Diogo Barradas*, Nuno Santos, and Luís Rodrigues

# DeltaShaper: Enabling Unobservable Censorship-resistant TCP Tunneling over Videoconferencing Streams

**Abstract:** This paper studies the possibility of using the encrypted video channel of widely used videoconferencing applications, such as Skype, as a carrier for unobservable covert TCP/IP communications. We propose and evaluate different alternatives to encode information in the video stream in order to increase available throughput while preserving the packet-level characteristics of the video stream. We have built a censorship-resistant system, named DeltaShaper, which offers a data-link interface and supports TCP/IP applications that tolerate low throughput / high latency links. Our results show that it is possible to run standard protocols such as FTP, SMTP, or HTTP over Skype video streams.

**Keywords:** Censorship Circumvention, Traffic Analysis, Traffic Encapsulation, Video Stream Synthesis

DOI 10.1515/popets-2017-0024
Received 2017-02-28; revised 2017-06-01; accepted 2017-06-02.

## 1 Introduction

Over the last years, a number of authors have developed censorship-resistant systems which aim to provide access to blocked information over the Internet [17, 21, 25]. A common approach to achieve this goal is to stealthily *tunnel* covert data over protocols that are unlikely to be blocked by a censor, such as Skype. Skype traffic is encrypted, which prevents a state-level censor that controls the network from inspecting the content of packets and look for the presence of covert data.

Our work builds on previous related work, in particular on FreeWave [17], while aiming at addressing its main limitations. FreeWave allows users to encode covert data into the acoustic signals of VoIP connections over Skype. A significant strength of FreeWave is that the transmission of covert data can be performed as full-duplex TCP/IP streams. Unfortunately, FreeWave was found to be vulnerable to traffic analysis techniques [12]. In particular, FreeWave's modified Skype streams exhibit certain patterns that can be detected by a censor that can tap on the network (e.g., changes in packet size distributions). By comparing such patterns against those of regular Skype calls, a censor can identify suspicious Skype streams with high probability and proceed with dropping down such connections, identifying and prosecuting endpoints based on their IPs, or carry out other censorship measures. FreeWave was designed without defense mechanisms that can thwart such attacks. Given that most electronic communication over the Internet can be controlled today by governments and/or by a few corporations [3, 9, 18], the lack of such mechanisms renders FreeWave connections *observable*.

To mitigate traffic analysis attacks, more recent censorship-resistant systems attempt to ensure that the resulting traffic remains *unobservable*. In other words, the traffic embedding covert data must not display individuating patterns that enable a censor to distinguish it from regular traffic using state-of-the-art traffic analysis techniques. An effective technique that has proven effective at achieving unobservability is to modulate covert data over carrier *video streams*. Notably, Facet [22] enables clients to secretly watch censored videos over a Variable Bit Rate (VBR) video stream. At the server side, Facet embeds covert video frames into a small region of the video frame area of a videoconferencing Skype call, effectively allowing a client to watch blocked videos. CovertCast [24] is another relevant system that uses video streaming as message carrier. CovertCast enables the content of blocked websites to be transmitted via modulated images of live-streaming feeds uploaded to YouTube. The main limitation of both Facet and CovertCast, however, is that they do not support the transmission of covert TCP/IP streams. In these systems, the format of covert messages are restricted to certain data types, namely videos (in Facet) or web content (in CovertCast). Operations as simple as sending an email are not possible under these systems. In particular, CovertCast does not support bidirectional communication, which means that neither interactive web browsing nor a simple chat message exchange are supported under this system. Thus, although

***Corresponding Author: Diogo Barradas:** INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, E-mail: diogo.barradas@tecnico.ulisboa.pt
**Nuno Santos:** INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, E-mail: nuno.santos@inesc-id.pt
**Luís Rodrigues:** INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, E-mail: ler@tecnico.ulisboa.pt

these systems can effectively achieve *unobservability*, the lack of support for covert TCP/IP streams can considerably impair their *coverage*, i.e. the content, services and protocols they can access in the Internet.

Our goal is to devise a system that can provide both unobservability and a higher degree of coverage than that offered by previous video streaming protocol tunneling systems. With that aim, this paper presents the design, implementation, and evaluation of DeltaShaper, a new censorship resistance system that enables unobservable TCP/IP tunneling over videoconferencing Skype streams. In DeltaShaper, the covert TCP/IP packets are encoded and embedded into the video stream transmitted by Skype between the communication endpoints. Similarly to FreeWave, DeltaShaper is meant for scenarios where a user linked to a censoring ISP wishes to establish a covert TCP/IP connection with a remote computer located outside the censoring region. After installing both Skype and DeltaShaper clients, the user can set up a Skype videoconferencing call to a proxy computer and open a DeltaShaper TCP tunnel over that call. The proxy must be configured to run Skype and DeltaShaper clients, be located outside the censoring region, and be maintained by some trusted third party (e.g., a family member or friend). Once the tunnel has been established, the user can run unmodified networked applications such as a mail client, a web browser, or a file transfer agent to access the services of a remote computer over the tunnel. DeltaShaper ensures that the covert TCP/IP packets are encoded in such a way that the resulting Skype streams remain unobservable.

The key challenge overcome by DeltaShaper is: *how to encode covert TPC/IP packets into Skype video frames such that unobservability is preserved while delivering enough channel bandwidth for TCP to work*. If we adopt an aggressive scheme and encode as much payload data per frame as we can (e.g., encode bits of TCP/IP packets in each pixel of the carrier fames), the bandwidth available on the covert channel is high, but the resulting packet stream will differ substantially from regular Skype calls. Moreover, we must be careful not only about the number of pixels that can be changed, but also about the pixel area dedicated to the encoding of payload, the modified pixels color, and the rate at which new payload can be encoded into the video frames. However, these restrictions will cause the amount of covert channel bandwidth to drop. Available bandwidth will be further dwarfed by Skype's lossy compression algorithm which introduces errors in payload encoding. As a result, we must increase the amount of pixels which encode payload units and introduce redundancy. It is also necessary to take into account additional TCP/IP meta-data that needs to be encoded into the frames in order to synchronize both tunnel endpoints. All these issues concur to reducing the channel bandwidth to a point where transmissions would take an unacceptable amount of time.

In DeltaShaper, we address this challenge by devising an encoding technique that is robust to noise and that can be dynamically tuned to adapt to the underlying network conditions (e.g., bandwidth and packet loss). We propose and evaluate different alternatives to encode information in Skype's video-streams, in order to maximize the available throughput while preserving the characteristics of unmodified streams. We have implemented a prototype of our system that offers a data-link interface and that can support applications which run over TCP/IP. Our results show that it is possible to achieve a throughput of 2.56 Kbps with no significant impact on the stream, which allows to run TCP/IP applications that can tolerate low throughput / high latency links, such as FTP, SMTP, Web clients or chat clients.

## 2 Related Work

Numerous practical solutions have been proposed over the last years to address the problem of Internet censorship [8, 20].

**Decoy routing:** Decoy routing systems [16, 19, 35] are based upon special routers deployed within ISPs outside the censor's sphere of influence. In this scheme, clients issue HTTPS steganographically marked requests to an overt destination whose path crosses a decoy router. Decoy routers recognize such a mark (which reveals a client's true desired destination) and divert traffic to blocked destinations. Recently, Bocovich and Goldberg [1] have proposed a decoy routing system which imitates the traffic patterns of the chosen overt website. Unfortunately, ISPs still have little incentive to deploy censorship circumvention mechanisms like the one presented.

**Protocol randomization:** A different class of systems helping in circumvention aims to obfuscate covert traffic so that it cannot be linked to the underlying application layer protocol [34]. An instance of this approach consists in *protocol randomization*, where traffic is manipulated to make it seem random and fool a censor's protocol blacklist. Obfsproxy [4] and ScrambleSuit [33] respectively encrypt and randomize Tor's [5] network traffic features. Yet, entropy tests have been successfully used for distinguishing regular TLS traffic from Obfsproxy encrypted traffic [29]. Moreover, by transforming network traffic into some unknown protocol, this approach fails to evade a censor that performs protocol whitelisting.

**Protocol imitation:** Another obfuscation strategy is *protocol imitation*, which aims at mimicking popular protocols allowed across a censor's network. For instance, StegoTorus [31] steganographically conceals chops of Tor traffic on the messages of a cover protocol, while SkypeMorph [25] and CensorSpoofer [30] mimic the statistical properties of video and

VoIP calls, respectively. In its turn, Format-Transforming Encryption (FTE) [6] can be used to foil regex-based DPI systems by producing ciphertexts that match the content definition of some allowed protocol. Unfortunately, due to the difficulties of mimicking all aspects of a protocol, the former systems are vulnerable to several attacks [15] while FTE can be detected through the use of entropy tests [29]. Marionette [7] employs automata composition to control fine-grained aspects of mimicry. Still, candidates for imitation may be proprietary software, demanding its reverse engineering in order to build a model for imitation. Not only is this a tedious effort, but it must be repeated for each software release.

**Protocol tunneling:** To avoid the pitfalls of protocol mimicry, another category of censorship circumvention systems directly *tunnels* covert data through a protocol's application layer. In SWEET [36], covert traffic is relayed through encrypted or steganography-protected email messages that are temporarily staged on standard mail servers. CloudTransport [2] adopts a similar principle, but uses public cloud storage services for covert message forwarding. SWEET is limited by the unusual email utilization patterns it generates while CloudTransport is prone to denial of service and traffic analysis. *meek* [11] leverages domain fronting to tunnel traffic over HTTPS connections to allowed hosts, while establishing a covert connection to a prohibited host. However, it does not attempt to match the traffic patterns which would emerge when visiting an overt destination. As a result, machine learning techniques applied on traffic classification are able to identify traffic generated by *meek*'s implementation over Tor [29]. Castle [14] and Rook [28] provide an alternative approach to exchange covert messages over Real-Time Strategy (RTS) games. Most of these systems, however, are either vulnerable to traffic analysis attacks or provide insufficient covert channel bandwidth.

