Richard McPherson*, Amir Houmansadr, and Vitaly Shmatikov

CovertCast:

Using Live Streaming to Evade Internet Censorship

Abstract: We design, implement, and evaluate CovertCast, a censorship circumvention system that broadcasts the content of popular websites in real-time, encrypted video streams on common live-streaming services such as YouTube. CovertCast does not require any modifications to the streaming service and employs the same protocols, servers, and streaming software as any other user of the service. Therefore, CovertCast cannot be distinguished from other live streams by IP address filtering or protocol fingerprinting, raising the bar for censors.

DOI 10.1515/popets-2016-0024
Received 2015-11-30; revised 2016-03-01; accepted 2016-03-02.

1 Introduction

Existing technologies for circumventing Internet censorship [5, 25, 29] are not robust against technically competent censors. The network protocols and connection endpoints used by circumvention systems are distinct from those used by any other Internet service. Therefore, ISP-level censors can easily recognize and block them at the network level at a very low cost, without disrupting any other Internet activities.

We believe that censorship circumvention systems can take advantage of the fact that many popular online services already provide end-to-end encryption and are therefore opaque to censoring ISPs. By using such services to transmit censored content, a circumvention system can make it much more difficult for network-level censors to distinguish permitted and censored content. At the very least, basic censorship techniques such as blocking the IP addresses of circumvention servers and recognizing the distinct protocols that carry circumvention traffic will no longer work.

In this paper, we investigate how to use *live streaming* as a censorship circumvention medium. Live streaming has become an extremely popular way to broadcast gameplays, e-sports competitions, podcasts, live-music and sports events, pirated TV shows and movies, etc. Live-streaming platforms such as Twitch boast over 100 million unique monthly viewers[1] and new platforms emerge every year. Unlike conventional online video services, which primarily support viewing of previously uploaded media files, live streaming enables real-time, high-bandwidth one-to-one and one-to-many transmission of video content.

Our contributions. We design, implement, and evaluate CovertCast, a new system that distributes digital content (in particular, news websites) by encoding it as a sequence of images and transmitting these images to users via a live-streaming service such as YouTube. The design and implementation of CovertCast are service-agnostic and can be deployed with any live-streaming service. We chose YouTube because, unlike some other services, it protects all streams end-to-end using HTTPS. This ensures that censors can neither inspect the contents of any stream, nor tamper with it.

A CovertCast server, located outside the censored network, initiates a live stream. For each website being served, the server crawls the site, encodes its content into images, and broadcasts the images via the live stream. A CovertCast client consists of a local Web proxy and the code that downloads and demodulates images from live streams. The user simply changes his or her browser's proxy settings to use the CovertCast proxy. After the CovertCast client demodulates an image back into Web content, it intercepts any requests sent via the proxy and responds to them locally. Because live-streaming services are engineered to support a large number of clients "watching" a single stream, CovertCast scales easily and one CovertCast server can handle many clients.

To the censor controlling the user's network, CovertCast traffic looks like any other traffic from a given live-streaming platform (e.g., users watching someone broadcasting from their webcam, which is a very popular use of live streaming). CovertCast uses exactly the same video codecs, network protocols, and video servers, all supplied by the live-streaming platform, as any other stream on the same platform. HTTPS-encrypted YouTube streams produced by CovertCast exhibit the same traffic patterns as many other YouTube streams and, to the best of our knowledge, cannot be accurately distinguished using previously proposed traffic analysis techniques.

***Corresponding Author: Richard McPherson:** The University of Texas at Austin, email: richard@cs.utexas.edu
Amir Houmansadr: University of Massachusetts Amherst, email: amir@cs.umass.edu
Vitaly Shmatikov: Cornell Tech, email: shmat@cs.cornell.edu

1 http://www.twitch.tv/year/2014

CC BY-NC-ND

Encoding Web content into live video streams presents several technical challenges. The main challenge is the initial lag between the time the "caster" initiates his live stream and the time the service starts displaying the stream to the subscribers. Depending on the platform, this delay ranges from 25 to 50 seconds. Because of this latency, CovertCast broadcasts websites instead of using the normal HTTP request-response model. This sidesteps the latency issue and allows a server to scale to any number of clients at the cost of reduced browsing options for CovertCast's users.

We evaluate CovertCast by loading various news sites, which is reportedly the most frequent use of censorship circumvention software [4]. Although CovertCast takes a relatively long time to load a page (one to two minutes for most news sites we examined), this latency is due to ensuring that the streaming service broadcasts images without visual artifacts and is not an inherent limitation of CovertCast. We also evaluate CovertCast under various network conditions, including low bandwidth and high packet-drop rates.

2 Related Work

Proxying. Filtering certain Internet destinations by IP address is the simplest, most widely deployed censorship technology. Consequently, many circumvention systems relay users' traffic through a *proxy* server. Examples of proxy services include Psiphon [25], UltraSurf [29], and Anonymizer [1].

Users must trust proxy operators to not reveal users' addresses. Furthermore, proxies' own IP addresses are easily discoverable by *insider attackers*, i.e., censors who pretend to be users of the system [19, 20, 27, 32]. There is no robust defense against this attack because proxies cannot reliably distinguish genuine and fake users. Once censors learn the proxy's address(es), they can blacklist them, identify past users from network traces, or even leave the proxy accessible to identify its future users [21].

Tor is a network of relays that can be used to construct overlay routes to Internet destinations without trusting every individual relay. These routes can then be used, for example, for anonymous Web browsing. The addresses of Tor relays are public, thus censors can and do block them. A Tor bridge is a hidden proxy that clients can use as a gateway to the Tor network [5]. As with other proxy-based services, censors use insider attacks and active probes [34, 35] to discover and blacklist the bridges' IP addresses, identify their users, and even block access to other bridges via a zig-zag attack [28].

Steganography. Several proposed systems for censorship circumvention rely on covert communications. In Infranet [7], the client pretends to normally browse an unblocked website that has secretly volunteered to serve censored content. The client uses a secret codebook that it shares with the server to generate a sequence of HTTP requests encoding its request for censored content. The server returns the requested content by embedding it in image objects on the unblocked site. As soon as the censors become aware that a certain site is being used for censorship circumvention (e.g., via an insider attack), they can easily block it.

Collage [3] aims to mitigate this issue by utilizing public, oblivious websites that share user-generated content such as photos and tweets. Clients' requests, as well as censored content, are hidden steganographically inside images hosted on the content-sharing site. These images are tagged in a certain way, which is known only to the participants, so that they are easily retrievable. Collage has low bandwidth and is not suitable for tasks such as Web browsing.

Parrots. In addition to blacklisting the IP addresses of known circumvention servers, censors also analyze network traffic and look for content identifiers and patterns characteristic of circumvention systems. In response, a new class of circumvention proposals aim to generate traffic that looks like popular Internet protocols. Examples include SkypeMorph [23], StegoTorus [33], and CensorSpoofer [31] that imitate protocols such as Skype, HTTP, and SIP-based VoIP.

Houmansadr et al. [11] demonstrate a range of attacks of different types and complexity that distinguish the traffic of parrot systems from the genuine traffic of the Internet protocols they attempt to imitate, and argue that perfect imitation of complex network protocols is fundamentally infeasible.

