Censorship resistant overlay publishing

Eugene Y. Vasserman[*]
Kansas State University
eyv@ksu.edu

Victor Heorhiadi[*]
University of North Carolina
victor@cs.unc.edu

Yongdae Kim, Nicholas Hopper
University of Minnesota
{kyd, hopper}@cs.umn.edu

November 1, 2011

Abstract

The fundamental requirement of censorship resistance is content availability and discoverability — it should be easy for users to find and access documents. At the same time, participating storage providers should be unaware of what they are storing to preserve plausible deniability. Fulfilling these requirements simultaneously seems impossible — how does a system maintain a searchable index of content for users and yet hide it from storage providers? These paradoxical requirements have been previously reconciled by requiring out-of-band communication to either find ways to connect to the system, locate files, or learn file decryption keys — an unacceptable situation when easy content discovery is critical. This paper describes a design for a peer-to-peer, permanent, and unblockable content store which is easily searchable and yet self-contained, i.e. does not require out-of-band communication. To achieve this, we separate file data, metadata, and encryption keys such that someone searching for information about a specific topic can retrieve all three components and reconstruct the file, but someone who only stores at most two components can neither determine the nature of the file content nor locate the missing component. We begin by identifying the core requirements for unblockable storage systems to resist state-level Internet censorship, construct a system that fulfills those requirements, and analyze how it avoids the problem of prior attempts at censorship resistance. Finally, we present measurements of a deployed proof-of-concept implementation, demonstrating the feasibility of our design.

1 Introduction

Freedom of expression has long been recognized as a fundamental human right, listed in Article 19 of the United Nations' Universal Declaration of Human Rights [62]. Unfortunately, *effective* freedom of expression — "freedom ... to seek, receive and impart information and ideas through any media and regardless of frontiers" [62] — is not currently possible when using the Internet due to censorship by private and state interests, who use a variety of social and technological means to limit expression or availability of information [18–21, 31]. Consequently, we have seen increased usage of technologies which bypass censorship by enabling any user to publish information and keep it available even under technological or legislative attack [11, 24, 28, 48, 57]. A notable example is the case of WikiLeaks [68], which bore the brunt of both government and private-sector censorship attempts after publishing a large cache of leaked diplomatic cables in late 2010 [56]. The website eventually emerged victorious but not unscathed — the documents were unavailable (or difficult to find) for hours or days at a time in the period immediately after their release [32]

[*]Part of this work was performed while at the University of Minnesota

even though WikiLeaks was specifically designed to resist censorship. This example shows how ad-hoc solutions are often vulnerable to attack in practice due to unforeseen interactions of government and private interests, varying laws and jurisdictions, and different technologies that make up the Internet.

To enable effective freedom of expression online we need a rigorously-developed solution for censorship resistant publishing (and access to data) which is robust to attack by powerful adversaries who are willing and able to prevent access to popular services, websites, or even sections of the IP address space [9, 45, 47] in order to block some targeted content, regardless of collateral damage. This scheme would require an "unblockable" communication, or transport, layer, and must additionally provide three key properties: plausible deniability for publishers and storers, efficient search and retrieval, and resistance to targeted and existential censorship. Furthermore, all of these properties must be provided without resorting to out-of-band channels since users may not be able or willing to risk social contact with each other in order to access the system. Unfortunately, as we detail in section 5, previous censorship-resistant systems have typically failed to satisfy at least one of these key properties.

Our design. In this work, we introduce CROPS, a Censorship-Resistant Overlay Publishing System which provides robust permanent storage while ensuring plausible deniability for storers, enables easy and efficient search for users, and resists targeted and existential censorship at the protocol level. Most importantly, *CROPS is self-contained — no out-of-band communication is required to search for desired content nor obtain decryption keys* for downloaded files. This promotes usability, improves uptake, and reduces the risk to users (who may face real-world dangers in attempting to use out-of-band channels). CROPS provides these properties even when up to 70% of nodes leave the network through four key design features:

Unblockable transport via MCON. To prevent an adversary from blocking access to CROPS at the network level by enumerating the membership set [22,24,63], we build CROPS on top of MCON [63], a membership-concealing Distributed Hash Table (DHT) that provides identity privacy and efficient, secure lookup even when up to 70% of all nodes are adversarial.

One-way search indexing. One of the key challenges is how to store files so that users can find content of interest but the nodes storing the files can plausibly deny knowledge of their contents. CROPS solves this using a process we call "one-way search indexing." When Alice publishes file F with keyword kw, she partitions it into three logical portions: the *content* portion consists of encrypted then erasure-coded *blocks* $b_1, \ldots, b_k$ each stored in the DHT under key $hash_1(b_i)$; the *content manifest* is a file that lists all of the block hashes (allowing retrieval of the file), stored under key $hash_2(kw)$; and the *key manifest* is the symmetric key used to encrypt the file, stored under key $hash_3(kw)$. Thus to retrieve a file it is sufficient to search for $hash_2(kw)$ and $hash_3(kw)$, but someone storing one of the manifests must conduct a dictionary attack in order to retrieve the other manifest and reconstruct the file.

In-network replication management. Once the file has been stored in this way, it becomes possible to delegate the task of ensuring that a file is sufficiently replicated to the nodes storing the file's content manifest. This allows the file to stay online indefinitely, even if the publisher goes offline or misplaces the decryption key. CROPS additionally uses erasure coding to decrease the replication factor required to maintain availability of all files. (Note that MCON's identity privacy prevents targeted censorship based on discovering and blocking the manifest holders' IP addresses.)

Curated garbage collection. In order to prevent an adversary from overwhelming the system with unused "garbage files," CROPS incorporates lazy garbage collection, in which unused contents are randomly selected for deletion. To prevent the deletion of important but unpopular content, CROPS also incorporates the notion of "editors" that can "bless" a file by signing its manifest so that the manifest holder knows not

to delete the manifest. Since the manifest holder periodically retrieves the file, its blocks will not be unused and will avoid garbage collection.

The rest of the paper is organized as follows: we define the requirements for censorship-resistant storage systems in Section 2, present the full design for CROPS in Section 3, and evaluate the design and an implementation in Section 4. Finally, we discuss related work in Section 5, and conclude in Section 6.

2 Requirements

There are several partially conflicting definitions of censorship. Danezis and Anderson define it as an external entity's success in imposing a particular distribution of files on a set of nodes [17]. Perng et al. define it as the likelihood a third-party can restrict a targeted document while allowing at least one other document to be retrieved [46]. Fiat and Saia use the term "censorship resistant" to mean that after adversarial removal of an arbitrarily large constant fraction of network nodes, all but an arbitrarily small fraction of the remaining nodes can obtain all but an arbitrarily small fraction of the original data items [30]. We subscribe to the third definition since it incorporates Danezis and Anderson's definition and does not limit the adversary by disallowing complete blocking.

Censorship resistant systems must provide transport, storage, or both, such that these services are not subject to targeted censorship or existential censorship (blocking) by malicious insiders or outsiders; adversaries controlling one or more nodes in the system, and those who monitor the system but do not control any members, respectively. To be useful, these systems must also support efficient content storage and retrieval, scale to a large number of participants and data objects, and survive member churn.[1] All *security requirements* of censorship resistant systems derive from three core goals: *availability* of content, *plausible deniability* for those storing content, and *identity privacy* for publishers and clients. Availability implies resistance to a broad spectrum of denial of service (DoS) attacks at all levels of the protocol stack, from the lower-level communication protocols (e.g. TCP/IP, SSL/TLS, etc.) to application-layer protocols (e.g. file sharing). *The adversary must not be able to deny service to most system users individually, nor broadly block the protocol without also shutting down Internet access.* Identity privacy implies that clients cannot be observed as accessing the system or specific content, and publishers should not expose their own real-world identities when posting content. Both clients and publishers require identity privacy to resist coercion and self-censorship. Publishers also need identity privacy to protect themselves from "rubber-hose cryptanalysis." [2] To that end, all content, once published, should be immutable — new versions can be published, but old ones cannot be removed. Plausible deniability is required to remove the legal burden from nodes contributing storage to the system — individuals must not be liable for content they volunteer to store.