**Tunneling over audio streaming:** A recent tunneling approach leverages existing audio streaming protocols to enable communication between two parties engaged in circumvention. FreeWave [17] leverages VoIP connections to tunnel censored Internet traffic. However, it is vulnerable to passive attacks since the packet length distribution of the traffic containing covert messages is nothing similar to that of a recognizable language. Furthermore, a censor can disable FreeWave by launching active attacks which prevent endpoint synchronization [12]. SkypeLine [21] works under a stronger threat model than FreeWave by assuming that the censor can eavesdrop on the contents of any ongoing VoIP call. SkypeLine modulates the background noise found in VoIP connections in order to transmit covert data, being robust both to steganalysis and attacks that make use of traffic analysis techniques. However, SkypeLine achieves a very low covert channel throughput.

**Tunneling over video streaming:** Facet [22] enables tunneling censored videos through videocalls of applications such as Skype. This system ensures unobservable covert data transmissions, by embedding the censored video in a portion of each frame, filling the remaining space with a legitimate conversation video. This approach provides active attack resistance by design, since any perturbation in the network will cause exactly the same effect on a regular or covert video transmission. When compared with FreeWave, Facet [22] provides greater resilience to both active and passive attacks. Unfortunately, Facet is only able to serve video content which limits the applicability of the system to other types of communication. CovertCast [24] leverages live-streaming feeds to transmit the content of blocked websites to multiple clients at once. CovertCast servers modulate censored data into images which are aggregated and transmitted through live video feeds, resorting to platforms such as YouTube. Clients scrape and demodulate the images served through the live stream, extracting the blocked web content. Although CovertCast data modulation is resilient to traffic analysis, the system provides only one-to-many communication channels.

## 3 Goals and Threat Model

Our main goal is to embed a TCP/IP covert data channel in a regular Skype stream in a way that it cannot be tagged as disallowed by an adversary. Instead of attempting to mimic the Skype protocol, an exercise that may be vulnerable to active attacks from the adversary, we aim at using Skype "as is" and encode the information in the video stream, using Skype as a blackbox. In particular, we are driven by four design goals:

1. *Unblockability:* The censor must not be able to block the transmission of covert messages without significant degradation of the Skype service for legitimate users.
2. *Unobservability:* We want to ensure that a censor will not be able to distinguish regular Skype streams from those that carry a covert channel.
3. *Reasonable throughput:* The performance of the covert channel must allow for the execution of standard TCP/IP applications which are able to tolerate low throughput / high latency links.
4. *Skype as a black-box:* Our system must not require changes to the application software running at the channel endpoints. Moreover, the videoconferencing software should be used "as is", without modifications to its binary.

**Threat model:** We assume the existence of applications that use encrypted video-streams, such as Skype, that the adversary is not willing to block. Therefore, the adversary will only

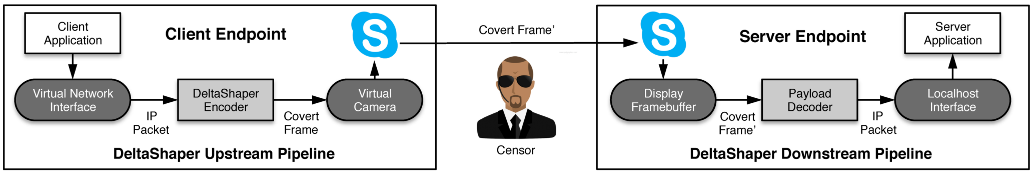

**Fig. 1.** DeltaShaper architecture.

block or disrupt those streams if it can observe that the stream is being used to convey some covert channel.

In order to detect covert channels, we assume that the censor can resort to the tools that are typically available to a state-level omniscient adversary, i.e., the censor is able to observe, store, interfere with, and analyze all the network flows between the parties that are engaged in the communication. However, we assume that the adversary is unable to control the software installed on end-users' computers. Thus, the communication endpoints where clients run are deemed trusted.

The adversary has the power to perform deep packet inspection but is computationally bounded, and cannot break the underlying cryptographic primitives used to encrypt the packet content. Also, we assume that the videoconferencing provider (i.e., the Skype service provider) will not collude with the adversary, for example by allowing the adversary to inspect rendered video content at the communication endpoints. Therefore, the adversary cannot detect the covert channel by observing directly the content of the stream. It may however perform statistical analysis on the traffic patterns of each flow (in face of different network conditions) and detect outliers. For that purpose, the adversary will use state-of-the-art techniques to classify the streams and to rank the similarity among different flows. In our evaluation, we will also use these same techniques to assess the unobservability of DeltaShaper.

# 4 Design

Figure 1 illustrates the operation of our censorship-resistant system, named DeltaShaper. On the sending side, the transmitter receives the payload and encodes it in a video stream that is fed to Skype using a virtual camera interface. Skype transmits this video to the remote Skype instance and the received stream is captured from the Skype video buffer. A decoder then extracts the payload from the video stream and delivers it to the application. The same procedure is applied at both endpoints of a Skype call, thus, effectively, we support a bi-directional channel in this way. To make the system as general as possible, the architecture exposes a data-link level protocol to the upper layers, such that an IP packet can be accepted, encoded, decoded, and delivered remotely using this technique. As a result, the system can support any TCP/IP application that can tolerate low throughput / high latency links.

## 4.1 Design Challenges

Although the principles behind the development of DeltaShaper are relatively simple, there are many design challenges that need to be addressed, namely:

**1. Conflicting data encoding requirements:** Intuitively, embedding many covert message bits per frame is desirable in order to achieve high throughput. Unfortunately, this may not be possible. On the one hand, the video stream is re-encoded and potentially compressed by Skype for transmission, using lossy algorithms. One needs to ensure that the payload is encoded with additional redundancy so that it can still be retrieved at the receiver, regardless of the transformations performed by Skype. On the other hand, an aggressive encoding scheme will likely generate network streams that differ significantly from a typical Skype call, making them prone to be detected by a censor. Hence, we require a video encoding scheme that can produce unobservable streams and offer acceptable performance despite the video quality degradation induced by Skype.

**2. Characterization of unobservable streams.** The traffic signature of a Skype call with an encoded covert channel should be indistinguishable from a "normal" Skype call. The challenge, however, is to define what a "normal" Skype call is. We need to establish objective metrics that allow for the identification of such streams through traffic analysis and to generate covert traffic that retains a similar signature.

**3. Adaptation to network conditions.** The properties of a "normal" Skype call may change depending on the network conditions. As a result, DeltaShaper must be able to adapt its data encoding algorithm according to such characteristics, otherwise a censor could leverage them to compromise the unobservability of covert streams.

**4. Synchronization of covert channel endpoints.** A consequence of network adaptation is that channel endpoints must synchronize with each other to enable the receiver to interpret the covert data according to the encoding scheme used by the sender. However, such a mechanism must be resilient to active

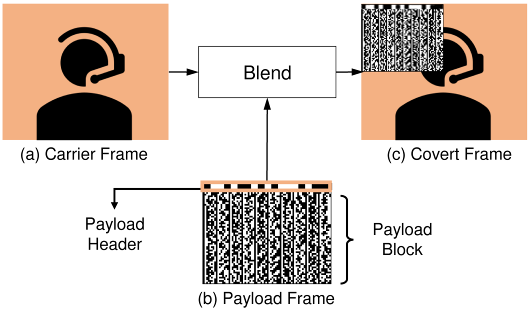

**Fig. 2.** Blending payload into carrier frame.

attacks issued by the censor aimed to cause denial of service. In the next sections, we address these challenges in detail.

## 4.2 Data Encoding and Decoding

In theory, provided that an RGB encoded pixel takes 24 bits (8 bits per channel), one could encode 24 bits of covert data in each pixel. Assuming we are dealing with a 640x480 frame (VGA resolution), each frame could carry, at most, approximately 7 Mbits. Unfortunately, that maximum throughput cannot be achieved in practice for several reasons.

Firstly, video processing may modify the frame pixels in multiple ways: (R1) change the colors of each pixel, thus altering the information being transferred, (R2) omit differences among adjacent pixels, losing all information encoded in those pixels, and (R3) omit differences among adjacent frames, losing information in the portion of the frames that are omitted. Secondly, it is necessary to preserve unobservability (R4). If all pixels of an image are used to encode data to the maximum capacity, the "image" complexity would be significantly different from a typical image transferred in Skype, where many pixels are similar. As we show in Section 6.2, the traffic generated when transmitting such images is different from the traffic resulting from the transmission of images composing a "normal" Skype stream. To deal with these concerns, we propose a data encoding scheme based on two basic ideas:

**1. Blend synthetic payload video into "normal" Skype video:** Our covert data encoding scheme generates transmitted video frames (*covert frames*) from the combination of: *carrier frames* and *payload frames*. Carrier frames can either be obtained from pre-recorded Skype calls or from a live webcam capture. Payload frames consist of synthetic video frames that encode application data to be transmitted. Payload and carrier frames are blended together into covert frames and passed over to Skype. Figure 2 shows an example of how a (a) carrier frame and a (b) payload frame are blended into a (c) covert frame to be dispatched by the sender. The payload frame is overlapped in the top-left corner of the carrier frame. Carrier frames aim to mimic a realistic Skype call by modulating the network stream observed by the censor thus preserving unobservability.

| Name | Description | Example |
|---|---|---|
| $a_p$ | payload frame area (pixel×pixel) | $160 \times 120$ |
| $a_c$ | cell size (pixel×pixel) | $4 \times 4$ |
| $b_c$ | color encoding (bits) | 1 |
| $r_p$ | payload frame rate (frame/sec) | 3 |

**Table 1.** Payload frame encoding parameters.

**2. Support tunable payload frame encoding:** Our payload encoding scheme depends on several parameters. Each payload frame encodes N bits of the covert message on a *payload block*. Each payload block is a synthetic image that consists of a grid of *cells*. Each cell consists of a fix-sized area of contiguous pixels featuring the same color. The color code is used to encode $b_c$ bits of information of the payload block. The total amount of bits that can be encoded per frame N is then defined by the geometry of the payload frame and given by: $N = b_c \times n_c$, where $n_c$ is the number of cells per frame. As a result, the communication throughput T is given by $N \times r_p$, where $r_p$ is the rate of payload frames sent per unit of time. To recover the data, the receiver must collect covert frames at rate $r_p$, extract the payload area $a_p$ from the frame, average out the color of each pixel of each cell, and streamline the $b_c$ bits of each cell. Consequently, to decode a payload block, the receiver must know which encoding parameters were used. For this reason, the sender appends these parameters into a fixed-format band atop the payload frame (payload header).