Hide-within systems. Unlike parrot systems that only imitate popular network protocols, hide-within systems tunnel circumvention traffic using the actual protocols. FreeWave [13] modulates traffic into acoustic signals carried via VoIP, while SWEET [15] hides traffic in email messages. Both FreeWave and SWEET have very low bandwidth. By contrast, CovertCast encodes circumvention traffic into video streams, resulting in much higher throughput and enabling the broadcast model of content distribution. CloudTransport [2] tunnels traffic via cloud storage services. Using commercial services such as Amazon S3 for CloudTransport has monetary costs and may not be affordable for users in many countries.

CovertCast's design is similar to hide-within systems in that CovertCast also tunnels circumvention traffic through a widely used network protocol (live video streaming). CovertCast's broadcast model, however, lets it scale easily with the number of circumvention users, in contrast to the hide-within systems like Freewave where the bandwidth and computation costs of circumvention servers grow linearly with the number of users. Superior scalability also makes CovertCast robust to denial-of-service (DoS) attacks on circumvention servers. For

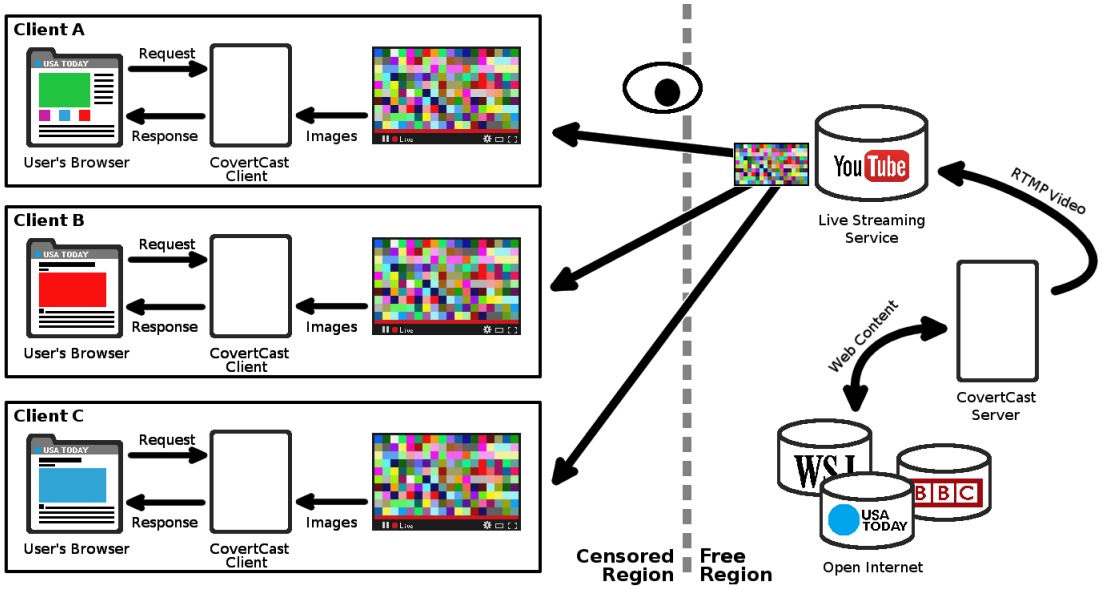

Fig. 1. Architecture of CovertCast.

instance, censors can launch a Sybil-based DoS attack against a FreeWave server by setting up large numbers of FreeWave clients and using them to send HTTP requests for big webpages. This attack does not work against CovertCast because the server workload in CovertCast is independent of the number of clients receiving its live streams.

Domain fronting. Domain fronting [8] is a new technique for setting up proxies in proxy-based circumvention systems in a way that makes them resistant to IP address blocking. The main idea of domain fronting is to run circumvention proxies, e.g., Tor bridges [5], on Web services that share their IP addresses with (many) other services. Therefore, any blocking of the circumvention proxies' addresses will cause collateral damage by also blocking co-located services.

Domain fronting has been adopted by several proxy-based circumvention systems, including Tor, Psiphon [25], FireFly Proxy,[2] and Flashlight HTTP proxy.[3] Tor's meek pluggable transport [22] implements domain fronting on Google App Engine, Microsoft Azure cloud infrastructure, and CloudFront CDN. Wang et al. [30] demonstrate that censors can detect meek with high accuracy and low false positives using decision-tree-based classifiers trained on the distinguishing features of circumvention and non-circumvention traffic.

With CacheBrowser [10], users directly download censored content from an arbitrary edge server of the hosting CDN. Similar to meek and other domain fronting systems, CacheBrowser assumes that censors will not block CDNs' IP addresses due to high collateral damage to other services. CacheBrowser is limited to serving only CDN-hosted content.

Facet. Facet [18] is a special-purpose circumvention system for watching blocked videos. Facet downloads videos from a blocked service such as YouTube, Vine, or Vimeo and resends them to users via a Skype videoconferencing call, at a significantly lower resolution. In contrast to Facet, CovertCast is a generic, content-agnostic circumvention system. CovertCast encodes arbitrary Internet content into video streams and thus supports key circumvention tasks such as Web browsing, which is not possible in Facet.

Decoy routing. Decoy routing [12, 16, 37] aims to incorporate censorship circumvention into inter-domain routing. Cooperating ISPs are supposed to have their routers identify covert, steganographically marked traffic generated by users in the censorship region and deflect it to the destinations specified by the senders. Schuchard et al. [26] showed that a state-level adversary capable of controlling the routing policies of ISPs in the censorship region can force them to route traffic so that it does not pass through the known decoy routers. Houmansadr et al. [14] demonstrate that such a rerouting-based attack will impose significant monetary and social costs on the censoring ISPs. In any case, deploying decoy routing systems is very challenging because ISPs have no incentive to actively assist censorship circumvention.

2 https://github.com/yinghuocho/firefly-proxy

3 https://github.com/getlantern/flashlight

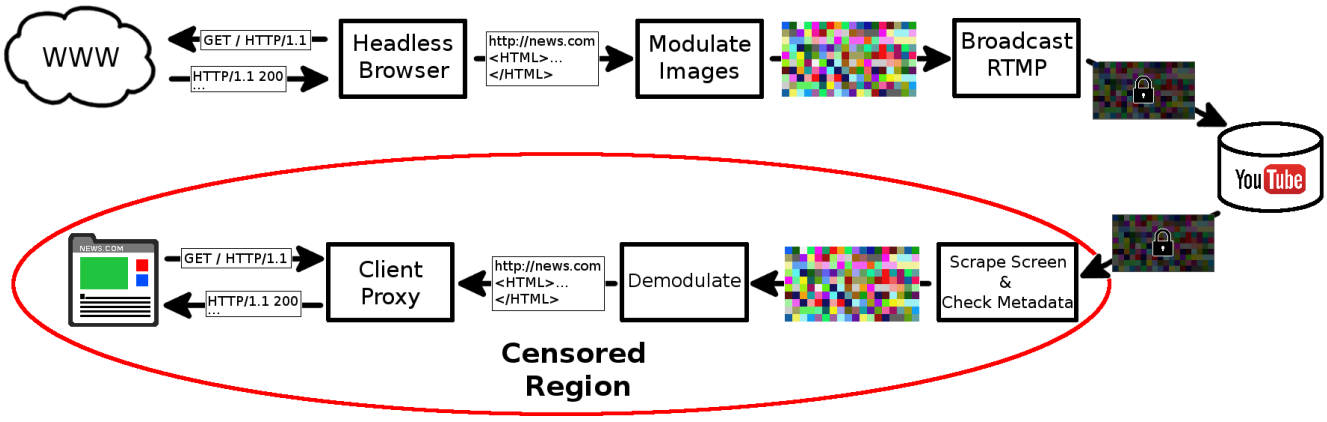

Fig. 2. The individual parts of a CovertCast client and server. Web content is downloaded by a server, modulated into images, and then uploaded to a live-streaming service as video. A client monitors the video and demodulates the image back into content, which it uses to respond to the requests to its proxy.