It is important to note that *censorship resistant content stores are not filesystems*, and have very different requirements. Foremost, censorship resistant systems *do not aim to provide data privacy*, unlike distributed filesystems where objects have associated access permissions. Published data should be accessible to all.[3] There are two exceptions to this rule: a censor, even one who has infiltrated the network, should not be able to determine the content of data in transit, and storers should maintain plausible deniability, or no knowledge of what they are storing. The critical challenge for censorship resistance, not addressed by distributed filesystems, is how to safely store both keys and files in the same system. Leaving key management to users would be counter-productive when content is meant to be distributed as widely as possible.

[1] Arbitrary nodes in the system going offline and coming online at unpredictable times.

[2] An adversary exerting real-world influence (i.e. coercion or force) to extract keys.

[3] Some users may opt for private storage and out-of-band key management, but this is strictly optional.

2.1 Security requirements

Identity privacy. When censorship is primarily enforced through social deterrence (by applying post-facto punishment to those who attempt to circumvent it), censorship resistant systems must prevents identification of individuals who supply or access censored materials or access them. This is formalized in [63] as "membership concealment" which decouples online pseudonyms from real-world identities (e.g. IP addresses) and conceals the link between them. Membership concealment must also be used to protect availability: if a censor can locate all participants of a censorship resistant network, it can block access to all individual nodes at the network level. Note that this differs from anonymizing systems like Tor [25], which provide unlinkability between sender and receiver, but not membership concealment (since the sender and receiver both may still be observed as taking part in the system). CROPS leverages MCONs [63] as a communication layer to provide identity privacy.

Plausible deniability. To prevent prosecution of storers on the basis of hosted content, node-local content must be opaque to real-time or post-hoc examination by an adversary and the storer itself. It is sufficient to ensure that while stored content can be examined, *the process is too resource-intensive* to apply to a large fraction of content. The storer should also be prevented from determining the nature of a specific query to which it is responding, i.e. the storer knows that the query matches data stored locally (which the storer returns in response), but cannot determine the cleartext content of the query or the data.

Availability. While some censorship resistant designs [65] have taken the all-or-nothing approach, assuming that an adversary would want to censor targeted content only (not the entire system), this assumption has not held up in practice. Recent evidence of complete blocking of certain services [9, 45, 47] reinforces the need for a system that can withstand existential censorship. We use MCONs to provide blocking (existential censorship) resistance and network-level availability in addition to identity privacy. However, this does not guarantee content targeted censorship resistance. An adversary may attempt to censor specific content through multiple means, such as technological countermeasures against access to content with specific keywords [21, 74], and social means such as legal action (or threats of legal action) [8, 21, 58]. This leads us to derive two functional requirements: *robust storage*, to protect availability, and *scalability* to support many users. A third functional requirement, *discoverability*, is necessary for popular uptake of the storage network — if files are difficult to find, or if users must discover decryption keys out-of-band, the network is unlikely to be successful.

2.2 Functional requirements

Robust storage. Storage is provided by network nodes in a peer-to-peer manner, with each node contributing a small portion of the overall network storage capacity. The network must replicate content across enough storers to guarantee, with high probability, that it will be retrievable even with high storer churn and attempted blocking. For this purpose we employ a combination of erasure coding and aggressive replication.

Scalability and efficiency. Censorship resistant networks should be designed to scale wells both in terms of the number of supported users (searchers and storers) and in terms of the amount of storage the network provides. Storage and traffic load should be distributed equitably across peers for fault tolerance as well as to avoid unduly taxing any particular peer. Robust replication and storage/bandwidth efficiency is a fundamental trade-off in such systems [33]. For search we use distributed hash tables (DHTs), which are structured overlay networks that allow for very efficient publication and lookup of arbitrary key-value pairs [41, 51, 52, 54, 59, 73]. Each DHT node has a logical address (pseudonym), and participates in routing and lookup, and contributes storage space.

Discoverability. To make the network easy to use content should be easy find, preferably using keyword

search. (As a counter-example, imagine indexing files by difficult-to-remember "content hashes," or alphanumeric strings at least a dozen characters long.) Simple search would dramatically improve the learning curve of the system, allowing users with minimal technical knowledge to search information relevant to them, regardless of whether or not it is censored.

3 System Design

CROPS is designed to be a write-once, read-maybe [60, 64] long-term data archival system, implemented as a storage layer on top of an identity-concealing DHT [63]. (While some censorship resistance properties of CROPS depend on properties of the underlying transport, such as blocking resistance and identity hiding *CROPS itself is DHT-agnostic*, compatible with any DHT implementation.) Member nodes can assume combinations of the following roles: a *publisher*, who uploads content; a *storer*, who stores content; an *intermediary*, who helps route control and data messages through the network; and a *searcher*, who searches and downloads content. Nodes may assume any subset of these roles simultaneously, although the intermediary role (and perhaps the storer role) are required at all times. Furthermore, all nodes are considered peers, or equal participants in the network.

CROPS does not have a notion of access permissions: since our design aims for censorship resistance and ease of searching, confidentiality guarantees are out of scope, except to ensure plausible deniability of content storers. This simplification avoids most issues that plague long-term archival and/or collaborative secure storage [33, 64, 69]. The design is further simplified by making files *immutable* — it allows us to use simple digital signature schemes to ensure integrity, authenticity, and publisher-continuity (if a publisher's pseudonym is, for instance, a hash of her public key, files encrypted and signed by the author are self-authenticating). *The major problems we face are file integrity, replication, key management, and content discovery.* Interestingly, in the case where censorship resistance meets discoverability, *key management and ease of discovery are linked*, since decryption keys and encrypted files are not useful without each other.

Key management. In filesystems, key management is usually handled on the client side, i.e. on the user's machine or trusted hardware. The user is also required to remember passwords. Requiring user-supplied secrets is out-of-band key management — the user must know some information to ensure access to content. In filesystems, this design decision stems from the need to protect confidentiality and integrity of files from everyone other than the owner(s) and authorized users. Conversely, a censorship resistant system should ensure content confidentiality only for the node storing the file — all others should be free to reconstruct and decrypt all files in the network. Therefore, our key management challenge stems from the requirement for our system to be *self-contained* — a user should be able to find content using plain-text keywords, and we cannot make the assumption that the user knows a password, the hash of the file's plaintext, etc. All information required to retrieve a file should be stored in the network, but to preserve plausible deniability of storers, files must be encrypted on disk, and the storer must not be able to easily learn the decryption key (nor must the key storer be able to locate the content which the key decrypts). A number of systems [23, 60, 61, 71] avoid encryption by using secret sharing — files are split across multiple servers, so no one server holds enough secrets to reconstruct the file, nor learn anything about its contents. Other designs rely on storing encrypted data, leaving key management to the clients [37, 66], specialized hardware [26], or directory servers [1]. None of these approaches can be used with our system since client-based (out-of-band) key management would make search and retrieval difficult, and neither specialized hardware nor directory servers are secure against a powerful adversary. To achieve our goals, *we separate file content from file metadata and from the key, making this information only known to the publisher and a searcher, but not storers or arbitrary network nodes.*

3.1 Publishing

Figure 1 shows the CROPS publishing process. Each publisher $\mathcal{P}$ has a persistent pseudonym based on the hash of the public key of an asymmetric key pair (using a pre-image- and collision-resistant hash function h). Therefore, if the public and secret keys are PK and SK, respectively, the pseudonym is $h(PK)$. Before publishing a file F, $\mathcal{P}$ compiles a list of keywords that describe the contents. The publisher selects a random integer K of appropriate bit-length and encrypts the file F using a symmetric cipher keyed with K, producing $E_K(F)$. Let the length of $E_K(F)$ be x *blocks* each of which is b bytes. We apply an m-of-n erasure code to each block, yielding a total of xn *encoded chunks* of b bytes each, producing $EC^n_m(E_K(F))$, where EC is the erasure-coding function accepting parameters m and n. Note that erasure coding is performed after encryption. Each of the n encoded chunks can now be inserted into the DHT, stored using the hash ($hash_1$) of its content as the key. Once all chunks and manifests are uploaded, the publisher is free to discard the random encryption key for added plausible deniability. Keeping the key does not grant the publisher any special permissions.

