Chipping Away at Censorship Firewalls with User-Generated Content

Sam Burnett, Nick Feamster, and Santosh Vempala
School of Computer Science, Georgia Tech
{*sburnett, feamster, vempala*}*@cc.gatech.edu*

Abstract

Oppressive regimes and even democratic governments restrict Internet access. Existing anti-censorship systems often require users to connect through proxies, but these systems are relatively easy for a censor to discover and block. This paper offers a possible next step in the censorship arms race: rather than relying on a single system or set of proxies to circumvent censorship firewalls, we explore whether the vast deployment of sites that host user-generated content can breach these firewalls. To explore this possibility, we have developed Collage, which allows users to exchange messages through hidden channels in sites that host user-generated content. Collage has two components: a message vector layer for embedding content in cover traffic; and a rendezvous mechanism to allow parties to publish and retrieve messages in the cover traffic. Collage uses user-generated content (*e.g.*, photo-sharing sites) as "drop sites" for hidden messages. To send a message, a user embeds it into cover traffic and posts the content on some site, where receivers retrieve this content using a sequence of tasks. Collage makes it difficult for a censor to monitor or block these messages by exploiting the sheer number of sites where users can exchange messages and the variety of ways that a message can be hidden. Our evaluation of Collage shows that the performance overhead is acceptable for sending small messages (*e.g.*, Web articles, email). We show how Collage can be used to build two applications: a direct messaging application, and a Web content delivery system.

1 Introduction

Network communication is subject to censorship and surveillance in many countries. An increasing number of countries and organizations are blocking access to parts of the Internet. The Open Net Initiative reports that 59 countries perform some degree of filtering [36]. For example, Pakistan recently blocked YouTube [47]. Content deemed offensive by the government has been blocked in Turkey [48]. The Chinese government regularly blocks activist websites [37], even as China has become the country with the most Internet users [19]; more recently, China has filtered popular content sites such as Facebook and Twitter, and even require their users to register to visit certain sites [43]. Even democratic countries such as the United Kingdom and Australia have recently garnered attention with controversial filtering practices [35, 54, 55]; South Korea's president recently considered monitoring Web traffic for political opposition [31].

Although existing anti-censorship systems—notably, onion routing systems such as Tor [18]—have allowed citizens some access to censored information, these systems require users outside the censored regime to set up infrastructure: typically, they must establish and maintain proxies of some kind. The requirement for running fixed infrastructure outside the firewall imposes two limitations: (1) a censor can discover and block the infrastructure; (2) benevolent users outside the firewall must install and maintain it. As a result, these systems are somewhat easy for censors to monitor and block. For example, Tor has recently been blocked in China [45]. Although these systems may continue to enjoy widespread use, this recent turn of events does beg the question of whether there are fundamentally new approaches to advancing this arms race: specifically, we explore whether it is possible to circumvent censorship firewalls with infrastructure that is more pervasive, and that does not require individual users or organizations to maintain it.

We begin with a simple observation: countless sites allow users to upload content to sites that they do not maintain or own through a variety of media, ranging from photos to blog comments to videos. Leveraging the large number of sites that allow users to upload their own content potentially yields many small cracks in censorship firewalls, because there are many different types of me-

dia that users can upload, and many different sites where they can upload it. The sheer number of sites that users could use to exchange messages, and the many different ways they could hide content, makes it difficult for a censor to successfully monitor and block all of them.

In this paper, we design a system to circumvent censorship firewalls using different types of user-generated content as cover traffic. We present Collage, a method for building message channels through censorship firewalls using user-generated content as the cover medium. Collage uses existing sites to host user-generated content that serves as the cover for hidden messages (*e.g.*, photo-sharing, microblogging, and video-sharing sites). Hiding messages in photos, text, and video across a wide range of host sites makes it more difficult for censors to block all possible sources of censored content. Second, because the messages are hidden in other seemingly innocuous messages, Collage provides its users some level of deniability that they do not have in using existing systems (*e.g.*, accessing a Tor relay node immediately implicates the user that contacted the relay). We can achieve these goals with minimal out-of-band communication.

Collage is not the first system to suggest using covert channels: much previous work has explored how to build a covert channel that uses images, text, or some other media as cover traffic, sometimes in combination with mix networks or proxies [3, 8, 17, 18, 21, 38, 41]. Other work has also explored how these schemes might be broken [27], and others hold the view that message hiding or "steganography" can never be fully secure. Collage's new contribution, then, is to design covert channels based on user-generated content and imperfect message-hiding techniques in a way that circumvents censorship firewalls that is robust enough to allow users to freely exchange messages, even in the face of an adversary that may be looking for such suspicious cover traffic.

The first challenge in designing Collage is to develop an appropriate *message vector* for embedding messages in user-generated content. Our goal for developing a message vector is to find user-generated traffic (*e.g.*, photos, blog comments) that can act as a cover medium, is widespread enough to make it difficult for censors to completely block and remove, yet is common enough to provide users some level of deniability when they download the cover traffic. In this paper, we build message vectors using the user-generated photo-sharing site, Flickr [24], and the microblogging service, Twitter [49], although our system in no way depends on these particular services. We acknowledge that some or all of these two specific sites may ultimately be blocked in certain countries; indeed, we witnessed that parts of Flickr were already blocked in China when accessed via a Chinese proxy in January 2010. A main strength of Collage's design is that blocking a specific site or set of sites will not fully stem the flow of information through the firewall, since users can use so many sites to post user-generated content. We have chosen Flickr and Twitter as a proof of concept, but Collage users can easily use domestic equivalents of these sites to communicate using Collage.

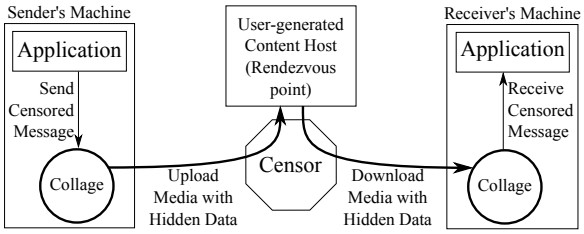


Figure 1: Collage's interaction with the network. See Figure 2 for more detail.

Given that there are necessarily many places where one user might hide a message for another, the second challenge is to agree on *rendezvous* sites where a sender can leave a message for a receiver to retrieve. We aim to build this *message layer* in a way that the client's traffic looks innocuous, while still preventing the client from having to retrieve an unreasonable amount of unwanted content simply to recover the censored content. The basic idea behind rendezvous is to embed message segments in enough cover material so that it is difficult for the censor to block all segments, even if it joins the system as a user; and users can retrieve censored messages without introducing significant deviations in their traffic patterns. In Collage, senders and receivers agree on a common set of network locations where any given content should be hidden; these agreements are established and communicated as "tasks" that a user must perform to retrieve the content (*e.g.*, fetching a particular URL, searching for content with a particular keyword). Figure 1 summarizes this process. Users send a message with three steps: (1) divide a message into many erasure-encoded "blocks" that correspond to a task, (2) embed these blocks into user-generated content (*e.g.*, images), and (3) publish this content at user-generated content sites, which serve as rendezvous points between senders and receivers. Receivers then retrieve a subset of these blocks to recover the original message by performing one of these tasks.

This paper presents the following contributions.

- We present the design and implementation of Collage, a censorship-resistant message channel built using user-generated content as the cover medium. An implementation of the Collage message channel is publicly available [13].
- We evaluate the performance and security of Collage. Collage does impose some overhead, but the overhead is acceptable for small messages (*e.g.*, ar-

ticles, emails, short messages), and Collage's overhead can also be reduced at the cost of making the system less robust to blocking.

- We present Collage's general message-layer abstraction and show how this layer can serve as the foundation for two different applications: Web publishing and direct messaging (*e.g.*, email). We describe and evaluate these two applications.

The rest of this paper proceeds as follows. Section 2 presents related work. In Section 3, we describe the design goals for Collage and the capabilities of the censor. Section 4 presents the design and implementation of Collage. Section 5 evaluates the performance of Collage's messaging layer and applications. Section 6 describes the design and implementation of two applications that are built on top of this messaging layer. Section 7 discusses some limitations of Collage's design and how Collage might be extended to cope with increasingly sophisticated censors. Section 8 concludes.