Our encoding scheme is then defined by the parameters in Table 1: size of payload frame in pixels ($a_p$), size of cells in pixels ($a_c$), color encoding in bits ($b_c$), and payload frame rate ($r_p$). We represent a *data encoder* by tuple $S : \langle a_p, a_c, b_c, r_p \rangle$. For example $\langle 160 \times 120, 4 \times 4, 1, 3 \rangle$ means that a cell takes 4x4 pixels and the payload frame size is 160x120 pixels, totaling $n_c = 1200$ cells. Since the payload data is encoded with 1 bit (yielding a binary black-white image), the payload block is $N = 1200$ bits. For a payload frame rate of 3 frames per second, the maximum throughput $T$ is then 3.6 Kbps.

This encoding scheme enables DeltaShaper to handle conflicting data encoding requirements by providing multiple degrees of freedom. Reducing the number of bits (less than 24 bits) to represent color codes improves resilience to per-pixel color change introduced by Skype (R1). Increasing the cell size above 1x1 helps tolerate loss of information between adjacent pixels as a result of video compression (R2). This is because more pixels will be used to encode a single bit. Moreover, rather than using all frames to encode payload data, our scheme allows for a reduction in the payload frame rate which is important to mitigate the video compression effects that can cause loss of information between consecutive frames (R3). Finally, by properly tuning DeltaShaper encoding parameters one can control the amount of information blended into the carrier video which will determine how close from a "normal" Skype call the resulting covert video will be (R4).

## 4.3 Preserving Unobservability

Unfortunately, due to the complexity of the optimization performed by video encoding/decoding algorithms, there is no trivial manner to estimate the traffic features of an encoded video stream. This makes it very hard to create an analytical model that guides the encoding of the covert channel such that the resulting stream produced by Skype is indistinguishable from a "normal" Skype stream. Ultimately, we want DeltaShaper to generate covert videos that preserve the signature of a "normal" Skype stream while encoding a large amount of payload data. To that end, we need to characterize what typical Skype streams are and then devise a technique to generate covert streams that follow similar traffic patterns.

We designate "normal" Skype streams as *regular streams*. A Skype stream is *regular* if it results from a legitimate videoconferencing call between Skype users carrying no covert messages. In such cases, users usually stand in front of the camera, moving sparingly as they speak. In contrast, we expect that the resulting traffic pattern to be quite different if Skype is used for streaming an action movie, for example. In such cases, frames will change more frequently and extensively causing Skype to send a larger number of long packets to reflect such changes. To express the intuition that regular calls tend to follow a common pattern, while inevitably have some differences, we consider a stream to be *irregular* if it differs by more than a given threshold $\Delta$ from a known regular stream, in which $\Delta$ is obtained by a given *similarity function* $\sigma$. Put more formally, considering $s_R$ to be a known regular stream, $f$ a *feature function* of the stream (e.g., packet length distribution), and $s_C$ an arbitrary stream (that may contain a covert channel), we say $s_C$ is indistinguishable from $s_R$ if:

$$\sigma(f(s_C[P]), f(s_R)) \leq \Delta$$

As it will become clear later in the text, DeltaShaper is able to tune the encoding parameters $P$ for $s_C$ such that the resulting covert stream satisfies the condition above (and, therefore, remains unobservable). Interestingly, the tuning mechanisms are independent of the actual feature and similarity functions used to assess observability. As discussed below, the current prototype uses the packet size as the relevant feature and the Earth Mover's Distance (EMD) [27] as the similarity function. However, DeltaShaper can be easily reconfigured to match other features or similarity metrics that can prove to offer a more accurate identification of the covert channel (in fact, we also use another function in the evaluation section). We discuss the current choices in detail in the next paragraphs:

**1. Find an effective feature function ($f$):** A feature function extracts some relevant quantitative attribute out of the packet traces that constitute a stream. Through experimental evaluation, we found the *frequency distribution of packet lengths* ($f_l$) to be effective at characterizing a given stream pattern in Skype. The frequency distribution of packet lengths of a stream depends on both the input video and compression applied by Skype. Therefore, blending payload frames into the carrier frames will alter the packet length distribution. A similar reasoning was proven to be successful at differentiating Skype streams from Tor streams [25]. An alternative function based on the 2-gram distribution of packet lengths has allowed for the differentiation of YouTube traffic from Skype traffic [22]. We found that in the context of DeltaShaper, this function produces similar results as $f_l$. A discussion on the accuracy of other feature functions is described in Section 6.4. We did not consider alternative features based on DPI, since Skype-generated packets are encrypted.

**2. Find an effective similarity function ($\sigma$):** A similarity function aims to calculate the difference between two feature functions. Since we adopt $f_l$, which outputs the distribution of a stream's packet length, we look for metrics that calculate the similarity between two probability distributions. Previous work has adopted the Kolmogorov-Smirnov test [25, 28]. However, we found Earth Mover's Distance (EMD) [27] to produce better results – a comparison between the accuracy of both similarity functions can be found in Section 6.4. Due to space constraints, we omit the mathematical formulation of EMD. Intuitively, EMD$(f_l(s_R), f_l(s_C))$ represents the amount of work that must be undertaken to transform the packet length frequency distribution of the regular stream $s_R$ to the packet length frequency distribution of stream $s_C$.

**3. Set a reference stream ($s_R$):** Now that we have defined $f$ and $\sigma$, we need to fix a known regular stream to serve as a *reference stream* around which DeltaShaper's generated covert streams will compare against. Such a regular stream will correspond to streaming the carrier video used by DeltaShaper in the payload blending process (as shown in Figure 2-a), and can be obtained by recording the packet trace of a real Skype call, for example. As it will be clear later in the text, we will have different reference streams for different network conditions.

**4. Compute the similarity threshold ($\Delta$):** The similarity threshold $\Delta$ aims to set a limit to the maximum difference that one can expect to find between legitimate regular Skype calls. To determine this value, we take an empirical approach which consists of creating a training set of $N$ legitimate Skype call videos, replaying each video $M$ times and record the packet length distribution of the resulting test stream $s_{ij}$, where $0 \leq i < N$ and $0 \leq j < M$. Then, calculate the distribution similarity between each test stream and the reference stream and obtain the maximum value as $\Delta$. This value is determined by:

$$\Delta = max(\text{EMD}(f_l(s_{ij}), f_l(s_R)))$$

**5. Obtain a valid encoding selector ($P$):** The final step consists of determining valid sets of parameter instances ($P$) to the payload encoding scheme. We call encoding selector to a specific instance of $P$. To be valid, an encoding selector must produce unobservable streams. Encoding selectors that satisfy such condition can be found by exploring the space of $P$ generating a training stream $s_C[P]$ and verify that $s_C$ is indistinguishable from $s_R$. More precisely:

$$\text{EMD}(f_l(s_C[P]), f_l(s)) = \delta, P \text{ is } \textit{valid} \text{ if } \delta < \Delta$$

The set of valid encoding selectors must be obtained experimentally and provided to DeltaShaper as possible encoding selectors to be adopted. If multiple encoding selectors are valid, DeltaShaper selects the one that delivers the highest throughput, which is also determined experimentally. Section 6 presents the results of our empirical analysis. Moreover, the video compression procedures used by different videoconferencing applications are based in similar algorithms. The exploration of the space of encoding selectors provides DeltaShaper a general approach to compute valid encoding selectors without being tied to a specific videoconferencing application's codec implementation.

## 4.4 Adaptation to Network Conditions

As it turns out, the reference stream $s_R$ and respective threshold $\Delta$ cannot be permanently fixed and hard-coded in DeltaShaper. In fact, according to our experiments (as explained in Section 6), the Skype stream distributions that result from playing a given (carrier) video greatly depend on the specific network conditions under which the transmission has occurred, such as bandwidth or packet loss rate. This observation brings two immediate consequences:

*The reference stream and the similarity threshold must be set dynamically:* In order to preserve the properties of unobservability on a given connection, it is necessary to adopt reference stream ($s_R$) and threshold value ($\Delta$) according to the specific network conditions. Furthermore, we must take into account that such network conditions may change over time either due to contingencies of the network infrastructure or to active attacks launched by the censor.

*The encoder selector must be set dynamically:* In response to changes in network conditions, it is necessary to change the frame encoding parameters in order to preserve the stream indistinguishability. Moreover, the negotiation of new parameters between both endpoints must also be resilient to active attacks issued by the censor aimed to prevent the agreement and cause denial of service.

To make our system adapt to the network conditions, the client endpoint performs the following operations: 1) before starting data transmission, determines what is the baseline distribution for the current network conditions, 2) from that baseline distribution, obtains the ideal parameters from a reference table, 3) starts encoding data frames according to the determined parameters and embeds the encoding parameters directly into each frame, 4) periodically, readjusts to network conditions, by repeating this procedure starting from 1. Next, we explain these steps in detail:

**1. Finding the reference stream distribution:** The main cause for the change of the threshold value is a modification of the reference stream distribution upon changes in network conditions. To accommodate to those changes, before the client starts encoding payload data into carrier frames, the client performs a calibration operation in which it transmits the carrier video alone without embedding any payload blocks. The client transmits this video for a certain calibration time $T_C$ and in this process collects relevant features about the stream packets, namely their sizes. These samples will allow the client to obtain a fingerprint of the carrier video stream for those particular network conditions.

**2. Determining the encoding parameters:** Based on the collected samples, the client will determine which reference stream the current stream is closer to. The client is preconfigured with a set of reference streams for different network conditions and checks which of these reference streams are similar to the current one by computing the similarity metric $\sigma$ for each of them. The client takes the reference stream that results in the lowest $\sigma$ value and then obtains the ideal parameters from a reference table. This table is simply a map that tells for each reference stream which parameter values should be used for generating the encoding selector. Both reference streams and reference tables are determined empirically.