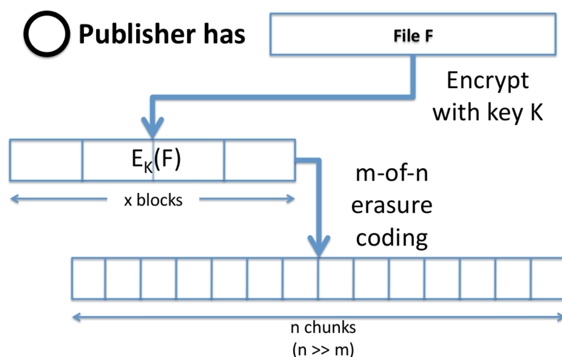

Figure 1: A publisher encrypts a file and applies an m-of-n erasure coding scheme.

Manifests. Each file has two associated *manifests*, or metadata files containing various identifying and authenticating information: one manifest for the key K, and one for the file content. The latter can be thought of as as a “rich content pointer,” (as opposed to simple content pointers used in e.g. Kad [27]), that contains information about the network nodes storing the erasure-coded chunks of the file. To preserve plausible deniability *nodes will never store both types of manifests for the same file*. In order to minimize the risk of the same node being asked to store both manifests, different hash functions are used for keywords in each manifest, e.g. hash function $hash_2$ will be used for all content manifests, while function $hash_3$ is used for all key manifests. Finally, a node may simply refuse to store both manifests if asked — this is in the best interest of honest nodes. Content manifests have the following format:

- $h(E_K(F))$, the hash of the file ciphertext to verify reconstruction success
- the list of hashed salted keywords chosen by the publisher for searching
- $C_1, C_2, \cdots, C_n$, the list of chunk storage locations (chunk content hashes) in the DHT
- $h(F)$, the hash of the file plaintext, before encryption and erasure coding, to verify decryption success
- $h(PK)$, the hash of the publisher’s pseudonym and public key for authentication and publisher-continuity
- $Sig_{SK}(\mathcal{M})$, the publisher’s signature over the entire manifest to preserve integrity

Key manifests are almost identical to content manifests, except that the list of chunk storage locations is replaced with a single plaintext copy of K, and the keywords are hashed using $hash_3$ instead of $hash_2$.

Salts can be arbitrarily long and different in every instance of a manifest. The only unsalted keyword in a manifest instance is the one meant for lookup at the node where that manifest is stored. While it may be sufficient to brute-force only one keyword, honest nodes have no incentive to do so (it breaks plausible deniability), and adversaries gain nothing, since they can directly search for any keywords of interest. Easily-guessed keywords unfortunately cannot be avoided, lest they themselves become the secret

information needed to locate a file. However, recent events have shown that users can easily avoid keyword-based attacks in the short term [16, 72].

Manifests are not erasure-coded, but rather stored whole and heavily replicated. Each manifest is indexed by the contained keywords as well as replica numbers, so a manifest with replication factor r and k keywords is stored under the rk hashes $h(i, h(keyword_j))$ for $1 \leq i \leq r, 1 \leq j \leq k$. Note that in order to deny access to a file it is sufficient to deny access to all copies of either its key manifest or content manifest, so a particularly aggressive replication strategy is needed to ensure manifest availability.

Erasure codes. We use erasure codes [50] to ensure file *availability*: we apply an m-of-n code such that out of n post-encoding chunks, only m need to be retrieved to reconstruct the original file. These chunks are stored in the network based on the hash of the chunk content [51], meaning that chunks are all self-verifying — a client can verify that each chunk was fetched correctly because the hash of the chunk is the chunk identity [49]. The parameters of the erasure code specify the replication and storage overhead — an m-of-n code imposes a factor of n increase on the amount of data stored in the system. Due to the nature of our publishing protocol (discussed below), it is possible to support publisher-specified erasure code algorithms and parameters. However, we will not discuss this at length and will instead focus on selecting m and n values that are "good enough" to store some particular amount of data indefinitely — see Section 4, below.

3.2 Search and retrieval

To retrieve a file from CROPS, a searcher first obtains the file manifest by hashing meaningful search terms and fetching manifest replicas stored at those hash keys. (Note that since search terms are hashed, everyone forwarding the search request or storing the results has plausible deniability as to the target of the query.) Since each keyword query will return a number of manifests, the searcher takes the intersection of all returned manifests to find the one that matches most of query keywords. Alternatively, to minimize query time and bandwidth requirements, clients can search for the least common keyword under which content of interest might be indexed, download those manifests, and then check additional salted keyword hashes locally. The searcher repeats the process for key manifests, but without the need for keyword intersection search, since key manifests can be matched with file manifests by an identical $h(E_K(F))$.[4] Once both manifests have been retrieved, the searching node can determine the location of all the content chunks in the network and fetch any m-chunk subset of them to reconstruct the encrypted file. She can verify that the file has been correctly reconstructed by checking that the results match $h(E_K(F))$. She then uses the key K (contained in the key manifest) to decrypt the content and verify that it matches $h(F)$.

Note that keyword search support is meant as a proof-of-concept example. Like other keyword-based schemes, it is vulnerable to "hot" keywords — particularly popular words whose "owner" nodes become heavily loaded with traffic. Keyword search is designed to promote content discoverability, and thus system uptake, while avoiding trusted or out-of-band indexes. Garbage collection and editors (discussed in detail below) should be able to help keep the popular keyword space free from junk. Popular keywords may need to be curated more aggressively.

3.3 Storage, maintenance, and retirement

Manifest "guarantors." Each peer in the network who had a copy of a manifest is referred to as a *manifest guarantor*, and is responsible for maintaining the replication factor of the manifest. Each node holding a file manifest also maintains the replication factor of all file chunks listed in the manifest. Although the file

[4]Key manifests can be distinguished from file manifests since key manifests do not include a list of chunks, because they use a different hash function for keywords, or simply by adding a manifest type tag to manifest files.

manifest-holder knows the location of every file chunk and manifest, she only has access to salted hashes of keywords, and so cannot learn anything about the nature of the content except by performing a lengthy brute-force search on the keyword list. Furthermore, fetching and reconstructing the file does not expose its contents to the file manifest-holder, since the file has been encrypted and the manifest-holder does not know the key. Key manifest holders, on the other hand, do not know the location of the content chunks and cannot reconstruct the file. The desired properties for manifest-holder are:

- Can re-assemble an entire erasure-coded encrypted file
- Cannot obtain the file plaintext
- Cannot obtain the file keywords except by exhaustive search
- Cannot alter the manifest without breaking the signature scheme used to sign it
- Cannot remove file chunks or manifests from the network other than dropping its own

Plausible deniability. Files in the CROPS network are not stored whole, but are divided into chunks, which are then distributed throughout the network based on the cryptographic hash of the block itself. Even if a peer is storing multiple parts of the same file, he has no way to determine this, nor can he learn the nature of the content since the files were encrypted before storage. Manifest holders are likewise protected even if they are also storing chunks of the file, because reconstruction of the file would yield encrypted data and not the file cleartext. However, if a peer stores both the key manifest and the content manifest, he can link those manifests together, reconstruct and decrypt the file, learning the cleartext. We note that *it is not in the best interest of an honest storer to be in possession of both manifests* to maintain plausible deniability if his computer were confiscated. Storing both manifests does not benefit malicious nodes either, since they could simply search for keywords of interest. Therefore, peers should refuse to store key manifests if they already hold a content manifest, and vice versa. Peers can determine whether the manifests correspond to the same file by checking that their file hash $h(E_K(F))$ matches. Forward-secure encryption must be used by the publisher when making the storage request, since otherwise a network-monitoring adversary may record the transaction, compromising the storer's deniability. If the publisher itself is malicious, the storer can claim to simply be following the protocol.

