2013 42nd International Conference on Parallel Processing

Freeweb: P2P-Assisted Collaborative Censorship-Resistant Web Browsing

Haiying Shen[1], Alex X. Liu[2], Lianyu Zhao[1]
[1]Dept. of ECE, Clemson University,Clemson, SC 29631
[2]Dept. of CSE, Michigan State University, East Lansing, MI 48824
{shen, lianyuz}@clemson.edu, alexliu@cse.msu.edu

***Abstract*—In many countries, the Internet is under stringent censorship for political or religious reasons which severely undermines the free flow of information. A censorship-resistant web browsing system must be scalable, blocking resistant, and tracing resistant. However, current censorship-resistant web browsing systems, which use a group of dedicated proxies to bypass censorship, fail to meet these requirements. To tackle these challenges, we propose Freeweb, which relies on widely-distributed peer-to-peer (P2P) nodes in a decentralized manner rather than specified proxies in a centralized manner. Freeweb is built on top of a Distributed Hash Table (DHT)-based P2P network, where nodes not under censorship help nodes under censorship to access blocked webpages. Freeweb has a web browser front-end whose user interface resembles existing web browsers. The underlying complex process of retrieving blocked webpages is therefore hidden from users. We implemented and open-sourced Freeweb and conducted extensive real-world experiments on PlanetLab. The experimental results show that Freeweb has a high success rate and reasonable browsing latency.**

***Keywords*-Censorship-Resistant; Web Browsing; Peer-to-Peer Network; Distributed Hash Table**

I. Introduction

A. Background and Motivation

The World Wide Web (WWW), containing around 160 million web sites [4], has become the most popular means of publishing and accessing information. Unfortunately, a long list of countries deem this vast amount of free and heterogeneous information as a threat and therefore vigorously block the sites that may publish content that is sensitive to government interests. Such Internet censorship has severely undermined the free flow of information. For example, the Great Firewall of China blocks most western news sites and many popular Web 2.0 sites (such as Facebook, Twitter, Youtube, Flickr, Blogspot, WordPress, Typepad, Bebo, Imageshack, and even Wikipedia) [2], [3] because the opinions expressed by some users may be offensive. The search engines in China, such as google.cn, are required to perform censorship by filtering out certain sensitive information from their search results. Some search engines, such as Bing and Live, are simply blocked by the Great Firewall of China because they can be used to search for sensitive materials [2].

Commonly used Internet censorship methods include, but are not limited to, (1) IP blocking, which means to block certain IP addresses such as all the IP addresses allocated to a particular site, (2) DNS filtering and redirection, which denies DNS requests to blocked sites or returns incorrect IP addresses for such requests, (3) content filtering, which drops packets whose payload contains sensitive keywords, (4) web feed blocking, which blocks incoming URLs starting with the words “rss”, “feed”, or “blog”, and (5) URL filtering, which blocks requested URLs that contain sensitive keywords.

B. Problem Statement and Challenges

We aim to build a system that allows users who are under Internet censorship to access blocked web sites. Such a system must satisfy the following three requirements. First, the system must be *blocking resistant*, which means that it is very difficult for censors to block the web browsing service provided by this system. This requirement is critical because it ensures the long-term viability of such a service. Second, the system must be *tracing resistant*, which means that it is very difficult to trace end users who request to access blocked webpages. This requirement is critical to ensure the safety of the users of this system because the governments practising internet censorship may punish citizens who violate such censorship with imprisonment or other sanctions. Third, this system must be *scalable*, which means that the system must scale to a large number of users with no significant performance degradation.

Current censorship-resistant web browsing systems fail to meet these three requirements. The systems [8], [9], [12], [15], [17], [23] use a group of dedicated proxies to access webpages for censored nodes in order to bypass censorship. However, any solution based upon a limited number of permanent proxies is not blocking resistant and tracing resistant because the proxies can be easily detected and blocked by censors. Also, the centralized approach, in which a limited number of proxies serve a large number of censored nodes, prevents the systems from achieving high scalability. Building a censorship resistant web browsing system that can meet these three requirements has become a formidable challenge.

C. Our Approach

To deal with the challenge, we propose Freeweb, the first peer-to-peer (P2P) approach to censorship-resistant web browsing. The basic idea of Freeweb is to allow the P2P nodes in uncensored regions serve as the proxies of the P2P nodes in censored regions for accessing blocked web sites. Specifically, the URL of a webpage will be used as the search key in the Distributed Hash Table (DHT)-based P2P network. The user interface of this system is a Firefox- or Internet Explorer-like web browser. After a user types in a URL in the address bar of this special browser, the URL, as a search key, traverses the

P2P network and finally reaches a node that can access the URL. This destination node will act as a proxy for this URL request: obtaining the page and sending the page back to the URL requester through a tunnel. In this work, we design and implement Freeweb based on a DHT structured P2P network [24]. It is not very difficult to adapt Freeweb to unstructured P2P networks.

Freeweb meets the above three requirements. First, Freeweb achieves blocking resistance because every node in an uncensored region may serve as a proxy for some URLs at some time in this P2P network, and blocking all such nodes is practically infeasible. Attempts to block the distribution of the Freeweb software can be defeated by distributing Freeweb in existing P2P networks and dynamically changing bootstrapping nodes. Second, Freeweb achieves tracing resistance because Freeweb encrypts URL requests and does not distinguish request initiators and request forwarders. Tracing the request initiators is therefore impractical. Third, Freeweb achieves high scalability because the massive number of nodes from uncensored regions and its decentralized manner allow Freeweb to scale to a large number of users.

We argue that P2P is the right approach to enable free web browsing in censored areas. To browse blocked sites, proxies are inevitably needed. The number of proxies has to be massive; otherwise, they can be identified and blocked by censors, and fail to support a scalable system. Furthermore, the massive number of proxies have to be self-organized because any central management server can be identified and blocked by censors. Thus far, P2P is the leading technology to achieve such a massive self-organized network.

D. Key Contribution

First, we propose a censorship-resistant web browsing system that achieves the goals of blocking resistance, tracing resistance, efficient transmission, and high scalability. Second, we implemented Freeweb and open-sourced Freeweb at http://freewebcu.sourceforge.net/. We further conducted extensive experiments on PlanetLab [5]. The experimental results show that Freeweb has a high success rate and reasonable browsing latency.

The rest of the paper proceeds as follows. We first examine prior work in Section II. In Section III, we present the design and implementation of Freeweb. In Section IV, we discuss a number of issues in Freeweb and analyze its reliability. We present experimental results in Section V. Finally, we give concluding remarks in Section VI.

