# SWEET: Serving the Web by Exploiting Email Tunnels

Amir Houmansadr, *Member, IEEE*, Wenxuan Zhou, Matthew Caesar, *Member, IEEE*, and Nikita Borisov, *Member, IEEE*

***Abstract*— Open communications over the Internet pose serious threats to countries with repressive regimes, leading them to develop and deploy censorship mechanisms within their networks. Unfortunately, existing censorship circumvention systems do not provide high *availability* guarantees to their users, as censors can easily identify, hence disrupt, the traffic belonging to these systems using today's advanced censorship technologies. In this paper, we propose Serving the Web by Exploiting Email Tunnels (SWEET), a highly available censorship-resistant infrastructure. SWEET works by encapsulating a censored user's traffic inside email messages that are carried over public email services like Gmail and Yahoo Mail. As the operation of SWEET is not bound to any specific email provider, we argue that a censor will need to block email communications all together in order to disrupt SWEET, which is unlikely as email constitutes an important part of today's Internet. Through experiments with a prototype of our system, we find that SWEET's performance is sufficient for Web browsing. In particular, regular Websites are downloaded within couple of seconds.**

***Index Terms*— Censorship circumvention, email communications, traffic encapsulation.**

## I. Introduction

THE Internet provides users from around the world with an environment to freely communicate, exchange ideas and information. However, free communication continues to threaten repressive regimes, as the open circulation of information and speech among their citizens can pose serious threats to their existence. Recent unrest in the middle east demonstrates that the Internet can be widely used by citizens under these regimes as a very powerful tool to spread censored news and information, inspire dissent, and organize events and protests. As a result, repressive regimes extensively monitor their citizens' access to the Internet and restrict open access to public networks [1] by using different technologies, ranging from simple IP address blocking and DNS hijacking to the more complicated and resource-intensive Deep Packet Inspection (DPI) [2], [3].

With the use of censorship technologies, a number of different systems were developed to retain the openness of the Internet for the users living under repressive regimes [4]–[9]. The earliest circumvention tools are HTTP proxies [4], [9], [10] that simply intercept and manipulate a client's HTTP requests, defeating IP address blocking and DNS hijacking techniques. The use of more advanced censorship technologies such as DPI [2], [11], rendered the use of HTTP proxies ineffective for circumvention. This led to the advent of more advanced tools such as Ultrasurf [5] and Psiphon [6], designed to evade content filtering. While these circumvention tools have helped, they face several challenges. We believe that the biggest one is their lack of *availability*, meaning that a censor can disrupt their service frequently or even disable them completely [12]–[16]. The common reason is that the network traffic made by these systems can be distinguished from regular Internet traffic by censors, i.e., such systems are not *unobservable*. For example, the popular Tor [8] network works by having users connect to an ensemble of nodes with public IP addresses, which proxy users' traffic to the requested, censored destinations. This public knowledge about Tor's IP addresses, which is required to make Tor usable by users globally, can be and is being used by censors to block their citizens from accessing Tor [17], [18]. To improve availability, recent proposals for circumvention aim to make their traffic unobservable to the censors by pre-sharing secrets with their clients [19]–[21]. Others [22]–[25] suggest to conceal circumvention by making infrastructure modifications to the Internet. Nevertheless, deploying and scaling these systems is a challenging problem, as discussed in Section II.

A more recent approach in designing unobservalbe circumvention systems is to imitate popular applications like Skype and HTTP, as suggested by Skype-Morph [26], CensorSpoofer [27], and StegoTorus [28]. However, it has recently been shown [29] that these systems' unobservability is breakable; this is because a comprehensive imitation of today's complex protocols is sophisticated and infeasible in many cases. A promising alternative suggested [29], [30] is to not mimic protocols, but run the actual protocols and find clever ways to tunnel the hidden content into their genuine traffic; this is the main motivation of the approach taken in this paper.

In this paper, we design and implement SWEET, a censorship circumvention system that provides high availability by leveraging the openness of email communications. Fig. 1 shows the main architecture. A SWEET client, confined by a censoring ISP, tunnels its network traffic inside a series of email messages that are exchanged between herself and an email server operated by SWEET's server. The SWEET server acts as an Internet proxy [31] by proxying the encapsulated

Manuscript received February 27, 2016; revised September 28, 2016; accepted November 20, 2016; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Chen. Date of publication January 16, 2017; date of current version June 14, 2017. This work was supported in part by the National Science Foundation CAREER under Grant CNS-1553301 and in part by the National Science Foundation under Grant CNS-1525642.

A. Houmansadr is with the University of Massachusetts Amherst, Amherst, MA 01003 USA (e-mail: amir@cs.umass.edu).

W. Zhou, M. Caesar, and N. Borisov are with the University of Illinois at Urbana–Champaign, Champaign, IL 61801 USA.

Digital Object Identifier 10.1109/TNET.2016.2640238

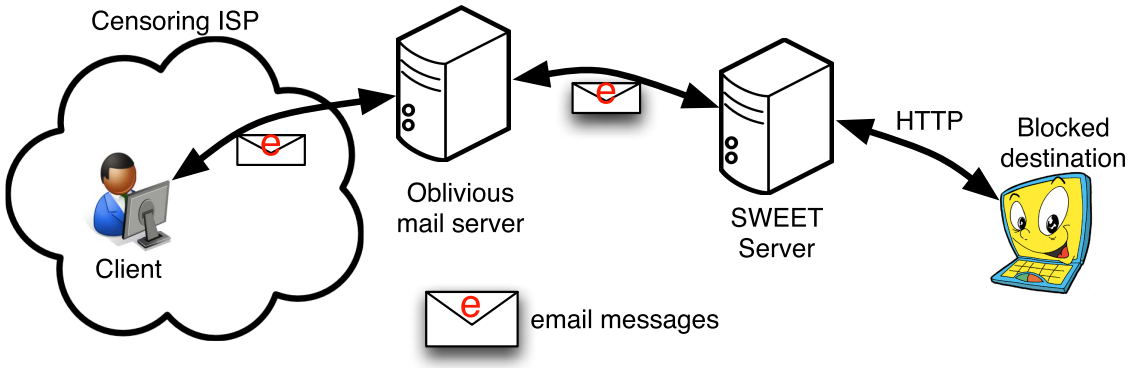


Fig. 1. Overall architecture of SWEET.

traffic to the requested blocked destinations. The SWEET client uses an oblivious, public mail provider (e.g., Gmail, Hotmail, etc.) to exchange the encapsulating emails, rendering standard email filtering mechanisms ineffective in identifying/blocking SWEET-related emails. More specifically, to use SWEET for circumvention a client needs to create an email account with *some* public email provider; she also needs to obtain SWEET's client software from an out-of-bound channel (similar to other circumvention systems). The user configures the installed SWEET software to use her public email account, which sends/receives encapsulating emails on behalf of the user to/from the email address of SWEET.

*SWEET's unobservability:* We claim that a censor is not easily able to distinguish between SWEET's email messages and benign email messages. As described later in Section IV, a SWEET client has two options in choosing her email account: 1) *AlienMail* a non-domestic email that encrypts emails (e.g., Gmail for users in China), and 2) *DomesticMail* a domestic email account with no need for encryption (e.g., 163.com for users in China). As described in Section IV, when AlienMail is used by a client all of its SWEET emails are sent to a publicly known email address, e.g., `tunnel@sweet.org`, encrypted; however, a censor will not be able to identify these emails since they are *proxied* by the AlienMail server running outside the censoring area. In simpler words, the censor only observes that the client is exchanging encrypted messages with the AlienMail server (e.g., Gmail's mail server in U.S.), but he will not be able to observe neither the recipient's email address (`tunnel@sweet.org`), nor the IP address of the sweet.org mail server. As a result, **existing approaches for spam filtering such as shooting the spamming SMTP servers and dropping spam emails are entirely infeasible**. In the case of DomesticMail, the SWEET server uses a secondary *secret* email account, which is only shared with that particular client, for exchanging SWEET emails (i.e., `myotheremail@163.com` instead of `tunnel@sweet.org` address). As a result, the censor will not be able to identify SWEET messages from their recipient fields (since the censor does not know the association of `myotheremail@163.com` with SWEET). Also, the use of steganography/encryption to embed tunneled data renders DPI infeasible.