2 Background and Related Work

We survey other systems that provide anonymous, confidential, or censorship-resistant communication. We note that most of these systems require setting up a dedicated infrastructure of some sort, typically based on proxies. Collage departs significantly from this approach, since it leverages existing infrastructure. At the end of this section, we discuss some of the challenges in building covert communications channels using existing techniques, which have also been noted in previous work [15].

Anonymization proxies. Conventional anti-censorship systems have typically consisted of simple Web proxies. For example, Anonymizer [3] is a proxy-based system that allows users to connect to an anonymizing proxy that sits outside a censoring firewall; user traffic to and from the proxy is encrypted. These types of systems provide confidentiality, but typically do not satisfy any of the other design goals: for example, the existence of any encrypted traffic might be reason for suspicion (thus violating deniability), and a censor that controls a censoring firewall can easily block or disrupt communication once the proxy is discovered (thus violating resilience). A censor might also be able to use techniques such as SSL fingerprinting or timing attacks to link senders and receivers, even if the underlying traffic is encrypted [29]. Infranet attempts to create deniability for clients by embedding censored HTTP requests and content in HTTP traffic that is statistically indistinguishable from "innocuous" HTTP traffic [21]. Infranet improves deniability, but it still depends on cooperating proxies outside the firewall that might be discovered and blocked by censors. Collage improves availability by leveraging the large number of user-generated content sites, as opposed to a relatively smaller number of proxies.

One of the difficult problems with anti-censorship proxies is that a censor could also discover these proxies and block access to them. Feamster *et al.* proposed a proxy-discovery method based on frequency hopping [22]. Kaleidoscope is a peer-to-peer overlay network to provide users robust, highly available access to these proxies [42]. This system is complementary to Collage, as it focuses more on achieving availability, at the expense of deniability. Collage focuses more on providing users deniability and preventing the censor from locating all hosts from where censored content might be retrieved.

Anonymous publishing and messaging systems. CovertFS [5] is a file system that hides data in photos using steganography. Although the work briefly mentions challenges in deniability and availability, it is easily defeated by many of the attacks discussed in Section 7. Furthermore, CovertFS could in fact be implemented using Collage, thereby providing the design and security benefits described in this paper.

Other existing systems allow publishers and clients to exchange content using either peer-to-peer networks (Freenet [12]) or using a storage system that makes it difficult for an attacker to censor content without also removing legitimate content from the system (Tangler [53]). Freenet provides anonymity and unlinkability, but does not provide deniability for users of the system, nor does it provide any inherent mechanisms for resilience: an attacker can observe the messages being exchanged and disrupt them in transit. Tangler's concept of document entanglement could be applied to Collage to prevent the censor from discovering which images contain embedded information.

Anonymizing mix networks. Mix networks (*e.g.*, Tor [18], Tarzan [25], Mixminion [17]) offer a network of machines through which users can send traffic if they wish to communicate anonymously with one another. Danezis and Dias present a comprehensive survey of these networks [16]. These systems also attempt to provide unlinkability; however, previous work has shown that, depending on its location, a censor or observer might be able to link sender and receiver [4, 6, 23, 33, 39, 40]. These systems also do not provide deniability for users, and typically focus on anonymous point-to-point communication. In contrast, Collage provides a deniable means for asynchronous point-to-point communication. Finally, mix networks like Tor traditionally use a pub-

lic relay list which is easily blocked, although work has been done to try to rectify this [44, 45].

Message hiding and embedding techniques. Collage relies on techniques that can embed content into cover traffic. The current implementation of Collage uses an image steganography tool called `outguess` [38] for hiding content in images and a text steganography tool called SNOW [41] for embedding content in text. We recognize that steganography techniques offer no formal security guarantees; in fact, these schemes can and have been subject to various attacks (*e.g.*, [27]). Danezis has also noted the difficulty in building covert channels with steganography alone [15]: not only can the algorithms be broken, but also they do not hide the identities of the communicating parties. Thus, these functions must be used as components in a larger system, not as standalone "solutions". Collage relies on the embedding functions of these respective algorithms, but its security properties do not hinge solely on the security properties of any single information hiding technique; in fact, Collage could have used watermarking techniques instead, but we chose these particular embedding techniques for our proof of concept because they had readily available, working implementations. One of the challenges that Collage's design addresses is how to use imperfect message hiding techniques to build a message channel that is both available and offers some amount of deniability for users.

3 Problem Overview

We now discuss our model for the censor's capabilities and our goals for circumventing a censor who has these capabilities. It is difficult, if not impossible, to fully determine the censor's current or potential capabilities; as a result, Collage cannot provide formal guarantees regarding success or deniability. Instead, we present a model for the censor that we believe is more advanced than current capabilities and, hence, where Collage is *likely* to succeed. Nevertheless, censorship is an arms race, so as the censor's capabilities evolve, attacks against censorship firewalls will also need to evolve in response. In Section 7, we discuss how Collage's could be extended to deal with these more advanced capabilities as the censor becomes more sophisticated.

We note that although we focus on censors, Collage also depends on content hosts to store media containing censored content. Content hosts currently do not appear to be averse to this usage (*e.g.*, to the best of our knowledge, Collage does not violate the Terms of Service for either Flickr or Twitter), although if Collage were to become very popular this attitude would likely change. Although we would prefer content hosts to willingly serve Collage content (*e.g.*, to help users in censored regimes), Collage can use many content hosts to prevent any single host from compromising the entire system.

3.1 The Censor

We assume that the censor wishes to allow *some* Internet access to clients, but can monitor, analyze, block, and alter subsets of this traffic. We believe this assumption is reasonable: if the censor builds an entirely separate network that is partitioned from the Internet, there is little we can do. Beyond this basic assumption, there is a wide range of capabilities we can assume. Perhaps the most difficult aspect of modeling the censor is figuring out how much effort it will devote to capturing, storing, and analyzing network traffic. Our model assumes that the censor can deploy monitors at multiple network egress points and observe all traffic as it passes (including both content and headers). We consider two types of capabilities: targeting and disruption.

Targeting. A censor might *target* a particular user behind the firewall by focusing on that user's traffic patterns; it might also target a particular suspected content host site by monitoring changes in access patterns to that site (or content on that site). In most networks, a censor can monitor all traffic that passes between its clients and the Internet. Specifically, we assume the censor can eavesdrop any network traffic between clients on its network and the Internet. A censor's motive in passively monitoring traffic would most likely be either to determine that a client was using Collage or to identify sites that are hosting content. To do so, the censor could monitor traffic *aggregates* (*i.e.*, traffic flow statistics, like NetFlow [34]) to determine changes in overall traffic patterns (*e.g.*, to determine if some website or content has suddenly become more popular). The censor can also observe traffic streams from *individual* users to determine if a particular user's clickstream is suspicious, or otherwise deviates from what a real user would do. These capabilities lead to two important requirements for preserving deniability: traffic patterns generated by Collage should not skew overall distributions of traffic, and the traffic patterns generated by an individual Collage user must resemble the traffic generated by innocuous individuals.

To target users or sites, a censor might also use Collage as a sender or receiver. This assumption makes some design goals more challenging: a censor could, for example, inject bogus content into the system in an attempt to compromise message availability. It could also join Collage as a client to discover the locations of censored content, so that it could either block content outright (thus attacking availability) or monitor users who download similar sets of content (thus attacking deniability). We

also assume that the censor could act as a content publisher. Finally, we assume that a censor might be able to coerce a content host to shut down its site (an aggressive variant of actively blocking requests to a site).

Disruption. A censor might attempt to *disrupt* communications by actively mangling traffic. We assume the censor would not mangle uncensored content in any way that a user would notice. A censor could, however, inject additional traffic in an attempt to confuse Collage's process for encoding or decoding censored content. We assume that it could also block traffic at granularities ranging from an entire site to content on specific sites.

The costs of censorship. In accordance with Bellovin's recent observations [7], we assume that the censor's capabilities, although technically limitless, will ultimately be constrained by cost and effort. In particular, we assume that the censor will *not* store traffic indefinitely, and we assume that the censor's will or capability to analyze traffic prevents it from observing more complex statistical distributions on traffic (*e.g.*, we assume that it cannot perform analysis based on joint distributions between arbitrary pairs or groups of users). We also assume that the censor's computational capabilities are limited: for example, performing deep packet inspection on every packet that traverses the network or running statistical analysis against all traffic may be difficult or infeasible, as would performing sophisticated timing attacks (*e.g.*, examining inter-packet or inter-request timing for each client may be computationally infeasible or at least prohibitively inconvenient). As the censorship arms race continues, the censor may develop such capabilities.