**3. Agreeing upon common frame encoding parameters:** Since the frame encoding parameters change dynamically, the client must tell the server which parameters to adopt when decoding the payload frames. To this end, each covert frame has a small meta-data block, with the parameter values used in the associated payload block. The meta-data block uses a conservative fixed encoding scheme so as to be resilient to errors and prevent observability anomalies. This makes each frame self-contained and the parameter agreement resilient to DoS

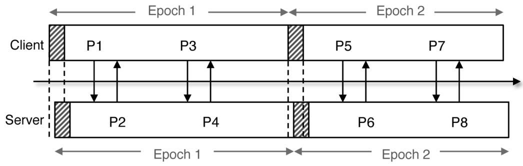

**Fig. 3.** Epochs in a bidirectional covert channel. IP packets are exchanged during the data transmission phases.

attacks specifically aimed at disrupting parameter exchange. We defer a discussion on the resilience of the covert channel against active attacks launched by a censor (e.g. dropping packets of Skype connections) to Section 6.7.

**4. Readjusting the frame encoding parameters:** Since the network conditions may change during a communication session, DeltaShaper allows for the readjustment of the encoding parameters. Essentially, this is achieved by enabling the client to repeat the calibration process to determine the new parameters and then reuse such parameters to encode the ensuing data frames. Each period comprising a calibration phase and a data transmission phase is called an *epoch*. Each session can comprise multiple epochs. The time span dedicated to the calibration phase and to the data transmission phase are defined by configuration. To support epochs, the meta-data block includes a *data bit* that indicates whether the frame carries useful data or is used for calibration purposes only. Figure 3 illustrates the concept of epochs visually; it depicts a timeline diagram that represents a two-way communication channel established by DeltaShaper. There are two independent communication flows, used to send IP packets in both directions: from client to server (C→S) and from server to client (S→C).

# 5 Implementation

We implemented a DeltaShaper prototype for Linux. Our system comprises several components that implement the client and server pipelines of DeltaShaper (see Figure 1). Some components were built from scratch in C++ and Python; others are based on existing tools. To intercept TCP/IP packets sent by applications at the client side, our prototype uses Linux network namespaces and the *netfilter* packet filtering framework [26]. Outgoing IP packets are captured by a kernel module using netfilter and handled by a user-space program which encodes and transmits them over Skype. At the server side, IP packets are decoded and routed to the "localhost" interface to be delivered transparently to the server application. Video processing operations are performed with the help of Snowmix [23], FFMPEG [10] and GStreamer [13], which are used to overlay the payload video on a carrier video. At the server side, a pool of worker threads extracts the payload out of the frames, and sends the resulting IP packets to the Linux kernel. To interface with unmodified Skype client software, the video is routed to a virtual camera device and fed into Skype. At the receiver's endpoint, a thread periodically runs the tool XWD to obtain a screenshot of the virtual display framebuffer.

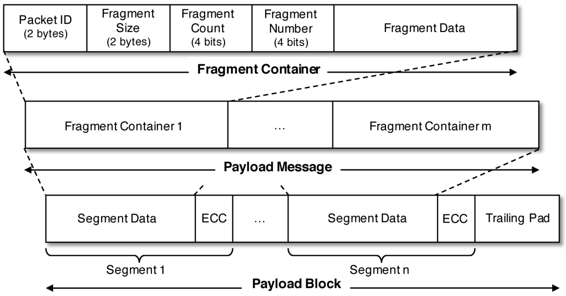

**Fig. 4.** Format of DeltaShaper messages.

## 5.1 System Setup and Operation

To initialize the system and establish a covert TCP/IP tunnel, users must launch DeltaShaper at both channel endpoints. DeltaShaper instantiates both downstream and upstream pipelines at each endpoint. Once the channel has been established, a server application listening at one endpoint can be contacted by a client application running on the counterpart endpoint. Communication occurs over standard TCP/IP sockets without the need to modify the client-server application.

In order to exchange IP packets on overlapping data transmission phases only (see Section 4.4), channel endpoints synchronize each other by checking that the data bit of outgoing and incoming payload frames is cleared to 0. Upstream frame rates of each endpoint must also be adjusted to the maximum supported frame rate. IP packets are encapsulated into payload blocks and encoded into payload frames before transmission. For efficient data transmission, we implement error recovery and packet fragmentation mechanisms. Since Skype provides ordered frame delivery, we do not have to worry about the delivery of out of order or duplicate covert frames. Since covert frames may be lost, e.g., due to network breakups, we simply let TCP request the retransmission of the lost payload blocks.

## 5.2 Message Format and Error Recovery

To support error correction, packet fragmentation, and efficient bit utilization, we specify a simple message format protocol. Figure 4 represents the format of DeltaShaper messages highlighting three internal abstraction layers.

Below, we find the *payload block*, whose bits are directly encoded into cells of a payload frame. The payload block consists of a body divided into segments which are designed to support bit error recovery, resorting to error-correcting codes (ECC). Each segment contains a chunk of application data followed by ECC-type specific redundancy bits.

The intermediate abstraction layer results from extracting the payload block from error redundancy meta-data and concatenating the resulting data segments into a single byte sequence named *payload datagram*. This data structure contains fragments of IP packets. There can be multiple fragments contained in a payload datagram.

The uppermost layer specifies the *fragment container*, which represents an individual IP packet fragment. Fragment containers are included in the body of payload datagrams. The body of each fragment container contains IP packet data, and the header comprises: a packet ID, fragment size, fragment count, and fragment number. These fields enable the recipient to reconstruct the received fragments into complete IP packets.

Since the color of a received frame may be altered by Skype's video compression algorithm, the recovered payload block may include bit errors. Given that discarding entire frames in the presence of errors would penalize throughput, we employ error correcting codes. In particular, we have defined a general payload block layout that supports configurable error-correcting codes. In our current prototype, we adopt Reed-Solomon [32] ECC as a result of our empirical evaluation. We use a commonly used code denoted as $(n, k) = (255, 223)$, where $n$ corresponds to 255 bytes of data symbol, out of which $k = 223$ bytes consist of application data and the remaining 32 bytes encode parity bit symbols (see Figure 4). This code can correct up to 16 symbol errors per symbol block.

## 5.3 Encoding Selector Algorithm

Finally, we discuss the network adaptation algorithm implemented by each party during the calibration phase. This algorithm is fundamental to determine the ideal parameters for payload frame encoding to achive good throughput under the current network conditions while preserving unobservability.

Algorithm 1 provides a sketch of this algorithm. It starts with parameters set to null. When the channel state enters the calibration phase, the algorithm starts playing the carrier video and intercepting packets of the corresponding Skype stream. For each packet, it stores relevant information about the packet in a local database, namely the packet size. When the calibration phase ends, the similarity metric (EMD) is calculated between a set of reference streams and the recorded packet samples. Each of these computations yields a $\Delta_r$ value. The next step is to select the smallest of these values $\Delta_{min}$ which corresponds to the reference distribution mostly similar to the sampled distribution. If $\Delta_{min}$ is greater than the threshold $\Delta$ then the communication is likely to be unsafe and the user is recommended to abort the transmission (Line 8). Otherwise, the communication can be safely undertaken while preserving unobservability. As a result, the algorithm sets the frame encoding parameters from the recommended parameters for the selected reference distribution $s_R$ (Line 9). When starting the data transmission phase, these parameters will be adopted by the payload encoding scheme (see Section 4.2).

**Algorithm 1** Adaptive encoding selector algorithm

```
1: procedure EncodingSelector($s, T_r$)
2:     $s_r, \Delta_{min} \leftarrow null, \infty$
3:     for all $r$ in reference streams $T_r$ do
4:         $\Delta_r \leftarrow \text{EMD}(f_l(s), f_l(T_r.dist(r)))$
5:         if $\Delta_r < \Delta_{min}$ then
6:             $s_r, \Delta_{min} \leftarrow r, \Delta_r$
7:     if $\Delta_r > \Delta$ then
8:         return $null$
9:     return $T_r.params(s_r)$
```

# 6 Evaluation

To evaluate our system, we test the success of EMD and $\Delta$ threshold metrics in characterizing Skype streams (Section 6.1), assess the ability of DeltaShaper to generate unobservable covert channels based on such metrics (Section 6.2) and measure the performance of DeltaShaper channels while preserving unobservability (Section 6.3). We explore alternative traffic features and similarity functions for the purpose of the classification of streams (Section 6.4) and study the impact of network perturbations in the ability to distinguish between regular and irregular streams (Section 6.5). Lastly, we evaluate our system's coverage (Section 6.6) and discuss some security properties offered by DeltaShaper (Section 6.7).

