Unobservable Covert Streaming for Internet Censorship Circumvention

(extended abstract of the MSc dissertation)

Diogo Miguel Barrinha Barradas
Departamento de Engenharia Informática
Instituto Superior Técnico

Advisors: Professors Luís Rodrigues and Nuno Santos

***Abstract*—This work studies the possibility of using the encrypted video channel of widely used video-conferencing applications, such as Skype, as a carrier for an unobservable covert channel that can be used to access arbitrary information on the Internet. Such channel can be used by citizens living in countries ruled by repressive regimes to exert their civil rights without being detected and/or blocked by a censor. Even if the video is encrypted, a covert channel may still be detected if it induces statistically relevant changes in the traffic pattern of the application. We propose and evaluate different alternatives to encode information in the video-stream, in order to maximize the available throughput while preserving the characteristics of the unmodified stream. We have implemented a prototype of our system, named DeltaShaper, that offers a data-link interface and that can support any protocol that runs over TCP/IP. Our results show that it is possible to achieve a throughput of 0.4 kB/s with no significant impact on the stream, which allows to run standard applications such as FTP, SMTP or a simple web client.**

I. Introduction

Today, most electronic communication over the Internet can be controlled by governments and/or by a few corporations. This allows repressive regimes to monitor and control the access to the Internet by employing several censorship techniques. As a result, citizens may be prevented from exercising their civil rights, for instance they may be prevented from accessing information, communicate, or express opinions freely[1].

In certain cases, to prevent access to specific sources of information, the censor employs blacklisting techniques in which it instructs ISPs to block direct connections to these sources. To circumvent such restrictions, a typical strategy consists in accessing this information via a trusted proxy. However, this is only viable as long as the proxy address is not public and the connections to it cannot be easily flagged as suspicious. In order to prevent all proxies addresses to become public, Tor [2] has suggested the use of *bridges*, proxies which addresses are not publicly distributed, to access its overlay network. Even so, if no explicit effort is made to obfuscate the traffic towards a bridge, it may exhibit patterns that make it easily recognizable [3]. Unfortunately, the task of obfuscating the traffic is a challenge by itself. If the resulting pattern does not match any known protocol, the flow can also be deemed as suspicious. Therefore, for the obfuscation to succeed, the resulting traffic should mimic existing protocols, ideally protocols that a censor may not be willing to block. But protocol mimicking can be extremely hard to implement [4] and needs to be adjusted every time the protocol is updated.

Fortunately, even the most oppressive regimes cannot afford to block all electronic communication channels with the outside world, as these channels may be instrumental to preserve the economy of the country and the sustainability of the regime itself. In particular, there is evidence that several countries do restrict access to information but maintain operational widely used services such as Skype. In fact, only in extreme scenarios, governments can afford to completely block the access to the Internet [5].

Based on the observation above, in the last years a number of authors have been exploring a promising approach, that consists in providing access to rightful information by covertly *tunneling* data through protocols that are unlikely to be blocked by the censor. Here, the assumption is that it is possible to implement the tunneling in such a way that the traffic pattern of the carrier protocol is not modified in an observable way. Examples of this line of work include FreeWave[6], which encodes network traffic into acoustic signals sent over VoIP connections, Facet[7], which enables clients to secretly transmit censored videos over a Variable Bit Rate (VBR) stream, and CovertCast[8] that transmits the content of blocked websites via modulated images of live-streaming feeds.

This paper follows this line of work and aims at circumventing some significant limitations of previous approaches. In fact, we argue that an ideal system should combine the following features: i) support the transfer of arbitrary data; ii) support flexible communication patterns (in order to allow the execution of a wide range protocols such as HTTP, FTP, SMTP, etc); and iii) be robust against passive attacks (such as traffic analysis) and active attacks (such as packet dropping). However, previous systems can address some but not all these requirements simultaneously. FreeWave can transfer arbitrary data but is prone to several passive and active attacks, Facet only supports the transmission of video, and CovertCast only supports unidirectional multicast.

The goal of our work is to leverage the basic principle of video morphing and extend it in order to address the three requirements listed above. Although this task may seem conceptually simple, there are several technical challenges involved. By embedding arbitrary data on a carrier videocall, the properties of the resulting stream change, which may render the stream observable to the censor. Moreover, for efficient frame transmission, video-conferencing software relies on lossy video compression algorithms which will introduce decoding errors. We address these challenges by using an encoding technique that is robust to noise and that can be dynamically tuned to match to the underlying network conditions, such that it maximizes the achievable throughput without compromising unobservability.

In this paper we present DeltaShaper, a system that enables the encoding of arbitrary data frames in an encrypted video stream, by effectively embedding payload data into carrier streams (the current prototype is based on Skype). DeltaShaper is designed with the objective of establishing a data link layer between a given video-call end hosts, enabling the forwarding of network layer packets between them. This allows for the use of several higher level protocols such as Telnet or HTTP while delegating the responsibility for the reliability of the connection to TCP/IP. DeltaShaper introduces several techniques that allow for the preservation of unobservability of covert streams. We propose and evaluate different alternatives to encode information in the video-stream, in order to maximize the available throughput while preserving the characteristics of the unmodified stream. We have implemented a prototype of our system that offers a data-link interface and that can support any protocol that runs over TCP/IP. Our results show that it is possible to achieve a throughput of 0.4 kB/s with no significant impact on the stream, which allows to run standard applications such as FTP, SMTP, or a Web client.