3.2 Circumventing the Censor

Our goal is to allow users to send and receive messages across a censorship firewall that would otherwise be blocked; we want to enable users to communicate across the firewall by exchanging articles and short messages (*e.g.*, email messages and other short messages). In some cases, the sender may be behind the firewall (*e.g.*, a user who wants to publish an article from within a censored regime). In other cases, the receiver might be behind the firewall (*e.g.*, a user who wants to browse a censored website).

We aim to understand Collage's performance in real applications and demonstrate that it is "good enough" to be used in situations where users have no other means for circumventing the firewall. We therefore accept that our approach may impose substantial overhead, and we do not aim for Collage's performance to be comparable to that of conventional networked communication. Ultimately, we strive for a system that is effective and easy to use for a variety of networked applications. To this end, Collage offers a messaging library that can support these applications; Section 6 describes two example applications.

Collage's main performance requirement is that the overhead should be small enough to allow content to be stored on sites that host user-generated content and to allow users to retrieve the hidden content in a reasonable amount of time (to ensure that the system is usable), and with a modest amount of traffic overhead (since some users may be on connections with limited bandwidth). In Section 5, we evaluate Collage's storage requirements on content hosting sites, the traffic overhead of each message (as well as the tradeoff between this overhead and robustness and deniability), and the overall transfer time for messages.

In addition to performance requirements, we want Collage to be robust in the face of the censor that we have outlined in Section 3.1. We can characterize this robustness in terms of two more general requirements. The first requirement is *availability*, which says that clients should be able to communicate in the face of a censor that is willing to restrict access to various content and services. Most existing censorship circumvention systems do not prevent a censor from blocking access to the system altogether. Indeed, regimes such as China have blocked or hijacked applications ranging from websites [43] to peer-to-peer systems [46] to Tor itself [45]. We aim to satisfy availability in the face of the censor's targeting capabilities that we described in Section 3.1.

Second, Collage should offer users of the system some level of *deniability*; although this design goal is hard to quantify or formalize, informally, deniability says that the censor cannot discover the users of the censorship system. It is important for two reasons. First, if the censor can identify the traffic associated with an anti-censorship system, it can discover and either block or hijack that traffic. As mentioned above, a censor observing encrypted traffic may still be able to detect and block systems such as Tor [18]. Second, and perhaps more importantly, if the censor can identify specific users of a system, it can coerce those users in various ways. Past events have suggested that censors are able and willing to both discover and block traffic or sites associated with these systems *and* to directly target and punish users who attempt to defeat censorship. In particular, China requires users to register with ISPs before purchasing Internet access at either home or work, to help facilitate tracking individual users [10]. Freedom House reports that in six of fifteen countries they assessed, a blogger or online journalist was sentenced to prison for attempting to circumvent censorship laws—prosecutions have occurred in Tunisia, Iran, Egypt, Malaysia, and India [26]—and cites a recent event of a Chinese blogger who was recently attacked [11]. As these regimes have indicated

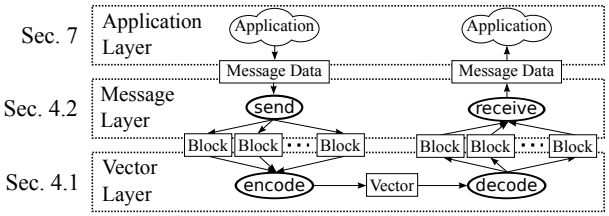


Figure 2: Collage's layered design model. Operations are in ovals; intermediate data forms are in rectangles.

their willingness and ability to monitor and coerce individual users, we believe that attempting to achieve some level of deniability is important for any anti-censorship system.

By design, a user cannot disprove claims that he engages in deniable communication, thus making it easier for governments and organizations to implicate arbitrary users. We accept this as a potential downside of deniable communications, but point out that organizations can already implicate users with little evidence (*e.g.*, [2]).

4 Collage Design and Implementation

Collage's design has three layers and roughly mimics the layered design of the network protocol stack itself. Figure 2 shows these three layers: the vector, message, and application layers. The *vector layer* provides storage for short data chunks (Section 4.1), and the *message layer* specifies a protocol for using the vector layer to send and receive messages (Section 4.2). A variety of applications can be constructed on top of the message layer. We now describe the vector and message layers in detail, deferring discussion of specific applications to Section 6. After describing each of these layers, we discuss *rendezvous*, the process by which senders and receivers find each other to send messages using the message layer (Section 4.3). Finally, we discuss our implementation and initial deployment (Section 4.4).

4.1 Vector Layer

The vector layer provides a substrate for storing short data chunks. Effectively, this layer defines the "cover media" that should be used for embedding a message. For example, if a small message is hidden in the high frequency of a video then the *vector* would be, for example, a YouTube video. This layer hides the details of this choice from higher layers and exposes three operations: encode, decode, and isEncoded. These operations encode data into a vector, decode data from an encoded vector, and check for the presence of encoded data given a secret key, respectively.

Collage imposes requirements on the choice of vector. First, each vector must have some capacity to hold encoded data. Second, the population of vectors must be large so that many vectors can carry many messages. Third, to satisfy both availability and deniability, it must be relatively easy for users to deniably send and receive vectors containing encoded chunks. Fourth, to satisfy availability, it must be expensive for the censor to disrupt chunks encoded in a vector. Any vector layer with these properties will work with Collage's design, although the deniability of a particular application will also depend upon its choice of vector, as we discuss in Section 7.

The feasibility of the vector layer rests on a key observation: *data hidden in user-generated content serves as a good vector for many applications, since it is both populous and comes from a wide variety of sources (*i.e.*, many users).* Examples of such content include images published on Flickr [24] (as of June 2009, Flickr had about 3.6 billion images, with about 6 million new images per day [28]), tweets on Twitter [49] (Twitter had about half a million tweets per day [52], and Mashable projected about 18 million Twitter users by the end of 2009 [50]), and videos on YouTube [56], which had about $200,000$ new videos per day as of March 2008 [57].

For concreteness, we examine two classes of vector encoding algorithms. The first option is *steganography*, which attempts to hide data in a cover medium such that only intended recipients of the data (*e.g.*, those possessing a key) can detect its presence. Steganographic techniques can embed data in a variety of cover media, such as images, video, music, and text. Steganography makes it easy for legitimate Collage users to find vectors containing data and difficult for a censor to identify (and block) encoded vectors. Although the deniability that steganography can offer is appealing, key distribution is challenging, and almost all production steganography algorithms have been broken. Therefore, we cannot simply rely on the security properties of steganography.

Another option for embedding messages is *digital watermarking*, which is similar to steganography, except that instead of hiding data from the censor, watermarking makes it difficult to remove the data without destroying the cover material. Data embedded using watermarking is perhaps a better choice for the vector layer: although encoded messages are clearly visible, they are difficult to remove without destroying or blocking a large amount of legitimate content. If watermarked content is stored in a lot of popular user-generated content, Collage users can gain some level of deniability simply because all popular content contains some message chunks.

We have implemented two example vector layers. The first is image steganography applied to images hosted on Flickr [24]. The second is text steganography applied to user-generated text comments on websites such as blogs,

```
send(identifier, data)
1  Create a rateless erasure encoder for data.
2  for each suitable vector (e.g., image file)
3      do
4          Retrieve blocks from the erasure coder to
               meet the vector's encoding capacity.
5          Concatenate and encrypt these blocks using
               the identifier as the encryption key.
6          encode the ciphertext into the vector.
7          Publish the vector on a user-generated
               content host such that receivers
               can find it. See Section 4.3.
```

```
receive(identifier)
 1  Create a rateless erasure decoder.
 2  while the decoder cannot decode the message
 3      do
 4          Find and fetch a vector from a
                user-generated content host.
 5          Check if the vector contains encoded
                data for this identifier.
 6          if the vector is encoded with message data
 7            then
 8                decode payload from the vector.
 9                Decrypt the payload.
10                Split the plaintext into blocks.
11                Provide each decrypted block to
                      the erasure decoder.
12  return decoded message from erasure decoder
```

Figure 3: The message layer's send and receive operations.

