxy in its subset.

To facilitate the mapping, each proxy is assigned a globally unique identifier *ProxyID*, such as a hash of the proxy's IP address. The set of proxies for a client is then determined by computing a client-specific index into the total keyspace, and by selecting a constant number of proxies whose identifiers most closely follow the index. The index is computed from a client-specific identifier (which could be, for example, the client's IP address) and a shared secret *hkey* that prevents an adversary from computing the subspace.

A client determines the proxy with which it communicates by adding the output of a uniform collision-resistant hash function, B_i (its base index in the keyspace), to a time-dependent *PreProxyID*, which is determined from the output of a universally-agreed upon pseudorandom number generator, $\mathcal{G}$. *ProxyID* must be based on *hkey* to prevent a censor from recomputing a suspected client's sequence of *ProxyID*s and tracking a suspected client's path through a sequence of proxies (this is particularly important for Infranet, which seeks to preserve client deniability).

The following equations present the assignment more formally:

$$
\begin{aligned}
B_i &\leftarrow \mathcal{H}(\text{Client ID}, \mathit{hkey}) \\
\mathit{PreProxyID}_{t,i} &\leftarrow \mathcal{G}(\text{Client ID}, \mathit{hkey}, t) \\
\mathit{ProxyID}_{t,i} &\leftarrow (B_i + (\mathit{PreProxyID}_{t,i} \mod |S_i|)) \mod |P|
\end{aligned}
$$

where S_i is the set of proxies that client i knows about, and P is the set of all proxies in the system.[1] Both $|S_i|$ and $|P|$ are well-known constants, and $|S_i|$ is the same for all clients i. $\mathit{ProxyID}_{t,i}$ is rounded up to the closest *ProxyID* in the client's set.

The size of the subset of the keyspace that client i uses, $|S_i|$, addresses a fundamental design tradeoff—the flexibility gained through using more proxies vs. independence from the fate of other clients (obtained by not sharing proxies with other clients). Smaller proxy subsets decrease the likelihood that one client's proxies will share a proxy with some other client, but mean that a client may appear more conspicuous by sending more traffic to a smaller number of hosts. A client with smaller set of proxies is also less resilient to having proxies blocked.

To minimize the likelihood that the censor discovers a proxy and blocks it, we require that any proxy only serve a small subset of clients at any time. For a given request, the proxy must determine whether or not it should serve that

[1] We assume for simplicity that this value is constant. We describe how to relax this assumption in Section 2.3.

particular client's request for censored content. This is easily done—the proxy can simply determine the IP address from which the client request originated and check whether the computed *ProxyID* for the current time interval t matches its own *ProxyID*. The proxy only treats the client's request as a request for censored content if this value matches (regardless of whether or not it is such a request) .

The client may need to rediscover proxies when the proxies that it is using either leave the system or become unreachable; this can be done in the same way that the client bootstraps its original set of proxies, as described below.

Lookup and Initialization To use keyspace hopping to contact proxies, the client must know two things: the mapping from *ProxyID*s to IP addresses for its set of proxies, and the value for *hkey*.

To prevent a censor from harvesting IP addresses to discover and block proxies by performing arbitrary lookups, a client should only be able to lookup IP addresses for *ProxyID*s that it will use for keyspace hopping. The first proxy that a client contacts will return the *ProxyID* to IP address mappings for *only the ProxyIDs that the client needs to know*. Because the proxy that the client originally contacts knows the client's IP address and the value of *hkey* that it assigned to that client, the proxy also knows the set of proxies that the client will attempt to contact. To make it more difficult for a censor to assume the identities of many clients (e.g., by changing IP addresses, etc.), the proxy can make this discovery more expensive for the client by encrypting the mapping and forcing the client to solve a puzzle to recover the mapping, such as a partial brute force decryption given only k bits of an n-bit encryption key [6]. To ensure that the client cannot bias its keyspace assignment, *hkey* must be set by a proxy, rather than chosen by the client. The proxy that assigns *hkey* must also inform the other proxies in the client's proxy set; if proxies are not trusted, this must be done differently, as discussed in Section 2.3.

Bootstrapping With the approach we have described above, the client must contact some proxy that will send it *hkey* and *ProxyID* mappings. However, to perform this bootstrapping operation, the client must first know the IP address of at least one proxy. This sounds like the original problem: if the censor happens to block a well-known bootstrapping proxy, new clients will not be able to discover the subset of proxies that they need for keyspace hopping. Thus, we must also ensure that clients discover their first proxy reasonably independently.