II. Related Work

Numerous practical solutions have been proposed over the last years to address the problem of Internet censorship. A common strategy for censorship circumvention is to leverage proxy-based traffic re-routing, oftentimes combined with digital steganography techniques. Notably, decoy routing systems [9] are based upon routers deployed within ISPs which are able to recognize steganographic marks hidden from the censor and divert the traffic to the client's desired location. However, it is often difficult to hide the proxies from the censor and, once proxies are detected, they can be blocked in a similar manner as the original source.

An improved class of systems helping in censorship circumvention aim to obfuscate covert traffic so that it cannot be linked to the underlying application layer protocol. Such technique is termed traffic morphing[10]. For instance, SkypeMorph [11] mimics the statistical properties of Skype video calls. Unfortunately, it is hard to mimic all aspects of a protocol, including responses to exceptions such as malformed packets or message losses, which makes the approach vulnerable to active attacks. Marionette [12] attempts to solve this issue by enabling the control of several aspects of protocol imitation through the composition of probabilistic automata. Still, the mimicking of proprietary protocols may require continuous reverse engineering efforts in order to keep up with protocol updates.

Some censorship circumvention systems leverage a tunnel-based staged communication approach, where an oblivious server relays the communication between the client and server of the system. In SWEET [13], covert traffic is relayed through encrypted or steganography-protected email messages that are temporarily staged on standard mail servers. CloudTransport [14] adopts a similar principle, but uses public cloud storage services for covert message forwarding. The *meek* system [15] leverages domain fronting to tunnel traffic over HTTPS connections to allowed hosts, while establishing a covert connection to a prohibited host. Castle [16] and Rook [17] provide an alternative approach to exchange covert messages over a publicly available service, namely Real-Time Strategy (RTS) games. Most of these systems, however are vulnerable to various attacks, such as denial of service or traffic analysis techniques.

A different tunneling approach leverages existing multimedia streaming protocols to enable communicating between two parties engaging in censorship circumvention. FreeWave [6] leverages VoIP connections to tunnel Internet traffic, allowing for uncensored web browsing. However, this system is vulnerable to passive attacks. In particular, the detection of FreeWave streams is based on the observation that the packet length distribution of the generated network traffic containing the covert message is nothing similar to that of a recognizable language, expected to be found in an actual conversation. Furthermore, a censor can thwart FreeWave when it is used over VBR codecs. This is achieved by launching active attacks to prevent FreeWave's modem from synchronizing, rendering the covert channel inviable.

When compared with FreeWave, Facet [7] provides a greater resilience to active attacks. This system leverages video conferencing connections such as Skype in order to tunnel censored videos through regular videocalls. Facet employs video morphing, a technique developed to ensure that the network packets generated by the video conferencing software do not directly reflect the characteristics of the censored video, but approximate those of regular video calls instead. To this end, Facet embeds the censored video in a portion of each frame, filling the remaining space with a chat 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. A censor who tries to completely disrupt Facet from functioning will block a large percentage of legitimate video conferencing calls. However, Facet is only able to serve video content which limits the applicability of the system to other types of communication.

CovertCast [8] leverages live-streaming feeds to transmit the content of blocked websites by modulating the respective data into images. These images are aggregated in order to be transmitted through live video feeds, for instance, resorting to video-streaming websites such as Youtube. In turn, the client component demodulates images served through the

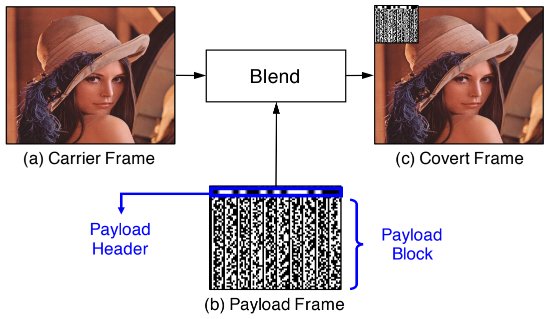


Figure 1. Blending payload into carrier frame.

live stream, extracting and saving the blocked web content. Although CovertCast data modulation is resilient against traffic analysis on such video-streaming platforms, it is only able to provide a satisfactory throughput when implementing a one-way communication channel. This means that the system is unable to support interactive communication and can only be used to download censored content. Furthermore, it requires operators of CovertCast servers to set up multiple live-feeds to serve different content.

III. 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 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.

Furthermore, the adversary has the power to perform deep packet inspection but is computationally bounded, and cannot break the underlying cryptographic primitives used to cypher the packet's 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 directly observing the content of the stream. It may however perform statistically 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 streams and to rank the similarity among different flows.

IV. Design

Our goal is to embed a (bi-directional) covert data channel in a regular Skype stream in a way that it cannot be tagged as disallowed by the adversary. For establishing the covert data channel, DeltaShaper depends on a upstream and downstream pipeline. 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 device. 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 server application. 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.

A. *Data Encoding and Decoding*

Given that a video stream is a sequence of frames, that each frame is composed by a set of pixels, and that each pixel can be defined by RGB components, one needs to find the best way to encode the data bits in the available pixels. According to the XWD format specification that is used to store screen dumps created by the X Window System, an RGB encoded pixel takes 24 bits, allowing the encoding of 16,777,216 different colors. Thus, in theory, one could encode 24 bits in each pixel. Assuming a 640x480 frame (VGA resolution), it would be possible to encode, at max, 7372800 bits in a single frame.