YouTube [56], Facebook [20], and Twitter [49]. Despite possible and known limitations to these approaches (*e.g.*, [27]), both of these techniques have working implementations with running code [38, 41]. As watermarking and other data-hiding techniques continue to become more robust to attack, and as new techniques and implementations emerge, Collage's layered model can incorporate those mechanisms. The goal of this paper is *not* to design better data-hiding techniques, but rather to build a censorship-resistant message channel that leverages these techniques.

4.2 Message Layer

The message layer specifies a protocol for using the vector layer to send and receive arbitrarily long messages (*i.e.*, exceeding the capacity of a single vector). Observable behavior generated by the message layer should be deniable with respect to the normal behavior of the user or users at large.

Figure 3 shows the send and receive operations. send encodes message data in vectors and publishes them on content hosts, while receive finds encoded vectors on content hosts and decodes them to recover the original message. The sender associates a message identifier with each message, which should be unique for an application (*e.g.*, the hash of the message). Receivers use this identifier to locate the message. For encoding schemes that require a key (*e.g.*, [38]), we choose the key to be the message identifier.

To distribute message data among several vectors, the protocol uses rateless erasure coding [9, 32], which generates a near-infinite supply of short chunks from a source message such that any appropriately-sized subset of those chunks can reassemble the original message. For example, a rateless erasure coder could take a 80 KB message and generate 1 KB chunks such that any 100-subset of those chunks recovers the original message. Step 1 of send initializes a rateless erasure encoder for generating chunks of the message; step 4 retrieves chunks from the encoder. Likewise, step 1 of receive creates a rateless erasure decoder, step 11 provides retrieved chunks to the decoder, and step 12 recovers the message.

Most of the remaining send operations are straightforward, involving encryption and concatenation (step 5), and operation of the vector layer's encode function (step 6). Likewise, receive operates the vector layer's decode function (step 8), decrypts and splits the payload (steps 9 and 10). The only more complex operations are step 7 of send and step 4 of receive, which publish and retrieve content from user-generated content hosts. These steps must ensure (1) that senders and receivers agree on locations of vectors and (2) that publishing and retrieving vectors is done in a deniable manner. We now describe how to meet these two requirements.

4.3 Rendezvous: Matching Senders to Receivers

Vectors containing message data are stored to and retrieved from user-generated content hosts; to exchange messages, senders and receivers must first *rendezvous*. To do so, senders and receivers perform sequences of *tasks*, which are time-dependent sequences of actions. An example of a sender task is the sequence of HTTP requests (*i.e.*, actions) and fetch times corresponding to "Upload photos tagged with 'flowers' to Flickr"; a corresponding receiver task is "Search Flickr for photos tagged with 'flowers' and download the first 50 images." This scheme poses many challenges: (1) to achieve deniability, all tasks must resemble observable actions completed by innocuous entities not using Collage (*e.g.*, browsing the Web), (2) senders must identify vectors suitable for each task, and (3) senders and receivers must

agree on which tasks to use for each message. This section addresses these challenges.

Identifying suitable vectors. Task deniability depends on properly selecting vectors for each task. For example, for the receiver task "search for photos with keyword *flowers*," the corresponding sender task ("publish a photo with keyword *flowers*") must be used with photos of flowers; otherwise, the censor could easily identify vectors containing Collage content as those vectors that do not match their keywords. To achieve this, the sender picks vectors with attributes (*e.g.*, associated keywords) that match the expected content of the vector.

Agreeing on tasks for a message. Each user maintains a list of deniable tasks for common behaviors involving vectors (Section 4.1) and uses this list to construct a *task database*. The database is simply a table of pairs (T_s, T_r), where T_s is a sender task and T_r is a receiver task. Senders and receivers construct pairs such that T_s publishes vectors in locations visited by T_r. For example, if T_r performs an image search for photos with keyword "flowers" then T_s would publish only photos with that keyword (and actually depicting flowers). Given this database, the sender and receiver map each message identifier to one or more task pairs and execute T_s and T_r, respectively.

The sender and receiver must agree on the mapping of identifiers to database entries; otherwise, the receiver will be unable to find vectors published by the sender. If the sender's and receiver's databases are identical, then the sender and receiver simply use the message identifier as an index into the task database. Unfortunately, the database may change over time, for a variety of reasons: tasks become obsolete (*e.g.*, Flickr changes its page structure) and new tasks are added (*e.g.*, it may be advantageous to add a task for a new search keyword during a current event, such as an election). Each time the database changes, other users need to be made aware of these changes. To this end, Collage provides two operations on the task database: add and remove. When a user receives an advertisement for a new task or a withdrawal of an existing task he uses these operations to update his copy of the task database.

Learning task advertisements and withdrawals is application specific. For some applications, a central authority sends updates using Collage's own message layer, while in others updates are sent offline (*i.e.*, separate from Collage). We discuss these options in Section 6. One feature is common to all applications: delays in propagation of database updates will cause different users to have slightly different versions of the task database, necessitating a mapping for identifiers to tasks that is robust to slight changes to the database.

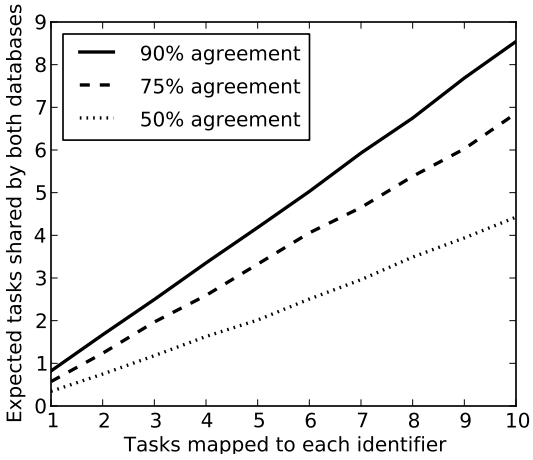


Figure 4: The expected number of common tasks when mapping the same message identifier to a task subset, between two task databases that agree on varying percentages of tasks.

To reconcile database disagreements, our algorithm for mapping message identifiers to task pairs uses *consistent hash functions* [30], which guarantee that small changes to the space of output values have minimal impact on the function mapping. We initialize the task database by choosing a pseudorandom hash function h (*e.g.*, SHA-1) and precomputing $h(t)$ for each task t. The algorithm for mapping an identifier M to a m-subset of the database is simple: compute $h(M)$ and take the m entries from the task database with precomputed hash values closest to $h(M)$; these task pairs are the mapping for M.

Using consistent hashing to map identifiers to task pairs provides an important property: updating the database results in only small changes to the mappings for existing identifiers. Figure 4 shows the expected number of tasks reachable after removing a percentage of the task database and replacing it with new tasks. As expected, increasing the number of tasks mapped for each identifier decreases churn. Additionally, even if half of the database is replaced, the sender and receiver can agree on at least one task when three or more tasks are mapped to each identifier. In practice, we expect the difference between two task databases to be around 10%, so three tasks to each identifier is sufficient. Thus, two parties with slightly different versions of the task database can still communicate messages: although some tasks performed by the receiver (*i.e.*, mapped using his copy of the database) will not yield content, most tasks will.

Choosing deniable tasks. Tasks should mimic the normal behavior of users, so that a user who is performing these tasks is unlikely to be pinpointed as a Collage

user (which, in and of itself, could be incriminating). We design task sequences to "match" those of normal visitors to user-generated content sites. Tasks for different content hosts have different deniability criteria. For example, the task of looking at photos corresponding to a popular tag or tag pair offers some level of deniability, because an innocuous user might be looking at popular images anyway. The challenge, of course, is finding sets of tasks that are deniable, yet focused enough to allow a user to retrieve content in a reasonable amount of time. We discuss the issue of deniability further in Section 7.

4.4 Implementation

Collage requires minimal modification to existing infrastructure, so it is small and self-contained, yet modular enough to support many possible applications; this should facilitate adoption. We have released a version of Collage [13].

We have implemented Collage as a 650-line Python library, which handles the logic of the message layer, including the task database, vector encoding and decoding, and the erasure coding algorithm. To execute tasks, the library uses Selenium [1], a popular web browser automation tool; Selenium visits web pages, fills out forms, clicks buttons and downloads vectors. Executing tasks using a real web browser frees us from implementing an HTTP client that produces realistic Web traffic (*e.g.*, by loading external images and scripts, storing cookies, and executing asynchronous JavaScript requests).