3 Threat Model

We assume that a CovertCast user is connecting to the Internet from a region that is subject to state-level Internet censorship. The user's ISP (or some higher-level autonomous system controlling the traffic of the user's ISP) monitors all network traffic and attempts to enforce some restrictions, e.g., prevent users from reaching certain Internet destinations.

To this end, the censoring ISP can block prohibited IP addresses, terminate prohibited network protocols, and selectively perturb network flows (e.g., by dropping or delaying packets) in order to identify and/or disrupt users who attempt to circumvent censorship. Censors can also perform traffic analysis to identify circumvention traffic.

For the purposes of this paper, we assume that censors are rational and refrain from actions that would inconvenience the large fraction of users who are not engaging in circumvention. For example, we assume that the censors are not blocking all encrypted traffic and permit certain protocols, in particular, popular live-streaming services that are used to watch live sports, gaming competitions, pirated movies, etc.

CovertCast uses the live-streaming service "as is." We assume that the service is neutral: it neither cooperates with CovertCast, nor actively tries to block streams generated by CovertCast servers, nor helps censors discover CovertCast streams or the users receiving these streams.

4 System Overview

4.1 Live-streaming services

Live-streaming video services allow anyone with a computer and an Internet connection to record and broadcast videos to online viewers in real time. Many of these services, such as YouTube, carry a wide variety of streams, from talk shows and TV programs to live music and sports; other services focus on a single type of stream, as Twitch.tv[4] does with video games.

Live-streaming services are rapidly growing in popularity. In 2014, Twitch was acquired by Amazon for $970 million.[5] In August of 2015, Google launched YouTube Gaming to cater to live video-game streamers.[6] That same month Twitch reported over 10.2 million unique visitors, with an average viewer watching 106 minutes a day.[7]

To use a live-streaming service, an aspiring "caster" simply has to begin recording a video, typically of their screen or webcam, and stream it to the service using a protocol such as Adobe's RTMP. Anyone wanting to view the stream can then go to the service's website and watch the live stream on its own page or *channel*.

The streaming service is an intermediary between the caster and the viewers, enabling massive scaling of content distribution. The caster only needs sufficient bandwidth to

4 http://www.twitch.tv/

5 http://gamasutra.com/view/news/224090/Amazon_to_acquire_Twitch.php

6 http://www.theverge.com/2015/8/26/9212071/youtube-gaming-app-hands-on

7 http://www.bloomberg.com/features/2015-the-big-business-of-twitch

stream his or her video to the service. The service provides the bandwidth and connections to broadcast this stream to the viewers, who can number in millions for popular streams.

Another common feature of live-streaming services is their support for the interaction between casters and viewers, with chatrooms or live comments that viewers can use to talk to each other and to the caster.

While many live streams are open for anyone to watch and comment on, several services allow a caster to restrict who can watch their channel, via passwords or lists of approved viewers. Coupled with the use of HTTPS by some of the services (in particular, YouTube), these options give casters control over access to their content.

Live-streaming services offer videos of much higher resolution and quality than one-to-one chat services such as Skype and Google Hangouts. The latter services aim to enable devices of any computational power and bandwidth to communicate in real time. This requires video-compression algorithms that work best on mostly static videos with incremental changes, i.e., talking heads on infrequently changing backgrounds. Another optimization involves automatic, incremental changes in video resolution over time as the user's bandwidth fluctuates. Because of the higher bandwidth, scalability, and control that live-streaming services offer in comparison to online video-chat services, we chose to base CovertCast on live streaming rather than Skype or Google Hangouts.

4.2 Architecture of CovertCast

CovertCast is designed to broadcast Web content by modulating it into a video and streaming the resulting video online. CovertCast software consists of two parts, the *client* and the *server*. The server communicates with one or more clients through the live-streaming service. The video stream between the server and the clients forms a tunnel that allows each connected client to receive data from outside of the censored region. Clients "watching" the channel demodulate the video into individual files that make up a website. The user's Web browser can then access the content via a local proxy.

Live-streaming channel. From the client's perspective, a live-streaming channel is an HTML page that contains an embedded video player. The video player is often Adobe's Flash Player streaming Flash Video using Adobe's Real Time Messaging Protocol (RTMP) or RTMP over SSL (RTMPS). YouTube has an option to play its videos with Flash Player or HTML5 and uses its own streaming protocol instead of RTMP.

Although we use YouTube in this paper, the live-streaming service is essentially a "dumb pipe" between the client and the server. CovertCast can be easily adopted to use a different live-streaming platform.

Fig. 3. A modulated CovertCast image for use in YouTube. Every colored block is 8×8 pixels and encodes 6 bits of data.

Client. The client part of CovertCast is run by a user located inside a censorship region. The client behaves like a normal user of the live-streaming service who is "watching" a video stream. The CovertCast client software constantly monitors the stream for new images. When it detects one, it demodulates the image and saves the extracted Web content. When the user's Web browser sends a request through the client's proxy, it is intercepted, and a response with the corresponding Web content is created locally and returned to the browser.

The client's demodulation code consists of two parts. The first part communicates with the screen scraper (see Section 5.1), crops the CovertCast image from the scraper's output, and demodulates the image's metadata (Section 4.3). If the image has already been seen or is blank, it is ignored. If the image is the next expected image, it is passed to the other part of the client, which fully demodulates it and sends the extracted raw data to the proxy. Splitting the CovertCast client into two parts lets it continuously check the live stream for new images and not slow down while demodulating.

The client includes a simple HTTP/1.1 proxy. CovertCast also supports an HTTP/2 proxy, but Firefox and Chrome still have some bugs related to HTTP/2 SSL/TLS proxies, thus our prototype relies on HTTP/1.1. The proxy intercepts all requests sent through it and responds with the requested content if it previously demodulated it. If it doesn't have the content, it replies with a 404. The proxy never forwards requests to the Internet lest it leaks information to censors.

Server. The server resides outside of the censored region and acts as a caster, creating and managing password-protected channel(s) on the live-streaming service. It also crawls websites and modulates their content into images that it broadcasts over the live-streaming service. Because the server doesn't rely on any communication from the clients and its workload is independent of the number of clients, one CovertCast server can broadcast to any number of clients, subject to the scalability of the live-streaming platform.