II. Related Work

Prior censorship-resistant web browsing systems all use a group of dedicated proxies to bypass censorship. Web MIXes (JAP) [8] is based on MIXes [9], which provides anonymity by relay forwarding. In JAP, a node waits until it has received a certain number of messages, then mixes them up before forwarding them. In this way, it hides the sender and the receiver from an eavesdropper on network traffic. Kopsell *et. al* proposed an add-on [17] to enable volunteers to act as forwarders in the MIXes by letting them register in the centralized information server, but it still requires a set of proxy servers. Infranet [12] uses commercial web sites as proxies. In Infranet, a user encodes a series of normal HTTP requests into a covert request and sends it to a responder. The responder retrieves the required content from the web server and uses steganography to encode forbidden content into harmless images, thus ensuring the deniability of users. Kaleidoscope [23] limits every user's knowledge to a small and consistent proportion of all the proxies; however, due to the limited number of proxies, all proxies can still be detected after a number of tries. Proxify [6] does not require users to install software; although convenient, it can be easily defeated through URL filtering. Psiphon [15] allows volunteer users to register their computers as proxies, which can dynamically increase the number of proxies; however, Psiphon still relies on a central server that can be easily blocked. Furthermore, because a proxy directly connects with both a blocked site and a URL requester, a censor may "volunteer" their computers as proxies. Thus, Psiphon is not tracing resistant.

Prior censorship resistant web browsing systems all have difficulty achieving blocking resistance, tracing resistance, and scalability due to the use of a small number of fixed proxies that can be easily blocked by censors. Psiphon tries to increase the number of proxies, but it cannot prevent its central server from being blocked. Infranet tries to provide tracing resistance by steganography, but censors can easily hunt users down by monitoring accesses to the small number of fixed proxies. Tor [10], Tarzon [13] and MorphMix [21] provide anonymity services using proxies or P2P networks. OneSwarm [14] provides file sharing users privacy by enabling users have configurable control over the amount of trust they place in peers and in the sharing model for their data. Unlike these works, FreeWeb mainly focuses on censorship circumvention though it also offers anonymity.

III. Freeweb Design and Implementation

A. Assumptions and Threat Model

We envision Freeweb as consisting of a massive number (tens of thousands to millions as in a P2P network) of joined nodes. The nodes are censored in some areas while uncensored in other areas. In Freeweb, nodes in an uncensored area help nodes in a censored area to retrieve blocked webpages. In this process, we refer to the uncensored node as the *server* and the censored node as the *client*. Some uncensored nodes all over the world may volunteer to help censored nodes to access blocked web sites. This is true because in uncensored regions (such as North America), people usually have extra computing and networking resources to share [7] and they are willing to help others for a good cause. Also, we can provide incentives, such as virtual credits, to encourage nodes to provide the service. The virtual credits can be used to buy magazines, games, and movies. The servers earn virtual credits from clients, and nodes can buy virtual credits using real money.

Since providing protection against a strong censor is not feasible in designing a low-latency system [10], we assume the censor is capable of controlling only a fraction of all traffic and nodes in the network. Traffic control includes analyzing, intercepting, generating and deleting traffic. Node control includes deploying a number of censor operated nodes in Freeweb and/or compromising a fraction of all nodes in the network. These assumptions hold in reality because, after all, censors have limited resources.

The censor's expected actions mainly include (1) Identifying the initiator of a communication (*client*), since the law where the censor is located may view such clients as illegal. Even with encrypted data packets, the censor can still find out clients by identifying traffic patterns. We aim to protect clients from a *traffic analysis* attack. (2) Identifying the nodes that provide service. Since such nodes are usually located outside the censor's traffic control range, the censor can only determine their existence by analyzing the intercepted packets and blocking the traffic containing their IPs. (3) Passive traffic analysis and active traffic controlling. Censors can mimic/hijack typical service requesters to send out massive requests to the network or act like functional service providers to collect overhead traffic. They can analyze or correlate the traffic to identify abnormality. Moreover, censors can mimic nodes or cooperate with compromised nodes in generating false information, traffic redirecting, or even running Sybil [11] or Eclipse [22] attacks.

B. Overview of Freeweb

Figure 1 shows a high-level overview of the workflow in Freeweb. The client and server pair are nodes in the DHT network. Firstly, the client sends out its request, including a URL, an onion path, and a symmetric key K_s, with the URL's hashed value as the destination (Step 1). Via DHT routing, the request is forwarded towards its destination (Step 2). Once the request reaches an intermediate node that can serve as the server or its destination that serves as the server, the server node stores the request into its *Request Pool*. From the pool, the server fetches the URL request and then fetches the webpage specified by the URL (Step 3). After a webpage is retrieved (Step 4), the server compresses the webpage into one file, encrypts the compressed file using the K_s in the request (Step 5), and sends the file back along the onion path (Steps 6&7). Finally, the client decrypts and decompresses the received file, and displays the webpage in the Freeweb web browser (Step 8).

In the following, we first briefly describe DHT networks and then present the details of the design and implementation of Freeweb in terms of two aspects: request forwarding and reply forwarding.

C. Introduction of DHT Networks

We build Freeweb on the Chord DHT [24], although any DHT network is applicable. DHTs are a class of decentralized systems in the application level that partition ownership of a set of objects among participating nodes and efficiently route messages to the unique owner of any given object. In a DHT network, peers function as clients as well as servers. A DHT provides two main functions: `Insert(key,object)`, to store an object in its owner node, and `Lookup(key)`, to retrieve the object by the DHT routing algorithm. Each object has a key that is the consistent hash [16] value of the object's name; similarly, each node has a key that is the consistent hash value of the node's IP address. Each node maintains a routing table of $\log N$ size to store its neighbors in the network, where N is the number of nodes in the system. A DHT provides $O(\log N)$ lookup time complexity. DHTs excel in scalability and reliability. To maintain topology in node dynamism, including node joins, departures, and failures, DHTs use stabilization and self-organizing mechanisms. Specifically, each node periodically updates its successor, predecessor, and its neighbors in its routing table. Please refer to [24] for the details of the DHT networks.

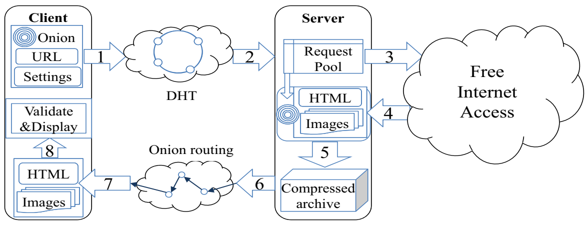


Figure 1: Workflow in Freeweb.