We represent tasks as Python functions that perform the requisite task. Table 1 shows four examples. Each application supplies definitions of operations used by the tasks (*e.g.*, `FindPhotosOfFlickrUser`). The task database is a list of tasks, sorted by their MD5 hash; to map an identifier to a set of tasks, the database finds the tasks with hashes closest to the hash of the message identifier. After mapping, receivers simply execute these tasks and decode the resulting vectors. Senders face a more difficult task: they must supply the task with a vector suitable for that task. For instance, the task "publish a photo tagged with 'flowers"' must be supplied with a photo of flowers. We delegate the task of finding vectors meeting specific requirements to a *vector provider*. The exact details differ between applications; one of our applications searches a directory of annotated photos, while another prompts the user to type a phrase containing certain words (*e.g.*, "Olympics").

5 Performance Evaluation

This section evaluates Collage according to the three performance metrics introduced in Section 3: storage overhead on content hosts, network traffic, and transfer time. We characterize Collage's performance by measuring its behavior in response to a variety of parameters. Recall that Collage (1) processes a message through an erasure coder, (2) encodes blocks inside vectors, (3) executes tasks to distribute the message vectors to content hosts, (4) retrieves some of these vectors from content hosts, and (5) decodes the message on the receiving side. Each stage can affect performance. In this section, we evaluate how each of these factors affects the performance of the message layer; Section 6 presents additional performance results for Collage applications using real content hosts.

- **Erasure coding** can recover an n-block message from $(1+\frac{\epsilon}{2})n$ of its coded message blocks. Collage uses $\epsilon = 0.01$, as recommended by [32], yielding an expected 0.5% increase in storage, traffic, and transfer time of a message.
- **Vector encoding** stores erasure coded blocks inside vectors. Production steganography tools achieve encoding rates of between 0.01 and 0.05, translating to between 20 and 100 factor increases in storage, traffic, and transfer time [38]. Watermarking algorithms are less efficient; we hope that innovations in information hiding can reduce this overhead.
- **Sender and receiver tasks** publish and retrieve vectors from content hosts. Tasks do not affect the storage requirement on content hosts, but each task can impose additional traffic and time. For example, a task that downloads images by searching for them on Flickr can incur hundreds of kilobytes of traffic before finding encoded vectors. Depending on network connectivity, this step could take anywhere from a few seconds to a few minutes and can represent an overhead of several hundred percent, depending on the size of each vector.
- **The number of executed tasks** differs between senders and receivers. The receiver performs as many tasks as necessary until it is able to decode the message; this number depends on the size of the message, the number of vectors published by the sender, disagreements between sender and receiver task databases, the dynamics of the content host (*e.g.*, a surge of Flickr uploads could "bury" Collage encoded vectors), and the number of tasks and vectors blocked by the censor. While testing Collage, we found that we needed to execute only one task for the majority of cases.

  The sender must perform as many tasks as necessary so that, given the many ways the receiver can fail to obtain vectors, the receiver will still be able to retrieve enough vectors to decode the message. In practice, this number is difficult to estimate and

Content host	Sender task	Receiver task
Flickr	PublishAsUser('User', Photo, MsgData)	FindPhotosOfFlickrUser('User')
Twitter	PostTweet('Watching the Olympics', MsgData)	SearchTwitter('Olympics')

Table 1: Examples of sender and receiver task snippets.

vectors are scarce, so the sender simply uploads as many vectors as possible.

We implemented a Collage application that publishes vectors on a simulated content host, allowing us to observe the effects of these parameters. Figure 5 shows the results of running several experiments across Collage's parameter space. The simulation sends and receives a 23 KB one-day news summary. The message is erasure coded with a block size of 8 bytes and encoded into several vectors randomly drawn from a pool for vectors with average size 200 KB. Changing the message size scales the metrics linearly, while increasing the block size only decreases erasure coding efficiency.

Figure 5a demonstrates the effect of vector encoding efficiency on required storage on content hosts. We used a fixed-size identifier-to-task mapping of ten tasks. We chose four *send rates*, which are multiples of the minimum number of tasks required to decode the message: the sender may elect to send more vectors if he believes some vectors may be unreachable by the receiver. For example, with a send rate of 10x, the receiver can still retrieve the message even if 90% of vectors are unavailable. Increasing the task mapping size may be necessary for large send rates, because sending more vectors requires executing more tasks. These results give us hope for the future of information hiding technology: current vector encoding schemes are around 5% efficient; according to Figure 5a, this a region where a significant reduction in storage is possible with only incremental improvements in encoding techniques (*i.e.*, the slope is steep).

Figure 5b predicts total sender and receiver traffic from task overhead traffic, assuming 1 MB of vector storage on the content host. As expected, blocking more vectors increases traffic, as the receiver must execute more tasks to receive the same message content. Increasing storage beyond 1 MB decreases receiver traffic, because more message vectors are available for the same blocking rate. An application executed on a real content host transfers around 1 MB of overhead traffic for a 23 KB message.

Finally, Figure 5c shows the overall transfer time for senders and receivers, given varying time overheads. These overheads are optional for both senders and receivers and impose delays between requests to evade timing analysis by the censor. For example, Collage could build a distribution of inter-request timings from the user's normal (*i.e.*, non-Collage) traffic and impose this timing distribution on Collage tasks. We simulated the total transfer time using three network connection speeds. The first (768 Kbps download and 384 Kbps upload) is a typical entry-level broadband package and would be experienced if both senders and receivers are typical users within the censored domain. The second (768/10000 Kbps) would be expected if the sender has a high-speed connection, perhaps operating as a dedicated publisher outside the censored domain; one of the applications in Section 6 follows this model. Finally, the 6000/1000 Kbps connection represents expected next-generation network connectivity in countries experiencing censorship. In all cases, reasonable delays are imposed upon transfers, given the expected use cases of Collage (*e.g.*, fetching daily news article). We confirmed this result: a 23 KB message stored on a real content host took under 5 minutes to receive over an unreliable broadband wireless link; sender time was less than 1 minute.

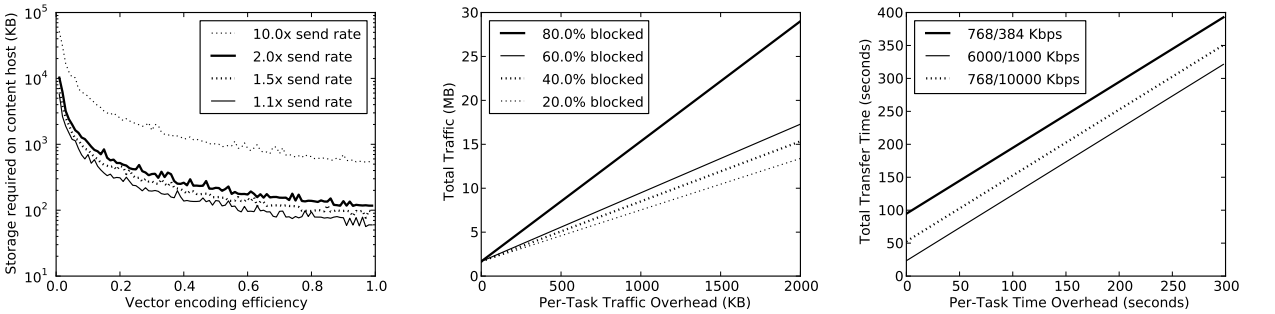
6 Building Applications with Collage

Developers can build a variety of applications using the Collage message channel. In this section, we outline requirements for using Collage and present two example applications.

6.1 Application Requirements

Even though application developers use Collage as a secure, deniable messaging primitive, they must still remain conscious of overall application security when using these primitives. Additionally, the entire vector layer and several parts of the message layer presented in Section 4 must be provided by the application. These components can each affect correctness, performance, and security of the entire application. In this section, we discuss each of these components. Table 2 summarizes the component choices.

Vectors, tasks, and task databases. Applications specify a class of vectors and a matching vector encoding algorithm (*e.g.*, Flickr photos with image steganography) based on their security and performance characteristics. For example, an application requiring strong content deniability for large messages could use a strong steganography algorithm to encode content inside of videos.