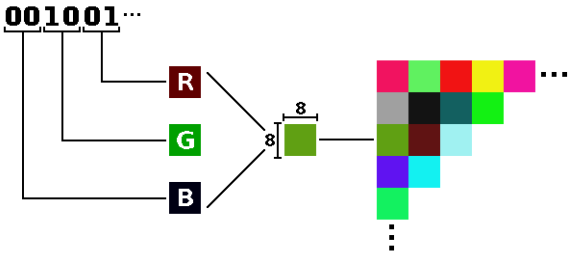

Fig. 4. Every two bits are translated into one of the RGB color channels.

The server, like the client, consists of two parts. The first part manages Web crawling. Using a headless browser, it loads a webpage and downloads all content requested by the page, including any resources fetched with JavaScript. Once the page's DOM has finished loading, the data is sent to the other part of the server, which modulates it into images (Section 4.3). Twice a second, it updates the image being broadcast to the live-streaming service via RTMP. When there are no new images to send, the server broadcasts white images and instructs the headless browser to download the next webpage.

Algorithm 1 Modulation

Require: A is an array of bits

function MODULATE(A)
  $Img \leftarrow$ create 1280×720 array of 3-tuples
  $i \leftarrow 0$
  while $i < len(A)$ **do**
    $r \leftarrow 160 \cdot A[i] + 32 \cdot A[i+1]$
    $g \leftarrow 160 \cdot A[i+2] + 32 \cdot A[i+3]$
    $b \leftarrow 160 \cdot A[i+4] + 32 \cdot A[i+5]$
    $blk_x \leftarrow (i/6) \pmod{1280}$
    $blk_y \leftarrow \lfloor (i/6)/720 \rfloor$
    for $x \in 0 \ldots 8$ **do**
      for $y \in 0 \ldots 8$ **do**
        $Img[8 \cdot blk_x + x, 8 \cdot blk_y + y] \leftarrow [r, g, b]$
      end for
    end for
    $i \leftarrow i + 6$
  end while
  return Img
end function

Alg. 1. Pseudocode to modulate an array of bits into an RGB image. Every six bits are encoded as the red, green, and blue color channels of an 8×8 block of pixels. blk_x and blk_y determine the location of the pixel block.

4.3 Video modulation

The stream sent by the caster to the streaming service is not identical to the stream received by online viewers. Streaming services process and store the video before it is shown on a channel. YouTube does a relatively large amount of preprocessing, which—we conjecture—includes format conversion.

CovertCast modulates HTTP content into images that are sent as frames in the streaming video. For robustness against compression and format conversions, CovertCast encodes bits into colored 8×8 pixel blocks (Fig. 3). In our experience, all live-streaming services allow videos to be uploaded using RTMP, thus CovertCast uses RTMP.

Even with 8×8 pixel blocks, slight color variations in online videos restricted us to encoding only 2 bits per color channel per block, or 6 bits for every set of 8×8 pixels (see Fig. 4). However, because live-streaming services allow high-resolution videos, CovertCast is able to stream 720p videos with the resolution of 1280×720. This means CovertCast is able to encode over 86,400 bits in each modulated image.

Because the video player may clip the outer pixels, we convert the outermost boxes of each frame into a black border. This reduces the number of bits per image to 84,906.

To create an image, CovertCast splits a piece of Web content into sets of 6 bits. The color of the image's i-th pixel block is determined by the i-th set of bits. The first two bits determine the values of the red color component, the next two bits determine the green component, and the final two bits determine the blue component. The bits `00` correspond to the component value of 32, `01` to 96, `10` to 160, and `11` to 225. See Alg. 1 for the pseudocode.

The content's length, URL, and HTTP status are sent alongside the data. If a piece of content is larger than 84,906 bits, it is split into multiple images. Similarly, if there is space left in an image, the next piece of content is added to it.

The demodulation algorithm (see Alg. 2) is the reverse of the modulation algorithm. For every colored block, the average values of the red, green, and blue color components are calculated. If a color component is closest to 32, it corresponds to the `00` bits, if it is closest to 96, it corresponds to `01`, etc. To increase the speed of demodulation, CovertCast only samples the innermost 4 pixels. This has a secondary effect of also increasing the accuracy of demodulation as the inner pixels are less susceptible to color bleeding from a neighboring pixel.

To ensure that the client receives the modulated image correctly, the server must send the same image for hundreds of milliseconds before switching to the next image. We found that sending 2 images per second achieves the best speed without sacrificing reliability, yielding effective bandwidth of over 169,000 bits per second. As technology evolves, we expect live-streaming services to offer more accurate video streams. CovertCast's bandwidth will increase correspondingly.

Each image starts with 5 bytes of metadata specifying the image number and the amount of modulated data in the image. The image number is used by the client to quickly check if it has already read an image and to determine whether it has missed an image. One bit of the image's metadata is used by the client for synchronization (Section 5.2).

4.4 Bootstrapping

To use CovertCast, a user in the censorship region needs to obtain the CovertCast client software via an out-of-band channel (e.g., file sharing or online social networks) and install it on his or her machine. The user then needs to discover the URL or name of the live-streaming channel that is broadcasting a website he or she wants to view. They may also need to establish appropriate credentials (i.e., password or approved username) to access the channel. Again, this can be done via out-of-band channels. For example, the user can send the request by email to the public address of a CovertCast provider. Alternatively, a user can request this information via a trusted friend who is already using CovertCast.

As in many similar systems, there are risks involved in using out-of-band channels for distribution. Unencrypted channels such as email or social networks can leak URLs of CovertCast channels and potentially help censors discover and track CovertCast users (e.g., by following friends of known users, learning who has sent or seen the URLs, etc.). This puts the onus of operational security for URL and credential distribution on CovertCast users and providers.

Unlike Tor and other censorship circumvention systems that tunnel all connections over the same circuit, CovertCast needs to distribute a separate URL for each broadcaster. If a provider broadcasts multiple channels, he can advertise them together (e.g., in the video description), but this makes it easier for a censor to discover all of them at once. Separating the channels would increase resistance to censorship, but also require more out-of-band communication to distribute their URLs. A broadcaster of multiple channels thus faces the trade-off between ease of use and security.

4.5 Content availability within channel

CovertCast only broadcasts a limited number of webpages. This raises the issue of how a client knows what webpages they can browse for a specific broadcaster. We have implemented a particular solution, but each broadcaster is free to use a different strategy.

In our prototype of CovertCast, we focus on distributing news websites. For each site, our server sends the main page (e.g., http://www.bbc.com/news) and a number of stories. Each transmitted story is linked from the main page, thus a user can load the site and browse it normally.

An alternative strategy involves using a "site map": a caster can put the list of available pages into the YouTube stream description or send this map in place of the main page. Other approaches involve sending the list of available pages along with the main page or sending multiple main pages (e.g., one each for U.S. news, European news, Asian news, etc.).