D. Design Details of Freeweb

In Freeweb, each node generates a public key K_{pub} and a private key K_{pri} upon joining. Each node exchanges its public key, IP address, and port number with its neighbors in its routing table. The DHT routing table of a node in Freeweb has one additional column for storing the public key of each neighbor of the node. Thus, in DHT routing in Freeweb, a request is always forwarded from a node to its neighbor in its routing table whose public key is known to the node. For further optimization, a node can use the public keys to establish a symmetric key with each neighbor; however, public key cryptography is much more computing expensive than symmetric key cryptography. This symmetric key can be periodically refreshed for better security.

1) Request Forwarding: Because a webpage can be uniquely identified by its URL, we use a URL as the search key for a webpage. Figure 2 illustrates the process of a requester finding a service provider. When a censored node c wants to access a URL u, c first applies the consistent hash function h used by the DHT to URL u; it then sends its request r using $h(u)$ as the search key (i.e., destination key) in the DHT by `Insert(key,object)`, where `object` is r encrypted by the public key of the next hop. The next hop decrypts r using its private key, then encrypts r using the next hop's public key before forwarding the message to the next hop. The request will finally arrive at the node that is the owner of $h(u)$, denoted s_1. If s_1 is uncensored, s_1 retrieves the webpage as a file, compresses the file, and sends it back to c along an onion routing path specified by c in its request; if s_1 cannot access URL u, s_1 uses $h^2(u) = (h(h(u)))$ as the new search key for this request and forwards the request to the owner node s_2 of $h^2(u)$. This process repeats v times until the owner node of $h^v(u)$ is able to access URL u; then, this node serves client c by accessing u for c. Thus, in Freeweb, the owner node of $h^v(u)$ always act as the server of the URL u. The deterministic routing provided by the DHT lookup function ensures that nodes find the webpage server. The server node caches its retrieved webpage in order to serve the subsequent requests within a certain time window without fetching the webpage again. We call the above request forwarding scheme a *deterministic* scheme. Always relying on one server to access a webpage may overload the server and make it traceable. Thus, we extend the *deterministic* scheme to an *opportunistic* scheme by complementing it with an additional algorithm. Recall that a message needs to travel a number of hops before arriving at its destination. While traveling, the message may arrive at an uncensored node that can access u. At this time, the travel terminates, and this intermediate node acts as the server for the request to access u from node c.

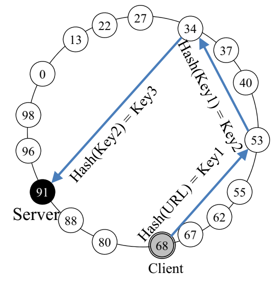


Figure 2: Request forwarding.

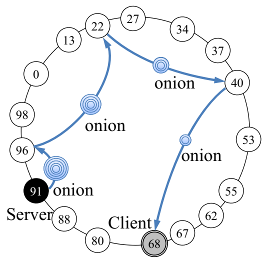


Figure 3: Reply forwarding.

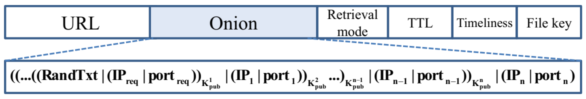
Next, we introduce the way in which a node generates its request. When a client c initiates a request r, c needs to specify the reply path for the server s to send the retrieved webpage back. To form a reply path, client c can use the nodes in its routing table. Suppose that the specified reply path is $a \to b \to c$. Thus, the full reply path is $s \to a \to b \to c$. To preserve c's anonymity, server s should only know that it needs to send the reply to a but does not know the rest of the path. Similarly, node a should only know that it needs to send the reply to b and node b only knows that it needs to send the reply to c. To achieve this goal, we use the technique of onion routing [10], [20]. After c generates the reply path $a \to b \to c$, it includes the following onion in the request r:

$$((random|IP_c|Port_c)_{K^b_{pub}}|IP_b|Port_b)_{K^a_{pub}}|IP_a|Port_a.$$

In the onion, $random$ (e.g., $23abcABC$) is a string randomly generated by c. Later on, when c retrieves string "23abcABC" from a reply message, it knows that the string was generated by itself and hence the reply is for itself.

There are two modes in webpage fetching: text-only mode and full-page mode. In the text-only mode, the server only fetches the plain text of the HTML page specified by the URL. In the full-page mode, the server fetches the rich content of the HTML page specified by the URL including images, etc. We include the text-only mode in Freeweb because many times the reader may be mainly interested in the text content of a webpage, and getting only the text content is faster than getting the full page. The requester specifies a fetching mode in its request.

Figure 4 shows the format of the request message. The *URL* field stores the requested URL. The *onion* field is a layered onion structure, with each layer containing the IP and port information of one onion routing hop encrypted by the proper public key. The *TTL* field indicates the maximum number of hops in a client-server path that a message is forwarded before being discarded for prevention of perpetual looping. The *retrieval mode* field indicates the webpage retrieval mode: text-only mode or full-page mode. The *timeliness* field is used only in the case when a requested webpage is in a node's cache. It is a time period specified by the client to require that the cached period of the webpage does not exceed the specified value. The last *file key* field is the symmetric key K_s generated by c and will be used by the server to encrypt the retrieved webpage.


URL
Onion
Retrieval mode
TTL
Timeliness
File key
$((...((\mathbf{RandTxt}\,|\,(\mathbf{IP}_{req}\,|\,\mathbf{port}_{req}))_{K^1_{pub}}\,|\,(\mathbf{IP}_1\,|\,\mathbf{port}_1))_{K^2_{pub}}\,...)_{K^{n-1}_{pub}}\,|\,(\mathbf{IP}_{n-1}\,|\,\mathbf{port}_{n-1}))_{K^n_{pub}}\,|\,(\mathbf{IP}_n\,|\,\mathbf{port}_n)$
Figure 4: Request message format.

In the Freeweb implementation, each node provides the service of sending/forwarding a request by `SendRequest(URL, onionPath, retrievalMode, TTL, Timeliness)`. This function provides a customized request interface for a web browsing service. Every parameter corresponds to one field in a request message. The function `SendRequest()` uses the public key of the next hop to encrypt all the fields then executes the DHT function `Insert(key,object)`, where `key` is the hashed value of the URL and `object` is the encrypted message. For example, when request r is passing from node A to its neighbor node B through `SendRequest()`, node A encrypts r using node B's public key. After node B receives the encrypted request $(r)_{K^B_{pub}}$, node B decrypts it using its private key K^B_{pri}. If node B cannot serve the request, it needs to forward the request to node C. Node B then encrypts r using node C's public key and sends $(r)_{K^C_{pub}}$ to node C through `SendRequest()`. Node C repeats the same process.