*SWEET's availability:* Given SWEET's unobservability discussed above, a censor can not efficiently distinguish between SWEET emails and benign email messages. Hence, in order to block SWEET a censor needs to block all email messages to the outside world. However, email is an essential service in today's Internet and it is very unlikely that a censorship authority will block *all* email communications to the outside world, due to different financial and political reasons. This, along the fact that SWEET can be reached through a wide range of domestic/non-domestic email providers provides a high degree of *availability* for SWEET.

*Prototype implementation:* We have built a prototype implementation for SWEET and evaluated its performance. We have also proposed and prototyped two different designs for SWEET client. The first client design uses email protocols, e.g., POP3 and SMTP, to communicate with the SWEET system, and our second design is based on using the webmail interface. Our measurements show that a SWEET client is able to browse regular-sized web destinations with download times in the order of couple of seconds.

In fact, the high availability of SWEET comes for the price of higher, but bearable, communication latencies. Fig. 2 compares SWEET with several popular circumvention systems regarding their availability and communication latency. As our measurements in Section VII show, SWEET provides communication latencies that are convenient for latency-sensitive activities like web browsing (i.e., few seconds). Such additional, tolerable latency of SWEET comes with the bonus of better availability, as discussed in Section V-B.

*Our contributions:* In summary, this paper makes the following main contributions: i) we propose a novel infrastructure for censorship circumvention, SWEET, which provides high availability, a feature missing in existing circumvention systems; ii) we develop two prototype implementations for SWEET (one using webmail and the other using email exchange protocols) that allow the use of nearly all email providers by SWEET clients; and, iii) we show the feasibility of SWEET for practical censorship circumvention by measuring the communication latency of SWEET for web browsing using our prototype implementation.

*Paper's organization:* The rest of this paper is organized as follows; in Section II, we discuss the related work on

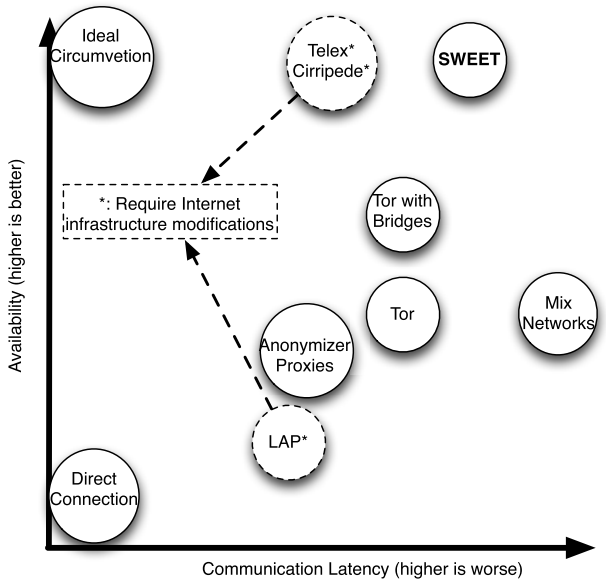


Fig. 2. Availability and communication latency comparison of circumvention systems.

unobservable censorship circumvention. In Section III, we reviews our threat model. We provide the detailed description of the proposed circumvention system, SWEET, in Section IV. We discuss SWEET's censorship features, including its availability, in Section V and compare it with the literature. Our prototype implementation and evaluations are presented in Sections VI and VII, respectively. Finally, we conclude the paper in Section VIII.

## II. Related Work

There has been much work on unobservable censorship circumvention systems [23], [24], [26]–[28], [30], [32]–[35]. Similar to SWEET, FreeWave [30], CloudTransport [32], and CovertCast [35] also work by tunneling circumvention traffic into the actual runs of popular network protocols. For instance, FreeWave [30] tunnels Internet traffic inside VoIP communications. This tunneling approach provides much stronger unobservability against the censors compared to imitation-based circumvention systems [26]–[28], as demonstrated by Houmansadr *et al.* [29].

Several designs [19]–[21] seek unobservability by sharing secret information with their clients, which are not known to censors. For instance, the Tor network has recently adopted the use of *Tor Bridges*, a set of volunteer nodes connecting clients to the Tor network, whose IP addresses are selectively distributed among Tor users by Tor. As another example, Infranet [19] shares a secret key and some secret URL addresses with a client, which is then used to establish an unobservable communication between the client and the system. Collage [20] works by having a client and the system *secretly* agree on some user-generated content sharing websites, e.g., flickr.com, and communicate using steganography. Unfortunately, sharing secrets with a wide range of clients is a serious challenge, as a censor can obtain the same secret information by pretending to be a client.

Some recent research suggests circumvention being built into the Internet infrastructure to better provide unobservability [22]–[24]. These systems rely on collaboration from some Internet routers that intercept users' traffic to uncensored destinations to establish covert communication between the users and the censored destinations. Telex [23] and Cirripede [24] provide this unobservable communication without the need for some pre-shared secret information with the client, as the secret keys are also covertly communicated inside the network traffic. Cirripede [24] uses an additional client registration stage that provides some advantages and limitations as compared to Telex [23] and Decoy routing [22] systems. Recent studies investigate the real-world deployment of decoy routing systems by evaluating the placement of decoy routers on the Internet in adversarial settings [36]–[38].

There are two projects that work in a similar manner to SWEET: FOE [39] and MailMyWeb [40]. Instead of tunneling traffic, which is the case in SWEET, these systems simply download a requested website and send it as an email attachment to the requesting user. This highly limits their performance compared to SWEET, as discussed in Section IV-D.

## III. Threat Model

We assume that a user is confined inside a censoring ISP. The ISP blocks the user's access to certain Internet destinations, namely *blocked destinations*. The censor is assumed to use today's advanced filtering technologies, including IP address blocking, DNS hijacking, and deep packet inspection techniques [3]. The ISP also monitors all of its egress/ingress traffic to detect any use of circumvention techniques.

We assume that the censorship is constrained not to degrade the *usability* of the Internet. In other words, even though it *selectively* blocks certain Internet connections, she is not willing to block key Internet services *entirely*. In particular, the operation of SWEET system relies on the fact that a censoring ISP does not block *all* email communications, even though she can selectively block emails/email providers. We also assume that the ISP has as much information about SWEET as any SWEET client.

We also consider active behaviors of the ISP. In addition to traffic monitoring, the censor manipulates its Internet traffic, e.g., by selectively dropping packets, and adding latency to some packets, to disrupt the use of circumvention systems and/or to detect the users of such systems. Again, such perturbations are constrained to preserve the usability of the Internet for benign users.

## IV. Design of SWEET

In this section, we describe the detailed design of SWEET. Fig. 1 shows the overall architecture. SWEET tunnels network connections between a client and a server, called SWEET server, inside email communications. Upon receiving the tunneled network packets, the SWEET server acts as a transparent proxy between the client and the network destinations requested by the client.