Algorithm 2 Demodulation

Require: Img is an 1280×720 array of $[r,\ g,\ b]$ values between 0 and 255

```
1:  function DEMODULATE(Img)
2:      A ← [ ]
3:      for y ∈ 0...90 do
4:          for x ∈ 0...160 do
5:              r, g, b ← 0
6:              for i ∈ 0...8 do
7:                  for j ∈ 0...8 do
8:                      r ← r + Img[8*x+i, 8*y+j][0]
9:                      g ← g + Img[8*x+i, 8*y+j][1]
10:                     b ← b + Img[8*x+i, 8*y+j][2]
11:                 end for
12:             end for
13:             for c ∈ r, g, b do
14:                 if c ≤ 8 · 64 then
15:                     A.append([0, 0])
16:                 else if c ≤ 8 · 128 then
17:                     A.append([0, 1])
18:                 else if c ≤ 8 · 192 then
19:                     A.append([1, 0])
20:                 else
21:                     A.append([1, 1])
22:                 end if
23:             end for
24:         end for
25:     end for
26:     return A
27: end function
```

Alg. 2. Pseudocode to demodulate an RGB image into an array of bits.

5 Implementation Issues

5.1 Reading the video feed

Different streaming services show live video in different ways, and it is difficult to design a uniform method for extracting frames from the players used by different services. In the

case of services that use RTMP for their live video, there are programs that can bypass the video player and download RTMP feeds directly. Using such programs, however, is typically against the terms of service and, in our experience, can easily result in a ban, locking the user out of a live-streaming service and consequently out of CovertCast. Furthermore, accessing the feed directly instead of through the service's video player may provide a behavioral fingerprint that can help an active censor find CovertCast users.

Therefore, instead of accessing the feed directly, CovertCast uses screen-scraping to read the video frames. We thus avoid making CovertCast dependent on the specific implementation of live streaming and reduce the risk of CovertCast users being banned from the service.

5.2 Recovering from errors

Normal network perturbations, such as dropped or delayed packets, can create visual artifacts that corrupt the video. These visual artifacts can cause a CovertCast client to misread data and to fail to correctly demodulate Web content. Network errors between the server, the streaming service, and the client can also cause images to not be displayed to the client at all. In this case, the client may not know where the next piece of Web content begins and become desynchronized. In this case, all further images in the stream become unusable.

Distorted or dropped image errors rarely happened in our experiments. To help protect against visual artifacts, CovertCast can use error-correcting codes (ECCs) in its video encoding, at the cost of reduced bandwidth available for the actual tunnelled Web content. Because most of our tests used a high-bandwidth, low-latency network, we opted not to use ECCs in our experiments, but we also tested CovertCast in more challenging network environments.

To help a client recover from missed images, the server can indicate when the start of an image aligns with the start of a new piece of content. The server indicates this alignment with a flag in the image's metadata. Because the server is constantly sending data, the chance that a piece of content aligns with the start of an image is small. To increase the number of chances for a client to resynchronize, every time the CovertCast server sends a new HTML page, it uses a new image and sets the flag in the image's metadata. The frequency can be increased even further, for example, to every n images, but this also increases the number of images sent by the server.

5.3 Removing unnecessary content

Although CovertCast has higher bandwidth than other hide-within systems, it still takes a non-negligible time to broadcast larger websites. To help mitigate this latency, CovertCast uses several techniques to reduce the amount of Web content sent, as well the size of the content.

First, the server proxies all requests through an ad blocker. Because many ads are big, this can significantly reduce the amount of data that needs to be sent, freeing up the video-encoding capacity for the actual content that users want to see.

Second, CovertCast uses a mobile user agent when fetching content so as to receive smaller pages from websites. Combined with the ad blocker, these heuristics greatly reduce the number and size of the responses. In our testing, using an ad blocker and requesting the mobile version of the front page of *The New York Times* prevented sending around 150 pieces of unneeded content and reduced the amount of sent data from 4 MB to 1.6 MB. Most of the other news sites in our evaluation saw similar reductions.

Third, CovertCast compresses all responses before modulating them into images.

5.4 Interactive and dynamic Web content

Due to the one-way nature of CovertCast, websites that make heavy use of Ajax or user input cannot be broadcast. A previous version of CovertCast sought to address this by allowing clients to send requests as comments and receive HTML responses interactively via live streams, but this introduced a large amount of latency and did not scale with the number of clients. Similarly, websites that wait until after the DOM is fully loaded to fetch images will fail to to display these images when broadcast via CovertCast.

6 Performance

6.1 Implementation

We implemented a fully working prototype of CovertCast. The server is written in Python 3.5. We used Ghost.py[8] for fetching Web content, Privoxy[9] as our ad blocker, and ffmpeg[10] to create the video and send it to YouTube.

8 http://jeanphix.me/Ghost.py/

9 http://www.privoxy.org/

10 https://www.ffmpeg.org/

Most of the server's code is agnostic to the content being sent over CovertCast, but a small amount of custom code is needed for each site. This code guides the server by keeping a list of news stories to broadcast and updating the list if the main page is updated (e.g., a new story appears on the front page). In the future, this code could be generated automatically from the DOM of a page.

The server first loads and broadcasts the main page of a news site. It also parses the main page's DOM for specific HTML attributes to find the URLs of other pages a user might click. For example, the mobile version of the *New York Times* assigns the `sfgAsset-link` class to all stories listed on the main page. The server creates a list of the found stories and then begins to broadcast them. Our code also has the option for the main page to be re-broadcast once every x stories. None of the site-specific code involves more then 30 lines of Python.

The CovertCast client is also implemented primarily in Python. The client views the streaming channel on YouTube using Firefox (although any modern browser would work as well). Its screen scraper, written in C++, continuously sends images to the client's proxy, which serves them over HTTP.

For all of our experiments, the server ran on a Linux machine with an Intel 1.73 GHz i7 processor and 8GB of RAM. The client ran on a Windows machine with an Intel 1.90 GHz i7 processor and 4GB of RAM. Both the client and the server were on a wireless network. Their Internet connection had latency of around 22ms, with download and upload bandwidths both above 15 Mbps.

6.2 System resources

To measure CovertCast's resource consumption, we ran it under normal network conditions with the server broadcasting the *BBC News* website to one client. The server used 15-20% of its total CPU and 180MB of memory. The client peaked at 40% of its CPU and 480MB of memory. The client's screen scraper peaked at 15% of its CPU and 31MB of memory.

6.3 Normal network conditions

According to OpenITP, one of the primary reasons Chinese users want to bypass the Great Firewall is "to read foreign news, like *The New York Times*" [4]. We used CovertCast to broadcast major news sites to a prototype CovertCast client, measuring the time between the client receiving the first and last images associated with each webpage. We picked five news sites and loaded the main page and three articles from each site. This experiment was repeated five times for each site. The results are shown in Fig. 5.

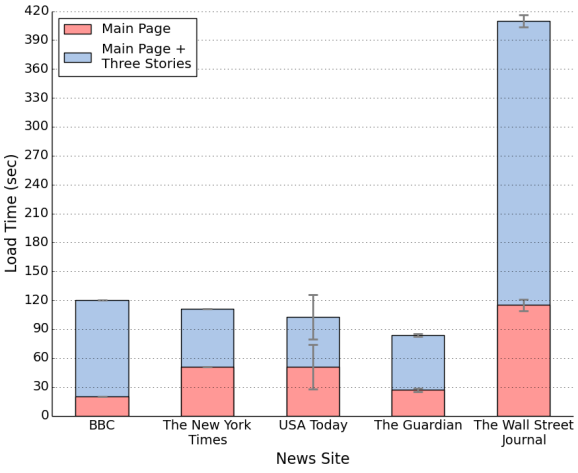

Fig. 5. The average loading time for news sites using CovertCast over YouTube. The time to load the site's main page and the time to load three news articles were measured. Each site was loaded five times. The variance for *USA Today* is due to non-determinism in Ghost.py deciding when a page is fully loaded.