2) Reply Forwarding: In each request originating from client c, c specifies the reply path for the server node of this request to send the retrieved webpage back to c. Figure 3 shows the process of reply forwarding. After server s retrieves the webpage of URL u as a file f, s encrypts f using key K_s, which is generated by client c and included in the request. Thus, after receiving $(f)_{K_s}$, c can obtain the webpage f using key K_s. Server s sends the webpage $(f)_{K_s}$ along the specified reply path in the request back to

c. Recall that the path is encrypted using the onion routing technique. The onion included in the request indicates that server s needs to send $(f)_{K_s}$ to node a with IP address IP_a at port $Port_a$. After node a receives $(f)_{K_s}$ along with $((random|IP_c|Port_c)_{K^b_{pub}}|IP_b|Port_b)_{K^a_{pub}}$, node a peels off one layer of the onion by decrypting the onion using its private key. Then, a knows that it needs to forward the message $(f)_{K_s}$ to node b with IP address IP_b at port $Port_b$. After node b receives $(f)_{K_s}$ along with the onion $(random|IP_c|Port_c)_{K^b_{pub}}$, it peels off one more layer of the onion by decrypting the onion using its private key. Then, b knows that it needs to forward the message $(f)_{K_s}$ to node c with IP address IP_c at port $Port_c$. After node c receives $(f)_{K_s}$ along with the onion $random$, it notices that it is the final receiver according to its generated $random$ and then uses key K_s to obtain webpage f.

Consequently, server node s and every intermediate node in the reply path only know the next node that they should forward the reply to and do not know the final receiver, *i.e.*, the request originator. In this way, the identity of the request originator is kept confidential. An alternative method of reply forwarding is to let the webpage traverse the reverse path that the request follows; however, this path may be unnecessarily long for a reply path. The number of hops that the request traverses before reaching the server is $O(v \log n)$, where v is the number of times that the owner node is unfortunately under censorship. Because c specifies a reply path, the length of the reply path can be a constant, which is more efficient than using a path of length $O(v \log n)$.

E. Implementation

Freeweb is built upon OpenChord, an open source implementation of the Chord DHT. In the current version of Freeweb, we mainly focus on web browsing functionalities. We have not yet implemented other complex functions such as POST, PUT, and video streaming support. We plan to add these functionalities to Freeweb in the future version. Freeweb provides APIs to store all serializable Java objects in a DHT network. For ease of deployment, we developed two versions of Freeweb: a Windows-based version and a Linux-based version. Freeweb consists of about 5000 lines of Java code in addition to the OpenChord infrastructure. We have put our source code on the open source repository sourceforge at http://sourceforge.net/projects/freewebcu/.

Freeweb has three additional layers built on top of OpenChord as shown in Figure 5. The first layer is *DHT Server*. It provides customized APIs based on the "raw" APIs of Openchord for object storage and fetching, as well as operations that deal with node dynamism. The second layer contains two modules: a *network daemon* and a *communication management module*. The network daemon module constantly monitors the change of the request pool, taking necessary actions such as forwarding or providing service. The communication management module takes care of TCP-based communication, including public key dissemination and webpage distribution. The third layer provides cryptography primitives, cache management, and APIs for HTTP accesses.

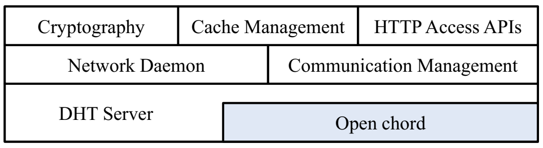


Figure 5: Freeweb modules.

```
1  while ! Interrupted do
2      Set entrySet = getRequestPool() ;
3      for Each entry in entrySet do
4          if canAccess(entry.getURL()) then
5              String onion = entry.getOnion();
6              String IP = entry.getIP();
7              Integer port = entry.getPort();
8              Integer $K_s$ = entry.getKs();
9              Update all cache mapping information;
10             if There is a cached package in download cache
               within the required entry.timeliness then
11                 Use URL to locate the package;
12                 Read object zipObj from the package;
13             else
14                 Download files and zip to package using the
                   URL;
15                 Read object zipObj from the downloaded
                   package;
16                 Add this package to download cache;
17             Archive archive = $((zipObj)_{K_s})|onion$;
18             sendContent(IP, port, archive);
19         else
20             if entry.getTTL()> 0 then
21                 reInsertRequest(entry);
22         remove(entry);
23     Thread.sleep(T);
```

Algorithm 1: Daemon.start()

1) Network Daemon: The *Network Daemon* periodically scans the *Request Pool* and serves the stored requests in a first-in-first-out order. Algorithm 1 shows how the Network Daemon works. It cycles to look for new requests and performs web accesses until interrupted by the user. For each request, the IP and port information (code lines 6-7) of the requester is retrieved, and then the fetched webpage is sent to the requester (code line 18). If the requested webpage is in the cache, the corresponding package is fetched from the cache (code lines 10-12). Otherwise, the requested package is downloaded from the URL (code lines 14-16). If the requested URL cannot be accessed for a predefined period of time, the request will be inserted back into the request pool (code lines 20-21). Note that this procedure is a simplified version of the actual procedure. For example, line 22 is used to remove the processed entry, but the actual execution of this command needs to avoid "concurrency modification" exception. Line 23 puts this thread to sleep for a time period of *T*, which is typically a few seconds. This step minimizes the CPU occupation in order to reduce the influence on other applications running on a volunteer computer; in practice, the response of Freeweb is not sensitive to this sleep interval.

2) Communication Management: The *Communication Management* module manages TCP communication to allow nodes to build TCP connections with other nodes. To make the application more stable, we allocate multiple TCP connections to each running communication management module.

```
1  Object recv = readRecv();
2  if recv is an instance of other nodes' public key then
3      Add this public key to public key pool;
4  if recv is an instance of Archive then
5      Write recv into local download cache folder;
6      Separate onion from recv;
7      String content = peelOnion(onion);
8      if getOnion(content) is the predetermined token then
9          Unzip recv and display the webpage;
10     else
11         IP = getIP(content);
12         port = getPort(content);
13         peeledOnion = getOnion(content);
14         Concatenate peeledOnion and recv (without original
           onion) to form archive
15         sendContent(IP, port, archive);
```

Algorithm 2: CommMgt.start()

Algorithm 2 shows how a communication manager handles a TCP connection. Currently, there are two different tasks performed by a TCP connection in Freeweb. One is the dissemination of public keys (code lines 2-3) and the other is file transmission on the reply forwarding path (code lines 4-15). In file transmission, after a node receives an instance of archive, it first stores the received package into its local download cache (code line 5). The node then separates the onion from it and peels (*i.e.*, decrypts) the onion (code lines 6-7). If the decrypted result is a predetermined token, this node is the webpage requester and the received webpage is displayed (code lines 8-9). If the decrypted result cannot be recognized, then this node must be a relay node on the reply forwarding path. It uses the *delimiter*, which is a 10 byte string in Freeweb, to divide the decrypted result into 3 parts: the IP address, the port information, and the peeled onion (code lines 11-13). Finally, this node sends the newly generated *archive* via TCP using the obtained IP and port number (code lines 14-15).