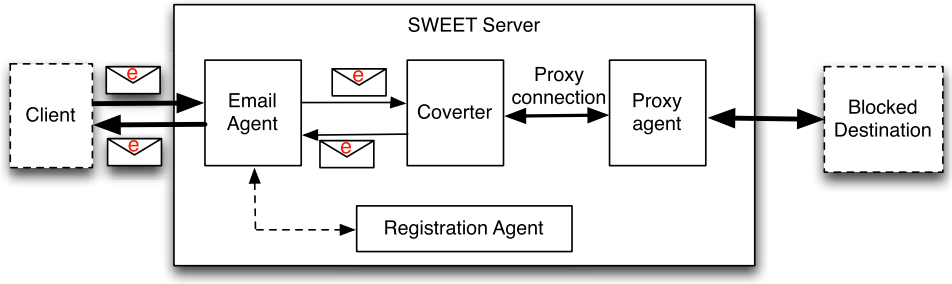


Fig. 3. The main architecture of SWEET server.

*A client's choices of email services:* A SWEET client has two options for his email provider: *AlienMail*, and *DomesticMail*.

1) **AlienMail** An AlienMail is a mail provider whose mail servers reside outside the censoring ISP, e.g., Gmail for the Chinese clients. We only consider AlienMails that provide email encryption, e.g., Gmail and Hushmail. A SWEET client who uses an AlienMail does not need to apply any additional encryption/steganography to her encapsulated contents. Also, she simply sends her emails to the publicly advertised email address of SWEET server, e.g., `tunnel@sweet.org`, since the censors will not be able to observe (and block) the `tunnel@sweet.org` address inside SWEET messages, which are exchanged ibetween the client and the AlienMail server in an encrypted format.
2) **DomesticMail** A DomesticMail is an email provider hosted inside the censoring ISP and possibly collaborating with the censors, e.g., 163.com for the Chinese clients. Since the censors are able to observe the email contents, the SWEET client using a DomesticMail should hide the encapsulated contents through steganography (e., by doing image/text steganography inside email messages). Also, the client can not send her SWEET emails to the public email address of SWEET server (`tunnel@sweet.org`) since the mail recipient field is observable to the DomesticMail provider and/or the censor. Instead, the client generates a secondary email address, `myotheremail@somedomain.com` (which could be either DomesticMail or AlienMail), and then provides the email credentials for this secondary account *only* to SWEET server through an out-of-band channel (e.g., through an online social network). The SWEET server uses this email address to exchange SWEET emails *only* with this particular client.

In the following, we describe the details of SWEET's server and client architectures. To avoid confusion and without loss of generality, we **only consider the case of AlienMail** being used by the client. If DomesticMail is used, the client and server should also perform some steganography operations to hide the encapsulated traffic, as well as they should exchange a secondary email address, as described above.

### A. SWEET Server

The SWEET server is the part of SWEET running outside the censoring region. It helps SWEET clients to evade censorship by proxying their traffic to blocked destinations. More specifically, a SWEET server communicates with censored users by exchanging emails that carry tunneled network packets. Fig. 3 shows the main design of SWEET server, which is composed of the following elements:

① *Email agent:* The email agent is an IMAP and SMTP server that receives emails that contain the tunneled Internet traffic, sent by SWEET clients to SWEET's email address. The email agent passes the received emails to another components of the SWEET server, the converter and the registration agent. The email agent also sends emails to SWEET clients, which are generated by other components of SWEET server and contain tunneled network packets or client registration information.

② *Converter:* The converter processes the emails passed by the email agent, and extracts the tunneled network packets. It then forwards the extracted data to another component, the proxy agent. Also, the converter receives network packets from the proxy agent and converts them into emails that are targeted to the email address of corresponding clients. The converter then passes these emails to the email agent for delivery to their intended recipients. As described later, the converter encrypts/decrypts the email attachments of a user using a secret key shared with that user.

③ *Proxy agent:* The proxy agent proxies the network packets of clients that are extracted by the converter, and sends them to the Internet destination requested by the clients. It also sends packets from the destination back to the converter.

④ *Registration agent:* This component is in charge of registering the email addresses of the SWEET clients, prior to their use of SWEET. The information about the registered clients can be used to ensure quality of service and to prevent denial-of-service attacks on the server. Additionally, the registration agent shares a secret key with the client, which is used to encrypt the tunneled information between the client and the server.

The email agent of the SWEET server receives two type of emails; *traffic emails*, which contain tunneled traffic from the clients (sent to `tunnel@sweet.org`), and

*registration emails*, which carry client registration information (sent to `register@sweet.org`).

*Client registration:* Before the very first use of the SWEET service, a client needs to register her email address with the system. This is automatically performed by the client's SWEET software. The objective of client registration is twofold: to prevent denial-of-service (DoS) attacks and to share a secret key between a client and the server. A DoS attack might be launched on the server to disrupt its availability, e.g., through sending many malformed emails on behalf of non-existing email addresses (this is discussed in Section V). In order to register (or update) the email address of a client, the client's SWEET software sends a registration email from the user's email address, to the SWEET's registration email address. i.e., `register@sweet.org`, requesting registration. The email agent forwards all received registration emails to the registration agent (④). For any new registration request, the registration agent generates and sends an email to the requesting email address (through the email agent) that contains a unique computational *challenge* (e.g., [41]). After solving the challenge, the client software sends a second email to `register@sweet.org` that contains the solution to the challenge, along with a Diffie-Hellman [42] public key $K_C = g^{k_C}$. If the client's response is verified by the registration agent, the client's email address will be added to a *registration list*, that contains the list of registered email addresses with their expiration time. Also, the registration agent uses its own Diffie-Hellman public key, $K_R = g^{k_R}$, to evaluate a shared key $k_{C,R} = g^{k_R k_C}$ for the later communications with the client. The registration agent adds this key to the client's entry in the registration list, to be used for communications with that client. The client is able to generate the same $k_{C,R}$ key using SWEET's publicly advertised public key and her own private key [42].

*Tunneling the traffic:* Any traffic email received by the email agent is processed as follows: the email agent (①) forwards the email to the converter(②). The converter processes the traffic email and extracts the tunneled information. The converter, then, decrypts the extracted information (using the key $k_{C,R}$ corresponding to the user) and sends it to the proxy agent (③). Finally, the proxy processes the received packet as required, e.g., sends the packet to the requested destination. Similarly, for any tunneled packet received from the proxied destinations, the proxy agent sends it to the converter. The converter encrypts the received packet(s) (using the corresponding $k_{C,R}$), and generate a traffic email containing the encrypted data as an attachment, targeted to the email address of the corresponding client. The generated email is passed to the email agent, who sends the email to the client. Note that to improve the latency performance, small packets that arrive at the same time get attached to the same email.

### B. SWEET Client

To use SWEET, a client needs to obtain a copy of SWEET's client software and install it on her machine. The client also needs to create one or two email account (depending on if she uses an AlienMail or a DomesticMail for her primary email). A client needs to configure the installed SWEET's software with information about her email account. Prior to the first use of SWEET by a client, the client software registers the email address of its user with the SWEET server and obtains a shared secret key $k_{C,R}$, as described in Section IV-A.

We propose two designs for SWEET client: a protocol-based design, which uses standard email protocols to exchange email with client's email provider, and a webmail-based design, which uses the webmail interface of the email provider. We describe these two designs in the following.

*1) Protocol-Based Design:* Fig. 4(a) shows the three main elements.

❶ *Web Browser:* The client can use any web browser that supports proxying of connections, e.g., Google Chrome, Internet Explorer, or Mozilla Firefox. The client needs to configure her browser to use a local proxy server, e.g., by setting `localhost:4444` as the HTTP/SOCKS proxy. The client can use two different browsers for browsing with and without SWEET to avoid the need for frequent re-configurations of the browser. Alternatively, some browsers (e.g., Chrome, and Mozilla Firefox) allow a user to have multiple browsing *profile*s, hence, a user can setup two profiles for browsing with and without SWEET.

❷ *Email Agent:* It sends and receives SWEET emails thorough the client's email account. The client needs to configure it with the settings of the SMTP and IMAP/POP3 servers of her email account. The client also needs to provide it with the account login information.