**Experimental testbed:** Our evaluation was performed on a quad-core Intel Xeon CPU E3-1220 v3 3.10GHz system provisioned with 32GB of RAM. We set up two 32bit Ubuntu 14.04.3 LTS virtual machines (VMs) with 2GB RAM and 4 virtual CPUs. Each VM runs an instance of Skype v4.3.0.37 and DeltaShaper acting, respectively, as caller and callee of videocall connections. To emulate varying network conditions between the VMs, we used the native *netem* Linux network emulation functionality. Regarding the dataset, we selected 30 videos representative of actual videocalls. We use these videos as our training set for regular streams. Such videos generally exhibit low movement as users typically sit in front of a computer. These videos have not been edited and are free of water-

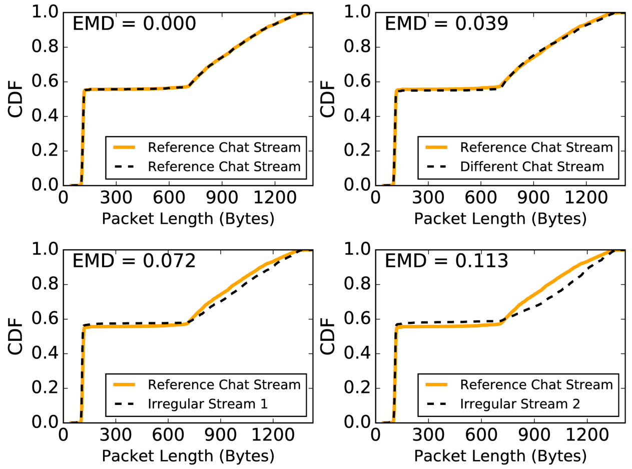

**Fig. 5.** Packet length CDF of sample streams.

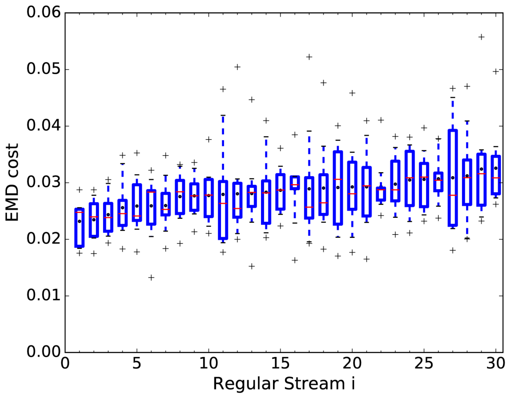

**Fig. 6.** EMD cost of multiple videocall streams.

marks or other visual artifacts. Our dataset of irregular streams is composed of 30 YouTube videos, where both rapidly changing scenes and artifacts introduced by video editing software are common. The duration of each video sample is 30 seconds.

## 6.1 Characterization of Skype Streams

We first study whether Skype calls do exhibit measurable patterns that allow us to differentiate regular from irregular calls. This question is important because such patterns can be used by a censor to detect suspicious videocalls (i.e., irregular ones) and block them. The data in Figure 5 indicates that such patterns do exist. It shows the cumulative distribution function (CDF) of packet lengths for four test Skype streams, each one represented in a different plot: (a) the stream of an actual videocall which we take as our reference stream; (b) a stream of a regular call from a different user; and two irregular streams corresponding to (c) a football match and, (d) a music concert. In each plot, we represent the distribution of the test stream along with the packet length distribution of our reference stream (the black curve), and determine their similarity by calculating the respective EMD value. We see that EMD increases progressively, reaching 0.113 for the most dynamic video, i.e., the music concert stream. The main differences can be observed for 40% of packets, which correspond to the largest packets (above 745 bytes) transmitted. These packets must encode a higher amount of inter- and intra-frame differences than in regular videocalls.

To better understand whether these traffic patterns are stable, and thus can be reliably used for characterizing regular Skype streams, we study if there are significant differences in packet length distribution when streaming the same videocall multiple times over Skype. For each regular video call sample of our dataset, we replay it 10 times and calculate the EMD of each resulting stream taking as baseline the average distribution of all 10 runs. Figure 6 plots the most relevant statistical indicators for the resulting EMD values of each video: min, max, mean, and percentiles 5, 25, 50, 75, and 95. On the one hand, packet length distributions of the same video tend to be quite similar. This is attested by the fact that the largest difference observed between 25th and 75th percentile of a single video is only 0.02. Also, the average EMD value tends to be very similar among different videos, varying between 0.025 and 0.031. Thus, under the same network conditions, regular Skype streams display a high degree of similarity.

Our next step is to study whether a censor can differentiate regular from irregular streams by computing the similarity of packet length distributions. To that end, we take a regular stream and use it as reference stream to calculate the EMD cost of other video streams. These video streams were generated by running each video of the data set 10 times and calculating statistical indicators of the resulting EMD cost. Figure 7 (a) shows the results obtained, plotting on the left hand side the EMD cost for regular streams, and on the right hand side the EMD cost for irregular streams. We can see a pattern in which regular streams tend to result in a constantly low EMD cost (below 0.1), whereas irregular streams produce a significantly scattered pattern varying EMD cost approximately from as low as 0.025 to as high as 0.25, i.e., by an order of magnitude.

The question is then whether it is possible to define an EMD cost $\Delta$ threshold that can be used as stream classifier such that a stream $s$ is considered regular if EMD$(s_R, s) < \Delta$ or irregular otherwise (see Section 4.3). For this particular experiment, $\Delta$ can be set to any value between 0 and the maximum observed EMD cost (0.275). To evaluate the effectiveness of this classifier, we show in Figure 7 (b) the probability of true positives (sensitivity) and true negatives (specificity) as we vary $\Delta$ (in x-axis). We can see that, as $\Delta$ increases the

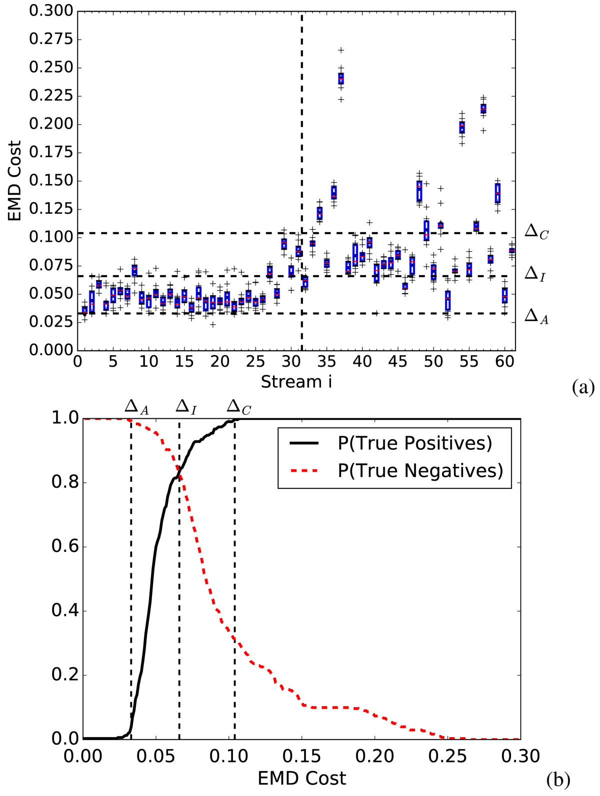

**Fig. 7.** EMD based on reference stream.

number of true negatives starts at 1, meaning that all irregular streams are correctly identified by the classifier, but eventually starts decreasing at 0.025 ($\Delta_A$) because some irregular streams start being classified as regular. In contrast, the true positive rate curve begins in 0 and starts increasing when the EMD cost of some regular streams becomes lower than $\Delta$. Eventually, when $\Delta$ reaches 0.11 ($\Delta_C$), the classifier is able to correctly identify all regular streams.

Based on how the $\Delta$ threshold is set, several classification policies are possible. Suppose that a censor wishes to apply an *aggressive policy* by blocking all streams that are truly irregular. In this case, $\Delta$ must be set to $\Delta_A$, which is the point where the true negative rate starts falling below 100%. The downside of this policy, however, is that a large number of regular streams would also be blocked, more specifically 95% of regular streams (false negatives) causing a massive denial of service of legitimate Skype users. On the other hand, if the censor aims to prevent blocking of any regular Skype transmissions (i.e., a *conservative policy*), $\Delta$ must be set to $\Delta_C$. The negative side-effect of this policy is, however, a loss in specificity since approximately 80% of irregular streams would also be classified as regular (false positives). An intermediate possibility that considers the best of both worlds is to take the break-even point where the probability of true negatives equals the probability of true positives. For our classifier, this point corresponds to EMD cost 0.066 ($\Delta_I$) which means that setting $\Delta$ to this value results in 83% accuracy in classifying a stream. Thus, one can define a $\Delta$ threshold to identify regular streams with high probability. This is crucial as DeltaShaper explores this property to hide within regular streams.

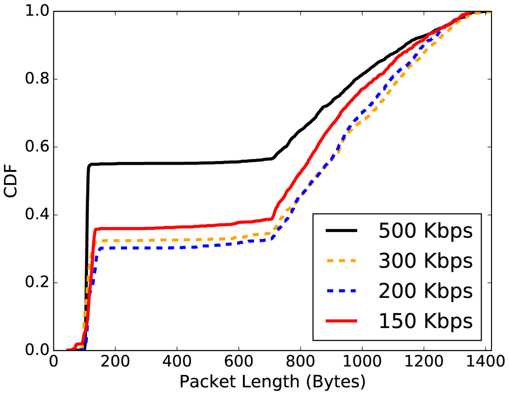

**Fig. 8.** Packet length varying bandwidth.

For the characterization of Skype streams performed up until now, we have used the same network conditions for Skype transmissions. However, the packet length distribution of Skype streams depends on the available network bandwidth. Figure 8 shows the packet length CDF of our reference stream as we vary the channel bandwidth between 150 and 500 Kbps. We can identify two main groups of streams which display different CDF shapes: In line with Skype's bandwidth requirements, one group corresponds to “Regular” video quality and comprises a bandwidth range between 128 Kbps and 300 Kbps, a second group includes streams between 400Kpbs and 500Kpbs which corresponds to “High Quality Video” transmission. In addition, amongst “Regular” video quality streams there are also notable differences amongst them, especially for larger packets (700+ bytes length). This confirms that the reference stream must be chosen for specific network conditions of the current communication channel.

## 6.2 Unobservability Assessment

Skype streams exhibit specific packet length patterns that allow a censor to distinguish regular from irregular streams based on an EMD classifier. Consequently, to produce unobservable covert Skype streams, DeltaShaper must be set up such that the EMD cost of the resulting stream remains below the $\Delta$ threshold. Since the properties of a resulting stream depend on the encoding parameters provided to DeltaShaper, we must study which range of encoding parameters is reliable to produce unobservable covert streams.