The time it takes to stream a site is directly proportional to the total amount of data sent. The front pages of the majority of news sites we examined were under 1.8MB, with *The Wall Street Journal* being the one exception at 5MB. *The Wall Street Journal*'s much larger size is probably due to it being the only site without a dedicated front page for mobile browsers.

During testing, *BBC News* and *The Guardian* failed to correctly display images, though for different reasons. *BBC News* first loads low-resolution images and later attempts to load higher-resolution images. Therefore, the page would load correctly and display the low-resolution images, but then, after requesting and failing to receive higher-resolution images, it would display blank images. *The Guardian*'s images are hosted on a separate CDN which uses TLS. CovertCast's proxy is unable to respond to requests for these images without using a custom root certificate (since the CovertCast server effectively acts as a man-in-the-middle for all Web content served over CovertCast). Asking CovertCast users to trust such certificates would open them to possible man-in-the-middle attacks, which we wished to avoid.

6.4 Poor network conditions

Not all users inside censored networks have high-quality Internet connections. Some may be located in areas with poor connectivity, while others might have censors actively disrupting their connections to live-streaming sites. To test these conditions, we loaded news sites using CovertCast while simulating low bandwidth and high packet-drop rates.

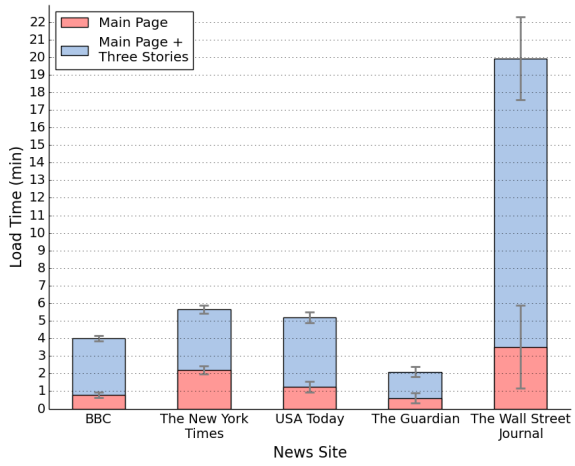

Fig. 6. Time to load the front pages of five news sites with a 800 Kbps connection.

We ran CovertCast using NetBalancer[11] to restrict the client's connection to the live-streaming service. When simulating low bandwidth, we restricted CovertCast to only 800 Kbps download bandwidth. The average Internet connection in China has bandwidth of 3.4 Mbps,[12] thus 800 Kbps represents relatively poor bandwidth for a Chinese user. 800 Kbps is also less than half of YouTube's minimum 720p bitrate (1.5 Mbps).[13] In our tests we found that latency up to 300ms had no effect on CovertCast quality and thus report only on bandwidth constraints. When simulating an active adversary, we set NetBalancer to drop 10% of all packets.

Both experiments used the same five news sites from Section 6.3. Each was loaded five times. The results for the low-bandwidth simulation are shown in Fig. 6 and the results for the high-drop-rate scenario are in Fig. 7. Due to changing news articles, different measurements involved loading different stories. We believe that the loading time for the sites' front pages yields the best comparison metric.

The low-bandwidth conditions caused YouTube videos to frequently pause and buffer, increasing load times by a factor of 2-3. After buffering, the video would often "speed up" and show many images in quick succession before returning to the correct speed. Unfortunately, this speed-up can result in images either being displayed too fast for CovertCast to scrape, or not being displayed at all. This experiment involved over 4000 images, but 20 of them were dropped or missed by CovertCast. The displayed images were mostly correct, with some errors that we believe were caused by the spinning icon that YouTube displays while buffering.

With 10% packet loss, CovertCast performed better than in the low-bandwidth conditions, but not as well as under the normal network conditions. Buffering occurred often and 35 images were missed by CovertCast due to YouTube temporarily increasing the video speed. We also experimented with a 20% drop rate, but found that under those conditions a 720p video could not be loaded.

We explain how CovertCast can recover from missing packets and image errors in Section 5.2.

7 Censorship Resistance

7.1 Disrupting bootstrapping

As described in Section 4.4, a CovertCast client has to obtain some information from the server (in particular, the URL of the covert streaming channel) in order to use CovertCast. The information itself need not be secret since the knowledge of the channel's URL does not reveal the IP addresses of its viewers, but the bootstrapping process should be unobservable to the censors, otherwise they may disrupt it or punish identified CovertCast users. Since bootstrapping need not be interactive, there are many secure channels available to users, e.g., popular email services that encrypt messages.

7.2 Blocking and disrupting live streams

An effective way to disable CovertCast is to block all live-streaming services in the censored network. Given the massive popularity of live streaming, this may cause significant discontent among regular, non-circumvention users. We believe that rational censors, as discussed in our threat model (Section 3), will refrain from blocking all live streaming. Furthermore, CovertCast is not unique in being vulnerable to this threat. Relying on a single type of service or application is common to censorship circumvention systems, including FreeWave [13] and Facet [18].

Censors may also tamper with encrypted streams, e.g., by dropping or delaying packets. Since video streams use TCP and are latency-sensitive, this will disrupt not only CovertCast, but also non-circumvention users of video streaming.

Collateral damage to non-circumvention users of live-streaming services raises potential ethical issues. We believe that these issues are inherent in the design of any censorship circumvention system, including Tor pluggable transports such as meek. Any such system faces the choice between (1)

11 http://seriousbit.com/netbalancer/

12 https://www.stateoftheinternet.com/downloads/pdfs/2015-q2-state-of-the-internet-report-infographic-asia.pdf

13 https://support.google.com/youtube/answer/2853702

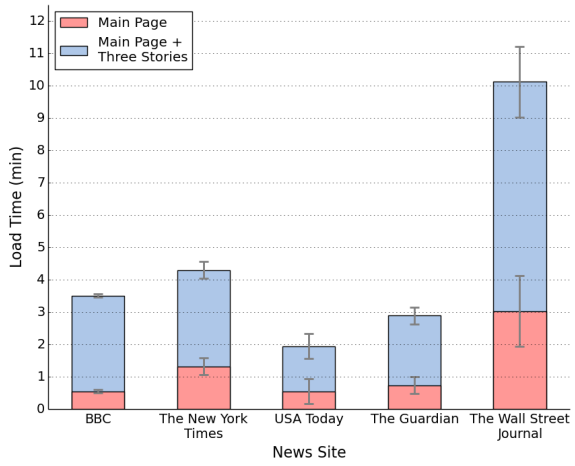

Fig. 7. Time to load the front pages of five news sites with 10% packet drop rate.

hiding within some popular Internet service and thus exposing its non-circumvention users to collateral damage, and (2) being distinct from any other service and thus easy to censor. Since (2) defeats the entire purpose of censorship circumvention, we believe that (1) is the more ethical choice. Furthermore, the operators of live-streaming services can easily opt out of CovertCast participation by disabling the accounts of CovertCast broadcasters. While distinguishing CovertCast streams from regular streams is hard for censors (Section 7.5), operators of streaming services can easily do this if they believe CovertCast is harming their users.