3) Cache Management: Freeweb employs cache management in order to reduce cost and speed up browsing speed. Freeweb builds a folder *cache/recv* in each node for webpage caching. Each node also maintains a *map* file that contains the information of every stored cached webpage file, including the file's DHT key, file name, creation time, and URL. A webpage file's DHT key, denoted `pkgID`, is a 128-bit unique key which is the consistent hash value of the file's name (*e.g.*, 90368311-b1d6-49c1-91c5-6a8ad5ad0f91). The unique file key helps to identify a requested webpage in a cache and avoids duplicated file caching operations for the same file.

Major APIs for cache management are listed in the following.

1) `AddPkgToCache(pkgID,URL)`. This function adds a webpage to the *cache/recv* folder. When a server finishes downloading a webpage for a requester or a requester receives a webpage, they put this webpage into their corresponding caching folder. The given *URL* and *pkgID* are used to record this package in the map file.
2) `UpdateMap(pkgID)`. When a cached file is created, deleted, or discarded, this function is used to make updates to the map file.
3) `GetPkgInCache(URL, Timeliness)`. This function calculates the time interval between the creation time of a cached file in the map file and the current system time. Only when the time interval is no more than the `Timeliness` will the cached file be returned.

4) HTTP Access APIs: HTTP access modules are used by a server to obtain webpages from webpage servers. To protect the client's identity, a server does not download JavaScript files embedded in the HTML files, because a node may expose its identity by executing such files. Freeweb aims to provide normal webpages with all essential elements in the webpages. Thus, a server retrieves important items for a webpage: (1) HTML source files, (2) all images, and (3) Cascading Style Sheets (CSS). Recall that a request message has a field called *retrieval mode* that indicates whether only HTML source files are desired or both HTML files and images are desired. This option enables users to choose higher browsing quality or higher browsing efficiency according to their preferences. Under most circumstances, users in censored areas only need the text information by choosing text-only mode in order to fetch the webpage quickly. The major APIs for HTTP accesses are listed below.

1) `AttemptURL(URL)`. Upon receiving a request, a server attempts to establish a connection with the requested webpage server for a few seconds, which is much less than the typical TCP timeout. If it is unable to make the connection, the server performs the rehashing and forwarding operations.
2) `DownloadHTMLPkg(URL, folderPath, pkgID, retrievalMode)`. A server uses this function to download the HTML source, image files, and CSS, which are put together into one file.

IV. Discussion and Analysis of Freeweb

A. Tracing Resistance

Note that a request r only contains a URL and the onion for sending the webpage back. Thus, request originators and request forwarders are not distinguishable. Because no node knows the request originator, Freeweb achieves tracing resistance on the request forwarding path. However, the censor may find communicating clients and servers through monitoring the inbound and outbound traffic of individual nodes and analyzing the traffic pattern [19]. Generally, the censor cannot monitor the traffic of uncensored nodes outside of its control regions. In regions where the censor can monitor the traffic of nodes, the nodes can defeat such traffic monitoring by padding traffic with dummy messages to camouflage the real traffic pattern. Dummy messages are identical to real messages, but are not accepted by nodes. For example, if request r is also sent to other, fake destinations, then the censor cannot detect the real destination.

Freeweb achieves tracing resistance on the reply forwarding path because of the onion routing technique; only the request originator knows who the final receiver is. Any intermediate node only knows to whom it should forward the message, because it can only see the outermost layer of the onion and cannot understand the content inside the onion.

B. Content Tampering Avoidance

When a censor joins Freeweb and receives a request, the censor may reply to a webpage with incorrect content or simply drop the request. Freeweb can defeat such attacks with redundant requests at a constrained additional cost. For example, when node c wants to access URL u, node c simultaneously sends out m ($m > 1$) requests: $u|1, u|2, \cdots u|m$ with hash values $h(u|1), h(u|2), \cdots, h(u|m)$, respectively. These m requests will be routed via different paths. After c receives m replies $f_1, f_2, \cdots, f_m$, it can use majority voting to choose the right reply. Because the number of nodes in the network is expected to be very large, the probability that all m paths contain a malicious censor node is ignorably small. This probability can be further minimized by carefully increasing the value of m. In addition to defeating malicious censor node attacks, this redundant request strategy helps to deal with communication failure caused by node dynamism (*i.e.*, node joining and leaving), and therefore increases the reliability of Freeweb. For a webpage that contains dynamic content, the multiple copies $f_1, f_2, \cdots, f_m$ that c receives may not match even if they are all genuine copies. To decide which copy to present to the user, c can compare the similarity between any two copies and choose a copy from the pair with the highest similarity.

C. Sybil and Eclipse Attack Discussion

Sybil attacks [11] on P2P networks are conducted by deploying a large number of a single adversary's identities and the adversary tries to occupy the routing table in other nodes to control the whole network. Tarzan [13] uses IP prefixes to avoid using nodes from the IPs with the same prefixes, which is also called *resource testing*. Therefore, the possibility of a Sybil attack is decreased. But Eclipse attacks [22], which utilize a small number of nodes with legitimate identities (or heterogeneous IP addresses), will spread more quickly than Sybil attacks by advertising only malicious nodes to Tarzan nodes when the Tarzan nodes try to discover new neighbors. Freeweb can incorporate resource testing used in Tarzan to prevent Sybil attacks. Moreover, Freeweb is naturally resistant to Eclipse attacks. This is because a node in Freeweb depends on the DHT policy for establishing new neighbors instead of asking for recommendations from other nodes. Thus, the Eclipse attack cannot spread quickly and the probability that a node's message in Freeweb encounters a malicious node only equals the percentage of malicious nodes in the node's routing table.