A client only needs to discover *one* proxy to bootstrap. A client might already know of an operational proxy (out-of-band), or clients could establish a web of trust (as in PGP [8]) where a client could divulge one of the proxies in its subset to trusted friends. To prevent out-of-band discovery, a proxy should bootstrap a client that it has never seen before only with a certain probability. Alternatively, a proxy might bootstrap only clients that are referred to it explicitly by clients that it already knows.

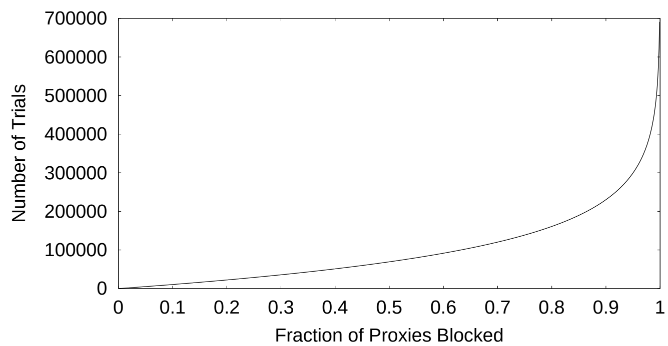

Fig. 4. Blocking 95% of 10^5 proxies would requires the censor to solve about 300,000 puzzles.

2.3 Analysis and Discussion

In this section, we analyze how well keyspace hopping resists discovery and blocking. We also discuss the deniability properties of keyspace hopping.

In-Band Discovery We analyze the likelihood that a given client will be denied access to any proxy in its subset of known proxies, given that a censor has the capability to impersonate a certain number of clients in a reasonable amount of time. In the bootstrapping phase, each client discovers a specific set of proxies based on some client-specific identifier (e.g., its IP address). Since the censor controls all of the IP address space behind the censorship firewall, it can impersonate any IP address behind its firewall to discover what set of proxies a client from that IP address might use.

Because a client cannot discover the proxies in its subset before solving a puzzle, a censor must solve one of these puzzles for each subset of proxies that it wants to discover. However, because each legitimate client will only have to solve the puzzle once, the puzzle can be sufficiently difficult (a draconian approach would require each client to spend a week decrypting its subset of proxies).

How many clients does the censor need to impersonate to know about a significant fraction of all proxies? Let's assume for simplicity that each puzzle allows the censor to discover one proxy selected randomly from P (i.e., $|S_i| = 1$). If the censor already knows about n proxies, then the probability of discovering a new proxy is $(P - n)/P$. Thus, assuming an independent Bernoulli process, the censor will discover a new proxy after impersonating $P/(P - n)$ clients, or $1/(P - n)$ of the total number of proxies. On average, the censor will have discovered N proxies after $\sum_{k=1}^{N} P/(P-k)$ impersonations (this is known as the "coupon collector problem"). For example, if $P = 10^5$, then a censor must solve about 70,000 puzzles to discover 50% of the proxies, and about 300,000 puzzles to discover 95% of all proxies.[2] Figure 4 shows the relationship for $P = 10^5$—it becomes increasingly hard for the censor to discover and block all proxies, or even a large fraction of them. If we design the system so that it is difficult enough to solve each puzzle (e.g., a day per puzzle), then it will take the censor almost 200 years to discover half of the proxies. If the system is able to detect that it is being scanned by a censor, it can also increase the difficulty of the client puzzles to slow the censor down even further.

If a censor can operate a proxy, it can discover clients by determining which clients make requests for censored content. This problem arises because the proxy can identify clients solely based on which hosts are contacting it with meaningful requests. To address this, the proxy functionality can be decomposed into an untrusted *messenger* and a trusted *portal*, where only trusted portals should be able to determine which hosts request censored content (we describe this approach in detail in Section 3). In this case, the censor can operate a malicious messenger, but that messenger will not be able to distinguish anti-censorship requests from innocent messages; this is particularly true in the case of Infranet, where innocent clients will be sending HTTP requests to that messenger under normal circumstances. For this technique to be effective with other anti-censorship systems, there must be an innocent reason for a client to send messages to that messenger; otherwise, there is no plausible deniability for sending messages through that messenger.