❸ *Converter:* It sits between the web browser and the email agent, and converts SWEET emails into network packets and vice versa. It uses the keys shared with SWEET, $k_{C,R}$, to encrypt/decrypt email content.

Once the client enters a URL into the configured browser (❶), the browser makes a proxy connection to the local port that the converter (❸) is listening on. The converter accepts the proxy connection and keeps the state of the established TCP/IP connections. For packets that are received from the browser, the converter generates traffic emails, targeted to `tunnel@sweet.org`, having the received packets as encrypted email attachments (using the key $k_{C,R}$). Such emails are passed to the email agent (❷) that sends the emails to the SWEET server through the public email provider of the client.

The email agent is also configured to receive emails from the client's email account through an email retrieval protocol, e.g., IMAP or POP3. This allows the email agent to continuously look for new emails from the server. Once new emails are received, the email agent passes them to the converter, who in turn extracts the packets from the emails, decrypts them, and sends them to the browser over the existing TCP/IP connection.

*2) Webmail-Based Design:* Alternatively, the SWEET client can use the webmail interface of the client's public email provider. as showed in Fig. 4(b). The main difference with the protocol-based design is that in this case the email agent (❷) uses a web browser to exchange emails. More specifically, the email agent uses its web browser to open a webmail interface with the client's email account, using the user's authentication

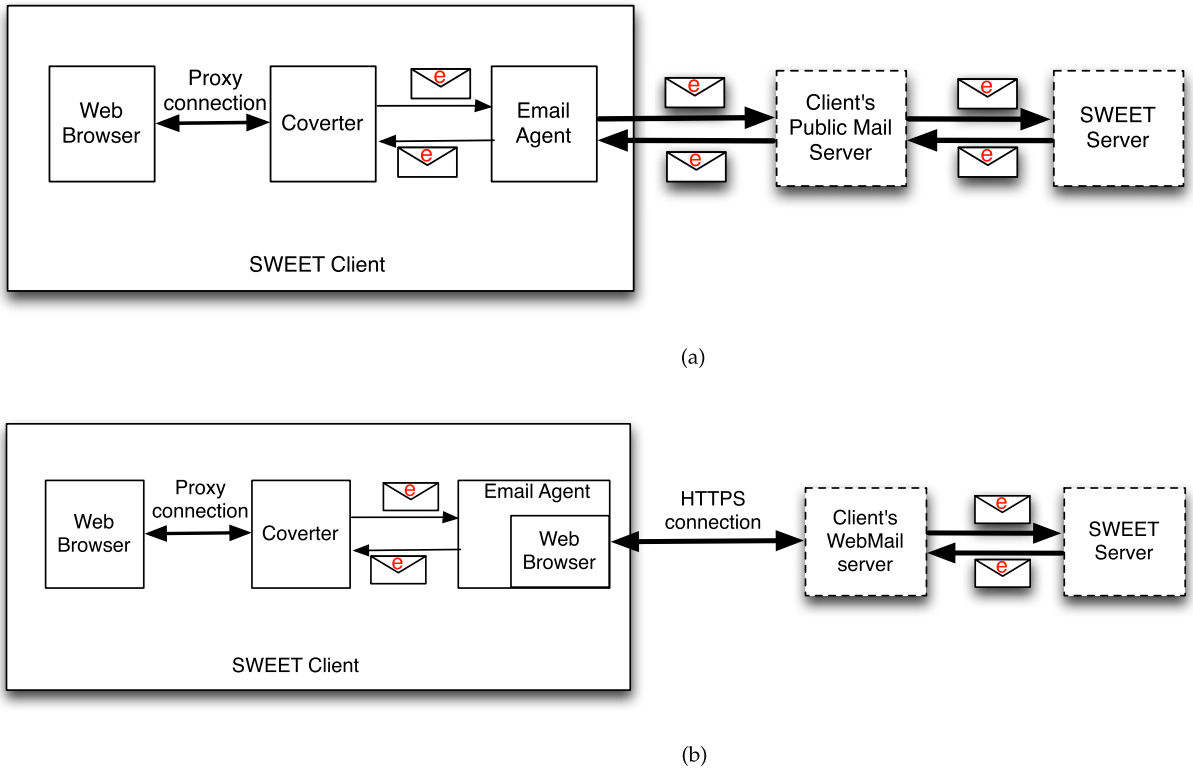


Fig. 4. Design of SWEET client software. (a) The protocol-based design. (b) The webmail-based design.

credentials for logging in. Through this HTTP/HTTPS connection, the email agent communicates with the SWEET server by sending and receiving emails.

### C. The Choice of the Proxy Protocol

As mentioned before, the SWEET server uses a proxy agent to receive the tunneled traffic of clients and to establish connections to the requested destinations. We consider the use of both SOCKS [31] and HTTP [43] proxies in the design, as each provides unique advantages. Our server's proxy agent runs a SOCKS proxy and an HTTP proxy in parallel, each on a different port. A user can choose to use the type of proxy by configuring her client to connect to the corresponding port.

The use of the SOCKS proxy allows the client to make *any* IP connection through the SWEET system, including dynamic web communications, such as Javascript or AJAX, and instant messaging. In contrast, an HTTP proxy only allows access to HTTP destinations. However, an HTTP proxy may speed up connections by using HTTP-layer optimizations such as caching or pre-fetching of web objects.

### D. An Alternative Approach: Web Download

A trivial approach in providing censorship circumvention using email is to download an entire webpage and attach it as an email attachment to emails that are targeted to the requesting users. In fact, this approach is under development by the open-source foe project [39], and the for-profit service of MailMyWeb [40]. Unfortunately, this simple approach only provides a limited access to the Internet: a user can only access static websites. In particular, this approach cannot be used to access destinations that require end-to-end encryption, contain dynamic web applications like HTML5 and Javascript sockets, or need user login information. Also, this approach does not support accessing web destinations that require a live Internet connection, e.g., video streaming websites, instant messaging, etc. In fact, the MailMyWeb service uses some heuristics to tackle some of these shortcomings partially, which are privacy-invasive and inefficient. For example, in order to access login-based websites MailMyWeb requests a user to send her login credentials to MailMyWeb by email. Also, a user can request for videos hosted *only* on the YouTube video sharing website, which are then downloaded by MailMyWeb and sent as email attachments; this causes a large delay between the time a video is requested until it is has received by the user. SWEET, on the other hand, provides a comprehensive web browsing experience to its users since it can tunnel any kind of IP traffic.

## V. Discussions and Comparisons

In this section, we evaluate SWEET's circumvention capabilities by discussing important features that are essential for an effective circumvention.

### A. Unobservability

We say a circumvention tool provides unobservability if censors are not able to identify neither the traffic, nor the clients using that tool. Unobservability has been considered as a main feature in the design of recent circumvention systems [22]–[24], [26]–[28].

We claim that a censor is not easily able to distinguish between SWEET's email messages and benign email messages. As described in Section IV, a SWEET client has two options in choosing her email account: 1) *AlienMail* a non-domestic email that encrypts emails (e.g., Gmail for users in China), and 2) *DomesticMail* a domestic email account with no need for encryption (e.g., 163.com for the users in China). When AlienMail is used by a SWEET client all of its SWEET emails are encrypted and are exchanged with a publicly known email address of SWEET, e.g., `tunnel@sweet.org`; however, a censor will not be able to identify these SWEET emails since they are *proxied* by the AlienMail server running outside the censoring area. In simpler words, the censor only observes that the client is sending/receiving messages with an AlienMail server (e.g., Gmail's mail server in U.S.), but he will not be able to observe neither the recipient's email address (`tunnel@sweet.org`), nor the IP address of the sweet.org mail server. As a result, existing approaches for spam filtering such as shooting spamming SMTP servers and dropping spam emails are entirely infeasible. In the case of DomesticMail, the SWEET server uses a *secret* secondary email account, which is only shared with that particular client, for exchanging SWEET emails (i.e., `myotheremail@163.com` instead of `tunnel@sweet.org` address). As a result, the censor will not be able to identify SWEET messages from their recipient field (since the censor can not associate the private email address with SWEET). Also, the use of steganography to embed tunneled data renders DPI infeasible.