7.3 IP address blocking

Unlike traditional circumvention systems such as Tor [6] and its pluggable transports [23, 24, 33], CovertCast clients do not establish direct network connections with CovertCast servers. Instead, CovertCast connections are tunnelled via the servers of a live-streaming service such as YouTube. Therefore, even if the censors discover the IP addresses of CovertCast servers, blacklisting these IP addresses will not help them identify and disrupt CovertCast connections.

7.4 Content analysis

If the streaming service used by CovertCast encrypts video streams end-to-end (like YouTube does), censors will not be able to detect CovertCast by analyzing video contents. The use of encryption, however, degrades quality of service. For example, YouTube has a bigger start delay than the streaming services that do not use encryption. If the service does not encrypt its streams, censors may be able to detect CovertCast traffic through content analysis techniques of varying complexity and accuracy. For example, censors can render the suspected video stream, using standard video processing, to check if it contains characteristic CovertCast images, as shown in Figure 3. Performing such analysis at line speeds over a large numbers of video streams is likely to impose significant computation and storage overheads on the censors' network monitors.

To increase the cost of content analysis in the absence of encryption, CovertCast can use steganography to hide circumvention traffic in the frames of actual videos. This will reduce the bandwidth available to the tunnelled circumvention traffic.

7.5 Passive attacks

Passive attacks aim to identify circumvention traffic through monitoring of network activity, without modifying or generating traffic.

Protocol analysis. Previous work [9, 11, 30] showed that censors can identify circumvention traffic by inspecting the underlying network protocols. For example, Houmansadr et al. [11] showed that censors can identify SkypeMorph [23], a system that tries to mimic Skype, since SkypeMorph traffic does not contain some essential Skype messages; Wang et al. [30] used a large set of network traces to identify Tor obfuscation techniques such as obfsproxy and meek.

CovertCast is not detectable by such attacks because it embeds circumvention traffic into *genuine* video streams, thus CovertCast traffic uses exactly the same protocols and algorithms as any other live-streaming traffic. In particular, a CovertCast connection over YouTube uses the same video codecs, same streaming protocol, same endpoints, etc., as any other YouTube live stream.

Traffic analysis. Even if the live stream is encrypted, traffic patterns such as packet sizes and inter-packet intervals may allow a censor to distinguish between CovertCast traffic and regular streaming traffic.

There has been little work on traffic analysis and content fingerprinting for streaming video. Our evaluation showed that basic categorical tests such as decision trees cannot distinguish CovertCast traffic from regular video streams. To design a more sophisticated classifier, we adopted ideas from the statistical classifier proposed by Wright et al. [36] to detect spoken language in VoIP calls.

The core of our classifier is the Kullback-Leibler (KL) divergence metric [17], which is a measure of relative entropy between two discrete probability distributions. The KL distance between distributions P and Q is given by $D_{KL}(P||Q) = \sum_i P(i) \ln \left(\frac{P(i)}{Q(i)} \right)$. Our classifier uses two sets of probability distributions, A and B. Each of these sets

Video Set	Total Accuracy	True Positives	True Negatives
YouTube Set 1	65.6%	33.3%	97.8%
YouTube Set 2	65.0%	44.4%	85.6%
Webcam Set 1	65.0%	77.8%	97.8%
Webcam Set 2	71.1%	65.6%	85.6%

(a) Packet sizes

Video Set	Total Accuracy	True Positives	True Negatives
YouTube Set 1	68.3%	41.1%	95.6%
YouTube Set 2	57.2%	36.7%	77.8%
Webcam Set 1	46.7%	61.1%	95.6%
Webcam Set 2	60.0%	60.0%	77.8%

(b) Inter-packet timings

Table 1. Classifying CovertCast streams vs. non-CovertCast streams using packet sizes (**a**) and inter-packet timings (**b**). Accuracy for CovertCast streams is highlighted.

contain N discrete probability distributions of features from 1-minute long video streams, quantized into 100 levels. For every distribution in A, our classifier computes the D_{KL} distance from all other distributions in A, as well as the D_{KL} distance from all distributions in B and vice versa. The classifier returns a *success* metric, which is the number of times the KL distance between two members of one set is smaller than the KL distance from a member of the other set, divided by the total number of KL distances.

We used our classifier to distinguish between CovertCast traffic and regular video streams, based on both packet sizes and inter-packet timings of QUIC[14] packets. One set contained probability distributions of CovertCast connections tunnelled via YouTube, the other set contained actual YouTube streams, namely, two subsets of ten 720p live streams featured on the front page of YouTube and two subsets of ten 720p live footage from a webcam in our lab.

Table 1 shows the success metric of our classifier for the YouTube videos, webcams, and CovertCast streams. While the overall accuracy of our classifier for YouTube videos is around 65%, the true positive rate, that is, the accuracy of correctly identifying a CovertCast stream, is between 33% and 45%, less than random guessing. Our classifier has a higher true positive rate when classifying CovertCast videos vs. webcam videos. This might indicate that it may be possible to classify encrypted YouTube videos by their content type (i.e., webcam footage, video games, sports, news, etc.), but a much larger study is needed before we can establish this for certain. If the content of encrypted YouTube videos can be classified in such a way, this raises privacy concerns for YouTube users that are unrelated to censorship circumvention.

Our tests examined only the size and inter-packet timings of QUIC packets. It may be possible to build a more accurate classifier using other features. Audio and user comments are two components of many live video streams that are absent in CovertCast. A deeper examination of QUIC and YouTube's architecture may inform the design of a classifier capable of recognizing silent, comment-less streams.

CovertCast does not currently include audio or comments, but it is easy to add them in order to mimic interactive live streams. For example, it would be trivial to add audio to ffmpeg when the server sends a video to the streaming service. This audio could be pre-recorded, generated on the fly, or taken from other streams. We leave the in-depth analysis and implementation of these features to future work.

In any case, accurate classification of encrypted video streams requires much more than deep packet inspection. In particular, it involves large-scale collection and analysis of traffic, thus significantly raising the technical bar for censors. To the best of our knowledge, no real-world censor performs this kind of analysis today, and existing censorship technologies are do not support this functionality.

7.6 Active attacks

Today's censors increasingly use active attacks. They can selectively perturb some network connections, e.g., by dropping and delaying packets, change routing policies, etc. Active attacks serve two purposes. First, these attacks can identify circumvention connections (and users) that are otherwise undetectable. For instance, Houmansadr et al. [11] show that cen-

14 Google's *Quick UDP Internet Connections*

sors can identify SkypeMorph [23] connections by selectively dropping a small number of UDP packets from a suspected Skype conversation. As with passive attacks discussed above, CovertCast is resistant to this kind of attack because a CovertCast connection uses exactly the same protocols as a regular video stream, thus its reaction to traffic manipulation will be the same as that of regular video streams.