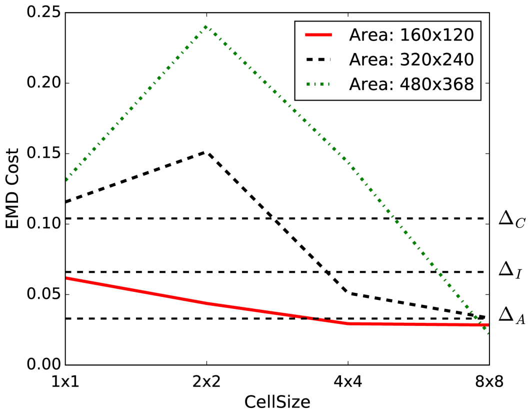

**Fig. 9.** EMD cost changing area and cell size.

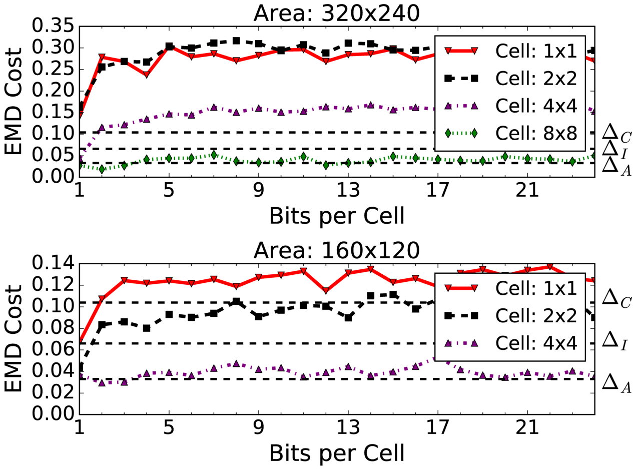

**Fig. 10.** EMD cost varying the bits per cell.

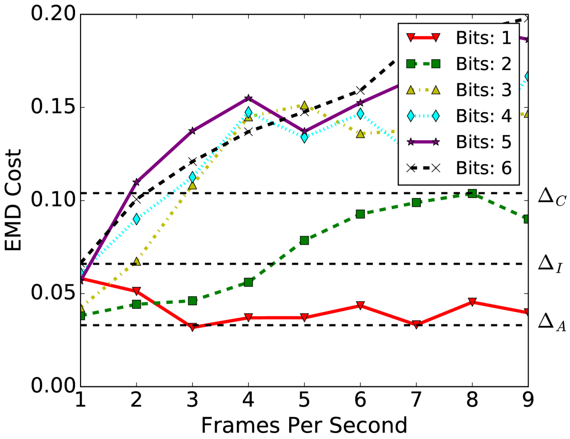

**Fig. 11.** EMD cost varying the frame rate.

As listed in Table 1, DeltaShaper can be configured with four parameters: payload area size, cell size, bit number, and frame rate. Since the number of possible configurations is very large, we focus on a subset of configurations that results in valid configurations, but not necessarily optimal in terms of the maximum throughput that can be achieved. In our study, we take the reference stream that was used in the previous section and the $\Delta$ threshold values that were found for that reference stream considering “High Quality Video” network bandwidth.

We start by analyzing the combined effects of the payload area size and the cell size, fixing the bit number in 1 bit/cell and the frame rate in 1 frame per second. Figure 9 shows the EMD cost for several configurations varying the cell size between 1x1 and 8x8 pixels and the area size ranging from 160x120, 320x240, and 480x368. The area sizes were chosen to cover roughly 1/16, 1/4, and 1/2 of the frame size, respectively, while aligning the payload size to the size of a macroblock. In the H.264 video encoding standard (used by Skype), frames are divided into small matrices of pixels (16x16) named macroblocks which are used for improving the efficiency of the video compression algorithm. The plot is annotated with $\Delta$ threshold values for the three policies discussed in the previous section: aggressive ($\Delta_A$), conservative ($\Delta_C$), and intermediate ($\Delta_I$). For example, for an intermediate policy, there are five configurations that produce unobservable streams, i.e., for area sizes 160x120 or 320x240 and cell sizes 4x4 or 8x8; and for area size 480x368 and cell size 8x8. As the cell size increases, the EMD cost tends to decrease because larger areas of the frames will be colored with the same color thereby improving the efficiency of the video compression algorithm. However, the payload area size 480x368 was consistently found to generate streams identified as irregular by the classifier, when encoding more than 1 bit per cell.

For the area / cell size configurations found to be valid, we studied how unobservability changes as a function of the number of bits per cell. Figure 10 shows our results covering the domain of data bit numbers. In general, unobservability tends to be degraded as the number of bits increases. Some configurations, however, have a more flattened evolution of the EMD cost. In particular, two configurations fall consistently below the $\Delta$ threshold value for intermediate blocking policy ($\Delta_I$), namely (160x120, 4x4) and (320x240,8x8). This means that both these encoding configurations are good candidates to generate covert streams that are both unobservable and can deliver a large data throughput range (due to the large data bit number). Note that since (320x240,8x8) is proportionally larger than (160x120, 4x4), both these configurations can encode the same amount of cells per frame.

Lastly, we study how the frame rate affects unobservability. Figure 11 shows how the EMD cost varies as we increase the frame rate. We define a fixed payload area size 320x240 and cell size 8x8, and measure the EMD cost for a cell bit encoding range varying between 1 and 6 bits per cell. Although

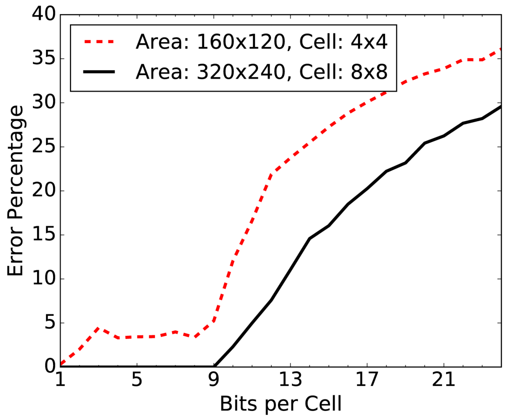

**Fig. 12.** Error rate increasing bits per cell.

the (160x120, 4x4) configuration encodes the same amount of bits per cell, these are more difficult to decode due to a higher error rate. Results show that increasing the frame rate quickly results in EMD costs above Δ. A notable exception is the 1 bit per cell encoding scheme, which remains below Δ for all tested frame rates. Schemes with higher bit numbers can only tolerate the minimal frame rate value (1 frame per second).

## 6.3 Channel Performance

Although it is possible to generate covert streams from numerous encoding configurations, DeltaShaper can only safely adopt those that result in unobservable streams. The need to satisfy this property constrains the maximum amount of encoded data per frame, placing a limit to the performance that can be delivered by a DeltaShaper channel. Performance can also be affected by decoding errors at the receiver when interpreting the cell color of payload frames. As the number of encoding bits used per cell increases, the video compression algorithm tends to introduce more changes in the less significant bits of the color of each pixel, introducing more decoding errors. This trend can be observed in Figure 12, which shows how the error rate increases as the number of encoding bits increases for configurations (160x120, 4x4) and (320x240, 8x8).

Taking into account both restrictions in terms of unobservability and decoding errors, we identify the candidate encoding configuration for DeltaShaper which consists of: 320x240 area size, 8x8 cell size, 6 bits per cell, at 1 frame per second. We decided to be more conservative with respect to the bit encoding scheme (less than 9 bits per cell) because even sporadic unrecoverable bit errors can result in the loss of an entire payload frame which will significantly affect throughput. Table 2 lists the maximum goodput that we can achieve under this scheme: 2.56 and 3.12 Kbps, respectively with and without error correction codes, measured while running an actual TCP connection and taking into account the overheads introduced by covert frames' meta-data and TCP/IP headers.

| Params | With ECC | Without ECC |
|---|---|---|
| Raw Throughput | 6.24 Kbps | 7.20 Kbps |
| Goodput | 2.56 Kbps | 3.12 Kbps |
| RTT | 2s 984ms | 2s 973ms |

**Table 2.** Performance measurements.

**Comparison with state-of-the-art systems:** Although we could not repeat the same experiments for related censorship-resistant systems, we provide some performance numbers of such systems as reported by their respective authors. As a tunneling system incapable of providing resistance against passive attacks, FreeWave achieves a throughput of 18.75 Kbps. In DeltaShaper, covert data is carefully encoded before being tunneled through the cover protocol, ensuring that the generated traffic resembles legitimate flows. This approach is similar to that of Castle and CovertCast. Interestingly, we observe that the goodput obtained by DeltaShaper is within the total throughput obtained by Castle – DeltaShaper reaches a goodput of 3.12 Kbps while Castle attains a covert transfer rate of 3.48 Kbps without accounting for the overheads introduced by high level protocols. The higher throughput attained by CovertCast – roughly 168 Kbps – can be explained by the lack of profile-specific optimizations performed by the underlying video codec, a higher video resolution, and the wide diversity of videos composing live-streaming platforms' traffic. Lastly, by focusing on steganographic security, SkypeLine and Rook attain much lower covert data transfer rates - the throughput of Rook caps between 0.024 Kbps and 0.04 Kbps while SkypeLine attains a maximum throughput of 0.064 Kbps.

## 6.4 Alternative Metrics

So far, we have used a single reference stream for computing the Δ thresholds. We have also assumed that the adversary was using the same functions as DeltaShaper to detect covert channels (i.e., packet length as the feature function and EMD as the similarity function). However, in a real setting, an adversary would compare the target stream being classified not with a single regular stream, but with a set of regular streams, thus taking into account the differences among regular streams that are inherent to Skype traffic. Also, the adversary may use alternative features or similarity functions to achieve higher accuracy in detecting streams with a covert channel.