CROPS provides storer protection even stronger than plausible deniability. Determining the nature of stored files is fairly difficult:[5] content manifest holders would have to either guess the file encryption key K or obtain the plaintext keyword hashes in order to infer file content. Key manifest holders must likewise obtain the keyword hash plaintexts in order to determine which file corresponds to their stored key. File chunk storers know even less, since they receive no metadata associated with the encrypted chunk.

Garbage collection. Content storers keep a timestamp associated with every locally stored chunk (initialized with the original publication time of that chunk), and update the timestamp every time that chunk is accessed by another user in the network. During idle times, storers lazily examine their stored chunks and probabilistically discard those with timestamps older than some global time cutoff (e.g. one month), clearing storage space. Recall the problem of pollution attacks, where adversaries can overwhelm the storage capacity of the entire network by inserting garbage. A number of archival schemes use periodic refresh to maintain content in a network and purge old or unpopular files [11, 33, 44]. This garbage collection scheme ensures that unrefreshed data does not remain in the network indefinitely, requiring adversaries to continue inserting or accessing their data in order to prolong the attack. This approach does not fully solve the pollution problem, since automated systems in general cannot distinguish "useful" content from junk (and prior attempts have lead to security vulnerabilities [38]). Rate-limiting publishers is not fully satisfactory either: consider a publisher who comes into possession of a large cache of documents — it would be good to allow

[5] Timing attacks may be used to identify correlated chunks, but the content of the decrypted file cannot be determined.

them all to be published fast, before the publisher faces real-world pressure to relinquish them. A fully satisfactory solution is left to future work.

Content-oblivious robust replication. Our replication strategy works best when many honest nodes cooperate to achieve resilience, but still functions when only a minority of nodes are honest. Each node storing a content manifest is "responsible" for the file to which the manifest points. Note, however, that the manifest holder does not know the content of the file since he cannot decrypt it without key K. To ensure availability in cases of failure of a large number of nodes, manifest holders constantly monitor the replication factor of the manifest target. Every time period τ, manifest-holders examine their stored content manifests and download and reconstruct every encrypted file to which the manifest refers. Applying the erasure code, the node can obtain copies of all chunks that should be stored. By searching for a sample of those chunks, the manifest holder can probabilistically determine the *current replication factor* of a file and compare it to the *desired replication factor*. If the difference is significant, the node inserts all missing chunks back into CROPS. The node does the same thing with the manifest itself, checking to see if enough replicas are available, and creating new replicas if not. (Key manifest holders are only responsible for the replication of the manifest itself, since they do not know the content chunk list.) Manifest holders, through checking replication factor and accessing content, implicitly serve to refresh content timestamps without requiring any action on the part of the publisher. Like the garbage collection scheme described above, manifests that are not being actively accessed or are over-replicated can be probabilistically discarded. As manifests are dropped, references to content chunks are lost with them, and content will no longer be accessed, eventually being discarded by storers. Thus *a single honest manifest-holder is sufficient to maintain the replication factor* of any piece of content with overwhelming probability, as long as she can successfully retrieve the minimum number of chunks required to reconstruct the encrypted file.

A file cannot be accessed without the manifest, so this is always the first item to be fetched. If a manifest is unpopular then it will never be accessed, and will be dropped (unless editor-signed), so maintenance will stop. File chunks at which it points will become "orphaned," and will subsequently be dropped as well (chunks are never editor-signed; only manifests). Furthermore, manifest refresh probes must be indistinguishable from "real" file access, lest we expose ourselves to adversaries who selectively respond only to refresh queries, but not real queries.

Curated content. CROPS is similar in concept to a massively distributed version of the WikiLeaks service [68]. In keeping with the spirit of WikiLeaks, we attempt to *preserve unpopular but important files by employing an editor-facilitated publishing model*. While a free-for-all model is attractive (where all published content is maintained indefinitely), it suffers from a number of drawbacks such as unregulated content quality, pollution and collision attacks, and storage space concerns. We also reject automatic data filtering by popularity, such as that used by GNUnet and Freenet [5, 11], since this can lead to vulnerabilities [38]. Moreover, unpopular content may be just as important, if not more important, than popular content. CROPS will be bootstrapped with a set of hard-coded *editor public keys*, the private counterparts to which are held by a select group of pseudo-administrators. Editor keys can be used to sign manifests to add a "stamp of approval." [6] Of course, an editor-assisted model introduces its own set of problems. A few malicious editors may sign "junk," but we must err on the side of content retention if we want to support storage in perpetuity. Furthermore, excessive publication of new content can constitute a denial of service attack on on editor time. To address this issue, we use a hybrid model such that both editor-approved and editor-unapproved data (whether explicitly or simply for lack of examination) can still be stored and retrieved, but editor-approved data has special protection in that it will not be dropped from the network. Honest manifest-holders do not apply pruning to editor-signed manifests, but rather keep them forever, independent of the

[6] Both the content and key manifests must be signed.

last time they were accessed. Since manifest-holders refresh content chunks in the network, the content is safe as well. Manifests not carrying editor signatures will be garbage-collected as descried above. Note that we are not particularly concerned with malicious editors — an editor cannot explicitly remove content from the system, so, like in the case of manifest holders, even *a single honest editor is sufficient* to ensure that important content makes it past the editorial process.

4 Evaluation

4.1 Security

The security of CROPS depends on three primary factors: the properties of the underlying blocking resistant communication layer, the plausible deniability available to content and manifest storers, and the resilience of the storage system to application-level attack such as pollution and storage exhaustion. The security properties of our communication layer are described in detail in [63]. The main contributions of this layer is blocking resistance and identity hiding, which are necessary to protect against enumeration, blocking, and real-world (physical) attack on users. CROPS itself if agnostic to the underlying communication layer, as long as it export a DHT abstraction.

Plausible deniability. Plausible deniability in CROPS is guaranteed by the separation of content from metadata, and of decryption keys from content location information. CROPS nodes who store individual erasure-coded content blocks do not have access to any information regarding the plaintext. Nodes who store content manifests (metadata) can access the list of chunk locations in the network, and a list of salted keyword hashes, but not the plaintext of the content. This node can reconstruct the encrypted file in its entirety, but cannot decrypt without a brute-force search of the keyspace. An alternative is to find the key manifest (metadata) in the network, but this would require a brute-force attack on the keyword hashes to derive at least one original keyword to fetch the correct key manifest. A single keyword, especially if it is popular, will return many manifests, imposing undesirable computational and bandwidth overhead (similar in spirit to rate limiting through resource-intensive proofs-of-work). Finally, the node storing the key manifest does not have access to the content, the content locations, nor to the keywords. It would likewise need to attempt to brute-force the keyword hashes to search for the content manifest and download the file itself. No storage node benefits from storing chunks in conjunction with metadata manifests, since not one but both manifests are needed to reconstruct and decrypt a file. A node who is storing both manifests *does* have the required information to reconstruct and decrypt the file, but we note that it is not in the user's best interest to store both manifests, since plausible deniability only benefits the node itself. Adversaries can certainly opt to store both manifests, and will then be able to fetch and decrypt the file, but this serves no purpose since the same goal can be accomplished by searching for keywords they wish to censor.

Locality and "keyword squatting." DHTs are generally quite vulnerable to both targeted and existential censorship. If the attack can deny access to a particular piece of content, the attack can be called "targeted." Targeted censorship is a particular problem if all copies of a given file are stored in the same DHT "neighborhood," as in systems like Kademlia [41]. This is common in content-addressable systems, since data is stored at nodes whose DHT addresses are logically close to the hash of the data. Adversaries can attempt to assume DHT pseudonyms (IDs) that are logically close to a given keyword, ensuring that only adversarial nodes store content indexed with that keyword. A natural way to avoid these chosen-ID attacks is to use externally-assigned IDs, computed by other nodes in the network using secure multiparty computation. This feature is provided by the MCON DHT layer. Adversaries who control network gateways can also disable access to large sections of the Internet with the intention of censoring data that may be hosted in a specific

physical (geographical) or logical (IP address range) location. The MCON layer helps evade this attack as well, as MCONs make determining the IP address of a given node prohibitive, preventing attacks from outside the network. However, colluding insiders can still launch *attacks from within the network*, sending a flood of requests for content hosted by a relatively small number of nodes, disrupting availability of those nodes only, and thus targeted content. Therefore, we need a more robust replication and load balancing strategy than one that only works within the neighborhood. CROPS uses multiple replica IDs which are incorporated into content hashes for each chunk and each manifest, e.g. with replication factor r, a single chunk b is stored under r hashes $h(r, h(b))$ for $1 \leq i \leq r$. Assuming a uniform distribution of the resulting hash values, chunks should be uniformly distributed throughout neighborhoods in the DHT.