(a) Storage, for various sender redundancies

(b) Traffic, for various vector block rates

(c) Transfer time, for various network connectivity rates (download/upload)

Figure 5: Collage's performance metrics, as measured using a simulated content host.

	Web Content Proxy (Sec. 6.2)	Covert Email (Sec. 6.3)	Other options
Vectors	Photos	Text	Videos, music
Vector encoding	Image steganography	Text steganography	Video steganography, digital watermarking
Vector sources	Users of content hosts	Covert Email users	Automatic generation, crawl the Web
Tasks	Upload/download Flickr photos	Post/receive Tweets	Other user-generated content host(s)
Database distribution	Send by publisher via proxy	Agreement by users	Prearranged algorithm, "sneakernet"
Identifier security	Distributed by publisher, groups	Group key	Existing key distribution infrastructure

Table 2: Summary of application components.

Tasks are application-specific: uploading photos to Flickr is different from posting tweets on Twitter. Applications insert tasks into the task database, and the message layer executes these tasks when sending and receiving messages. The applications specify how many tasks are mapped to each identifier for database lookups. In Section 4.3, we showed that mapping each identifier to three tasks ensures that, on average, users can still communicate even with slightly out-of-date databases; applications can further boost availability by mapping more tasks to each identifier.

Finally, applications must distribute the task database. In some instances, a central authority can send the database to application users via Collage itself. In other cases, the database is communicated offline. The application's task database should be large enough to ensure diversity of tasks for messages published at any given time; if n messages are published every day, then the database should have cn tasks, where c is at least the size of the task mapping. Often, tasks can be generated programmatically, to reduce network overhead. For example, our Web proxy (discussed next) generates tasks from a list of popular Flickr tags.

Sources of vectors. Applications must acquire vectors used to encode messages, either by requiring end-users to provide their own vectors (*e.g.*, from a personal photo collection), automatically generating them, or obtaining them from an external source (*e.g.*, a photo donation system).

Identifier security. Senders and receivers of a message must agree on a message identifier for that message. This process is analogous to key distribution. There is a general tradeoff between ease of message identifier distribution and security of the identifier: if users can easily learn identifiers, then more users will use the system, but it will also be easier for the censor to obtain the identifier; the inverse is also true. Developers must choose a distribution scheme that meets the intended use of their application. We discuss two approaches in the next two sections, although there are certainly other possibilities.

Application distribution and bootstrapping. Users ultimately need a secure one-time mechanism for obtaining the application, without using Collage. A variety of distribution mechanisms are possible: clients could receive software using spam or malware as a propagation vector, or via postal mail or person-to-person exchange. There will ultimately be many ways to distribute applications without the knowledge of the censor. Other systems face the same problem [21]. This requirement does not obviate Collage, since once the user has received the software, he or she can use it to exchange an arbitrary number of messages.

To explore these design parameters in practice, we built two applications using Collage's message layer. The first

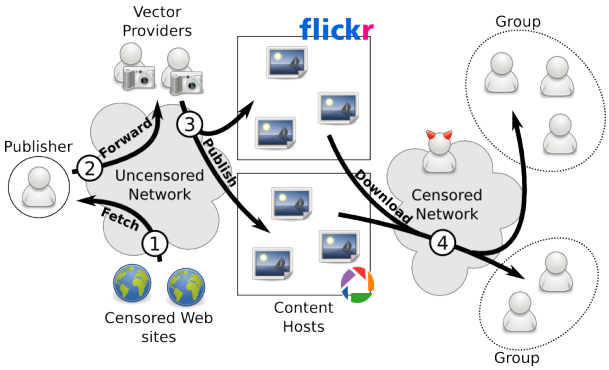


Figure 6: Proxied Web content passes through multiple parties before publication on content hosts. Each group downloads a different subset of images when fetching the same URL.

is a Web content proxy whose goal is to distribute content to many users; the second is a covert email system.

6.2 Web Content Proxy

We have built an asynchronous Web proxy using Collage's message layer, with which a publisher in an uncensored region makes content available to clients inside censored regimes. Unlike traditional proxies, our proxy shields both the identities of its users and the content hosted from the censor.

The proxy serves small Web documents, such as articles and blog posts, by steganographically encoding content into images hosted on photo-sharing websites like Flickr and Picasa. A standard steganography tool [38] can encode a few kilobytes in a typical image, meaning most hosted documents will fit within a few images. To host many documents simultaneously, however, the publisher needs a large supply of images; to meet this demand, the publisher operates a service allowing generous users of online image hosts to donate their images. The service takes the images, encodes them with message data, and returns the encoded images to their owners, who then upload them to the appropriate image hosts. Proxy users download these photos and decode their contents. Figure 6 summarizes this process. Notice that the publisher is outside the censored domain, which frees us from worrying about sender deniability.

To use a proxy, users must *discover* a publisher, *register* with that publisher, and be *notified* of an encryption key. Publishers are identified by their public key so discovering publishers is reduced to a key distribution exercise, albeit that these keys must be distributed without the suspicion of the censor. Several techniques are feasible: the key could be delivered alongside the client software, derived from a standard SSL key pair, or distributed offline. Like any key-based security system, our proxy must deal with this inherent bootstrapping problem.

Once the client knows the publisher's public key, it sends a message requesting registration. The message identifier is the publisher's public key and the message payload contains the public key of the client encrypted using the publisher's public key. This encryption ensures that only the publisher knows the client's public key. The publisher receives and decrypts the client's registration request using his own private key.

The client is now registered but doesn't know where content is located. Therefore, the publisher sends the client a message containing a *group key*, encrypted using the client's public key. The group key is shared between a small number of proxy users and is used to discover identifiers of content. For security, different groups of users fetch content from different locations; this prevents any one user from learning about (and attacking) all content available through the proxy.

After registration is complete, clients can retrieve content. To look up a URL u, a client hashes u with a keyed hash function using the group key. It uses the hash as the message identifier for receive.

Unlike traditional Web proxies, only a limited amount of content is available though our proxy. Therefore, to accommodate clients' needs for unavailable content, clients can suggest content to be published. To suggest a URL, a client sends the publisher a message containing the requested URL. If the publisher follows the suggestion, then it publishes the URL for users of that client's group key.

Along with distributing content, the publisher provides updates to the task database via the proxy itself (at the URL `proxy://updates`). The clients occasionally fetch content from this URL to keep synchronized with the publisher's task database. The consistent hashing algorithm introduced in Section 4.3 allows updates to be relatively infrequent; by default, the proxy client updates its database when 20% of tasks have been remapped due to churn (*i.e.*, there is a 20% reduction in the number of successful task executions). Figure 4 shows that there may be many changes to the task database before this occurs.

Implementation and Evaluation. We have implemented a simple version of the proxy and can use it to publish and retrieve documents on Flickr. The task database is a set of tasks that search for combinations (*e.g.*, "vacation" and "beach") of the 130 most popular tags. A 23 KB one-day news summary requires nine JPEG photos ($\approx$ 3 KB data per photo, plus encoding overhead) and takes approximately 1 minute to retrieve over a fast network connection; rendering web pages and large photos takes a significant fraction of this time. Note

that the document is retrieved immediately after publication; performance decays slightly over time because search results are displayed in reverse chronological order. We have also implemented a photo donation service, which accepts Flickr photos from users, encodes them with censored content, and uploads them on the user's behalf. This donation service is available for download [13].

6.3 Covert Email

Although our Web proxy provides censored content to many users, it is susceptible to attack from the censor for precisely this reason: because no access control is performed, the censor could learn the locations of published URLs using the proxy itself and potentially mount denial-of-service attacks. To provide greater security and availability, we present Covert Email, a point-to-point messaging system built on Collage's message layer that excludes the censor from sending or receiving messages, or observing its users. This design sacrifices scalability: to meet these security requirements, all key distribution is done out of band, similar to PGP key signing.

Messages sent with Covert Email will be smaller and potentially more frequent than for the proxy, so Covert Email uses text vectors instead of image vectors. Using text also improves deniability, because receivers are inside the censored domain, and publishing a lot of text (*e.g.*, comments, tweets) is considered more deniable than many photos. Blogs, Twitter, and comment posts can all be used to store message chunks. Because Covert Email is used between a closed group of users with a smaller volume of messages, the task database is smaller and updated less often without compromising deniability. Additionally, users can supply the text vectors needed to encode content (*i.e.*, write or generate them), eliminating the need for an outside vector source. This simplifies the design.