However, there are some reasons that prevent such an encoding scheme from functioning in practice. Firstly, video processing may modify a frame's pixels in multiple ways: change the colors of each pixel, thus altering the information being transferred; omit differences among adjacent pixels, loosing all information encoded in those pixels. Secondly, it is necessary to preserve unobservability. If all pixels of a frame are used to encode data to the maximum capacity, the resulting image complexity would be significantly different from a typical image transferred in Skype, where many pixels are similar. This would cause the traffic signature of the resulting Skype stream to be extremely different from a that of a "normal" Skype stream. To deal with such conflicting trade-offs, a data encoding scheme based on two basic ideas is proposed:

1. Blend synthetic payload video into "normal" Skype video: The covert data encoding scheme generates transmitted video frames (*covert frames*) from the combination of two components: *carrier frames* and *payload frames*. Carrier frames are taken up from a pre-recorded Skype call. Payload frames consist of synthetic video frames that encode the application data to be transmitted to the receiver. Payload and carrier frames are then blended together into covert frames and passed over to Skype. Figure 1 shows an example of how a (a) carrier frame and a (b) payload frame are blended into a (c) covert frame. The payload frame is overlapped to top-left corner of the carrier frame. The goal of carrier frames is to mimic a realistic Skype call by modulating the network stream observed by the censor thus preserving unobservability.

2. Support tunable payload frame encoding: 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 given by: $N = b_c \times n_c$, where n_c is the number of cells per frame.

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. The encoding scheme is then defined by the following parameters: size of payload frame in pixels (s_p), size of cells in pixels (s_c), color encoding in bits (b_c), and payload frame rate (r_p). A data encoder is represented by the tuple $S : \langle s_p, s_c, b_c, r_p \rangle$, for example $\langle 160 \times 120, 4 \times 4, 1, 3 \rangle$. 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).

Reducing the number of bits to represent color codes makes the system more resilient to per-pixel color change introduced by the Skype encoding pipeline whereas the increase of the cell size helps tolerate loss of information between adjacent pixels as a result of video compression. 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.

B. Preserving the Skype Traffic Signature

"Normal" Skype streams are designated as *regular streams*. A Skype stream is *regular* it if results from a legitimate video-conferencing call between Skype users carrying no covert messages. In such cases, users normally stand in front of the camera and move sparingly as they speak. In contrast, the resulting traffic pattern is expected to be quite different if Skype is used for streaming an action movie, for instance. In such cases, frames will change more frequently and extensively causing Skype's video encoding to reflect such changes. To express this intuition that regular calls tend to follow common pattern, while inevitably having some differences, a stream is considered to be *irregular* if it differs by more than a given threshold Δ from known regular streams, in which Δ is obtained by a given *similarity function* σ. Put more formally, considering s_R to be a set of known regular streams, f a *feature function* of the stream (e.g., packet length distribution), and s_C an arbitrary stream (that may contain a covert channel), s_C is said to be indistinguishable from s_R if:

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

Therefore, the frame encoding parameters P for s_C must be chosen in such a way that the resulting covert stream obeys this condition; to meet this goal, the following steps are taken:

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, the *frequency distribution of packet lengths* (f_l) was found to be effective at characterizing a given stream pattern in Skype. Similar reasoning was proven to be successful at differentiating Skype streams from Tor streams [11]. The packet length of the 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. An alternative function based on the 2-gram distribution of packet lengths has enabled to differentiate regular Skype calls from the transmission of YouTube videos over Skype [7]. In the context of DeltaShaper, this function produces similar results as f_l. Alternative feature functions based on the inter-packets' arrival time are also considered in the system's evaluation. Feature functions based on the packets' content were not considered since Skype-generated packets are encrypted.

2. Find an effective similarity function (σ): A similarity function calculates the difference between two feature functions. Given that f_l, which outputs the frequency distribution of a stream's packet length, was adopted, there is a need to find metrics that calculate the similarity between two probability distributions. Previous work has adopted the 2-sample Kolmogorov-Smirnov test [11], [17]. Informally, this test quantifies the maximum vertical distance between the empirical cumulative distribution functions of two given samples. However, the Earth Mover's Distance (EMD) [18] was found to yield better classification results and was selected as similarity function. Intuitively, EMD$(f_l(s_R), f_l(s_C))$ represents the total amount of work that is necessary to undertake in order to transform the packet length frequency distribution of a regular stream s_{Ri} into the packet length frequency distribution of s_C.

3. Define a set of reference streams (s_R): Now that f and σ have been defined, a set of known regular streams must be fixed to serve as *reference streams* around which DeltaShaper's generated covert streams will compare against. Such regular streams will correspond to streaming several carrier videos that may be used by DeltaShaper in the payload blending process (as shown in Figure 1-a), and can be obtained by recording the packet trace of real video-conferencing Skype calls, for example.