Storage exhaustion. Another such attack that bears special mention is the *pollution, or storage exhaustion attack*. Any P2P open-access network is vulnerable to adversaries "polluting" the results of any given search (by keyword) by repeatedly publishing faulty data to the target keyword. Furthermore, since only a finite amount of storage is available, the adversary also could publish large volumes of content to exhaust storage space in the entire network, independent of keywords. Unfortunately, this problem is exceedingly challenging. Although there are verifiable fair storage schemes (e.g. [14, 44]), the incentive to retain data in such systems is generally the removal of the offender's data from the network. In the context of censorship resistance, this is *not an impediment* to a censor. In fact, *it is beneficial* — if an adversary can "persuade" a given publisher to stop storing data, the publisher's data would be dropped from the network, effectively censoring it. Therefore, an alternate solution is required. CROPS' self-cleaning mechanism decreases (but does not completely eliminate) damage from pollution and exhaustion attacks, making adversaries continually work to refresh their content in the network by accessing it or uploading it again. Furthermore, editor-vetted content will be stored permanently and available for access. It is important to note that while editors and manifest guarantors have the power to keep content within a network, they do not have the ability to cause content to be dropped, therefore editors' and guarantors' power is geared toward censorship resistance (maintaining content) versus censorship (removal of content).

Availability and robustness. The authors of Glacier [33] derive the following expression for the *durability* D, of a file block when using m-of-n erasure coding:

$$D = P(s \geq m) = \sum_{k=m}^{n} \binom{n}{k} (1 - f_{max})^k \cdot f_{max}^{n-k}$$

Recall that files of x blocks are erasure-coded to produce encoded chunks, such that at least m chunks are required to successfully decode each block; here, s is the number of successfully-retrieved chunks, and f_{max} is the failure probability of a node, assuming uniformly random failures and that all chunks are stored at different nodes. Therefore, if x blocks are stored in the network, then the probability that *every file is recoverable*, or the robustness of the censorship-resistant system, is $\rho(x) = D^x$; a perfectly robust censorship-resistant system has $\rho(x) = 1$. In other words, *a perfectly robust censorship-resistant storage system always allows for the retrieval of every block, and thus every file*, assuming a node can successfully connect

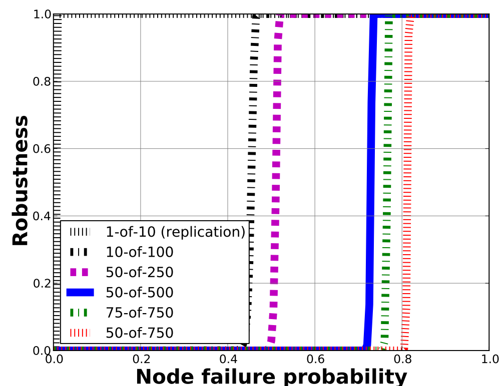

Figure 2: Robustness ρ of the censorship-resistant system when using a given erasure code configuration or replication.

to the system; if $1-\rho(x)$ is negligible, then we say that a system is robust. In CROPS, clients must retrieve key and content manifests before fetching content chunks. To simplify analysis we assume that the manifest replication factor is the same as that of the erasure code used.

Figure 2 (and Figure 7 in Appendix A.2) shows results of experiments to determine network and file robustness when using various erasure code parameters. We observed that nearly a factor of 10 storage overhead is required for a robust censorship-resistant system that can support 2^{60} chunks, independent of size. Specifically, the best trade-off between overhead and robustness was obtained using a 50-of-500 code. Not gain much advantage is gained by moving to lower overhead parameters, since robustness drops quite sharply at roughly factor of 8 overhead.

4.2 Performance

To demonstrate real-world feasibility of our design, we implemented a CROPS prototype using the Azureus/Vuze DHT [4] as an underlay (instead of MCON), and measured the performance of a 400-node CROPS network using the PlanetLab distributed testbed [10]. Testing CROPS using a different DHT than MCON, for which it was designed, allows us to determine the *additional overhead imposed purely by CROPS*; CROPS+MCON measurement is left to future work.

Architecture. Since the Vuze DHT is designed only for lookup and not bulk transfers or storage, we modified the Vuze client to augment its UDP-only communication mechanism with a TCP-based bulk file transfer subsystem. This constitutes only a small addition of $4,389$ lines of code to the Vuze Java codebase, which is $486,226$ lines. The structure of our implementation is diagrammed in Figure 3. The *cryptographic module* performs operations such as encryption and signing. The *DHT interface and Vuze client* modules, along with the *DHT routing module*, are responsible for locating CROPS nodes using the DHT interface, as well as verifying that nodes are online. This information is passed to the *CROPS interface* which keeps track of CROPS configuration parameters and routing table, and is responsible for all CROPS-specific protocol logic.

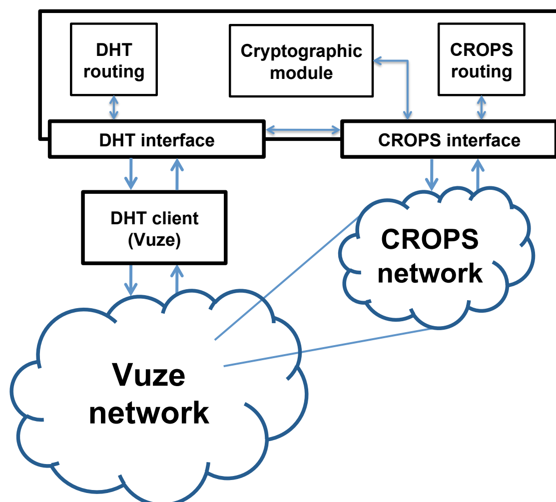

Figure 3: CROPS implementation, incorporating separate Vuze and CROPS DHT interfaces.

Experiments. CROPS nodes join the Vuze DHT in groups of 10 at half-minute intervals. When a node comes online, it publishes its ID to the closest 10 rendezvous points[7] and builds a CROPS routing table by querying each rendezvous point for random CROPS IDs. While we fixed the size of each node's CROPS routing table to 50, we found that due to transient network events, the average number of routing table entries was 27.5 ± 0.3. To accommodate network dynamics, nodes retry publication for 10 seconds, re-sending data until the desired content replication factor is achieved. Furthermore, each node refreshes its routing table every hour to discover new nodes and purge offline or failed nodes. We currently do not implement erasure coding, but rather simulate its effect by increasing the replication factor, using 10 logically adjacent and 15 logically distant replicas (of each previous replica, for a total of 100) for file content and, 15 replicas for

[7] 20 total rendezvous locations are currently used

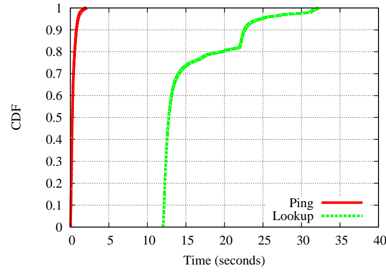

Figure 4: Cumulative distributions, in seconds, showing successful UDP `ping` and `lookup` times for CROPS node within the Vuze network.

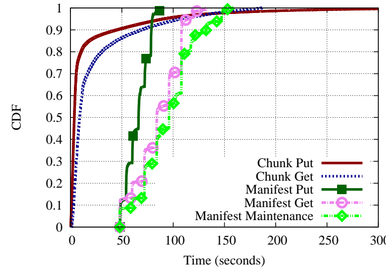

Figure 5: Cumulative distributions, in seconds, of successful `put` and `get` times for content chunks and manifests, and manifest maintenance times.

