/burl@stx null def /BU.S /burl@stx null def def /BU.SS currentpoint /burl@lly exch def /burl@llx exch def burl@stx null ne burl@endx burl@llx ne BU.FL BU.S if if burl@stx null eq burl@llx dup /burl@stx exch def /burl@endx exch def burl@lly dup /burl@boty exch def /burl@topy exch def if burl@lly burl@boty gt /burl@boty burl@lly def if def /BU.SE currentpoint /burl@ury exch def dup /burl@urx exch def /burl@endx exch def burl@ury burl@topy lt /burl@topy burl@ury def if def /BU.E BU.FL def /BU.FL burl@stx null ne BU.DF if def /BU.DF BU.BB [/H /I /Border [burl@border] /Color [burl@bordercolor] /Action ¡¡ /Subtype /URI /URI BU.L ¿¿ /Subtype /Link BU.B /ANN pdfmark /burl@stx null def def /BU.BB burl@stx HyperBorder sub /burl@stx exch def burl@endx HyperBorder add /burl@endx exch def burl@boty HyperBorder add /burl@boty exch def burl@topy HyperBorder sub /burl@topy exch def def /BU.B /Rect[burl@stx burl@boty burl@endx burl@topy] def /eop where begin /@ldeopburl /eop load def /eop SDict begin BU.FL end @ldeopburl def end /eop SDict begin BU.FL end def ifelse

SWEET: Serving the Web by Exploiting Email Tunnels

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

Abstract—Open communications over the Internet poses 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 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.

1 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 repressive regimes [**?**], [**?**], [**?**], [**?**], [**?**], [**?**]. The earliest circumvention tools are HTTP proxies [**?**], [**?**], [**?**] 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 [**?**], [**?**], rendered the use of HTTP proxies ineffective for circumvention. This led to the advent of more advanced tools such as Ultrasurf [**?**] and Psiphon [**?**], 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

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 2.

A more recent approach in designing unobservalbe circumvention systems is to imitate popular applications like Skype and HTTP, as suggested by SkypeMorph [**?**], CensorSpoofer [**?**], and StegoTorus [**?**]. However, it has recently been shown [**?**] 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 [**?**], [**?**] 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 [**?**] by proxying the encapsulated 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 4, 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 4, 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 **??** 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 5.2.

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

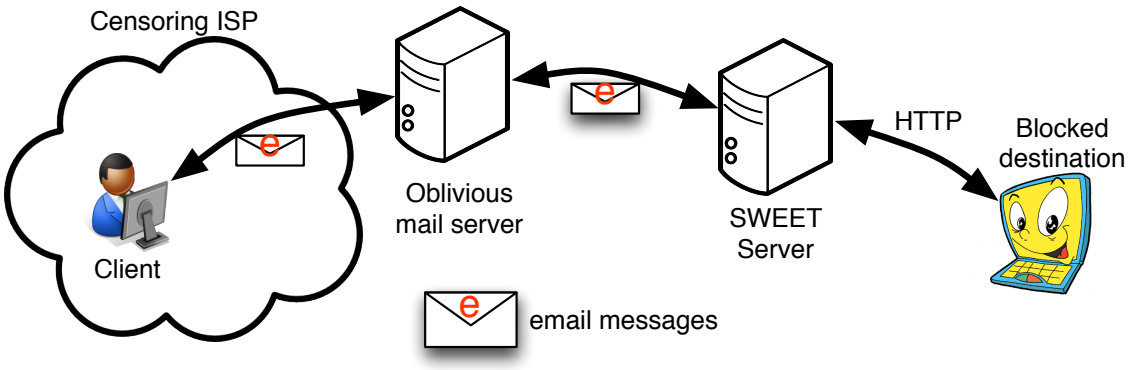


Fig. 1. Overall architecture of SWEET.

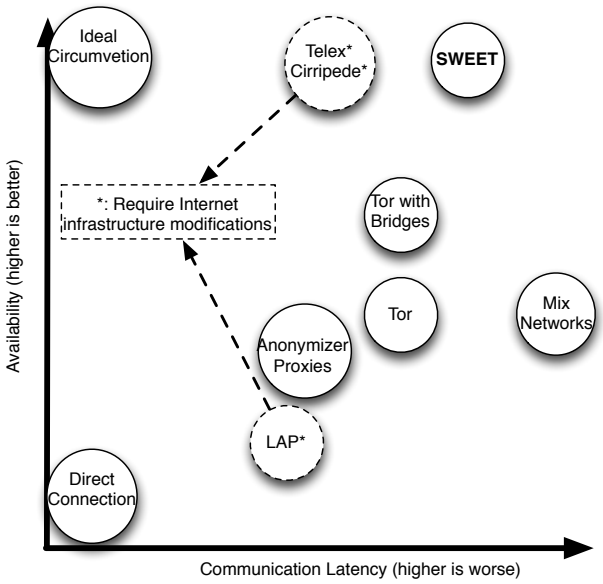


Fig. 2. Availability and communication latency comparison of circumvention systems.

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 2, we discuss the related work on unobservable censorship circumvention. In Section 3, we reviews our threat model. We provide the detailed description of the proposed circumvention system, SWEET, in Section 4. We discuss SWEET's censorship features, including its availability, in Section 5 and compare it with the literature. Our prototype implementation and evaluations are presented in Sections **??** and **??**, respectively. Finally, we conclude the paper in Section **??**.