The second use of active attacks is to disable potential circumvention connections even if they are not detectable. For instance, Geddes et al. [9] show that strategically dropping UDP packets from traffic that looks like Skype will not impact genuine Skype conversations, but it will disable FreeWave [13], a circumvention system that encodes traffic into Skype conversations, by dropping parts of FreeWave content that are essential for decoding the circumvention traffic. This attack does not work against CovertCast. Because CovertCast does not interact with the streaming service outside of scraping the video, any active attack that disrupts CovertCast would disrupt the streaming service as well.

8 Conclusion

CovertCast is a new censorship circumvention system that encodes censored Web content into live video streams and broadcasts them over popular streaming services such as YouTube. CovertCast is highly scalable, with the server workload independent of the number of clients receiving content. CovertCast resists common censorship techniques such as IP address filtering, protocol fingerprinting, and deep packet inspection.

Acknowledgments. We thank the anonymous reviewers and our shepherd, Matt Wright, for feedback and suggestions. This work was partially supported by NSF grants 1223396 and 1409438/1612872 and NIH grant R01 LM011028-01.

References

[1] Anonymizer. https://www.anonymizer.com/.

[2] BRUBAKER, C., HOUMANSADR, A., AND SHMATIKOV, V. CloudTransport: Using Cloud Storage for Censorship-Resistant Networking. In *PETS* (2014).

[3] BURNETT, S., FEAMSTER, N., AND VEMPALA, S. Chipping Away at Censorship Firewalls with User-Generated Content. In *USENIX Security* (2010).

[4] Collateral Freedom: A Snapshot of Chinese Internet Users Circumventing Censorship. https://openitp.org/pdfs/CollateralFreedom.pdf, 2013.

[5] DINGLEDINE, R., AND MATHEWSON, N. Design of a Blocking-Resistant Anonymity System. https://svn.torproject.org/svn/projects/design-paper/blocking.html.

[6] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The Second-generation Onion Router. In *USENIX Security* (2004).

[7] FEAMSTER, N., BALAZINSKA, M., HARFST, G., BALAKRISHNAN, H., AND KARGER, D. Infranet: Circumventing Web Censorship and Surveillance. In *USENIX Security* (2002).

[8] FIFIELD, D., LAN, C., HYNES, R., WEGMANN, P., AND PAXSON, V. Blocking-resistant Communication through Domain Fronting. In *PETS* (2015).

[9] GEDDES, J., SCHUCHARD, M., AND HOPPER, N. Cover Your ACKs: Pitfalls of Covert Channel Censorship Circumvention. In *CCS* (2013).

[10] HOLOWCZAK, J., AND HOUMANSADR, A. CacheBrowser: Bypassing Chinese Censorship without Proxies Using Cached Content. In *CCS* (2015).

[11] HOUMANSADR, A., BRUBAKER, C., AND SHMATIKOV, V. The Parrot Is Dead: Observing Unobservable Network Communications. In *S&P* (2013).

[12] HOUMANSADR, A., NGUYEN, G., CAESAR, M., AND BORISOV, N. Cirripede: Circumvention Infrastructure Using Router Redirection with Plausible Deniability. In *CCS* (2011).

[13] HOUMANSADR, A., RIEDL, T., BORISOV, N., AND SINGER, A. I Want My Voice to Be Heard: IP over Voice-over-IP for Unobservable Censorship Circumvention. In *NDSS* (2013).

[14] HOUMANSADR, A., WONG, E. L., AND SHMATIKOV, V. No Direction Home: The True Cost of Routing Around Decoys. In *NDSS* (2014).

[15] HOUMANSADR, A., ZHOU, W., CAESAR, M., AND BORISOV, N. SWEET: Serving the Web by Exploiting Email Tunnels. In *PETS* (2013).

[16] KARLIN, J., ELLARD, D., JACKSON, A., JONES, C., LAUER, G., MANKINS, D., AND STRAYER, W. Decoy Routing: Toward Unblockable Internet Communication. In *FOCI* (2011).

[17] KULLBACK, S., AND LEIBLER, R. A. On information and sufficiency. *The Annals of Mathematical Statistics* (1951), 79–86.

[18] LI, S., SCHLIEP, M., AND HOPPER, N. Facet: Streaming over Videoconferencing for Censorship Circumvention. In *WPES* (2014).

[19] MAHDIAN, M. Fighting Censorship with Algorithms. In *Fun with Algorithms* (2010).

[20] MCCOY, D., MORALES, J. A., AND LEVCHENKO, K. Proximax: A Measurement Based System for Proxies Dissemination. In *FC* (2011).

[21] MCLACHLAN, J., AND HOPPER, N. On the Risks of Serving Whenever You Surf: Vulnerabilities in Tor's Blocking Resistance Design. In *WPES* (2009).

[22] meek Pluggable Transport. https://trac.torproject.org/projects/tor/wiki/doc/meek.

[23] MOGHADDAM, H., LI, B., DERAKHSHANI, M., AND GOLDBERG, I. SkypeMorph: Protocol Obfuscation for Tor Bridges. In *CCS* (2012).

[24] Tor: Pluggable Transports. https://www.torproject.org/docs/pluggable-transports.html.en.

[25] Psiphon. http://psiphon.ca/.

[26] SCHUCHARD, M., GEDDES, J., THOMPSON, C., AND HOPPER, N. Routing Around Decoys. In *CCS* (2012).

[27] SOVRAN, Y., LIBONATI, A., AND LI, J. Pass It On: Social Networks Stymie Censors. In *IPTPS* (2008).

[28] Ten Ways to Discover Tor Bridges. https://blog.torproject.org/

blog/research-problems-ten-ways-discover-tor-bridges.

[29] Ultrasurf. http://www.ultrareach.com.

[30] WANG, L., DYER, K. P., AKELLA, A., RISTENPART, T., AND SHRIMPTON, T. Seeing Through Network-Protocol Obfuscation. In *CCS* (2015).

[31] WANG, Q., GONG, X., NGUYEN, G., HOUMANSADR, A., AND BORISOV, N. CensorSpoofer: Asymmetric Communication Using IP Spoofing for Censorship-Resistant Web Browsing. In *CCS* (2012).

[32] WANG, Q., LIN, Z., BORISOV, N., AND HOPPER, N. rBridge: User Reputation Based Tor Bridge Distribution with Privacy Preservation. In *NDSS* (2013).

[33] WEINBERG, Z., WANG, J., YEGNESWARAN, V., BRIESEMEISTER, L., CHEUNG, S., WANG, F., AND BONEH, D. StegoTorus: A Camouflage Proxy for the Tor Anonymity System. In *CCS* (2012).

[34] WILDE, T. Knock Knock Knockin' on Bridges' Doors. https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors, 2012.

[35] WINTER, P., AND LINDSKOG, S. How the Great Firewall of China Is Blocking Tor. In *FOCI* (2012).

[36] WRIGHT, C., BALLARD, L., MONROSE, F., AND MASSON, G. Language Identification of Encrypted VoIP Traffic: Alejandra y Roberto or Alice and Bob? In *USENIX Security* (2007).

[37] WUSTROW, E., WOLCHOK, S., GOLDBERG, I., AND HALDERMAN, J. Telex: Anticensorship in the Network Infrastructure. In *USENIX Security* (2011).