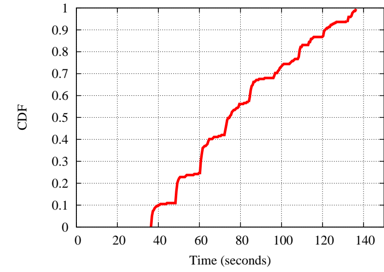

Figure 6: Time required for a client to locate, reconstruct, and decrypt a file from the CROPS network.

each manifest. We measured *publication time* by randomly selecting 20 nodes to upload a 20MB file in 41 blocks, along with corresponding manifests categorized with 5 to 15 keywords. Each node performed *hourly maintenance* for each locally stored key manifest only, checking its replication factor in the testbed.

The most basic Vuze communication operations — UDP ping and lookup — are shown in Figure 4. While UDP pings are reasonably fast, even trivial TCP operations, such as "chunk put" (upload), take 10 to 15 seconds (shown in Figure 5). Each "chunk put" and "chunk get" operation is measured for individual chunks, and does not take replication into account, establishing a baseline for communication speeds. Manifest put, on the other hand, is only considered complete and successful when *all manifest replicas* have been uploaded. Maintenance time depends on the size of the manifest, and so on the number of keywords and chunks per file.[8] Note that bandwidth-intensive operations, such as uploading a 512KB chunk, require similar amounts of time as low-bandwidth actions. Moreover, get operations, which require more lookups than put operations, are always slower. This suggests that latency is the limiting factor in our experiments.

As in expected real-world deployments, we observed a relatively large number of failed operations in our testbed (omitted from the graphs). 7.77% of UDP and 2.87% of ICMP pings failed, 15.12% of lookups failed or timed out, and 11.24% of maintenance operations failed due to either unsuccessful lookup and download, or inability to upload all needed replicas. 20% of network nodes were responsible for 80% of these failures, suggesting that the problem was related to instabilities of the node's own network connection rather than CROPS or DHT failures. Note, however, that due to our manifest replication factor, *it is highly unlikely that any given file became unavailable* during the course of the failure. This mirrors our expected real-world deployment scenario, where many queries and maintenance operations may transiently fail without compromising the long-term availability of content. Nodes with working network connections were able to complete lookups, and there were sufficient guarantors to maintain manifest replication.

Figure 6 shows the total time for a client to obtain a file, including search, manifest download, fetching all required content blocks, and then reconstructing and decrypting the file. Data shows that the median user would only have to wait between 65 and 85 seconds to fetch and decode a 20MB file — a reasonable amount of time for a non-interactive transaction (bulk transfer). This allows us to conclude that the performance of our *unoptimized* CROPS client is acceptable in practice even in highly unstable network

[8] File manifests will generally be significantly larger than key manifests due to the chunk list, with the exception of small files with few chunks, when the number of keywords and the size of the cryptographic key dominate.

environments. Adding features like pipelining, parallelization, and other static and dynamic performance tuning could only improve performance. Moreover, since MCON operations incur high latency penalties but minimal bandwidth overhead when pipelined, changing the underlying DHT from Vuze to MCON should be minimally disruptive [63].

5 Related work

Broadly, past censorship resistant designs can be separated into groups that provide functionality from *centralized or semi-centralized servers*, and those where functionality is *fully distributed* among system participants (P2P). Table 1 provides a quick comparison of several representative storage, distribution, and censorship resistant systems. Additional "ad-hoc" methods of bypassing censorship are discussed in Appendix A.1. A common issue with those designs is that they require additional sophisticated key management to prevent insiders from learning most or all possible keys or service providers and subsequently removing content or blocking the system. It is currently unclear how to solve this key management issue.

System types. Storage in federated systems is provided either by multiple servers controlled by the same logical entity (e.g. WikiLeaks [68]), or a small to moderate number of servers controlled by independent agents (e.g. Eternity [3], Free Haven [23], Publius [66], Tangler [65], and Tahoe-LAFS [69]). These systems can be easily blocked at the network border since every client must know the IP addresses or domain names of servers. Peer-to-peer systems do not suffer this problem: they spread information storage roles over far more entities than federated systems, and are thus potentially more resistant to wholesale blocking attacks. Examples include LOCKSS [40], Turtle [48], Kaleidescope [57], Freenet [11] and GNUnet [5], and Infranet [28, 29]. These designs are also more robust to short- and long-term insider attack [64]. Although these systems cannot be blocked in a static manner in the same way as federated designs, they are still vulnerable to *dynamic enumeration and real-time blocking* [63].

Robustness. Like many others [33, 34, 69], CROPS uses erasure codes and a high replication factor to mitigate attacks against specific content (targeted) and the system as a whole (existential). Replication and erasure coding together provide better robustness than replication alone. Non-locality is also a critical point — data must be available at multiple "logical" storage locations in the network as well as multiple physical locations. Logical separation is required to prevent adversaries from mounting targeted attacks against logical network "neighborhoods;" physical separation prevents an adversary from blocking a small number of key nodes in the same manner as existential attacks against federated systems.

Discoverability and indexing. Many of the above systems do not index content, and rely either on hard-to-remember content identifiers, trusted directories, or third-party indexes. The latter two options represent a clear violation of P2P principles, providing an attractive attack target for static blocking. Moreover, modifying an existing system to support in-band search while maintaining plausible deniability is not a trivial task. Any search facility would have to be in essence unidirectional — one can search for information and find it, but someone storing the information should not be able to successfully search for it, so encryption and metadata separation of some type are required in order to preserve plausible deniability.

Blocking resistance. CROPS uses a blocking-resistant P2P transport layer to protect against attacks based at the network layer which block individual storage nodes. Note that network-layer blocking may be based on numerous factors, such as the node IP address, keywords in DNS queries, protocol signatures, etc. "Darknets," or networks based on trust graphs, in which a node only directly connects to other nodes it trusts [36, 55], provide limited blocking resistance and source/identity hiding at the cost of efficient routing, but are still potentially blockable by protocol signature or other unique network-level identifier [24, 63].[9]

[9] Systems such as Free Haven use a communication layer that provides client-server *unlinkability*, but this does not achieve the

System	Type	Attack resistance		Functionality		
		Targeted censorship	**Existential censorship**	**Plausible deniability**	**Robustness**	**Content discovery**
Free Haven	federated	replication	none	**secret sharing**	replication	OoB
Publius	federated	replication	none	encryption/ limited	replication	OoB
WikiLeaks	centralized	none	mirrors/ limited	none	mirrors/ limited	**efficient/ self-indexed**
Freenet	**P2P**	source hiding and replication	darknet/ limited	encryption/ limited	replication/ popularity	OoB or flooding
Tangler	federated	**replication and "entanglement"**	none	**secret sharing**	replication	OoB
OneSwarm	**P2P**	source hiding	darknet/ limited	none	none	flooding
Glacier	**P2P**	**EC and replication**	none	**encryption**	**EC and replication**	OoB
Tahoe-LAFS	federated	**EC and replication**	none	**encryption**	**EC and replication**	OoB
CROPS	**P2P**	**EC and replication**	**MCON**	**encryption**	**EC and replication**	**efficient/ in-band**

Table 1: A representative summary of many existing designs. "EC" denotes erasure coding and "OoB" stands for out-of-band communication. Attributes in bold are more desirable.

Otherwise, our approach is most similar to that of Tahoe-LAFS [69], the least-authority filesystem, and Glacier [33], a DHT-based P2P filesystem designed to be robust against *massive correlated failures*. However, *CROPS is not a filesystem* and requires alternate design choices. *CROPS does not have a notion of access permissions* — since the design aims for censorship resistance and ease of searching, confidentiality guarantees are out of scope, except to ensure plausible deniability. This simplification avoids most issues that plague long-term archival and/or collaborative secure storage [33, 64, 69]. The design is further simplified by making files *immutable* — it allows us to use simple digital signature schemes to ensure integrity, authenticity, and publisher-continuity (if a publisher's pseudonym is, for instance, a hash of her public key, files encrypted and signed by the author are self-authenticating). *The major problems we face are file integrity, replication, key management, and content discovery*; distributed filesystems generally consider the last two issues out of scope. Interestingly, in the case of censorship resistant but discoverable storage, *key management and discovery are linked*, since keys and encrypted files are not useful without each other.