The requirement that the proxy know *ProxyID* to IP address mappings and IP address to *hkey* mappings is problematic because this implies that the censor can discover other proxies by becoming a proxy and can discover clients by discovering *hkey* mappings. Of course, proxies can inject a large number of false IP address to *hkey* mappings; however, a better solution uses untrusted messengers and trusted portals to control who knows this information. Trusted portals can assign *ProxyID* to IP address mappings to clients. In this case, portals can inform messengers about which clients it should serve during any time slot without requiring messengers to ever learn of other messengers. Portals can also tell messengers about only the *hkey* mappings for clients that have that messenger in its set. Portals can also simplify proxy assignment, since they can inform clients about changing values of $|P|$ or simply compute the keyspace subset S_i for each client i, using the current value of $|P|$. We discuss messengers in further detail in Section 3.

Out-of-Band Discovery A censor can discover clients out-of-band by watching for traffic anomalies (e.g., a client sending messages to a specific set of hosts outside the firewall) and can discover proxies out-of-band by probing for behavior typical of a proxy (e.g., serving visible HTTP requests more slowly than

[2] Recent studies suggest that the number of Web servers on the Internet is on the order of 10^7 and growing [7]; having 1% of these act as proxies is a reasonable goal.

a normal Web server, in the case of Infranet). Keyspace hopping makes out-of-band discovery more difficult because, given an arbitrary message from the censor, the proxy will ignore the censor's request.

A censor could mount out-of-band discovery by computing the sequence of proxies that a client would use to serve its requests and determining whether any clients send messages to proxies according to the same schedule. For this reason, the *ProxyID* for a particular client and time interval must depend not only on the client's IP address, but also some key *hkey* that is known only to the client and its set of proxies S_i. Thus, if the censor does not know *hkey*, it does not know either the keyspace for that client, nor does it know the progression of proxies that the client will take through that keyspace.

In the case where the censor operates at least one of the proxies in S_i, the censor knows all the information to hypothesize that a certain host might be operating an anti-censorship client. A simple solution relaxes frequency hopping to allow the client to pass requests to any of the *ProxyID*s that were valid for the n most recent time intervals. However, this still allows the censor to ascertain that a suspected client is contacting machines that are within the client's proxy set S_i. Another solution is to distribute a set of *hkey*s to the client and allow the client to send messages on any one of multiple channels. The censor would then have to know the secrets for all of these channels to successfully track the client through a series of proxies.

Deniability Infranet explicitly tries to make a client's requests for censored content as similar as possible to normal-looking Web traffic; we would like to preserve such deniability when incorporating keyspace hopping. Other anti-censorship systems do not provide deniability at all, so there is no risk of compromising client deniability in these cases.

Keyspace hopping presents several potential vulnerabilities that might compromise the client deniability goals of anti-censorship systems such as Infranet [3]. First, a client may arouse suspicion by attempting to contact a recently-blocked proxy. However, this weakness is no worse than that which exists in the original Infranet design. Second, the hopping sequence between proxies must be chosen so that both the hopping interval and the proxies between the client hops seems like a reasonable browsing pattern for a normal user. Because the keyspace hopping schedule we have presented does not rely on client-specific time intervals, a censor could potentially single out anti-censorship clients by correlating browsing request patterns with other known anti-censorship clients. Designing a hopping schedule that does not arouse suspicion is a challenging problem for future work.

Infranet clients should make visible requests to proxies in a way that preserves client deniability. Keyspace hopping does not affect this aspect of client deniability since it only affects how a client hops between proxies (i.e., between "responders", in Infranet parlance), not the visible requests the client makes to any particular responder.

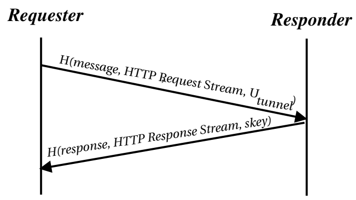

Fig. 5. Without messengers (original design).

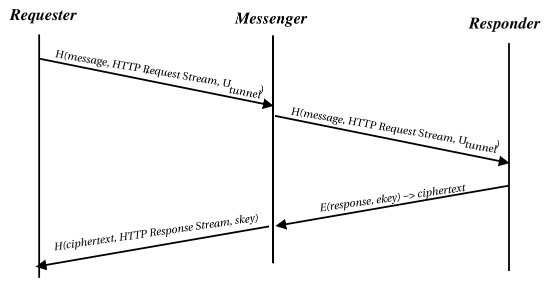

Fig. 6. With messengers.