In addition, to ensure unobservability the user's email traffic patterns should mimic that of normal email communications, to defeat traffic analysis by a censor; this limits the bandwidth available to the user, as discussed in Section VII-B.

### B. Availability

SWEET's availability is tied to the assumption that a censor is not willing to block *all* email communications. As the use of SWEET does not require using any specific email provider, users can always find an email service to get connected to SWEET. IP filtering and DNS hijacking would not be able to stop SWEET traffic as a SWEET user's traffic is destined to her public email provider, but not to an IP address or nameserver belonging to the SWEET system. Moreover, deep packet inspection (DPI) is rendered ineffective due to the use of encrypted emails in the case of AlienMail, and steganography in the case of DomesticMail.

As another approach to disrupt the operation of SWEET, a censor might try to launch a denial-of-service (DoS) attack on SWEET server. The common techniques for DoS attacks, e.g., ICMP flooding and SYN flooding, can be mitigated by protecting the SWEET server using up-to-date firewalls. Alternatively, a censor can play the role of a SWEET client and send traffic through its SWEET client software in a way that overloads the SWEET server. As an example, the attacker can flood the SWEET's SOCKS proxy by initiating many incomplete SOCKS connections, or sending SYN floods. A censor could even send such attacking requests on behalf of a number of rogue (non-existing) email addresses, to render an email blacklist maintained by SWEET server ineffective in preventing such attacks. To protect against possible DoS attacks, SWEET requires a new user to register her email address with SWEET server prior to her first use. Such registration can be performed in an unobservable manner by SWEET's client software through the email communication channel (see Section IV-A). Also, to ensure the quality of service for all users, the SWEET server can limit the use of SWEET by putting a cap on the volume of traffic communicated by each registered email address.

### C. Other Properties of SWEET

*Confidentiality:* As mentioned before, SWEET encrypts the tunneled traffic, i.e., email attachments are encrypted using a key shared between a user and SWEET server. This ensures the confidentiality of user communications from any entity wiretapping the traffic, including the censorship authorities and the public email provider. Note that the email attachments are encrypted even if the user choose a plaintext email service. To make a connection confidential from SWEET server, the user can use an end-to-end encryption with the final destination, e.g., by using HTTPS, or alternatively the user can use SWEET to connect to another circumvention system, like Anonymizer [44].

*Ease of deployment:* We argue that SWEET can be easily deployed on the Internet and provide service to a wide range of users. First of all, SWEET is low-cost and needs few resources for deployment. It can be deployed using a single server that runs a few light-weight processes, including a mail server and a SOCKS proxy. To service in a large scale SWEET server can be deployed in a distributed manner as several machines in different geographic locations. Secondly, the operation of SWEET is standalone and does not rely on collaboration from other entities, e.g., end-hosts or ISPs. This provides a significant advantage to recent research that relies on collaboration from ISPs [22]–[24], or end-hosts [19], [20]. In fact, the easy setup and low-resources of SWEET's deployment allows it to be implemented by individuals with different levels of technical expertise. For instance, an ordinary home user can deploy a personal SWEET server to help her friends in censored regions evade censorship, or a corporate network can setup such system for its agents residing in a censored country.

*User convenience:* As a recent study [44] shows, users give the most preference to the ease of use when choosing a circumvention tool. The use of SWEET is simple and requires few resources from a client. A SWEET client only needs to install the client software, which can be obtained from out-of-band channels like social networks or downloaded from the Internet. Due to its simple design, an expert user can also develop the client software herself. In addition, the user needs to have an email account with a public email provider, and needs to know the public information related to SWEET, e.g., the email addresses of SWEET.

## VI. Prototype Implementation

### A. Server Implementation

We implement the SWEET server on a Linux machine, which runs *Ubuntu 10.04 LTS* and has a 2 GHz quad-core CPU

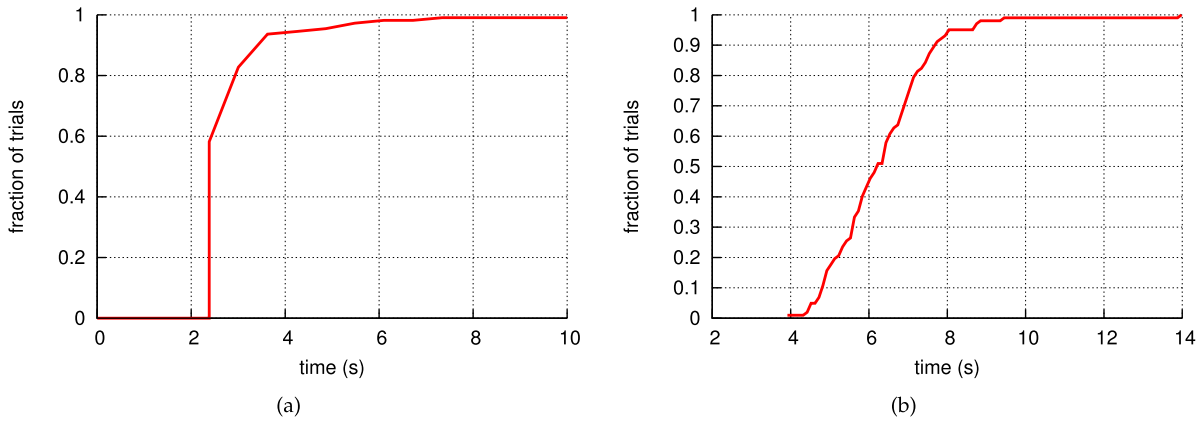


Fig. 5. The CDF of (a) the time that a SWEET email takes to travel from the SWEET client to the SWEET server; (b) the registration time.

and 4 GB of memory. We run Postfix,[1] a simple email server that supports basic functions. Postfix listens for new emails targeted to the `register@sweet.org` and `tunnel@sweet.org` email addresses. Postfix stores the received emails into designated file directories that are continuously watched by the converter and registration agent of SWEET server. Each stored email has a unique name, concatenating the email id of its corresponding client and an increasing counter. The converter agent is a simple Python-based program that runs in the background and continuously checks the folder for new emails. The converter also converts proxied packets, passed by SWEET's proxy, into emails and sends them to their intended clients. For the proxy agent, we use Squid[2] as our HTTP proxy and Suttree[3] as our SOCKS proxy. Squid listens on a local port for connections from the converter.

### B. Client Implementation

*Protocol-based design:* The client prototype is built on a desktop machine, running *Linux Ubuntu 10.04 TLS*. We set up a web browser to use the local port "localhost:9034" as the SOCKS/HTTP proxy. The converter is a simple python script that listens on port 9034 for connections, e.g., from our web browser. We implement the email agent of SWEET client using *Fetchmail*,[4] a popular client software for sending and retrieval of emails through email protocols. We generate a free Gmail account and configure Fetchmail to receive emails through IMAP[5] and POP3[6] servers of Gmail, and to send emails through the SMTP server of Gmail.[7] Note that our design does not rely on Gmail, and the client software can be set up with any email account.