4. Compute the similarity threshold (Δ): The similarity threshold Δ aims to set a bound to the differences that one can expect to find between legitimate regular Skype calls. To determine this value, an empirical approach is undertaken. This approach consists of creating a training set of N legitimate Skype call videos and record the packet length distribution of the resulting test stream s_i, where $0 \leq i < N$. Then, the average similarity between each test stream and every other regular stream is calculated. The threshold value Δ can then be assigned in several ways, for instance: the largest difference verified between multiple regular streams; the average similarity between all regular

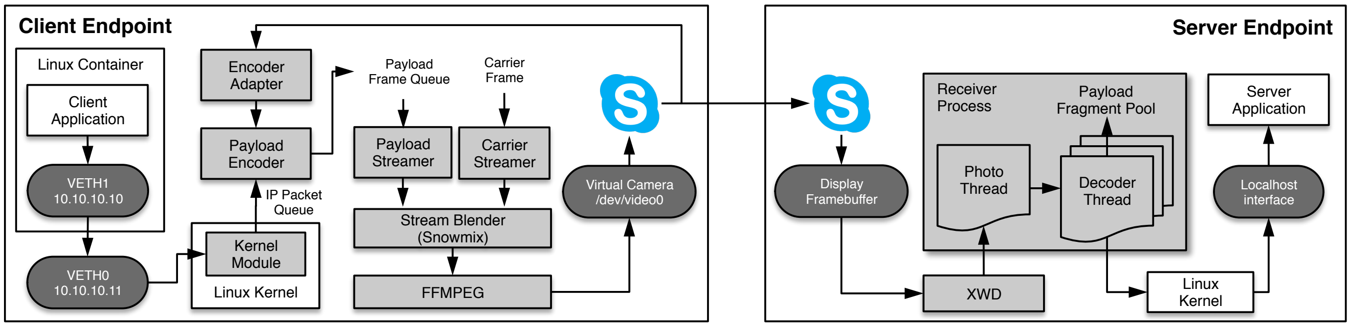


Figure 2. Architecture of the DeltaShaper prototype.

streams, plus a multiplicative factor based on the standard deviation. Furthermore, Δ must be assigned dynamically, as Skype's video encoder will adapt its bit rate to the current network conditions.

5. Obtain a valid encoding selector (P): The final step consists of determining valid sets of parameter instances (P) to the payload encoding scheme. A specific instance of P is called *encoding selector*. 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. For checking whether s_C can be identified as a regular stream, its similarity value δ can be obtained by computing the average similarity between s_C and all regular test streams in s_R. More precisely:

$$\frac{1}{N}\sum_{i=0}^{N-1} \text{EMD}(f_l(s_C[P]), f_l(s_{Ri})) = \delta,\ P \text{ is } \textit{valid} \text{ if } \delta < \Delta$$

C. Adaptation to Network Conditions

As Skype streams' distributions that result from playing a given (carrier) video greatly depend on the specific network conditions under which the transmission has occurred, a single reference stream set s_R and respective threshold Δ cannot be permanently fixed and hard-coded in DeltaShaper. This observation brings two consequences:

The reference stream set 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 a reference stream set (s_R) and threshold value (Δ) according to the specific network conditions. Furthermore, it must be taken into account that 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 responses to changes in network conditions, it may be necessary to change the frame encoding parameters in order to preserve a stream's indistinguishability. In the interest of maintaining unobservability, DeltaShaper employs a periodical calibration procedure, where both endpoints can adjust data encoding parameters. Due to lack of space, the details of the calibration procedure are omitted in this document.

V. Implementation

A DeltaShaper prototype was built for Linux. The architecture of the prototype is depicted in Figure 2. Several components implement the client and server pipelines of DeltaShaper. Some of these components were built from scratch in C++ and Python; others are based on existing tools, identified below. To build DeltaShaper, several challenges were faced at different levels: network interfacing, video processing, and Skype interfacing.

1. Network interfacing: The network interfacing between DeltaShaper and the client / server application must be performed without changes to the application. For this reason, complementary techniques were adopted when building the client and server endpoints. At the client side, the prototype takes advantage of both Linux's network namespaces and the netfilter packet filtering framework in order to build the backend of DeltaShaper's data link layer. The 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.

2. Video processing: For efficient video synthesis at the client side, DeltaShaper's payload encoder takes each IP packet, generates the corresponding payload frame, and forwards this frame to the payload streamer. The payload streamer is a user-level process that feeds payload frames into Snowmix, a live stream video mixer that is used to overlay the payload video on a dummy carrier video. Snowmix outputs the covert video which is then sent through Skype to the receiver. At the server side, the reverse decoding operation is implemented by a receiver process. Internally, the receiver process collects covert video frames from Skype, extracting the payload block out of the frames, and sends the resulting IP packets to the Linux kernel.

3. Skype interfacing: Lastly, it is necessary to interface with Skype without modifying the Skype client software. For this purpose, Snowmix's output is routed to a virtual camera device which acts as Skype's input source. This video routing operation is supported directly by Snowmix when coupled with the FFMPEG video encoding tool. At the receiver's endpoint, DeltaShaper captures the received images that are rendered by a Skype client on a virtual display. DeltaShaper relies on a frame thread of the receiver process to periodically launch the XWD tool to obtain a screenshot of the virtual display. The frame thread is calibrated so that the polling frequency is higher than the payload transmission rate. This condition ensures that no payload frames are lost.

A. *Message Format*

To support error correction and packet fragmentation, a simple message format protocol was specified. Essentially, this protocol maps messages between the high-level IP layer (IP packets) and the low-level frame layer supported by DeltaShaper (payload blocks). Due to lack of space, this document omits a full description of the messages' format.