Suppose a group of mutually trusted users wishes to communicate using Covert Email. Before doing so, it must establish a shared secret key, for deriving message identifiers for sending and receiving messages. This one-time exchange is done out-of-band; any exchange mechanism works as long as the censor is unaware that a key exchange takes place. Along with exchanging keys, the group establishes a task database. At present, a database is distributed with the application; the group can augment its task database and notify members of changes using Covert Email itself.

Once the group has established a shared key and a task database, its members can communicate. To send email to Bob, Alice generates a message identifier by encrypting a tuple of his email address and the current date, using the shared secret key. The date serves as a salt and ensures variation in message locations over time. Alice then sends her message to Bob using that identifier. Here, Bob's email address is used only to uniquely identify him within the group; in particular, the domain portion of the address serves no purpose for communication within the group.

To receive new mail, Bob attempts to receive messages with identifiers that are the encryption of his email address and some date. To check for new messages, he checks each date since the last time he checked mail. For example, if Bob last checked his mail yesterday, he checks two dates: yesterday and today.

If one group member is outside the censored domain, then *Covert Email can interface with traditional email.* This user runs an email server and acts as a proxy for the other members of the group. To send mail, group members send a message to the proxy, requesting that it be forwarded to a traditional email address. Likewise, when the proxy receives a traditional email message, it forwards it to the requisite Covert Email user. This imposes one obvious requirement on group members sending mail using the proxy: they must use email addresses where the domain portion matches the domain of the proxy email server. Because the domain serves no other purpose in Covert Email addresses, implementing this requirement is easy.

Implementation and Evaluation. We have implemented a prototype application for sending and retrieving Covert Email. Currently, the task database is a set of tasks that search posts of other Twitter users. We have also written tasks that search for popular keywords (*e.g.*, "World Cup"). To demonstrate the general approach, we have implemented an (insecure) proof-of-concept steganography algorithm that stores data by altering the capitalization of words. Sending a short 194-byte message required three tweets and took five seconds. We have shown that Covert E-mail has the potential to work in practice, although this application obviously needs many enhancements before general use, most notably a secure text vector encoding algorithm and more deniable task database.

7 Threats to Collage

This section discusses limitations of Collage in terms of the security threats it is likely to face from censors; we also discuss possible defenses. Recall from Section 3.2 that we are concerned with two security metrics: *availability* and *deniability*. Given the unknown power of the censor and lack of formal information hiding primitives in this context, both goals are necessarily best effort.

7.1 Availability

A censor may try to prevent clients from sending and receiving messages. Our strongest argument for Collage's availability depends on a censor's unwillingness to block large quantities of legitimate content. This section discusses additional factors that contribute to Collage's current and future availability.

The censor could *block message vectors*, but a censor that wishes to allow access to legitimate content may have trouble doing so since censored messages are encoded inside otherwise legitimate content, and message vectors are, by design, difficult to remove without destroying the cover content. Furthermore, some encoding schemes (*e.g.*, steganography) are resilient against more determined censors, because they hide the presence of Collage data; blocking encoded vectors then also requires blocking many legitimate vectors.

The censor might instead *block traffic patterns resembling Collage's tasks*. From the censor's perspective, doing so may allow legitimate users access to content as long as they do not use one of the many tasks in the task database to retrieve the content. Because tasks in the database are "popular" among innocuous users by design, blocking a task may also disrupt the activities of legitimate users. Furthermore, if applications prevent the censor from knowing the task database, mounting this attack becomes quite difficult.

The censor could *block access to content hosts*, thereby blocking access to vectors published on those hosts. Censors have mounted this attack in practice; for example, China is currently blocking Flickr and Twitter, at least in part [43]. Although Collage cannot prevent these sites from being blocked, applications can reduce the impact of this action by publishing vectors *across many user-generated content sites*, so even if the censor blocks a few popular sites there will still be plenty of sites that host message vectors. One of the strengths of Collage's design is that it does not depend on any specific user-generated content service: any site that can host content for users can act as a Collage drop site.

The censor could also try to *prevent senders from publishing content*. This action is irrelevant for applications that perform all publication outside a censored domain. For others, it is impractical for the same reasons that blocking receivers is impractical. Many content hosts (*e.g.*, Flickr, Twitter) have third-party publication tools that act as proxies to the publication mechanism [51]. Blocking all such tools is difficult, as evidenced by Iran's failed attempts to block Twitter [14].

Instead of blocking access to publication or retrieval of user-generated content, the censor could *coerce content hosts* to remove vectors or disrupt the content inside them. For certain vector encodings (*e.g.*, steganography) the content host may be unable to identify vectors containing Collage content; in other cases (*e.g.*, digital watermarking), removing encoded content is difficult without destroying the outward appearance of the vector (*e.g.*, removing the watermark could induce artifacts in a photograph).

7.2 Deniability

As mentioned in Section 3.1, the censor may try to compromise the deniability of Collage users. Intuitively, a Collage user's actions are *deniable* if the censor cannot distinguish the use of Collage from "normal" Internet activity. Deniability is difficult to quantify; others have developed metrics for anonymity [39], and we are working on quantitative metrics for deniability in our ongoing work. Instead, we explore deniability somewhat more informally and aim to understand how a censor can attack a Collage user's deniability and how future extensions to Collage might mitigate these threats. The censor may attempt to compromise the deniability of either the sender or the receiver of a message. We explore various ways the censor might mount these attacks, and possible extensions to Collage to defend against them.

The censor may attempt to **identify senders**. Applications can use several techniques to improve deniability. First, they can *choose deniable content hosts*; if a user has never visited a particular content host, it would be unwise to upload lots of content there. Second, *vectors must match tasks*; if a task requires vectors with certain properties (*e.g.*, tagged with "vacation"), vectors not meeting those requirements are not deniable. The vector provider for each application is responsible for ensuring this. Finally, *publication frequency must be indistinguishable* from a user's normal behavior and the publication frequency of innocuous users.

The censor may also attempt to **identify receivers**, by observing their task sequences. Several application parameters affect receiver deniability. As the *size of the task database* grows, clients have more variety (and thus deniability), but must crawl through more data to find message chunks. Increasing the *number of tasks mapped to each identifier* gives senders more choice of publication locations, but forces receivers to sift through more content when retrieving messages. Increasing *variety of tasks* increases deniability, but requires a human author to specify each type of task. The receiver must decide an *ordering of tasks* to visit; ideally, receivers only visit tasks that are popular among innocuous users.

Ultimately, the censor may develop more sophisticated techniques to defeat user deniability. For example, a censor may try to target individual users by mounting timing attacks (as have been mounted against other systems like Tor [4, 33]), or may look at how browsing patters change

across groups of users or content sites. In these cases, we believe it is possible to extend Collage so that its request patterns more closely resemble those of innocuous users. To do so, Collage could monitor a user's normal Web traffic and allow Collage traffic to only perturb observable distributions (*e.g.*, inter-request timings, traffic per day, etc.) by small amounts. Doing so could obviously have massive a impact on Collage's performance. Preliminary analysis shows that over time this technique could yield sufficient bandwidth for productive communication, but we leave its implementation to future work.

8 Conclusion

Internet users in many countries need safe, robust mechanisms to publish content and the ability to send or publish messages in the face of censorship. Existing mechanisms for bypassing censorship firewalls typically rely on establishing and maintaining infrastructure outside the censored regime, typically in the form of proxies; unfortunately, when a censor blocks these proxies, the systems are no longer usable. This paper presented Collage, which bypasses censorship firewalls by piggybacking messages on the vast amount and types of user-generated content on the Internet today. Collage focuses on providing both availability and some level of deniability to its users, in addition to more conventional security properties.

Collage is a further step in the ongoing arms race to circumvent censorship. As we discussed, it is likely that, upon seeing Collage, censors will take the next steps towards disrupting communications channels through the firewall—perhaps by mangling content, analyzing joint distributions of access patterns, or analyzing request timing distributions. However, as Bellovin points out: "There's no doubt that China—or any government so-minded—can censor virtually everything; it's just that the cost—cutting most communications lines, and deploying enough agents to vet the rest—is prohibitive. The more interesting question is whether or not 'enough' censorship is affordable." [7] Although Collage itself may ultimately be disrupted or blocked, it represents another step in making censorship more costly to the censors; we believe that its underpinnings—the use of user-generated content to pass messages through censorship firewalls—will survive, even as censorship techniques grow increasingly more sophisticated.