*Webmail-based design:* Our webmail-based implementation also runs on *Linux Ubuntu 10.04 TLS*, and uses the same converter as the one used in the protocol-based prototype. A Google Chrome browser is used for making connections through SWEET, configured to use "localhost:9034" as a proxy. We prototype the web-based email agent by running a *UserScript*[8] inside the Mozilla Firefox[9] browser. More specifically, we install a Firefox extension, *Greasemonkey*,[10] to allow a user to run her own JavaScript, i.e., Userscript, while browsing certain destinations. We write a UserScript that runs in Gmail's webmail interface and listens for the receipt of new emails. Our UserScript saves new emails in a local directory, which is watched by the converter. Note that the Firefox browser is directly connected to the Internet and does not use any proxies (user needs to use the configured Chrome browser to surf the web through SWEET).

## VII. Evaluation

We evaluate SWEET using our prototype implementation.

### A. Performance

We use Gmail as the oblivious mail provider in our experiments. Our SWEET server is located in Urbana, IL, resulting in approximately 2000 miles of geographic distance between the SWEET server and Gmail's email server (we locate Gmail's location from its IP address). Fig. 5(a) shows the CDF of the time that a SWEET email (carrying the tunnelled traffic) sent by a SWEET client takes to reach our SWEET server (the reverse path takes a similar time). As the figure shows, around 90% of emails take less than 3 seconds to reach the server, which is very promising considering the high data capacity of these emails. Note that based on our measurements, most of this delay comes from email handling (e.g, spam checks, making SMTP connections, etc.) performed by the oblivious mail provider (Gmail in our experiments), but not from the network latency (the network latency and client latency constitute only tens of milliseconds of the total latency). As a result, the latency would be very similar for users with an even longer geographical distance from the oblivious mail server.

[1] http://www.postfix.org/
[2] http://www.squid-cache.org/
[3] http://suttree.com/code/proxy/
[4] http://www.fetchmail.info/
[5] https://mail.google.com/support/bin/answer.py?answer=78799
[6] https://mail.google.com/support/bin/answer.py?answer=13287
[7] https://mail.google.com/support/bin/answer.py?answer=78799
[8] http://userscripts.org/
[9] http://www.mozilla.org/en-US/firefox/new/
[10] https://addons.mozilla.org/en-US/firefox/addon/greasemonkey/

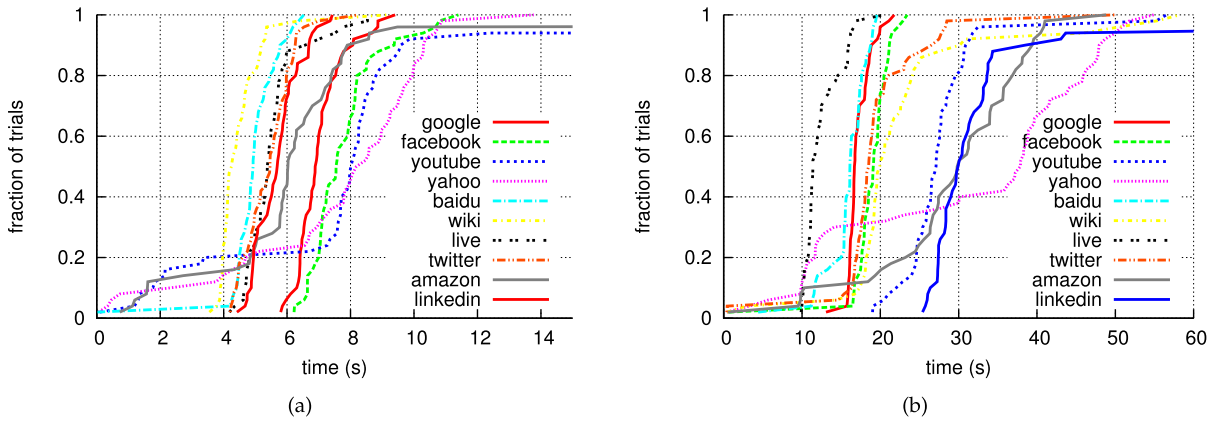


Fig. 6. The CDF of (a) the time to the first appearance (TFA) and (b) the total browsing time (TBT) using SWEET.

*Client registration:* Before being able to request data from Internet destinations, a user needs to be registered by the SWEET server. Fig. 5(b) shows the time taken to exchange registration messages between a client and the SWEET server. Note that the client registration needs to be performed *only once* for a long period of time. The figure shows that more than 90% of registrations establish in less than 8 seconds (with an average of 6.4 seconds).

We use two metrics to evaluate the latency performance of SWEET in browsing websites: the *time to the first appearance (TFA)* and the *total browsing time (TBT)*. The TFA is the time taken to receive the first response from a requested web destination. It is an important metric in measuring user convenience during web browsing. For instance, suppose that a client requests a URL, e.g., http://www.cnn.com/some_news.html. By the TFA time the client receives the first HTTP RESPONSE(s) from the destination, which include the URL's text parts (perhaps the news article) along with the URLs of other objects on that page, e.g., images, ads hosted by other websites, etc. At this time the client can start reading the received portion of the website (e.g., the news article), while her browser sends requests for other objects on that webpage. On the other hand, the total browsing time (TBT) is the time after which the browser finishes fetching all of the objects in the requested URL.

Using our prototype we measure the end-to-end web browsing latency for the client to reach different web destinations. Fig. 6(a) shows the TFA for the top 10 web URLs from Alexa's most-visited sites ranking [46]. The median is about 5 seconds across all experiments, which is very promising to user convenience.

On the other hand, Fig. 6(b) shows the total browsing time (TBT) for the same set of destinations (50 runs for each website). As can be seen, the destinations that contain more web objects (e.g., yahoo and linkedin) take more time to get completely fetched (note that after the TFA time the user can start reading the webpage). We also run similar experiments through the popular Tor [45] anonymous network to compare its latency performance with SWEET. Fig. 7 compares the latency CDF for SWEET and Tor. As expected, our simple implementation of SWEET takes more time than Tor to browse web pages, however, it provides a sufficient performance

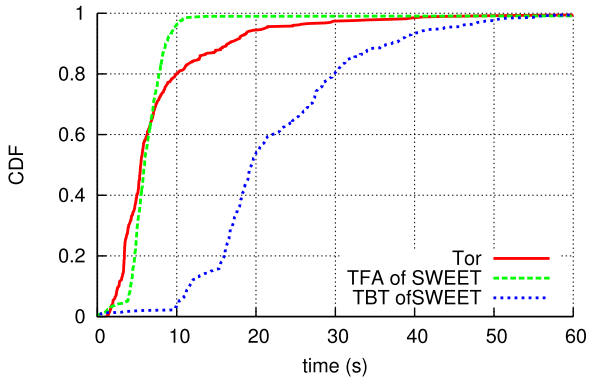


Fig. 7. Comparing the average latency of SWEET and Tor.

for normal web browsing. This is in particular significant considering the strong *availability* of SWEET compared to other circumvention systems. Additionally, we believe that further optimizations on SWEET server's proxy (like those implemented by Tor exit nodes) will further improve the performance. Our techniques are also amenable to standard methods to improve web latency, such as plugin-based caching and compression, which can make web browsing tolerable in high delay environments [47].

### *B. Traffic Analysis*

A powerful censor can perform traffic analysis to detect the use of SWEET, e.g., by comparing a user's email communications with that of a typical email user. As a result, a SWEET user who is concerned about unobservability needs to ensure that her SWEET email communications mimic that of a normal user (a user who does not fear reprisal from her government might opt to have lower unobservability in order to gain a higher communication bandwidth). It should be mentioned that such traffic analysis is expensive for censors considering the large volume of email communications; it is estimated[11] that 294 billion emails were sent per day in 2011.