B. *Error Recovery*

Due to the effects of the video compression algorithm employed by Skype, the recovered payload block may include bit errors. Thus, a general payload block layout was designed in order to support configurable error correction codes (ECC). In the current prototype, DeltaShaper adopts the Reed-Solomon [19] ECC. It employs a commonly used code denoted as $(n, k) = (255, 223)$, where n corresponds to 255 bytes of a data symbol, out of which $k = 223$ bytes consist of application data and the remaining 32 bytes encode parity bit symbols. This code can correct up to 16 symbol errors per symbol block.

VI. Evaluation

To evaluate our system, we focus on the following main goals: test the effectiveness of EMD and Δ threshold metrics in characterizing Skype streams (Section VI-B), assess the ability of DeltaShaper to generate unobservable covert Skype channels based on such metrics (Section VI-C) and measure the performance of DeltaShaper channels while preserving unobservability (Section VI-D). We also explore the impact of network perturbations in a censor's ability to distinguish between regular and irregular streams (Section VI-E). Then, a study over the use of alternative traffic features and similarity functions to classify Skype streams is presented (Section VI-F). Lastly, we evaluate the end-user experience of DeltaShaper for several use cases (Section VI-G).

A. *Experimental Settings*

The system's experimental evaluation was performed on two 32bit Ubuntu 14.04.4 LTS virtual machines (VMs) with 8GB RAM and 4 virtual Intel Core 2 Duo T7700 2.40GHz CPUs. Each VM runs an instance of Skype v4.3.0.37 and DeltaShaper acting, respectively, as caller and callee of video-conferencing calls. The native *netem* Linux network emulation functionality was used to enforce limitations over the network conditions between the VMs.

In order to characterize regular video-conferencing streams, 30 videos representative of actual videocalls have been selected. These videos are used as training set for regular streams. Such videos generally exhibit low movement as

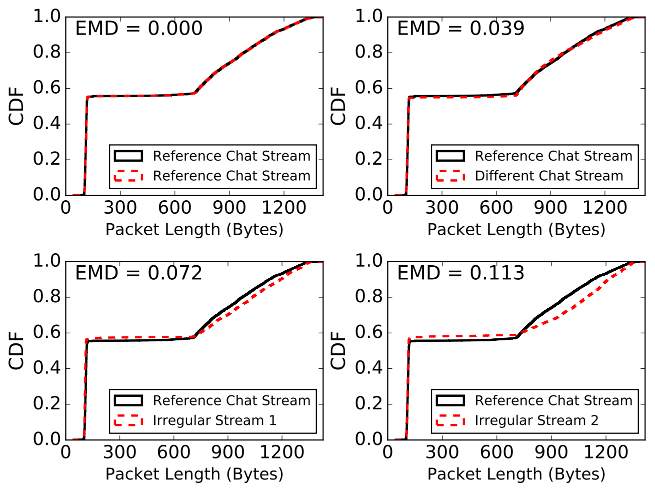


Figure 3. Packet length CDF of sample streams.

users typically sit in front of a computer, moving sparingly as they speak. These videos have not been edited and are free of watermarks or other visual artifacts. The training set for irregular streams consists of 30 YouTube videos, involving more dynamic movement, where both rapidly changing scenes and artifacts introduced by video editing software are common. The duration of each video sample is 30 seconds. Samples are captured after 10 seconds of the initial call establishment. For these experiments, the calls' audio packets carry data representing silence.

B. *Characterization of Skype Streams*

Firstly, it is studied whether Skype calls exhibit measurable patterns that allow the differentiation between regular and irregular calls. This question is of utmost importance since such patterns may be used by a censor to detect suspicious videocalls (i.e., irregular ones) and further block them. For conducting the experiment with reduced interference, the four test streams have been transmitted in a one-way Skype video-conferencing call. The data in Figure 3 indicates that such patterns do exist. It shows the cumulative distribution function (CDF) of packet lengths for each of the four test Skype streams, respectively represented in a different plot: (a) the stream of an actual videocall which is taken as 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. Each plot represents the distribution of the test stream along with the packet length distribution of the reference stream (the black curve), exhibiting their similarity by calculating the respective EMD value. It can be seen that the 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. This is congruent with VBR encoding procedures, where more dynamic scenes typically lead to the generation of larger network packets due

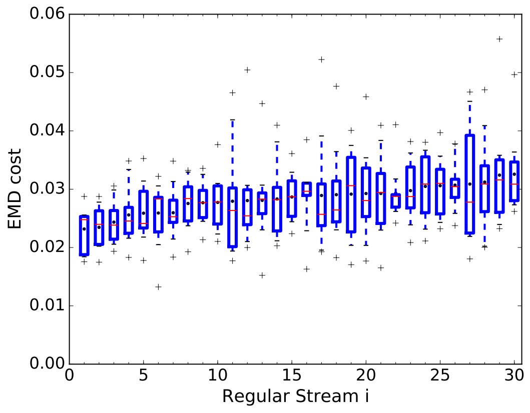
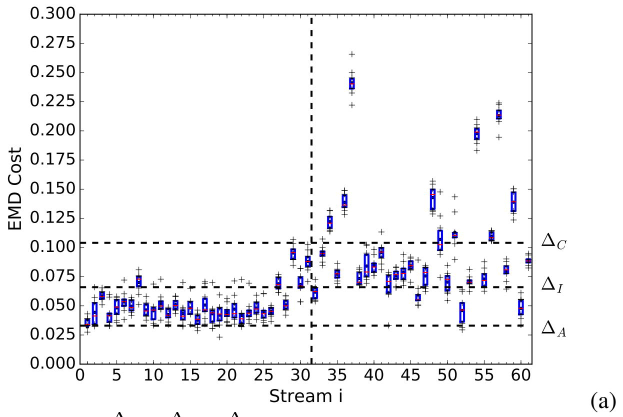


Figure 4. EMD cost of multiple videocall streams.

to the higher amount of inter- and intra-frame differences. In particular, this comparison shows that these differences are more acute in dynamic videos than in rather static videocalls.