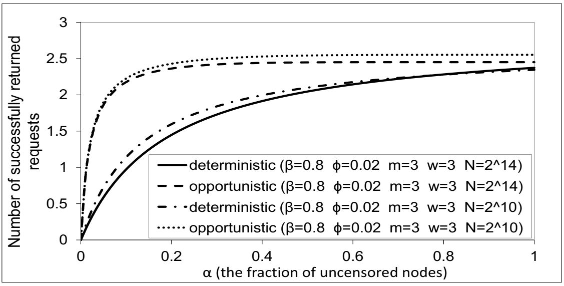


Figure 6: Performance of two forwarding schemes in Freeweb.

D. Reliability Analysis

Next, we analyze Freeweb's reliability in servicing web access requests. Let α be the fraction of uncensored nodes in the network, β be the fraction of nodes willing to provide their Internet connection resource, and φ be the fraction of nodes not willing to forward requests or webpages, which can be censors or any other malfunctioning nodes.

If we use the *deterministic request forwarding scheme* in Freeweb, for each request REQ, let $\hat{\Psi}$ ($\hat{\Psi} \in (0, m)$) be the expected number of requests that are served within the m duplicated requests. Therefore, the number of REQs that can traverse to the destination node D is $m(1-\varphi)^k$, where $k = \frac{1}{2}\log_2 N$ is the expected routing path length in Chord in the average case. If D is a censored node, then it rehashes REQ and sends it back to the network. The possibility that REQ is served under such a circumstance is $(1-\alpha)\frac{\hat{\Psi}}{m}$. If D is an uncensored node, it will satisfy this request with a probability of β, or otherwise re-send REQ to the network if it does not want to serve. In the latter case, the possibility that REQ is eventually served is $(1-\beta)\frac{\hat{\Psi}}{m}$. Therefore, we have

$$\hat{\Psi} = m(1-\varphi)^k \left\{ (1-\alpha)\frac{\hat{\Psi}}{m} + \alpha[\beta + (1-\beta)\frac{\hat{\Psi}}{m}] \right\}. \quad (1)$$

By solving $\hat{\Psi}$, we have

$$\hat{\Psi} = \frac{\alpha\beta m(1-\varphi)^k}{1-(1-\varphi)^k(1-\alpha\beta)}. \quad (2)$$

When a request is fulfilled, it will be sent back using the onion routing method with a probability of $(1-\varphi)^w$, where w is the number of relay nodes in the replying route. Based on the above analysis, we retrieve the number of expected successfully returned responses Ψ,

$$\Psi = \frac{\alpha\beta m(1-\varphi)^{k+w}}{1-(1-\varphi)^k(1-\alpha\beta)}. \quad (3)$$

If we use the *opportunistic request forwarding scheme* in Freeweb, a request REQ could be served *before* it arrives at the destination node with probability $1-(1-\alpha)^{k-1}$. Otherwise, it could be served with probability $(1-\alpha)^{k-1}$. Noticing such differences and applying similar analysis, letting $\hat{\Phi}$ be the expected number of requests that are served within the m duplicated requests, we have

$$\hat{\Phi} = m(1-\varphi)^k\{[1-(1-\alpha)^{k-1}]+ \tag{4}$$

$$(1-\alpha)^{k-1}\{(1-\alpha)\frac{\hat{\Phi}}{m}+\alpha[\beta+(1-\beta)\frac{\hat{\Phi}}{m}]\}\}. \tag{5}$$

By solving $\hat{\Phi}$, we derive

$$\hat{\Phi} = \frac{\{\alpha\beta(1-\alpha)^{k-1}+[1-(1-\alpha)^{k-1}]\}m(1-\varphi)^k}{1-(1-\varphi)^k(1-\alpha)^{k-1}(1-\alpha\beta)}. \tag{6}$$

By applying the probability that the backward onion routing process is successful, the expected number of successfully returned responses Φ is

$$\Phi = \frac{\{\alpha\beta(1-\alpha)^{k-1}+[1-(1-\alpha)^{k-1}]\}m(1-\varphi)^{k+w}}{1-(1-\varphi)^k(1-\alpha)^{k-1}(1-\alpha\beta)}. \tag{7}$$

We set $N = 2^{10}$ and $N = 2^{14}$ to simulate a medium and a large scale network. According to our assumptions in Section III-A, the fraction of nodes that are willing to provide web access service (β) is set to 8%. The fraction of nodes that do not forward any requests (φ), *e.g.*, censor nodes, is set to 2%. The number of duplicated requests is set to 3 to avoid content tampering.

Figure 6 shows the relationship between the fraction of uncensored nodes α and the number of successful replies for a request Ψ and Φ. When α is larger than 50%, we can observe that Ψ and Φ can both theoretically return more than 2 replies, so the users can benefit from content tampering avoidance (see Section IV-B). The performance of Freeweb becomes stable when $\alpha > 50\%$, which indicates Freeweb is highly usable in regions where even half of the nodes are censored. In addition, we can observe that Freeweb is scalable, since both Ψ and Φ in the medium- and large-scale networks are close.

V. Performance Evaluation

We chose PlanetLab [5] as the experiment testbed in order to evaluate the effectiveness of Freeweb on censorship circumvention in a real worldwide network environment. However, since each slice on PlanetLab is given very limited resources, the network transmission speed and encryption time is much longer than normal. Therefore, the performance presented in this paper reflects the performance of Freeweb in a harsh environment, and its performance in the practical network environment would be much better. The default test parameters are shown in Table I. Two content retrieval modes are used in the test: full-page and text-only. Full-page mode means that in addition to HTML files, image files of different formats (*e.g.*, GIF, PNG, JPG) and CSS files are also browsed. The length of the onion reply forwarding path (i.e., tunnel) is set to a maximum of 5 [18]. In the experiments, the default setting for the retrieval mode is text-only, the network size is 100 nodes, the proportion of the number of clients and servers is 3:1 (denoted by *client/server*), and the length of an anonymous tunnel is 2, unless otherwise specified.

The tested websites are retrieved from the 100 top sites in Alexa [1]. We randomly chose 100 nodes located in North America, Asia, Europe and Canada to join in Freeweb and assign the role of either "client (censored node)" or "server (uncensored node)" to each node. Upon receiving the "join" notification, a node randomly chooses a value c from $[0, 60]$ and joins in the system after c seconds. A client sends out one URL request every 100 seconds. Nodes in PlanetLab are not constantly connected, which provides a simulation environment with ungraceful node departure and failures. We tested the connectivity failure rate of the selected PlanetLab nodes by letting each node try to connect to all other nodes in the system. The connection failure rate of a node is between 4-12%, with an average of 6%. Thus, Freeweb is tested in an environment with node dynamism.