3 Communication through Untrusted Messengers

With the keyspace hopping technique that we described in Section 2, the client cannot verify the trustworthiness of every proxy that it contacts. In this section, we describe how to rectify this problem. Specifically, we decompose the functions of the proxy into two distinct modules: the *messenger*, which acts as the gateway, or access point, to the Internet, and the *portal*, which deciphers and serves clients' requests, as shown in Figure 2. The *messenger* acts as untrusted intermediary through which the client and portal communicate.[3]

Because traffic passes through the messenger in the same way that it passes through the censor, a messenger can mount every attack that a censor can mount (as outlined in previous work [3]); in addition, the messenger can disrupt communication between the client and the proxy by failing to deliver packets to the intended destination. In this section, we describe how an untrusted messenger can be implemented in the context of both Infranet and SSL-based systems.

3.1 Infranet Messenger

The original Infranet design proposes that clients circumvent censors by sending requests via an Infranet *requester*, which hides requests for censored content in a visible HTTP requests to the Web site of the Infranet *responder*, as shown in Figure 5. In the case of Infranet, the responder acts as the portal. To separate forwarding messages from decoding messages and serving requests, we use two separate entities: the Infranet *messenger*, which is the machine that the client directly communicates with; and the Infranet *responder*, which uses the same upstream and downstream modulation techniques as before. The messenger simply acts as a conduit for the requester and responder's messages. We present an overview of the necessary modifications to the original Infranet requester/responder protocol and follow with a discussion on the implications of the Infranet messenger on deniability and other security properties of Infranet [3].

[3] This differs from the Triangle Boy approach, where the messengers (i.e., Triangle Boy nodes) must be (1) widely announced and (2) trusted (since they are intermediaries in the SSL handshake with the Safeweb server).

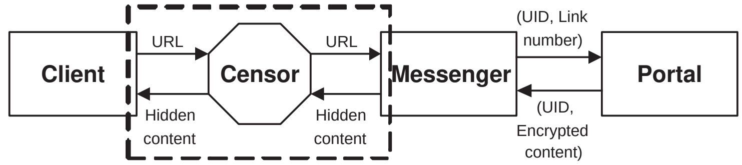

Fig. 7. An improved architecture separates the forwarding and decoding of hidden messages in both directions. This allows a potentially untrusted messenger to service requests and serve hidden content. The UID serves to demultiplex requesters.

Overview Figure 7 shows the Infranet architecture with the separation of the Infranet proxy into a messenger and responder (i.e., portal). The Infranet responder functions as before and assumes responsibility for translating the Infranet requester's visible HTTP requests into requests for censored content. The messenger informs the responder about the visible HTTP requests of certain clients (e.g., from Section 2, those which should be hopping to its *ProxyID*), and hides the appropriate encrypted content in its HTTP responses for the appropriate users. Figures 5 and 6 show the conceptual distinction between the two versions of the Infranet protocol with and without the messenger. For simplicity, we discuss the comparison for steady-state communication only.

Without Infranet messengers, the requester hides a message using the hiding function $\mathcal{H}$ and an upstream modulation function $\mathcal{U}_{tunnel}$ known only to the requester and responder. In the downstream direction, the responder hides the requested content with the downstream hiding function (e.g., steganography), using a secret hiding key *skey* known only to the requester and responder. Using untrusted messengers does not affect upstream hiding; the messenger simply tells the responder which request was made by a particular requester, but the message remains hidden, as far as the messenger is concerned. The output of the upstream hiding function is an HTTP request stream, and it suffices for the messenger to pass this request sequence directly to the responder. Note that this is an HTTP request stream for objects on the *messenger's* Web site, which the responder can then decode into message fragments (as described in previous work [3]). The responder no longer needs to run a Web site, although the messenger must do so. Only the requester and responder understand the semantics of the visible HTTP request stream.

The downstream communication protocol is similar to that proposed in the original Infranet design, except that *two* keys must be used in the downstream direction. In the original design, the Infranet responder encrypts and steganographically embeds the requested content in images from its Web site that it would subsequently serve to the client. In this case, the responder can use the same key to encrypt and steganographically embed the content.[4] However, because the messenger is not necessarily a trusted entity (i.e., it could in fact be

[4] Technically, encryption of the content to be hidden is a part of the steganographic embedding [10], but we mention both operations separately for clarity.

a malicious node), the responder must first separately encrypt the requested content *under a key that is unknown to the messenger*. However, the messenger must steganographically embed the requested content in its own visible HTTP responses, and thus needs a separate key to do so. Of course, the requester must know both of these keys to successfully retrieve the hidden content; the responder can generate these keys and send them to the requester as in the original Infranet design. Thus, the three parties must now come to agreement on two keys: an **encryption key**, *ekey*, that is shared between the requester and responder, and is unknown to the messenger; and a **hiding key**, *skey*, which the requester and the messenger must know, and the responder may also know.