To better understand whether these traffic patterns are stable, and therefore can be reliably used for characterizing regular Skype streams, a study must be conducted to assess whether there are significant differences in the packet length distribution when streaming the same videocall multiple times over Skype. Each regular video call sample in the dataset is replayed 10 times. The EMD of each resulting stream is calculated, taking as baseline the average distribution of all 10 runs. These experiments were performed through one-way video-conferencing calls, whereas network conditions were not artificially constrained. Figure 4 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. Moreover, the average EMD value tends to be very similar among different videos, varying between 0.025 and 0.031. It is possible to conclude that, under the same network conditions, regular Skype streams display a high degree of similarity.

The 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, a regular stream is used 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. Similarly to the previous experiments, the traffic samples were obtained from one-way video-conferencing calls, under unconstrained network conditions. Figure 5 (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. It is possible to immediately observe a pattern in which regular streams tend to result in a constantly low



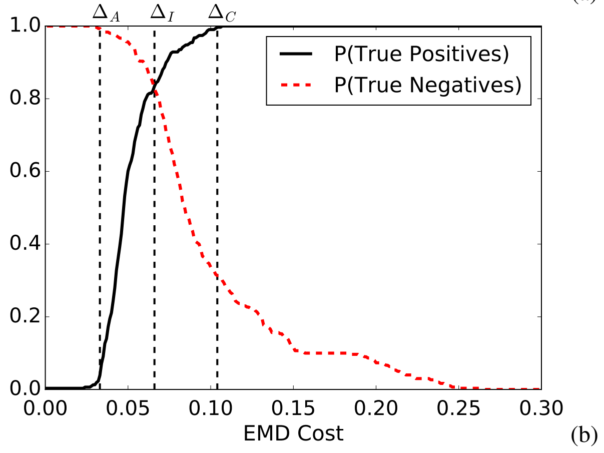


Figure 5. EMD based on reference stream.

EMD cost (below 0.1), whereas irregular streams produce a significantly more scattered pattern varying EMD cost approximately from as low as 0.025 to as high as 0.25, i.e., by a order of magnitude.

The question is then whether it is possible to define an EMD cost Δ threshold that can be used as stream classifier such that a stream s is considered regular if $\text{EMD}(s_R, s) < \Delta$ or irregular otherwise (see Section IV-B). For this particular experiment, Δ can be set to any value between 0 and the maximum observed EMD cost (0.275). To evaluate the effectiveness of this classifier, Figure 5 (b) shows the probability of true positives (sensitivity) and true negatives (specificity) as Δ varies (in the x-axis). It is possible to see that, as Δ increases the 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 (Δ_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 Δ. Eventually, when Δ reaches 10.1 (Δ_C), the classifier is able to correctly identify all regular streams.

Based on how the Δ threshold is set, several classification policies are possible. Suppose that a censor wishes to apply an *aggressive classification policy* by blocking all streams that are truly irregular. In this case, Δ must be set to Δ_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 classification policy*), Δ must be set to Δ_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 maximizes the classifier's accuracy is to take the cutoff point where the probability of true negatives equals the probability of true positives. For the classifier, this point corresponds to EMD cost 0.066 (Δ_I) which means that setting Δ to this value results in 83% accuracy in classifying a stream. Thus, it is possible to define a Δ threshold which allows for identification of regular streams with high probability. This is crucial as DeltaShaper explores this property to hide within regular streams.

C. Unobservability of DeltaShaper Channels

In order to produce covert Skype streams that can be deemed indistinguishable from a regular stream, DeltaShaper must be set up such that the EMD cost of resulting stream remains below the Δ threshold. Since the properties of a resulting stream depend on the encoding parameters provided to DeltaShaper, it is fundamental to study which range of encoding parameters can be reliably used to produce unobservable covert streams.

DeltaShaper can be configured with four parameters: payload area size, cell size, bit number, and frame rate. Since covering the entire configuration space requires covering a large number of configurations, this work focuses on a subset of parameters that result in valid configurations, but not necessarily optimal in terms of the maximum throughput that can be achieved. In this study, the reference stream that was selected in the previous section, as well as the Δ threshold values that were found for the same reference stream, are used. For this test, several payload frames are synthesized and combined with the carrier video that originally generated the reference stream. These "offline" samples are then transmitted over Skype, allowing the gathering of samples from the resulting network streams.