Fig. 8 shows the number of emails sent and received by a SWEET client to browse different websites. We observe that for any particular website the number of emails does not change at different runs. As can be seen, most of the web sites

[11] http://royal.pingdom.com/2011/01/12/internet-2010-in-numbers/

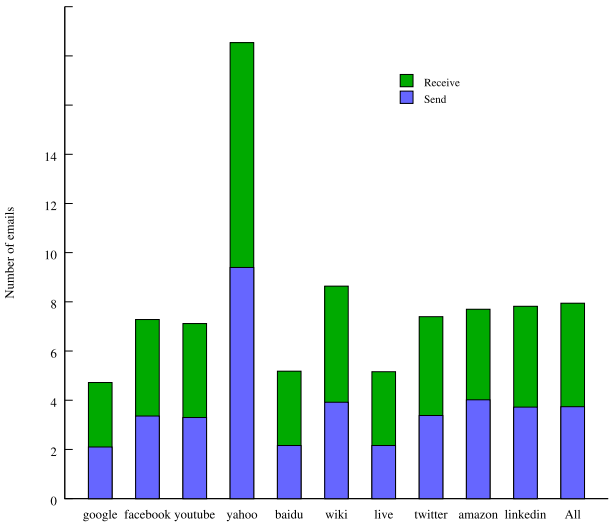


Fig. 8. The number of emails sent and received by a SWEET client to get one of the websites from Alexa's top ten ranking.

finish in less than three SWEET emails in each direction. The exception is the Yahoo web page as it contains many web objects, each hosted by different URLs (note that the number of emails affects the latency performance only *sub-linearly*, since some emails are sent and received simultaneously.). Also, the average number in each way of a connection is about 4 emails. A recent study [48] on email statistics predicts that an average user will send 35 emails and will receive 75 emails per day in 2012 (the study predicts the numbers to increase annually). In addition, membership in mailing lists[12] and Internet groups[13,14] is popular among Internet users, producing even more emails by normal email users. As an indication of the popularity of such services, Yahoo in 2010 announced[15] that 115 million unique users are collectively members of more than 10 million Yahoo Groups. Based on the mentioned statistics, we estimate that a conservative SWEET user can perform 35-70 web downloads per day, or make 10-20 interactive web connections, while ensuring unobservability of SWEET usage. Note that the censored users use SWEET only to browse "censored" Internet webpages and they use regular webbrowsing for non-censored websites. Also, normal citizens who do not fear being caught by the censors may decide to ignore resistance against traffic analysis in order to achieve higher bandwidths.

The censors may also try to detect SWEET users by analyzing the inter-arrival times of the email messages exchanged between SWEET users and their mail service providers. More specifically, a SWEET client may send and receive multiple emails in a shorter time interval compared to regular email clients. We, however, argue that this is not a serious vulnerability for SWEET. As discussed earlier, we require SWEET clients to use mail service providers that encrypt email exchanges (otherwise, the censors can simply filter SWEET emails by searching for the email addresses of SWEET servers). The use of encryption prevents the censors from identifying the number of emails exchanged between a SWEET user and her mail service provider.

[12]http://gcc.gnu.org/lists.html

[13]http://groups.yahoo.com

[14]http://groups.google.com

[15]http://www.eweek.com/c/a/Search-Engines/Yahoo-Refreshes-Upgrades-Some-Products-775120/

## VIII. Conclusions

In this paper, we presented SWEET, a deployable system for unobservable communication with Internet destinations. SWEET works by tunneling network traffic through widely-used public email services such as Gmail, Yahoo Mail, and Hotmail. Unlike recently-proposed schemes that require a collection of ISPs to instrument router-level modifications in support of covert communications, our approach can be deployed through a small applet running at the user's end host, and a remote email-based proxy, simplifying deployment. Through an implementation and evaluation in a wide-area deployment, we find that while SWEET incurs some additional latency in communications, these overheads are low enough to be used for interactive accesses to web services. We feel our work may serve to accelerate deployment of censorship-resistant services in the wide area, guaranteeing high availability.

## References

[1] J. Zittrain and B. Edelman, "Internet filtering in China," *IEEE Internet Comput.*, vol. 7, no. 2, pp. 70–77, Mar. 2003.

[2] (Nov. 2007). *Defeat Internet Censorship: Overview of Advanced Technologies and Products*. [Online]. Available: http://www.internetfreedom.org/archive/DefeatInternet Censorship White Paper.pd

[3] C. S. Leberknight, M. Chiang, H. V. Poor, and F. Wong. (2010). *A Taxonomy of Internet Censorship and Anti-Censorship*. [Online]. Available: http://www.princeton.edu/ chiangm/anticensorship.pdf

[4] I. Clarke, S. G. Miller, T. W. Hong, O. Sandberg, and B. Wiley, "Protecting free expression online with freenet," *IEEE Internet Comput.*, vol. 6, no. 1, pp. 40–49, Jan. 2002.

[5] *Ultrasurf*, accessed on Jan. 7, 2017. [Online]. Available: https://ultrasurf.us/

[6] J. Jia and P. Smith. (2004). *Psiphon: Analysis and Estimation*. [Online]. Available: http://www.cdf.toronto.edu/ csc494h/reports/2004-fall/psiphon_ae.html

[7] I. Cooper and J. Dilley, "Known HTTP proxy/caching problems," IETF, Fremont, CA, USA, Tech. Rep. Internet RFC 3143, Jun. 2001.

[8] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proc. USENIX Secur. Symp.*, 2004, pp. 21–37.

[9] J. Boyan, "The anonymizer: Protecting user privacy on the Web," *Comput.-Mediated Commun. Mag.*, vol. 4, no. 9, pp. 1–6, Sep. 1997.

[10] *DynaWeb*, accessed on Jan. 7, 2017. [Online]. Available: http://www.dongtaiwang.com/home_en.php

[11] R. Clayton, S. J. Murdoch, and R. N. M. Watson, "Ignoring the great firewall of China," in *Proc. Int. Workshop Privacy Enhancing Technol.*, 2006, pp. 20–35.

[12] Y. Sovran, A. Libonati, and J. Li, "Pass it on: Social networks stymie censors," in *Proc. 7th Int. Conf. Peer-to-Peer Syst.*, Feb. 2008, p. 3. [Online]. Available: http://www.iptps.org/papers-2008/73.pdf

[13] D. McCoy, J. A. Morales, and K. Levchenko, "Proximax: A measurement based system for proxies dissemination," *Financial Cryptogr. Data Secur.*, vol. 5, no. 9, pp. 1–10, 2011.

[14] N. Feamster, M. Balazinska, W. Wang, H. Balakrishnan, and D. Karger, "Thwarting Web censorship with untrusted messenger discovery," in *Int. Workshop Privacy Enhancing Technol.*, 2003, pp. 125–140.

[15] M. Mahdian, "Fighting censorship with algorithms," in *Proc. Int. Conf. Fun Algorithms*, 2010, pp. 296–306. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-13122-6_29

[16] J. McLachlan and N. Hopper, "On the risks of serving whenever you surf: Vulnerabilities in Tor's blocking resistance design," in *Proc. 8th ACM Workshop Privacy Electron. Soc.*, Nov. 2009, pp. 31–40. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1655188.1655193

[17] P. Winter and S. Lindskog. Apr. 2012. "How China is blocking Tor." [Online]. Available: https://arxiv.org/abs/1204.0447

[18] (Sep. 2007). *Tor Partially Blocked in China*. [Online]. Available: https://blog.torproject.org/blog/tor-partially-blocked-china

[19] N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. Karger, "Infranet: Circumventing Web censorship and surveillance," in *Proc. 11th USENIX Secur. Symp.*, Aug. 2002, pp. 247–262. [Online]. Available: http://www.usenix.org/events/sec02/feamster.html

[20] S. Burnett, N. Feamster, and S. Vempala, "Chipping away at censorship firewalls with user-generated content," in *Proc. USENIX Secur. Symp.*, 2010, pp. 463–468. [Online]. Available: http://www.usenix.org/events/sec10/tech/full_papers/Burnett.pdf

[21] R. Dingledine and N. Mathewson, "Design of a blocking-resistant anonymity system," Tor Project, Tech. Rep. 1, Nov. 2006.