Infranet [28] partially addresses the enumeration-and-blocking problem by using steganographic techniques to hide content requests and responses. It is fully decentralized, but requires limited cooperation of a number of web servers. While increasing the computational load on censors, these systems are nonetheless detectable and enumerable when adversaries become aware of their existence or proactively look for them. In [29], Feamster et al. extend Infranet to add a layer of indirection in the form of untrusted messengers, who pass requests to a forwarder, who then fetches the actual censored content. This serves as an excellent example of using transport providers to connect clients to storage-only systems. Another example is Collage [7], which uses covert channels to bypass censorship, leveraging any website that hosts user-contributed content

blocking resistance properties gained when using *identity hiding*.

to embed covert messages in cover traffic. This approach focuses on content availability, and complements discovery of the communication channels (e.g. [24, 29, 57]), but does not itself provide discovery, limiting usefulness. Moreover, as noted before, blocking resistant transport alone is not sufficient to build effective censorship resistant systems.

6 Conclusion

This paper described CROPS, a censorship resistant overlay publishing system designed to provide flexible and robust peer-to-peer storage. It is designed to resist strong adversaries (on the level of nation-states) that are willing to block most communication, short of shutting down the entire Internet. CROPS combines an unblockable communication layer with a publishing mechanism which separates file content, decryption keys, and metadata, and stores them at different locations, and offers robust and highly available storage, stronger-than-plausible deniability, and easy human-language content discovery without explicit indexing. CROPS also provides publisher anonymity, support for curated (vouched) content, and a self-cleaning system to mitigate pollution attacks, purging dead files from the network automatically. A key benefit of CROPS is that very few honest nodes are needed to successfully maintain a high file replication factor even in the presence of high member churn.

Through simulation and implementation we showed that CROPS is practical, simple to implement, highly robust even in low-reliability environments, and can support massive amounts of stored content (2^{60} blocks, regardless of size) *with negligible content loss*, even at node failure rates up to 70%.

Future work. We are working on combining CROPS with a deployed implementation of MCON to measure their simultaneous performance, but suspect that due to the high latency of our current evaluation testbed, CROPS performance measurements given in this paper will be within an order of magnitude of the performance of the entire CROPS+MCON system. Furthermore, performance limitations can be partly hidden from users by implicitly setting performance expectations by employing a particular user interface style. The "browser" paradigm is clearly a bad fit due to expectations of near-instantaneous results, but an interface that mimics file sharing applications may prove to be more aligned with CROPS performance.

We are likewise improving the editing mechanisms of CROPS. While we will always need human editors, we can take steps to make their jobs as easy and efficient as possible. First, we require a mechanism for interested publishers to notify editors that a document has been submitted for their approval. Since the volume of content is likely to overwhelm a small community of dedicated volunteers, we need a secure way to extend the editor pool as well as revoke editing credentials of individuals who are no longer active. This needs to be done in a way that does not fall victim to the tyranny of the majority, so that even unpopular editors may have their say.

References

[1] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. Farsite: federated, available, and reliable storage for an incompletely trusted environment. In *OSDI*, 2002.

[2] P. Ahsan and D. Kundu. Practical data hiding in TCP/IP. In *ACM Workshop on Multimedia Security*, 2002.

[3] R. Anderson. The Eternity service. In *PRAGOCRYPT*, 1996.

[4] Azureus, now called Vuze : Bittorrent Client, 2011. `http://azureus.sourceforge.net/`.

[5] K. Bennett, C. G. T. Horozov, and I. Patrascu. Efficient sharing of encrypted data. In *ACISP*, 2002.

[6] BitTorrent. `http://www.bittorrent.com/`, 2009.

[7] S. Burnett, N. Feamster, and S. Vempala. Chipping away at censorship firewalls with user-generated content. In *USENIX Security*, 2010.

[8] D. Carvajal. Britain, a destination for "libel tourism". *The New York Times*. January 20, 2008.

[9] China 'blocks' iTunes music store. *BBC News*. August 22, 2008.

[10] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *SIGCOMM Computer Communication Review*, 33(3), 2003.

[11] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *PET*, 2000.

[12] R. Clayton, S. J. Murdoch, and R. N. M. Watson. Ignoring the Great Firewall of China. In *PET*, 2006.

[13] Conehead. Stego hasho. *Phrack*, 9(55), 1999.

[14] L. Cox and B. Noble. Samsara: Honor among thieves in peer-to-peer storage. *SIGOPS Operating Systems Review*, 37(5), 2003.

[15] T. Cranton. Cracking down on botnets. `http://blogs.technet.com/microsoft_blog/archive/2010/02/25/cracking-down-on-botnets.aspx`. February 25, 2010.

[16] Creative chinese dodge censors to search for 'uncle jiang'. *The Indian Express*. July 7, 2011.

[17] G. Danezis and R. Anderson. The economics of censorship resistance. In *WEIS*, 2004.

[18] R. J. Deibert, J. Palfrey, R. Rohozinski, and J. Zittrain. ONI Home Page — OpenNet Initiative, 2009. `http://opennet.net/`.

[19] R. J. Deibert, J. G. Palfrey, R. Rohozinski, and J. Zittrain. *Access Denied: The Practice and Policy of Global Internet Filtering (Information Revolution and Global Politics)*. 2008.

[20] R. J. Deibert, J. G. Palfrey, R. Rohozinski, and J. Zittrain. *Access Controlled: The Shaping of Power, Rights, and Rule in Cyberspace*. 2010.

[21] R. J. Deibert and N. Villeneuve. *Firewalls and Power: An Overview of Global State Censorship of the Internet*. 2004.

[22] R. Dingledine. Tor and circumvention: Lessons learned. The 26th Chaos Communication Congress, 2009.

[23] R. Dingledine, M. J. Freedman, and D. Molnar. The Free Haven project: Distributed anonymous storage service. In *PET*, 2000.

[24] R. Dingledine and N. Mathewson. Design of a blocking-resistant anonymity system. Technical report, 2006.

[25] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *USENIX Security*, 2004.

[26] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *HotOS*, 2001.

[27] eMule Team. eMule project, 2009. `http://www.emule-project.net/`.

[28] N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. Karger. Infranet: Circumventing web censorship and surveillance. In *USENIX Security*, 2002.

[29] N. Feamster, M. Balazinska, W. Wang, H. Balakrishnan, and D. Karger. Thwarting web censorship with untrusted messenger delivery. In *PET*, 2003.

[30] A. Fiat and J. Saia. Censorship resistant peer-to-peer content addressable networks. In *SODA*, 2002.

[31] Freedom on the Net 2011: A global assessment of internet and digital media, April 2011.

[32] L. Greenemeier. How has WikiLeaks managed to keep its web site up and running? Scientific American. December 7, 2010.

[33] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *NSDI*, volume 5, 2005.

[34] J. Hendricks, G. R. Ganger, and M. K. Reiter. Low-overhead byzantine fault-tolerant storage. In *SOSP*, 2007.

[35] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling. Measuring and detecting fast-flux service networks. In *NDSS*, 2008.

[36] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Friend-to-friend data sharing with OneSwarm. Technical report, University of Washington, 2009. `http://oneswarm.cs.washington.edu/f2f_tr.pdf`.

[37] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An architecture for global-scale persistent storage. *ACM SIGPLAN Notices*, 35(11), 2000.

[38] D. Kügler. An Analysis of GNUnet and the Implications for Anonymous, Censorship-Resistant Networks. In *PET*, 2003.

[39] T. MacDermid. Stegtunnel, 2008. http://www.synacklabs.net/OOB/stegtunnel.html.

[40] P. Maniatis, M. Roussopoulos, T. Giuli, D. S. H. Rosenthal, and M. Baker. The LOCKSS peer-to-peer digital preservation system. *ACM Transactions on Computer Systems*, 23(1), 2005.