To assess the ability of the adversary to correctly classify streams when using different functions, we have performed additional tests, letting the adversary use alternative traffic

| Feature<br>Similarity | Packet Length<br>EMD% / KS% | Bi-Grams Packet Length<br>EMD% / KS% | Inter-Packet Time<br>EMD% / KS% | Bi-Grams Inter-Packet Time<br>EMD% / KS% |
|---|---|---|---|---|
| | Unperturbed Stream | | | |
| | 88 / 78 | 85 / 72 | 85 / 82 | 88 / 77 |
| | Perturbed Stream | | | |
| Bandwidth | | | | |
| 500 Kbps | 85 / 70 | 82 / 78 | 88 / 75 | 90 / 68 |
| 300 Kbps | 75 / 68 | 75 / 68 | 72 / 68 | 78 / 68 |
| Packet Loss | | | | |
| 5% | 76 / 72 | 82 / 72 | 68 / 75 | 78 / 70 |
| 10% | 75 / 82 | 78 / 78 | 72 / 73 | 75 / 60 |
| 20% | 78 / 75 | 85 / 78 | 75 / 58 | 88 / 72 |
| Network Jitter | | | | |
| 10ms | 82 / 78 | 85 / 72 | 75 / 72 | 77 / 68 |
| 20ms | 78 / 85 | 85 / 78 | 67 / 60 | 67 / 67 |
| 50ms | 82 / 78 | 82 / 78 | 68 / 55 | 68 / 62 |

**Table 3.** Accuracy of the adversary using different feature and similarity functions in unperturbed/perturbed streams. Underlined entries represent the accuracy of a classifier relying on packet lengths frequency distribution as feature function and EMD as similarity function.

features, namely: bi-gram distribution of packet sizes, inter-packet time, bi-gram distribution of inter-packet times. Some of these features have been previously adopted in the related work [22, 24, 25]. We have also experimented as a similarity metric the Kolmogorov-Smirnov (KS) distance-based classifier for all the studied traffic features, given that the 2-sample KS test for statistical significance has been successfully applied in traffic classification in the related literature [25, 28].

Table 3 depicts the classification results for the different combinations of the features listed above with both the EMD and KS similarity metrics. In all these results, each regular / irregular test stream was compared against all streams comprising the regular streams dataset (by taking the average of the different $\Delta$s). The first row of the table shows the classifier accuracy in unperturbed network conditions, while the remaining rows show the classifier accuracy for streams in perturbed network conditions. The induced perturbations will be described in Section 6.5 thus, for now, we concentrate on the results depicted in the first line. As expected, the adversary is able to distinguish between regular and irregular Skype streams when using other traffic features. In unperturbed conditions, a classifier based on the bi-grams of inter-packet times achieves a similar accuracy to one based in packet lengths (88%). However, building the latter model takes nearly tenfold the time spent in building the former ($\sim$64s vs $\sim$6s) due to the computations involved in the creation and processing of bi-grams. Thus, we consider that packet length / EMD offers the best results (which further substantiates the choice of these functions for parameter configuration on our prototype).

Furthermore, when comparing the EMD similarity metric with KS, it is possible to observe that EMD offers a better traffic classification accuracy. Particularly, for unrestricted network conditions, the combination of the KS similarity metric and the analysis of inter-packet times yields 82% accuracy, 6% short of the maximum classification accuracy that can be achieved by EMD under the same network conditions (88%).

## 6.5 Effect of Network Perturbations

A censor may introduce controlled network perturbations with the goal of increasing stream classification accuracy. We study the impact of bandwidth throttling, loss of random packets, and the introduction of jitter and packet delay on the ability of the censor to distinguish between regular / irregular streams. If such perturbations allow a censor to increase its accuracy without severely degrading the quality of the video calls, this fact could be used to more reliably detect DeltaShaper streams.

Table 3 presents the relevant results of our study. It shows the accuracy of the adversary, when it uses different feature and similarity functions to classify regular / irregular streams under different network perturbations. In the discussion below, when referring to concrete values, we use the values obtained by an attacker relying on the packet length / EMD functions to perform the classification (values underlined).

**Bandwidth throttling:** To assess the classification accuracy of throttled Skype streams, we repeat our tests for three bandwidth configurations: unrestricted, 500 Kbps, and 300 Kbps. A 300 Kbps throttle resulted in a high packet loss rate, which culminated in a high FPS drop from 30 to 5 FPS, and impairing visual experience. As shown in the table, the classifier accuracy worsens alongside bandwidth reductions, from 88% (unrestricted) to 75% (300 Kpbs). By throttling streams to 300 Kbps, ill-classifications are at least doubled versus unthrottled

| Use Case | Protocol Setup w/ DS (mm:ss) | Protocol Session w/ DS (mm:ss) | Protocol Session w/o DS (mm:ss) | Overhead |
|---|---|---|---|---|
| A. Wget | 0:13 | 0:22 | < 0:01 | 3,142.9× |
| B. FTP | 0:29 | 1:43 | 0:09 | 11.4× |
| C. SMTP | 0:51 | 2.41 | 0:38 | 4.2× |
| D. SSH | 1:09 | 1:29 | 0:06 | 14.8× |
| E. Telnet | 0:51 | 1:13 | 0:06 | 12.2× |
| F. Netcat Chat | - | 0:01 | < 0:01 | 166.7× |
| G. SSH Tunnel | 1:09 | 2:19 | 0:22 | 6.3× |

**Table 4.** Execution time for DeltaShaper use cases: fetch 4KB web page from receiver (A), connect to receiver through FTP, run 'ls' and download a 4KB file (B), open telnet to receiver and tunnel a small email through an SMTP server running on the receiver (C), open SSH session to receiver and run "ls" (D), open telnet to receiver and execute "ls" (E), send message to netcat server on receiver mimicking text chat (F), tunnel SSH session to remote SSH server through the receiver, and execute "ls" (G).

streams. Thus, this strategy is not effective due to the large rate of false positives and false negatives that it may generate.

**Packet loss:** A censor can also perform arbitrary packet dropping. Packet dropping may affect ongoing streams in unpredictable ways that may be used to increase the classification accuracy between regular and irregular streams. To study this effect, we repeat our tests for three different packet loss rates: 5%, 10%, and 20%. By visually examining ongoing streams, we observed that while a 5% packet dropping rate is sufficient for sustaining an acceptable viewing experience, a 20% packet dropping rate greatly affects the stream quality. As Table 3 shows, the classifier accuracy is severely affected by even a small packet dropping rate. In fact, when analyzing the minimal degree of packet loss tested (5%), the false positive / negative rates (24%) are doubled when compared with unimpaired network conditions' false positive / negative rates (12%).

**Packet delay and jitter:** Videoconferencing traffic tends to be sensitive to network jitter, since multimedia data must be delivered in a timely and sequential fashion. To study whether the manipulation of jitter translates into an increase of classification accuracy, we repeat our tests for three different configurations of packet delay and jitter: the connection delay was set to 20ms while jitter was adjusted in each different experiment to 10ms, 20ms, and 50ms, respectively. As shown in Table 3, the accuracy of the classifier is negatively affected, although moderately, by the introduction of jitter. One can also argue that a censor would refrain from employing perturbations other than the one which constitutes the first experiment, since the introduction of 20ms or 50ms of jitter greatly impacts the end-user experience for legitimate connections.

**Summary:** Results in Table 3 justify our design choices when building stream classifiers. We have observed that when alternative features were used to classify streams in perturbed conditions, classifiers generally followed a decline in accuracy vs unperturbed conditions. In fact, only a single classifier based on bi-grams of inter packet times (BIPT) / EMD was able to obtain a better classification accuracy in a specific setting (Bandwidth = 500 Kbps). Although the frequency distribution of packet lengths (PL) may not be the absolute best feature to inspect in every scenario (e.g. Bandwidth = 500 Kbps, PL / EMD = 85%, BIPT / EMD = 90%), this feature function still offers a reasonable accuracy while providing a sound and lightweight approach for building classifiers. The analysis of PL has also shown to be the best approach to classify streams in unperturbed environments. Hence, the analysis of such feature may be advantageous to a censor which attempts to classify streams while refraining itself from introducing perturbations that affect the quality of legitimate Skype streams. Lastly, when comparing between EMD and KS similarity functions, we can observe that among all experiments KS seldom offers better results than EMD (in only four ocasions).

## 6.6 Use Cases

Given that the data throughput that can be achieved while preserving unobservability is relatively small, DeltaShaper is not adequate for the transmission of bulk data. Nevertheless, it can sustain the execution of a plethora of applications that are not bandwidth hungry. To confirm this hypothesis, we tested DeltaShaper with seven use cases (A-G) depicted in Table 4.

In use cases A-F, the client communicates only with the receiver over a DeltaShaper channel. In use case G, the receiver acts as a relay tunneling traffic between the client and a remote party. Excepting case A, all other use cases are performed interactively, where a proficient user types the commands required to establish the different types of connections in a terminal. Table 4 provides a summary of the execution time for each use case when performed using DeltaShaper and without using DeltaShaper, i.e., using overt communication channels between client and receiver. The figures represented in the third and fourth column of Table 4 account for an average of the cumulative time of an entire session of the considered protocol, including connection setup phase. The second column of this table depicts the time spent in the connection setup phase of the protocol when using DeltaShaper. We note that the latency observed for the feedback of commands issued in interactive sessions is akin to the time spent in observing a request-response through netcat, i.e. in the order of seconds.

We see that the execution time is several orders of magnitude higher in DeltaShaper than in overt channels. Such a large overhead is expected given the low throughput that DeltaShaper can currently deliver. Nevertheless, in spite of these delays, all tested use cases are fully functional. We can observe that our system allows for the execution of traditional TCP/IP applications which can be used in practice to cover different needs exhibited by users actively evading censorship.

### 6.7 Security Discussion

We now discuss relevant security properties of DeltaShaper:

**1. Attacks detecting patterns in streams:** Since Skype traffic is encrypted, it is hard for a censor to detect patterns based on packet content. In addition, given that DeltaShaper generates covert streams indistinguishable from regular streams, censors cannot block our system without disrupting the Skype service to legitimate users. Nevertheless, it is possible that by replaying the same carrier video for covert channel modulation purposes, repeatable packet distribution patterns can emerge that may be spotted by a censor. To mitigate this attack, carrier videos may be directly captured from an actual webcam or pre-recorded carrier videos may be rotated periodically by DeltaShaper to limit pattern exposure to the censor.

**2. Denial of service attacks:** A censor may drop packets of Skype streams to prevent the delivery of video frames. This attack, however, does not permanently disrupt DeltaShaper's covert channel because the TCP layer will retransmit lost IP packets automatically in newly issued frames. Also, DeltaShaper frames are self-contained: meta-data and payload data are included in each frame being scraped. Thus, the censor is unable to prevent the decoding of covert frames which are successfully delivered. Moreover, shall the censor extend its attack period, Skype video calls will experience a reduction in quality (as studied in Section 6.5) affecting legitimate users.