Bootstrap IP	138.232.66.195
Warmup time	$320s$
Each test duration	1 hour
Network size	$25 - 100$ nodes
Client/server	$1:1-5:1$
Percentage of malicious nodes	$5\% - 40\%$
Length of tunnel	$1-5$ hops
Encryption method	DES, RSA
Request interval of a node	$100s$
Content retrieval mode	full-page, text-only
Request forwarding scheme	only deterministic

Table I: Experiment parameters.

Malicious servers in our experiment modify the content of a webpage before sending it back to the client. In the experiment, we did not encrypt requests using a public key because the cost of encryption on PlanetLab nodes would dominate the URL requesting process time. We conducted a test on modern computers, and the results show that their public key encryption time is usually 1/10 of that on PlanetLab nodes. PlanetLab nodes are heavily loaded most of the time, so their computing ability is limited. Thus, we believe Freeweb can perform much better on personal computers.

Figure 7 shows the success rate of Freeweb versus different tunnel lengths. The time bound was set to 15s and 20s, respectively. When the time bound is 20s and the tunnel is shorter than 5 hops, Freeweb can successfully handle 90% of requests for both text-only and full-page modes. When the tunnel length is 5, both success rates drop to around 80%. This is because more hops in a tunnel result in a higher possibility of failed TCP communication between two endpoints.

We find that the success rates with a 15s time bound are lower than those with a time bound of 20s. This is because Freeweb completes fewer requests in a shorter time bound. In this case, full-page mode generates a lower success rate than text-only mode because its webpage contains images in addition to the HTML file, which takes a longer time to process and transmit. The success rate also decreases as the length of the tunnel increases. The results show that Freeweb can achieve relatively high success rates on heavily loaded nodes in PlanetLab and that it is important to choose an appropriate tunnel length to achieve an optimized tradeoff between browsing timeliness and anonymity protection degree.

Figure 8 shows the success rate versus the network size (the total number of nodes) in Freeweb. We can see that the

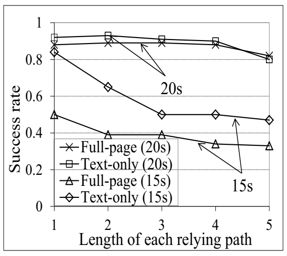


Figure 7: Success rate vs. tunnel length.

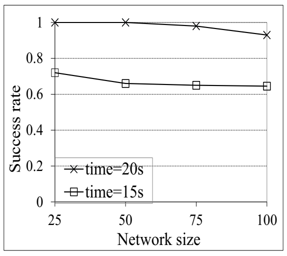


Figure 8: Success rate vs. network size.

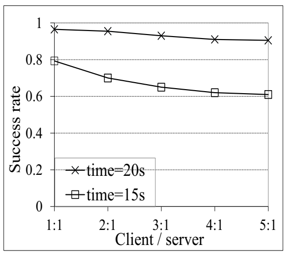


Figure 9: Success rate vs. client/server.

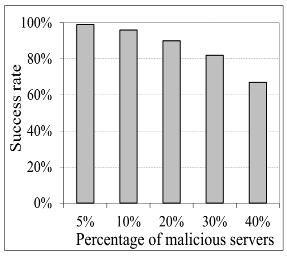


Figure 10: Success rate vs. malicious node percent.

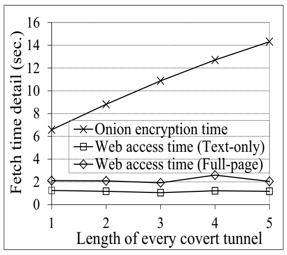


Figure 11: Fetch time vs. tunnel length.

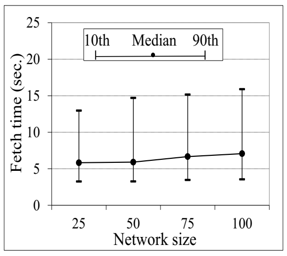


Figure 12: Fetch time vs. network size

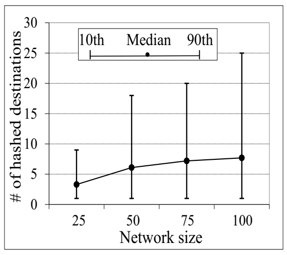


Figure 13: Number of hashed destination vs. network size.

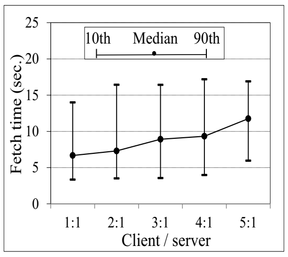


Figure 14: Fetch time vs. client/server.

success rate remains at [95%,100%] and [60%,70%] when the time bound is 20s and 15s, respectively. The success rate decreases slightly as the network size increases. Thus, the network size does not have a significant impact on the success rate. These results verify that Freeweb has a high scalability that is inherited from the DHT network. This figure confirms our analytical result in Figure 6 that the success rate increase as the network size decreases.

We varied the *client/server* rate to test its impact on the success rate. Figure 9 shows the success rate versus the *client/server* rate. We can observe that the success rate still remains above 95% under the 20s time bound and above 60% under the 15s time bound. We can also see that as the *client/server* rate increases, the success rate of the 15s bound exhibits a slight drop. A higher *client/server* rates mean that the number of servers decreases and the number of clients increases. Consequently, a request has a lower probability of reaching a server and needs more rehashing and forwarding operations to meet a server. This leads to longer transmission latency, which makes some requests unable to complete within 15s. It is intriguing to see that the increase of the *client/server* rate does not significantly affect the success rate of the 20s time bound. This is because most requests can still be completed within the time bound. The results imply that a request can always reach a server, even when the server/client ratio is as low as $\frac{1}{6}$, and verify the high success rate and censorship-resistant ability of Freeweb. The results confirm our analytical result in Figure 6 that the success rate increase as the fraction of uncensored nodes increases (i.e., client/server decreases).

Figure 10 shows the performance of Freeweb when there are malicious nodes sending back tampered webpages. If the percentage of malicious servers is p, then the expected success rate is $1-p$. Recall that in order to tackle tampering, a client in Freeweb sends three requests with different destinations and compares its received webpages to identify the correct one. The figure demonstrates that when only 5%-10% of nodes are malicious, the success rate is nearly 100%. When 20% of nodes are malicious, the success rate is approximately 90%, which is higher than the predicted value of $1 - 20\% = 80\%$, as is the cases where there are 30% and 40% malicious servers. The results demonstrate that Freeweb's redundant request strategy is effective in increasing the success rate. There are two main reasons why Freeweb sometimes cannot identify the correct webpage. First, sometimes only one or two webpages are returned. In the test, the first webpage is chosen in this case. Second, as the number of malicious nodes increases, two or three webpages among the three may be from malicious servers.