2 Related Work

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

Several designs [?], [?], [?] 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 [?] 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 [?] 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 [?], [?], [?]. 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 [?] and Cirripede [?] 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 [?] uses an additional client registration stage that provides some advantages and limitations as compared to Telex [?] and Decoy routing [?] 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 [?], [?], [?].

There are two projects that work in a similar manner to SWEET: FOE [?] and MailMyWeb [?]. 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 4.4.

3 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 [?]. 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.

4 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.

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.

4.1 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 5). 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., [**?**]). 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 [**?**] 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 [**?**].

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.

4.2 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

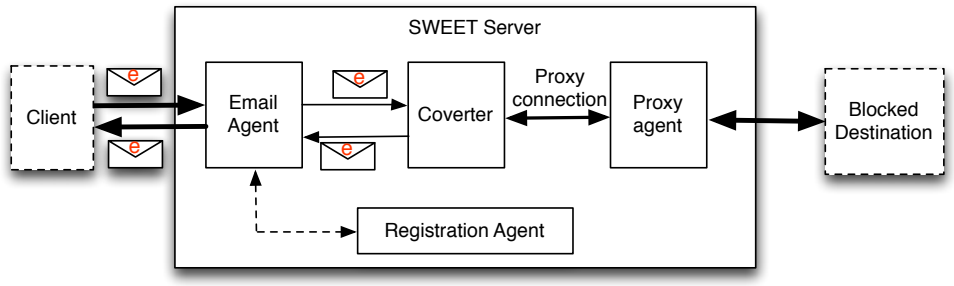


Fig. 3. The main architecture of SWEET server.

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 4.1.

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.

4.2.1 *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.

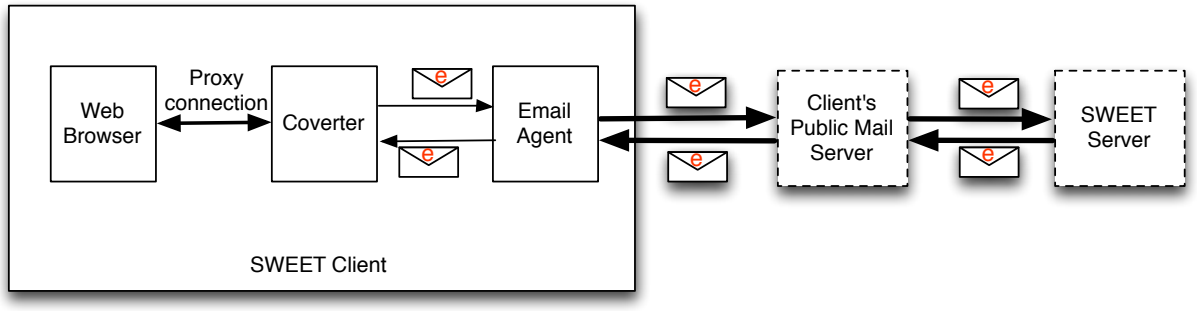
The email agent is also configured to receive emails from the client's email account through an email retrieval protocol, e.g., IMAP or POP3. This allows the email agent to continuously look for new emails from the server. Once new emails are received, the email agent passes them to the converter, who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

4.2.2 *Webmail-based design*

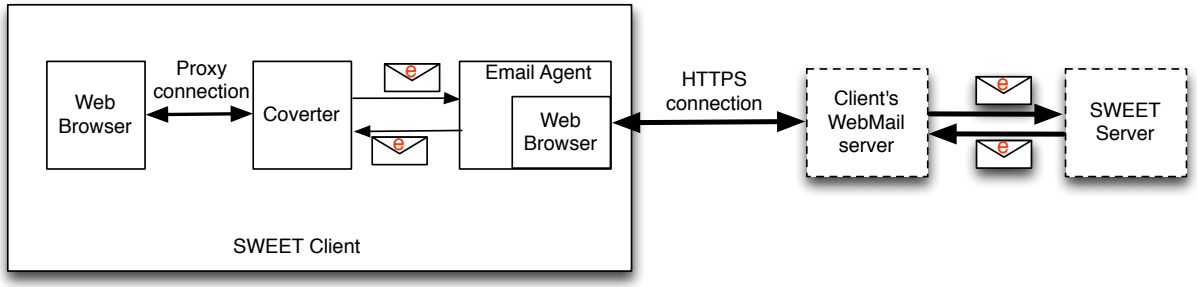
Alternatively, the SWEET client can use the webmail interface of the client's public email provider. as showed in Fig. 4(b).The main difference with the protocol-based design is that in this case the email agent (❷) uses a web browser to exchange emails. More specifically, the email agent uses its web browser to open a webmail interface with the client's email account, using the user's authentication credentials for logging in. Through this HTTP/HTTPS connection, the email agent communicates with the SWEET server by sending and receiving emails.

4.3 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 [?] and HTTP [?] 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



(a) The protocol-based design.



(b) The webmail-based design.

Fig. 4. Design of SWEET client software.

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.

4.4 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 [**?**], and the for-profit service of MailMyWeb [**?**]. 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.

5 Discussions and Comparisons

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

5.1 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 [**?**], [**?**], [**?**], [**?**], [**?**], [**?**].

We claim that a censor is not easily able to distinguish between SWEET's email messages and benign email messages. As described in Section 4, 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 **??**.

5.2 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 4.1). 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.

5.3 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 [**?**].

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 [**?**], [**?**], [**?**], or end-hosts [**?**], [**?**]. 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