[41] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In *IPTPS*, 2002.

[42] S. J. Murdoch and S. Lewis. Embedding covert channels into TCP/IP. In *Information Hiding Workshop (IH)*, 2005.

[43] A. Mushtaq. Smashing the Mega-d/Ozdok botnet in 24 hours. http://blog.fireeye.com/research/2009/11/smashing-the-ozdok.html. November 11, 2009.

[44] I. Osipkov, P. Wang, N. Hopper, and Y. Kim. Robust accounting in decentralized P2P storage systems. In *ICDCS*, 2006.

[45] Pakistan blocks YouTube website. *BBC News*. February 24, 2008.

[46] G. Perng, M. K. Reiter, and C. Wang. Censorship resistance revisited. In *Information Hiding Workshop (IH)*, 2005.

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

[48] B. C. Popescu. Safe and private data sharing with Turtle: Friends team-up and beat the system. In *Cambridge International Workshop on Security Protocols*, 2004.

[49] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In *FAST*, 2002.

[50] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2), 1989.

[51] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. *SIGCOMM Computer Communication Review*, 31(4):161–172, 2001.

[52] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: a public DHT service and its uses. *SIGCOMM Computer Communication Review*, 35(4), 2005.

[53] C. H. Rowland. Covert channels in the TCP/IP protocol suite. *First Monday*, 2(5), 1997.

[54] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.

[55] O. Sandberg. Distributed routing in small-world networks. In *ALENEX*, 2006.

[56] Secret US Embassy Cables (Cablegate), 1966-2010. http://mirror.wikileaks.info/wiki/Secret_US_Embassy_Cables_(Cablegate),_1966-2010, November 2010.

[57] Y. Sovran, A. Libonati, and J. Li. Pass it on: Social networks stymie censors. In *IPTPS*, 2008.

[58] C. Stillwell. Libel tourism: Where terrorism and censorship meet. *San Francisco Chronicle*. August 29, 2007.

[59] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *SIGCOMM*, 2001.

[60] M. W. Storer, K. M. Greenan, E. L. Miller, and K. Voruganti. POTSHARDS: secure long-term storage without encryption. In *USENIX Technical*, 2007.

[61] A. Subbiah and D. M. Blough. An approach for fault tolerant and secure data storage in collaborative work environments. In *StorageSS*, 2005.

[62] United Nations General Assembly. The Universal Declaration of Human Rights. December 10, 1948.

[63] E. Y. Vasserman, R. Jansen, J. Tyra, N. Hopper, and Y. Kim. Membership-concealing overlay networks. In *ACM CCS*, 2009.

[64] M. w. Storer, K. Greenan, and E. L. Miller. Long-term threats to secure archives. In *StorageSS*, 2006.

[65] M. Waldman and D. Mazières. Tangler: a censorship-resistant publishing system based on document entanglements. In *ACM CCS*, 2001.

[66] M. Waldman, A. Rubin, and L. Cranor. Publius: A robust, tamper-evident, censorship-resistant and source-anonymous web publishing system. In *USENIX Security*, 2000.

[67] P. Wang, J. Tyra, E. Chan-Tin, T. Malchow, D. F. Kune, N. Hopper, and Y. Kim. Attacking the Kad network — real-world evaluation and high-fidelity simulation using DVN. *Security and Communication Networks*, 2009.

[68] WikiLeaks. `http://wikileaks.org/`, 2008.

[69] Z. Wilcox-O'Hearn and B. Warner. Tahoe: the least-authority filesystem. In *StorageSS*, 2008.

[70] D. Worth. CÖK: Cryptographic one-time knocking. In *Black Hat USA*, 2004.

[71] J. J. Wylie, M. W. Bigrigg, J. D. Strunk, G. R. Ganger, H. Kiliçcöte, and P. K. Khosla. Survivable information storage systems. *Computer*, 33, 2000.

[72] J. Ye and G. A. Fowler. Chinese bloggers scale the 'great firewall' in riot's aftermath. *The Wall Street Journal*. July 2, 2008.

[73] B. Zhao., H. Ling, J. Stribling, S. Rhea, A. Joseph., and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), 2004.

[74] J. Zittrain and B. Edelman. Internet filtering in China. *IEEE Internet Computing*, 7(2), 2003.

A Appendix

A.1 Ad-hoc approaches to censorship resistance

Unfortunately, many naïve schemes have been deployed in repeated attempts to create censorship-resistance systems. Such systems tend to focus on thwarting specific censorship techniques, and can be easily defeated if deployed widely-enough to be noticed by the adversary (such as ignoring TCP RST packets [12]). These systems include:

Fast flux and domain flux. Phishing sites have been known to use DNS fast flux [35] (returning different IP addresses for repeated DNS queries to the same domain) or domain flux to contact different domain names used as control centers, which change over time. These strategies are easily overcome by either shutting down the DNS server [15, 43], taking over current or future domains [15, 43], or blocking all the domain queries or URLs at the ISP level.

Steganography. There is extensive literature on TCP/IP steganography and covert channels, including HTTP-embedded messages and tunneling information through DNS queries [2, 13, 39, 42, 53, 70], which can be used when accessing censored content. However, Murdoch and Lewis [42] show that many of these proposals are easily detected. The location of censored documents has to be somehow disseminated, giving the adversary an easier target. While steganography increases the effort of a potential censor, such schemes nonetheless fail if an adversary looks for their existence. Keyed steganography potentially falls victim to trivial insider attacks unless complicated key distribution schemes are used, which brings us back to the dissemination and distribution problem, which affects cryptographic keys and node identities.

File-sharing networks. Likewise, using general-purpose file-sharing networks (e.g. Kad [27] or BitTorrent [6]) or public document storage sites (e.g. Google Docs) for censorship resistance is not secure against malicious insiders who can either block access to content [74] or disrupt the entire file sharing network [67]. More recently we have seen the rise of applications for "friend-to-friend" (F2F) sharing [36, 55]. They are meant to allow sharing recursively through trusted intermediaries, preventing the disclosure of the uploader's identity. They are fundamentally different from previous designs since they hide the network member set even from other network members. Unfortunately, because links are based on trust relationships, they generally suffer scalability issues and inefficient search.

A.2 Erasure coding and robustness

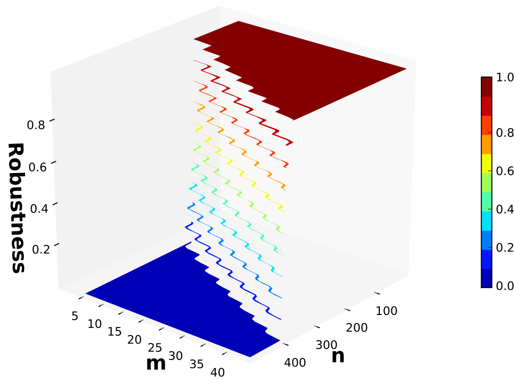

Figure 7: Determining the optimum erasure-coding variables to support up to 2^{60} stored chunks. n and m are erasure code parameters, uniformly sampled from $[1, 5]$ and $[5, 500]$, respectively. The Y axis is the robustness ρ of a censorship-resistant system, showing the fraction of nodes that must be offline or malicious before data loss begins to occur.

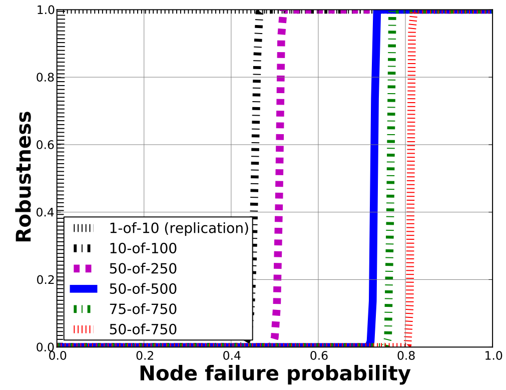

Figure 8: The robustness ρ of the censorship-resistant system using a given erasure code configuration or simple replication. The 50-of-500 configuration is a good tradeoff between overhead and robustness, as is 75-of-750. Both impose a factor of 10 storage overhead.