Figure 11 demonstrates the fetch time with different tunnel lengths. We can see that the fetch time increases almost linearly as the tunnel length increases. The length of a tunnel determines the number of hop-to-hop transmissions of a returned webpage package. We also see that the onion encryption time increases linearly as the length of tunnel increases. Onion encryption operation only encrypts a number of IP and port addresses, but takes 6.6-14.3 seconds, which is a large proportion of the entire fetch time. On modern computers as we tested, this encryption operation usually takes 1/5 of the shown time. Therefore, the encryption can be performed faster in normal computers than in PlanetLab nodes. In addition, we notice the web access time of HTML is only slightly less than that of rich content mode, which shows that web access time contributes little to the overall fetch time.

Figure 12 plots the 90th percentile, median and the 10th percentile of fetch time under different network scales. It can be observed the fetch time increases slightly as network scale increases. Recall that the average path length of DHTs is $\log n$. Thus, fetch time grows as the length increases in the client-server path. In addition, we can see the 10th percentiles of fetch time are all below 3.6 seconds and do not differ much in different network scales. The 90th percentile of fetch time generally increases as the network scale increase, because larger network scale leads to more nodes to traverse before a server node can be found. The result shows the high scalability of Freeweb due to its underlying DHT network.

Figure 13 demonstrates that the 90th percentile and the median of the number of hashed destinations increase as the network size increases. Given the same percentage of servers, it is easier to find a server in a smaller-scale network than in a larger-scale network. More nodes in the network increases the probability that a request reaches a non-server node. The 10th percentile remains at 1, which implies some requests can always reach servers after the first hashing. Also, we see that the median number stays between 4-8. This means most requests need to be rehashed and forwarded to a new destination a number of times before reaching a server.

Figure 14 shows the 90th percentile, median, and 10th percentile of fetch time with different *client/server* rates. We see that the fetch time grows as the rate increases. This is because with fewer servers, the request needs more time to find a server. We also observe that the increase in fetch time slows down as the client/server rate increases due to the slowdown of the server decrease rate. Another observation is that the 10th percentile of fetch time is not affected greatly by the *client/server* rate because a few requests can always reach the server quickly. The 90th percentile of fetch time increases marginally as the *client/server* rate increases because a request needs to traverse more hops when there are fewer servers.

VI. Conclusions

In this paper, we propose Freeweb, which is built on a P2P network to provide a blocking resistant and tracing resistant collaborative censorship circumvention service. It enables all nodes in the system to collaborate with their web accesses in order to circumvent censorship. A node in a censored area can retrieve its requested webpage with the aid of a node in an uncensored area. Freeweb protects node identify from censors and constructs an anonymous tunnel for web content transmission. It also employs the techniques of encryption, onion routing, and caching to enhance its censorship circumvention ability and reduce browsing cost and latency. Extensive experiments on PlanetLab show Freeweb achieves low cost and latency, resistance to malicious nodes, and high success rates for website browsing. Admittedly, Freeweb is not completely bulletproof. In our future work, we will consider other attacks, such as DoS, and develop anti-attack mechanisms for Freeweb.

Acknowledgements

This research was supported in part by U.S. NSF grants CNS-1254006, CNS-1249603, CNS-1049947, CNS-0917056, CNS-1025652, CNS-0845513, CNS-1017598, and CNS-1017588, Microsoft Research Faculty Fellowship 8300751. We would like to thank Mr. Guoxin Liu for his help in conducting the experiments.

References

[1] Alexa. http://www.alexa.com/.

[2] China blocks Bing.com, Hotmail.com, Live.com, Twitter, Flickr. http://www.liveside.net/main/archive/2009/06/02.

[3] China blocks Facebook and Twitter. http://www.allvoices.com/contributed-news/3624049-china-blocks-facebook-and-twitter.

[4] How many website in the world? www.boutell.com/newfaq/misc/sizeofweb.html.

[5] PlanetLab. http://www.planet-lab.org/.

[6] Proxify. http://www.proxify.com.

[7] US Broadband Speed 18th Worldwide. http://www.websiteoptimization.com/bw/0908/.

[8] O. Berthold, H. Federrath, and S. Kpsell. Web MIXes: A system for anonymous and unobservable Internet access. In *Proc. of workshop on design issues in anonymity and unobservability*, 2001.

[9] D. Chaum, C. O. T. Acm, R. Rivest, and D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 1981.

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

[11] J. R. Douceur. The sybil attack. In *Proc. of IPTPS*, 2002.

[12] N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. Karger. Infranet: Circumventing Web Censorship and Surveillance. In *Proc. of USENIX Security*, 2002.

[13] M. J. Freedman, E. Sit, J. Cates, and R. Morris. Introducing Tarzan, a Peer-to-Peer Anonymizing Network Layer. In *Proc. of IPTPS*, 2002.

[14] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Privacy-preserving p2p data sharing with oneswarm. In *Proc. of SIGCOMM*, 2010.

[15] J. Jia and P. Smith. Psiphon: Analysis and Estimation, 2002. http://pyre.third-bit.com/2004-fall/psiphonae.html.

[16] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proc. of STOC*, 1997.

[17] S. Köpsell and U. Hillig. How to achieve blocking resistance for existing systems enabling anonymous web surfing. In *Proc. of WPES*, 2004.

[18] S. J. Murdoch. Hot or not: Revealing hidden services by their clock skew. In *Proc. of CCS*, pages 27–36. ACM Press, 2006.

[19] J.-F. Raymond. Traffic Analysis: Protocols, Attacks, Design Issues and Open Problems. In *Proc. of International Workshop on Design Issues in Anonymity and Unobservability*, pages 10–29. Springer-Verlag New York, Inc., 2001.

[20] M. Reed, P. Syverson, and D. Goldschlag. Anonymous connections and onion routing. *JSAC*, pages 482–494, 1998.

[21] M. Rennhard. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *Proc. of WPES*, 2002.

[22] A. Singh, T. wan johnny Ngan, P. Druschel, and D. S. Wallach. Eclipse attacks on overlay networks: Threats and defenses. In *Proc. of INFOCOM*, 2006.

[23] Y. Sovran, A. Libonati, and J. Li. Pass it on: Social Networks stymie censors. In *Proc. of IPTPS*, 2008.

[24] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. In *Proc. of SIGCOMM*, 2001.