The analysis of the combined effects of the payload area size and the cell size starts by fixing the bit number in 1 bit/cell and the frame rate in 1 FPS. Figure 6 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. The plot is annotated with Δ threshold values for the three policies discussed in the previous section: aggressive (Δ_A), conservative (Δ_C), and intermediate (Δ_I). For example, it can be seen that, for an intermediate policy, there are five

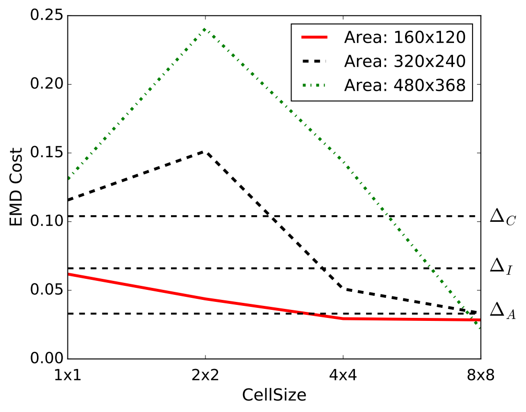


Figure 6. EMD cost changing area and cell size.

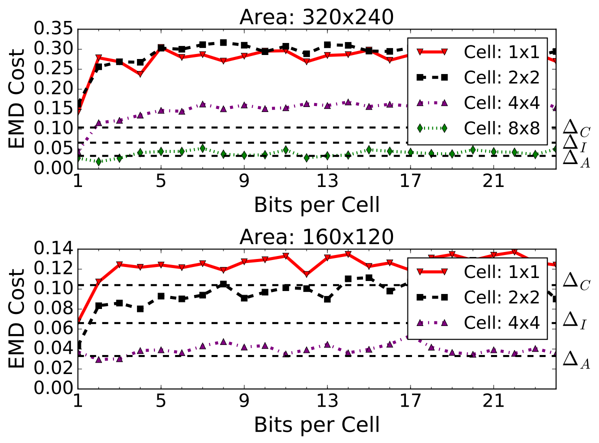


Figure 7. EMD cost varying the bits per cell.

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. The payload area size 480x368 was consistently found to generate streams identified as irregular by the DeltaShaper classifier, when encoding more than 1 bit per cell.

Results show that as the cell size increases, the EMD cost tends to decrease. This is because larger areas of the frames will be colored with the same color thereby improving the efficiency of the video compression algorithm.

A study of how unobservability changes, as a function of the number of bits per cell, was conducted for the area/cell size configurations found to be valid. Figure 7 shows the results, which cover the domain of data bit numbers, i.e., between 1 and 24 bits. 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 Δ threshold value for intermediate blocking policy (Δ_I), namely (160x120, 4x4) and (320x240,8x8). This means that both these encoding configurations are good candidates to generate unobservable covert streams.

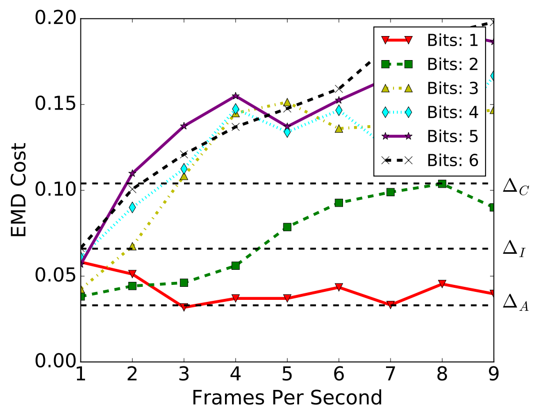


Figure 8. EMD cost varying the frame rate.

Lastly, it is conducted a study over how does the frame rate affects unobservability. To that end, a fixed payload area size 320x240 and cell size 8x8, are defined. Then, the EMD cost for a cell bit encoding range, varying between 1 and 6 bits per cell, is measured. Although the (160x120, 4x4) configuration encodes the same amount of bits per cell, these result in higher error rates in the decoding process. Figure 8 shows how the EMD cost varies as the frame rate is increased. The results show that increasing the frame rate will quickly result in EMD cost above Δ. A notable exception is the the data encoding scheme of 1 bit per cell, which remains below Δ for all tested frame rates. Encoding schemes with higher bit numbers can only tolerate the minimal frame rate value (1 FPS).

D. Performance of the Covert Channel

Although it is possible to generate covert streams from numerous encoding configurations, DeltaShaper can only safely adopt those that result in unobservable streams. Furthermore, performance can also be affected by decoding errors at the receiver when interpreting the cell color of payload frames. In fact, as the number of bits encoded per cell increases, the video compression algorithm tends to introduce changes in the less significant bits of the color of each pixel, therefore introducing more decoding errors.

Taking into account both restrictions in terms of stream unobservability and decoding errors, we have identified a candidate encoding configuration for DeltaShaper, which consists of: 320x240 area size, 8x8 cell size, 6 bits per cell, at 1 frame per second. Under this scheme, DeltaShaper is able to achieve 0.32 and 0.39 KBps, respectively with and without the use of error-correcting codes.

E. Impact of Network Perturbations in Classification

In the previous sections, for simplicity of exposition, we have used a single reference stream for computing Δ thresholds. However, in a real setting, a censor would be able to compare the target stream with a dataset of regular streams, thus taking into account the underlying differences among regular streams. In the further results, each regular / irregular stream was compared against all streams comprising the regular streams dataset (by taking the average of the different Δs).

A censor may introduce controlled perturbations in the network in an attempt to establish improved Δ thresholds that may enhance classification performance and ultimately unveil DeltaShaper's covert channels. For assessing whether classification performance is enhanced by artificially constraining the network, we have attempted the classification of streams upon several network limitations, namely: bandwidth throttling (unrestricted, 500kbps and 300kbps); loss of random packets (5%, 10% and 20%); and the introduction of jitter between packet delivery (20ms base delay plus 10ms, 20ms and 50ms of jitter, respectively).