Acknowledgments

This work was funded by NSF CAREER Award CNS-0643974, an IBM Faculty Award, and a Sloan Fellowship. We thank our shepherd, Micah Sherr, and the anonymous reviewers for their valuable guidance and feedback. We also thank Hari Balakrishnan, Mike Freedman, Shuang Hao, Robert Lychev, Murtaza Motiwala, Anirudh Ramachandran, Srikanth Sundaresan, Valas Valancius, and Ellen Zegura for feedback.

References

[1] Selenium Web application testing system. `http://www.seleniumhq.org`.

[2] Riaa sues computer-less family, 234 others, for file sharing. `http://arstechnica.com/old/content/2006/04/6662.ars`, apr 2006.

[3] Anonymizer. `http://www.anonymizer.com/`.

[4] A. Back, U. Möller, and A. Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In I. S. Moskowitz, editor, *Proceedings of Information Hiding Workshop (IH 2001)*, pages 245–257. Springer-Verlag, LNCS 2137, April 2001.

[5] A. Baliga, J. Kilian, and L. Iftode. A web based covert file system. In *HOTOS'07: Proceedings of the 11th USENIX workshop on Hot topics in operating systems*, pages 1–6, Berkeley, CA, USA, 2007. USENIX Association.

[6] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker. Low-resource routing attacks against tor. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2007)*, Washington, DC, USA, Oct. 2007.

[7] S. M. Bellovin. A Matter of Cost. *New York Times* Room for Debate Blog. Can Google Beat China? `http://roomfordebate.blogs.nytimes.com/2010/01/15/can-google-beat-china/#steven`, Jan. 2010.

[8] P. Boucher, A. Shostack, and I. Goldberg. Freedom systems 2.0 architecture. White paper, Zero Knowledge Systems, Inc., December 2000.

[9] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *Proc. ACM SIGCOMM*, pages 56–67, Vancouver, British Columbia, Canada, Sept. 1998.

[10] China Web Sites Seeking Users' Names. `http://www.nytimes.com/2009/09/06/world/asia/06chinanet.html`, Sept. 2009.

[11] Chinese blogger Xu Lai stabbed in Beijing bookshop. `http://www.guardian.co.uk/world/2009/feb/15/china-blogger-xu-lai-stabbed`, Feb. 2009.

[12] I. Clarke. A distributed decentralised information storage and retrieval system. Master's thesis, University of Edinburgh, 1999.

[13] Collage. `http://www.gtnoise.net/collage/`.

[14] Could Iran Shut Down Twitter? `http://futureoftheinternet.org/could-iran-shut-down-twitter`, June 2009.

[15] G. Danezis. Covert communications despite traffic data retention.

[16] G. Danezis and C. Diaz. A survey of anonymous communication channels. Technical Report MSR-TR-2008-35, Microsoft Research, January 2008.

[17] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 2–15, May 2003.

[18] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proc. 13th USENIX Security Symposium*, San Diego, CA, Aug. 2004.

[19] China is number one. *The Economist*, Jan. 2009. http://www.economist.com/daily/chartgallery/displaystory.cfm?story_id=13007996.

[20] Facebook. http://www.facebook.com/.

[21] N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. Karger. Infranet: Circumventing Web censorship and surveillance. In *Proc. 11th USENIX Security Symposium*, San Francisco, CA, Aug. 2002.

[22] N. Feamster, M. Balazinska, W. Wang, H. Balakrishnan, and D. Karger. Thwarting Web censorship with untrusted messenger discovery. In *3rd Workshop on Privacy Enhancing Technologies*, Dresden, Germany, Mar. 2003.

[23] N. Feamster and R. Dingledine. Location diversity in anonymity networks. In *ACM Workshop on Privacy in the Electronic Society*, Washington, DC, Oct. 2004.

[24] Flickr. http://www.flickr.com/.

[25] M. J. Freedman and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proc. 9th ACM Conference on Computer and Communications Security*, Washington, D.C., Nov. 2002.

[26] Freedom on the Net. Technical report, Freedom House, Mar. 2009. http://www.freedomhouse.org/uploads/specialreports/NetFreedom2009/FreedomOnTheNet_FullReport.pdf.

[27] J. Fridrich, M. Goljan, and D. Hogea. Attacking the outguess. In *Proceedings of the ACM Workshop on Multimedia and Security*, 2002.

[28] Future of Open Source: Collaborative Culture. http://www.wired.com/dualperspectives/article/news/2009/06/dp_opensource_wired0616, June 2009.

[29] A. Hintz. Fingerprinting websites using traffic analysis. In *Workshop on Privacy Enhancing Technologies*, San Francisco, CA, Apr. 2002.

[30] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663, New York, NY, USA, 1997. ACM.

[31] South Korea mulls Web watch , June 2008. http://www.theinquirer.net/inquirer/news/091/1042091/south-korea-mulls-web-watch.

[32] P. Maymounkov. Online codes. Technical Report TR2002-833, New York University, Nov. 2002.

[33] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of Tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*. IEEE CS, May 2005.

[34] Cisco netflow. http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html.

[35] Uproar in Australia Over Plan to Block Web Sites, Dec. 2008. http://www.nytimes.com/aponline/2008/12/26/technology/AP-TEC-Australia-Internet-Filter.html?_r=1.

[36] OpenNet Initiative. http://www.opennet.net/.

[37] Report on china's filtering practices, 2008. Open Net Initiative. http://opennet.net/sites/opennet.net/files/china.pdf.

[38] Outguess. http://www.outguess.org/.

[39] A. Serjantov and G. Danezis. Towards an information theoretic metric for anonymity. In R. Dingledine and P. Syverson, editors, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2002)*. Springer-Verlag, LNCS 2482, April 2002.

[40] A. Serjantov and P. Sewell. Passive attack analysis for connection-based anonymity systems. In *Proceedings of ESORICS 2003*, Oct. 2003.

[41] The SNOW Home Page. http://www.darkside.com.au/snow/.

[42] Y. Sovran, J. Li, and L. Subramanian. Pass it on: Social networks stymie censors. In *Proceedings of the 7th International Workshop on Peer-to-Peer Systems*, Feb. 2008.

[43] TechCrunch. China Blocks Access To Twitter, Facebook After Riots. http://www.techcrunch.com/2009/07/07/china-blocks-access-to-twitter-facebook-after-riots/.

[44] Tor: Bridges. http://www.torproject.org/bridges.

[45] Tor partially blocked in China, Sept. 2009. https://blog.torproject.org/blog/tor-partially-blocked-china.

[46] TorrentFreak. China Hijacks Popular BitTorrent Sites. http://torrentfreak.com/china-hijacks-popular-bittorrent-sites-081108/, May 2008.

[47] Pakistan move knocked out YouTube, Jan. 2008. http://www.cnn.com/2008/WORLD/asiapcf/02/25/pakistan.youtube/index.html.

[48] Turkey blocks YouTube access, Jan. 2008. http://www.cnn.com/2008/WORLD/europe/03/13/turkey.youtube.ap/index.html.

[49] Twitter. http://www.twitter.com.

[50] 18 Million Twitter Users by End of 2009. http://mashable.com/2009/09/14/twitter-2009-stats/, Sept. 2009.

[51] Ultimate List of Twitter Applications. http://techie-buzz.com/twitter/ultimate-list-of-twitter-applications-and-websites.html, 2009.

[52] State of the Twittersphere. http://bit.ly/sotwitter, 2009.

[53] M. Waldman and D. Mazières. Tangler: A censorship-resistant publishing system based on document entanglements. In *Proc. 8th ACM Conference on Computer and Communications Security*, Philadelphia, PA, Nov. 2001.

[54] The Accidental Censor: UK ISP Blocks Wayback Machine, Jan. 2009. Ars Technica. http://tinyurl.com/dk7mhl.

[55] Wikipedia, Cleanfeed & Filtering, Dec. 2008. http://www.nartv.org/2008/12/08/wikipedia-cleanfeed-filtering.

[56] Youtube - broadcast yourself. http://www.youtube.com/.

[57] YouTube Statistics. http://ksudigg.wetpaint.com/page/YouTube+Statistics, Mar. 2008.