Analysis and Discussion Infranet provides client deniability disguising client request stream as a user's "normal" browsing pattern because the requester's browsing pattern is determined in the same manner as before by an upstream modulation function as agreed upon by the requester and responder. Because introduction of a messenger does not affect the requester's use of an upstream modulation function, the stream of visible of HTTP requests and responses still looks innocuous to any entity that does not know the upstream modulation function. Solely based on seeing the HTTP request stream from a client, the messenger has no more knowledge about whether a client is an Infranet requester or an innocent client; only the responder knows how to map this request stream to the requester's hidden message.

In the downstream direction, the messenger knows that it is embedding ciphertext in one of its images and returning that ciphertext to some client. However, it does *not* know 1.) whether that ciphertext contains any useful data or what that data might be, or 2.) if that ciphertext corresponds to a request made by a particular client. A responder could return bogus content for clients that are not Infranet requesters without the messenger's knowledge.

Because we have separated the process of forwarding messages from decoding these messages, a requester does not need to trust the messengers, but it still needs to trust the responder. This separation allows the identity of responders to be widely published, since a censor's knowledge about the identities of Infranet responders does not enable it to block a client's access to the messengers. Thus, requesters can pass messages through a set of untrusted messengers (which, as we know from Section 2, can be made resistant to complete discovery and blockage) to well-known, trusted Infranet responders.

While a malicious messenger cannot distinguish clients that are making requests for censored content from ordinary Web clients (since only Infranet requesters and responders know whether the visible HTTP request stream has any hidden semantics), it can certainly disrupt the communication between the requester and responder by refusing to pass some messages or message fragments from the requester to the responder, and vice versa. For example, the messenger may fail to pass some URLs that it hears from a requester along to the responder; alternatively, it might neglect to embed certain pieces of content in responses to the requester. These are the same types of attacks that the censor can perform itself at the firewall; previous work provides detailed discussion about how to handle these types of attacks [3]. The client can easily detect these types of attacks—either the responder will serve an incomplete or wrong request, or the requester will not receive the full data that it requested. Presumably, this messenger could then be marked as malicious, faulty, or misbehaving, and removed from the set of candidate messengers.

3.2 Messengers for SSL-Based Systems

Other existing systems, including Safeweb/Triangle Boy [11], Zero Knowledge's WebSecure [13], and Peekabooty [15], use an encrypted channel between the client and the proxy to send requests and receive censored content. Although these systems do not provide *covert* censorship circumvention (SSL is vulnerable to fingerprinting attacks, for one [5, 12]), these systems nevertheless potentially allow clients to circumvent censorship techniques using one or more proxies. Nevertheless, these SSL-based proxy systems can also benefit by separating the proxy into a messenger and a portal, which would allow them to use the messenger discovery techniques described in Section 2.

It might seem that we could use a messenger as a conduit for an SSL connection in the same way that was possible for the Infranet messenger. In fact, SSL-based proxies are less amenable to the separation of the proxy into a messenger and a portal—traffic must appear to originate from the messenger, but the SSL handshake includes a step whereby the portal returns to the client a certificate with the portal's public key. Using this naive approach, these systems cannot attain the same level of resistance to blocking that a system that is not based on SSL can achieve. Any modifications to the SSL protocol itself (e.g., removing this portion of the handshake, etc.) would also arouse suspicion from the censor, which we would like to avoid.

Using onion-routing to tunnel the initial SSL handshake results in connection establishment that does not require suspicious modifications to SSL and is more robust to the presence of untrusted messengers [14]. For example, with knowledge of a messenger's public key, a client can encrypt its half of the SSL handshake with the messenger's public key, and the messenger can unwrap this and send it to the portal. The messenger must also establish the equivalent of a reply block, so that the messenger can send the portal's half of the SSL handshake encrypted back to the client. Using the discovery mechanisms proposed in Section 2, however, it is not possible for the client to trust the messenger's public key. To achieve greater assurance that these untrusted messengers will not compromise the client's confidentiality, the client can specify that the initial handshake be routed through multiple messengers. An alternative approach would be to use Tarzan [4] to establish the initial SSL handshake, or even to conduct the entire communication over Tarzan.