[22] J. Karlin *et al.*, "Decoy routing : Toward unblockable Internet communication," in *Proc. USENIX (FOCI)*, 2011, pp. 1–6.

[23] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman, "Telex: Anticensorship in the network infrastructure," in *Proc. 20th USENIX Secur. Symp.*, Aug. 2011, pp. 1–15.

[24] A. Houmansadr, G. T. K. Nguyen, M. Caesar, and N. Borisov, "Cirripede: Circumvention infrastructure using router redirection with plausible deniability categories and subject descriptors," in *ACM Conf. Comput. Commun. Secur. (CCS)*, Chicago, IL, USA, 2011, pp. 187–200.

[25] H.-C. Hsiao *et al.*, "LAP: Lightweight anonymity and privacy," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 506–520.

[26] H. M. Moghaddam, B. Li, M. Derakhshani, and I. Goldberg, "Skypemorph: Protocol obfuscation for Tor bridges," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2012, pp. 97–108.

[27] Q. Wang, X. Gong, G. Nguyen, A. Houmansadr, and N. Borisov, "Censor Spoofer: Asymmetric communication using IP spoofing for censorship-resistant Web browsing," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2012, pp. 121–132.

[28] Z. Weinberg *et al.*, "Stego Torus: A camouflage proxy for the Tor anonymity system," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2012, pp. 109–120.

[29] A. Houmansadr, C. Brubaker, and V. Shmatikov, "The parrot is dead: Observing unobservable network communications," in *Proc. IEEE Symp. Secur. Privacy*, May 2013, pp. 65–79.

[30] A. Houmansadr, T. Riedl, N. Borisov, and A. Singer, "I want my voice to be heard: IP over voice-over-IP for unobservable censorship circumvention," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2013, pp. 1–17.

[31] M. Leech *et al.* (Apr. 1996). *RFC 1928: SOCKS Protocol Version 5*. [Online]. Available: http://ftp://ftp.internic.net/rfc/rfc1928.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc1928.txt

[32] C. Brubaker, A. Houmansadr, and V. Shmatikov, "Cloud transport: Using cloud storage for censorship-resistant networking," in *Proc. PETS*, 2014, pp. 1–20.

[33] H. Zolfaghari and A. Houmansadr, "Practical censorship evasion leveraging content delivery networks," in *Proc. 23rd ACM Conf. Comput. Commun. Secur. (CCS)*, 2016, pp. 1–12.

[34] J. Holowczak and A. Houmansadr, "CacheBrowser: Bypassing Chinese censorship without proxies using cached content," in *Proc. 22nd ACM Conf. Comput. Commun. Secur. (CCS)*, 2015, pp. 70–83.

[35] R. McPherson, A. Houmansadr, and V. Shmatikov, "CovertCast: Using live streaming to evade Internet censorship," in *Proc. Privacy Enhancing Technol. (PETS)*, 2016, pp. 212–225.

[36] M. Schuchard, J. Geddes, C. Thompson, and N. Hopper, "Routing around decoys," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 85–96.

[37] A. Houmansadr, E. L. Wong, and V. Shmatikov, "No direction home: The true cost of routing around decoys," in *Proc. 21st Annu. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2014, pp. 1–14.

[38] M. Nasr and A. Houmansadr, "GAME OF DECOYS: Optimal decoy routing through game theory," in *Proc. 23rd ACM Conf. Comput. Commun. Secur. (CCS)*, 2016, pp. 1–12.

[39] *The FOE Project*, accessed on Jul. 1, 2013. [Online]. Available: http://code.google.com/p/foe-project/

[40] *MailMyWeb*, accessed on Jan. 7, 2017. [Online]. Available: http://www.mailmyweb.com/

[41] A. Juels and J. G. Brainard, "Client Puzzles: A cryptographic countermeasure against connection depletion attacks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 1999, pp. 151–165. [Online]. Available: http://www.isoc.org/isoc/conferences/ndss/99/proceedings/papers/juels.pdf

[42] M. Steiner, G. Tsudik, and M. Waidner, "Diffie–Hellman key distribution extended to groups," in *Proc. 3rd ACM Conf. Comput. Commun. Secur.*, Mar. 1996, pp. 31–37.

[43] R. Fielding *et al.*, "Hypertext transfer protocol—HTTP/1.1," IETF, Fremont, CA, USA, Tech. Rep. RFC 2616, Jun. 1999, [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[44] C. Callanan, H. Dries-Ziekenheiner, A. Escudero-Pascual, and R. Guerra. (Mar. 2010). *Leaping Over the Firewall: A Review of Censorship Circumvention Tools*. [Online]. Available: http://www.freedomhouse.org/sites/default/files/ inline images/Censorship.pdf

[45] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr, "Towards an analysis of onion routing security," in *Proc. Designing Privacy Enhancing Technol., Workshop Design Issues Anonymity Unobserva.*, Jul. 2000, pp. 96–114.

[46] (Feb. 2012). *Defeat Internet Censorship: Overview of Advanced Technologies and Products*. [Online]. Available: http://www.alexa.com/topsites

[47] J. Chen, D. Hutchful, W. Thies, and L. Subramanian. (2011). *Analyzing and Accelerating Web Access in a School in Peri-Urban India*. [Online]. Available: www(Companion Volume)

[48] S. Radicati and Q. Hoang. (2011). *Email Statistics Report 2011–2015*. [Online]. Available: http://www. radicati.com/wp/wp-content/uploads/2011/05/Email-Statistics-Report-2011-2015-Executive-Summary.pdf

**Amir Houmansadr** (M'11) received the Ph.D. degree from the University of Illinois at Urbana–Champaign in 2012. He is currently an Assistant Professor with the College of Information and Computer Sciences, University of Massachusetts Amherst. His research interests include network security and privacy, which includes problems, such as Internet censorship resistance, statistical traffic analysis, location privacy, cover communications, and privacy in next-generation network architectures. He was a recipient of several awards, including the Best Practical Paper Award at the IEEE Symposium on Security & Privacy (Oakland) in 2013, the Google Faculty Research Award in 2015, and the NSF CAREER Award in 2016.

**Wenxuan Zhou** received the bachelor's degree in electronic engineering from the Beijing University of Aeronautics and Astronautics, China, and the master's degree in computer science from the University of Illinois at Urbana–Champaign, where she is currently pursuing the Ph.D. degree in computer science, advised by Prof. M. Caesar. Her research focuses on network verification and synthesis, with an emphasis on software-defined networks, data centers, and enterprise networks.

**Matthew Caesar** (M'07) received the Ph.D. degree from the University of California at Berkeley, Berkeley, CA, USA, in 2007. He is currently the Co-Founder and President of Veriflow Systems, Inc., and an Associate Professor with the Department of Computer Science, University of Illinois at Urbana–Champaign. He has been involved in the area of network security for over two decades, publishing over 50 technical papers, which appear in highly selective academic conferences and have resulted in multiple best paper awards. He is a CAS Fellow (2013). He received the NSF CAREER Award (2011), the DARPA CSSG Membership (2011), and the Test of Time Award at the USENIX Symposium on Networked Systems Design and Implementation for his foundational contributions to software-defined networking.

**Nikita Borisov** (M'06) received the Ph.D. degree from the Department of Computer Science, University of California at Berkeley, Berkeley, CA, USA, in 2005. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Illinois at Urbana–Champaign. His research interests include anonymity, network security, and privacy. He is a recipient of the NSF CAREER Award.