**3. Defense against active probing:** In order to thwart active probing attempts, DeltaShaper servers can be configured to accept calls from added contacts, ignoring calls from unknown Skype IDs. To advertise DeltaShaper servers, volunteers may share circumvention servers' Skype IDs and respective access credentials (like a password) through some out-of-band channel among users within the censored region. A client can then place a contact request, including the corresponding credential, to a DeltaShaper server. This prevents the circumvention server to blindly accept calls from censors' Skype endpoints.

## 7 Future Work

In this section we outline some avenues of future work regarding the issue of dynamic adaptation for achieving unobservability and further studies on the applicability of our system to alternative videoconferencing applications.

**Adaptation mechanism:** As described in Section 4.4, the calibration procedure is conducted periodically to set the encoding parameters for the next data transmission phase. If the period between consecutive adaptations is large enough, the adversary may have enough time to perturb the stream and observe the covert channel. The quest for more sophisticated adaptive mechanisms which can strengthen the unobservability of the covert stream is an interesting topic of future research.

**Video-carrier independence:** It would be interesting to assess whether DeltaShaper can be used with alternative videoconferencing applications. Albeit we have not performed any experiments to verify this hypothesis, we have two arguments that suggest this is possible. Firstly, DeltaShaper requires no changes to the videoconferencing application; it simply transmits frames to a camera driver that feeds the application. Secondly, several videoconferencing systems use the same or similar codecs as Skype. Even if the videoconferencing system uses another codec, our calibration procedure is codec agnostic and does not require previous knowledge about the characteristics of the codec to derive encoding parameters.

## 8 Conclusions

In this paper we introduced DeltaShaper, a censorship-resistant system which leverages the video channel of a popular videoconferencing application to tunnel covert data. Our system offers a data-link interface, supporting the execution of several applications running over TCP/IP, offering users different possibilities for transferring information in an unobservable way. We have performed an extensive evaluation of our prototype so as to define which combinations of encoding parameters can defend against traffic analysis.

***Acknowledgments:*** We thank the anonymous reviewers and our shepherd, Mashael Al-Sabah, for their comments and suggestions. This work was partially supported by the EC through project H2020-645342 (reTHINK), and by Fundação para a Ciência e Tecnologia (FCT) through the project with reference UID/ CEC/ 50021/ 2013.

# References

[1] C. Bocovich and I. Goldberg, "Slitheen: Perfectly imitated decoy routing through traffic replacement," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Vienna, Austria, 2016, pp. 1702–1714.

[2] C. Brubaker, A. Houmansadr, and V. Shmatikov, "Cloudtransport: Using cloud storage for censorship-resistant networking," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, E. De Cristofaro and S. Murdoch, Eds. Springer International Publishing, 2014, vol. 8555, pp. 1–20.

[3] A. Chaabane, T. Chen, M. Cunche, E. De Cristofaro, A. Friedman, and M. A. Kaafar, "Censorship in the wild: Analyzing Internet filtering in Syria," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, Vancouver, BC, Canada, 2014, pp. 285–298.

[4] R. Dingledine, "Obfsproxy: the next step in the censorship arms race," https://blog.torproject.org/blog/obfsproxy-next-step-censorship-arms-race, 2012, accessed: 2017-06-12.

[5] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th Conference on USENIX Security Symposium*, San Diego, CA, USA, 2004.

[6] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Protocol misidentification made easy with format-transforming encryption," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, Berlin, Germany, 2013, pp. 61–72.

[7] K. P. Dyer, S. E. Coull, and T. Shrimpton, "Marionette: A programmable network-traffic obfuscation system," in *Proceedings of the 24th USENIX Conference on Security Symposium*, Washington, D.C., USA, 2015, pp. 367–382.

[8] T. Elahi, C. M. Swanson, and I. Goldberg, "Slipping past the cordon: A systematization of Internet censorship resistance," in *CACR Tech Report 2015-10*, 2015.

[9] R. Ensafi, D. Fifield, P. Winter, N. Feamster, N. Weaver, and V. Paxson, "Examining how the great firewall discovers hidden circumvention servers," in *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, Tokyo, Japan, 2015, pp. 445–458.

[10] FFmpeg, https://ffmpeg.org, 2000, accessed: 2017-06-12.

[11] D. Fifield, C. Lan, R. Hynes, P. Wegmann, and V. Paxson, "Blocking-resistant communication through domain fronting," in *Proceedings on Privacy Enhancing Technologies 2015.2*, Philadelphia, PA, USA, 2015, pp. 46–64.

[12] J. Geddes, M. Schuchard, and N. Hopper, "Cover your acks: Pitfalls of covert channel censorship circumvention," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, Berlin, Germany, 2013, pp. 361–372.

[13] GStreamer, https://gstreamer.freedesktop.org/, 2001, accessed: 2017-06-12.

[14] B. Hahn, R. Nithyanand, P. Gill, and R. Johnson, "Games without frontiers: Investigating video games as a covert channel," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. Saarbrucken, Germany: IEEE, 2016, pp. 63–77.

[15] A. Houmansadr, C. Brubaker, and V. Shmatikov, "The parrot is dead: Observing unobservable network communications," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, San Francisco, CA, USA, 2013, pp. 65–79.

[16] A. Houmansadr, G. T. Nguyen, M. Caesar, and N. Borisov, "Cirripede: Circumvention infrastructure using router redirection with plausible deniability," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, Chicago, IL, USA, 2011, pp. 187–200.

[17] A. Houmansadr, T. J. Riedl, N. Borisov, and A. C. Singer, "I want my voice to be heard: IP over Voice-over-IP for unobservable censorship circumvention." in *Proceedings of the 20th Annual Network & Distributed System Security Symposium*, San Diego, CA, USA, 2013.

[18] J. Angwin, C. Savage, J. Larson, H. Moltke, L. Poitras and J. Risen, "AT&T Helped U.S. Spy on Internet on a Vast Scale," https://www.nytimes.com/2015/08/16/us/politics/att-helped-nsa-spy-on-an-array-of-internet-traffic.html, 2015, accessed: 2017-06-12.

[19] J. Karlin, D. Ellard, A. Jackson, C. Jones, G. Lauer, D. Mankins, and T. Strayer, "Decoy routing: Toward unblockable Internet communication," in *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet*, San Francisco, CA, USA, 2011.

[20] S. Khattak, T. Elahi, L. Simon, C. M. Swanson, S. J. Murdoch, and I. Goldberg, "Sok: Making sense of censorship resistance systems," in *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 4, Darmstadt, Germany, 2016, pp. 37–61.

[21] K. Kohls, T. Holz, D. Kolossa, and C. Pöpper, "SkypeLine: Robust hidden data transmission for VoIP," in *Proceedings of the 2016 ASIA Computer and Communications Security*, Xi'an, China, 2016.

[22] S. Li, M. Schliep, and N. Hopper, "Facet: Streaming over videoconferencing for censorship circumvention," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, Scottsdale, AZ, USA, 2014, pp. 163–172.

[23] P. Maersk-Moller, "Snowmix," https://sourceforge.net/projects/snowmix/, 2012, accessed: 2017-06-12.

[24] R. McPherson, A. Houmansadr, and V. Shmatikov, "Covertcast: Using live streaming to evade internet censorship," in *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 3, Darmstadt, Germany, 2016, pp. 212–225.

[25] H. Moghaddam, B. Li, M. Derakhshani, and I. Goldberg, "Skypemorph: Protocol obfuscation for Tor bridges," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, Raleigh, NC, USA, 2012, pp. 97–108.

[26] Netfilter Framework, http://www.netfilter.org/, 1998, accessed: 2017-06-12.

[27] Y. Rubner, C. Tomasi, and L. J. Guibas, "The Earth Mover's Distance As a Metric for Image Retrieval," *Int. J. Comput. Vision*, vol. 40, no. 2, pp. 99–121, Nov. 2000.

[28] P. Vines and T. Kohno, "Rook: Using video games as a low-bandwidth censorship resistant communication platform," in *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society*. Denver, CO, USA: ACM, 2015, pp. 75–84.

[29] L. Wang, K. P. Dyer, A. Akella, T. Ristenpart, and T. Shrimpton, "Seeing through network-protocol obfuscation," in *Pro-*

*ceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, Denver, CO, USA, 2015, pp. 57–69.

[30] Q. Wang, X. Gong, G. T. Nguyen, A. Houmansadr, and N. Borisov, "Censorspoofer: Asymmetric communication using IP spoofing for censorship-resistant web browsing," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, Raleigh, NC, USA, 2012, pp. 121–132.

[31] Z. Weinberg, J. Wang, V. Yegneswaran, L. Briesemeister, S. Cheung, F. Wang, and D. Boneh, "Stegotorus: A camouflage proxy for the Tor anonymity system," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, Raleigh, NC, USA, 2012, pp. 109–120.

[32] S. B. Wicker, *Reed-Solomon Codes and Their Applications*. IEEE Press, 1994.

[33] P. Winter, T. Pulls, and J. Fuss, "Scramblesuit: A polymorphic network protocol to circumvent censorship," in *Proceedings of the 12th ACM Workshop on Privacy in the Electronic Society*, Berlin, Germany, 2013, pp. 213–224.

[34] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic morphing: An efficient defense against statistical traffic analysis," in *Proceedings of the 16th Network and Distributed Security Symposium*, San Diego, CA, USA, 2009, pp. 237–250.

[35] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman, "Telex: Anticensorship in the network infrastructure," in *Proceedings of the 20th USENIX Conference on Security*, San Francisco, CA, USA, 2011.

[36] W. Zhou, A. Houmansadr, M. Caesar, and N. Borisov, "Sweet: Serving the web by exploiting email tunnels," in *Proceedings of the 6th Workshop on Hot Topics in Privacy Enhancing Technologies*, Bloomington, IN, USA, 2013.