4 Conclusion

We have presented the *proxy discovery problem*, which is faced by every proxy-based anti-censorship system: how can clients discover the proxies that will assist them in gaining access to censored information without having the censor discover and block these proxies? Because a censor can discover proxies both in-band (by becoming a client itself) and out-of-band (by actively scanning for proxies, or by noticing odd traffic patterns between clients and suspected proxies), our techniques ensure that it is difficult for a censor to discover more than a small subset of all proxies using either method. We have proposed *keyspace hopping*, which defends against both in-band and out-of-band widespread discovery by any one client. Because each client selects its proxy from a set determined by client-specific information that is not easily forged (i.e., the client's IP network), it is difficult for any one client to discover a large set of proxies. In addition, proxies are configured to "hop" with clients, so each one will only act as a proxy for some small subset of clients at any given time.

Because keyspace hopping does not allow clients to choose specific proxies, clients must use untrusted hosts as gateways to the uncensored Internet. To remedy this problem, we have separated the functions of the proxy into two distinct components—an untrusted *messenger*, which clients discover through keyspace hopping and only serve to pass along clients' hidden messages to *portals*, widely-known and trusted hosts with which clients communicate to request and retrieve censored content. Although messengers have the ability to disrupt communication between clients and portals, messengers cannot distinguish anti-censorship clients from innocuous clients.[5] This separation also allows the identities and public keys of portals to be widely-published, since knowledge of these hosts does not allow a censor to block access to messengers.

This paper presents many possibilities for future work. We intend to develop a prototype of our proposed designs for use with Infranet. Designing a keyspace hopping sequence that more closely mimics the habits of normal browsing remains an open question. The proxy discovery problem mirrors the structure of "leaderless resistance" social networks, which are composed of small, independently-operating sets and are robust to infiltration by disruptive agents [2]; we may gain insight into the proxy discovery problem by studying the structure of these networks more closely.

Acknowledgments

We are grateful to David Andersen for many helpful discussions and for the suggestion of using client puzzles. Thanks also to Jean Camp and Daniel Rubenstein for thoughtful discussions, and to Sameer Ajmani, Kevin Fu, Stuart Schechter, and the anonymous reviewers for comments on drafts of this paper.

[5] In Infranet, an innocuous client is an ordinary Web client. For SSL-based schemes, an innocuous client would be an onion-routing or Tarzan client.

References

1. Anonymizer. `http://www.anonymizer.com/`.
2. Louis Beam. Leaderless resistance. `http://www.louisbeam.com/leaderless.htm`, February 1992.
3. Nick Feamster, Magdalena Balazinska, Greg Harfst, Hari Balakrishnan, and David Karger. Infranet: Circumventing Web censorship and surveillance. In *Proceedings of the 11th USENIX Security Symposium*, San Francisco, CA, August 2002.
4. Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, Washington, D.C., November 2002.
5. A. Hintz. Fingerprinting websites using traffic analysis. In *Workshop on Privacy Enhancing Technologies*, San Francisco, CA, April 2002.
6. A. Juels and J. Brainard. Client puzzles: A cryptographic defense against connection depletion attacks. In *Proceddings of the Network and Distributed System Security Symposium (NDSS'99)*, San Diego, CA, February 1999.
7. Netcraft web server survey. `http://www.netcraft.com/survey/`, 2003.
8. PGP FAQ. `http://www.faqs.org/faqs/pgp-faq/`.
9. J. Proakis and M. Salehi. *Communication System Engineering.* Prentice-Hall, Englewood Cliffs, NJ, 1994.
10. N. Provos. Defending against statistical steganalysis. In *Proceedings of the 10th USENIX Security Symposium*, Washington, D.C., August 2001.
11. SafeWeb. `http://www.safeweb.com/`.
12. Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkat Padmanabhan, and Lili Qiu. Statistical identification of encrypted Web browsing traffic. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 2002.
13. Zero-Knowledge Systems. Freedom WebSecure. `http://www.freedom.net/products/websecure/`.
14. Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous connections and onion routing. In *Proceedings of the 18th Annual Symposium on Security and Privacy*, Oakland, CA, May 1997.
15. The Cult of the Dead Cow (cDc). Peekabooty. `http://www.vnunet.com/News/1121286`.