The experiment's outcome shows that artificial network impairment contributes for a decrease in classification performance, ranging from 3% to 13%, when distinguishing streams with the help of the analysis of packet sizes and EMD. This suggests that a censor is able to establish more accurate Δ thresholds in unrestricted network conditions.

F. Exploring Traffic Features and Similarity Functions

We have assumed up until this point that the adversary uses the analysis of packet lengths and EMD to classify Skype streams. However, in a real setting, a censor could use different traffic features and similarity functions in an attempt to achieve higher accuracy in detecting streams with a covert channel. We have performed additional tests, by using alternative traffic features, namely: bi-gram distribution of packet sizes, inter-packet time, bi-gram distribution of inter-packet times. We have also attempted the classification of streams with the 2-sample Kolmogorov-Smirnov (KS) test, a popular similarity function used in the related literature.

The outcome of such experiments suggests that the analysis of packet lengths, when used in tandem with EMD, offers a sound and lightweight approach to classify Skype streams. In a general way, the classification accuracy offered by KS was inferior of that offered by EMD. Notably, for unrestricted network conditions, KS accuracy stood 6% short of EMD. A broader study over the use of different traffic features and similarity functions for classification can be found in the dissertation.

G. 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 applications that are not bandwidth hungry and are latency tolerant. To confirm this hypothesis, DeltaShaper has been tested with seven use cases: fetching a 4KB web page from the receiver (Case A), downloading a 4KB file from an FTP server running on the receiver (Case B), tunneling a small email (two small sentences) through an SMTP server running on the receiver (Case C), issuing an SSH session to the receiver and performing the "ls" command (Case D), issuing a telnet session to the receiver and performing the "ls" command

Use Case	W/ DS	W/o DS	Overhead
A. Wget	1m 9s 830ms	7ms	9,975.7×
B. FTP	2m 45s	8s 528ms	19×
C. SMTP	2m 42s	37s 913ms	4.3×
D. SSH	1m 51s 493ms	6s 485ms	17.2×
E. Telnet	1m 17s 471ms	7s 670ms	10.1×
F. Netcat Chat	1s 147ms	11ms	133×
G. SSH Tunnel	3m 46s 55ms	21s 940ms	10.3×

Table I
EXECUTION TIME FOR DELTASHAPER USE CASES.

(Case E), sending a message directed at a netcat server running on the receiver, mimicking a text chat (Case F), tunneling an SSH session to a remote SSH server through the receiver, and performing the "ls" command (Case G). In use cases A-F, the client communicates only with the receiver over a DeltaShaper channel. In case G, the receiver acts as a relay by 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 I provides a summary of the execution time for each use case when performed with and without DeltaShaper, i.e., using overt communication channels between client and receiver. As depicted, 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 and high latency that DeltaShaper can currently deliver. Nevertheless, in spite of the high delay experienced by users, all tested use cases are fully functional.

VII. CONCLUSIONS

This work describes DeltaShaper, a novel Internet censorship circumvention system which leverages the video channel of popular video-conferencing applications to tunnel covert data. The system offers a data-link interface, supporting any protocol running over TCP/IP, offering users a wide array of possibilities to transfer information in an unobservable way. An extensive evaluation of the prototype has been conducted so as to define which combination of encoding parameters can defend against traffic analysis.

ACKNOWLEDGMENTS

This work benefited from the fruitful comments of Professor Fernando Pereira, regarding the H.264 codec inner workings, as well as Professor Bruno Martins' suggestions on evaluation metrics. This work was performed at INESC-ID and was partially funded by FCT and PIDDAC as part of the UID/CEC/50021/2013 project.

REFERENCES

[1] S. Aryan, H. Aryan, and J. A. Halderman, "Internet censorship in Iran : A first look," in *Proc. of FOCI*, Washington, DC, USA, 2013.

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

[3] David Fifield, "A Child's Garden Of Pluggable Transports," https://trac.torproject.org/projects/tor/wiki/doc/AChildsGardenOfPluggableTransports, 2014.

[4] A. Houmansadr, C. Brubaker, and V. Shmatikov, "The parrot is dead: Observing unobservable network communications," in *Proc. of IEEE S&P*, San Francisco, CA, USA, 2013.

[5] A. Dainotti, C. Squarcella, E. Aben, K. C. Claffy, M. Chiesa, M. Russo, and A. Pescapé, "Analysis of country-wide Internet outages caused by censorship," in *Proc. of IMC*, Berlin, Germany, 2011.

[6] 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 *Proc. of NDSS*, 2013.

[7] S. Li, M. Schliep, and N. Hopper, "Facet: Streaming over videoconferencing for censorship circumvention," in *Proc. of WPES*, Scottsdale, Arizona, USA, 2014.

[8] R. McPherson, A. Houmansadr, and V. Shmatikov, "CovertCast: Using Live Streaming to Evade Internet Censorship," in *Proc. of PETS*, 2016.

[9] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman, "Telex: Anticensorship in the network infrastructure," in *Proc. of USENIX Security*, San Francisco, CA, USA, 2011.

[10] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic morphing: An efficient defense against statistical traffic analysis," in *Proc. of NDSS*, San Diego, CA, USA, 2009.

[11] H. Moghaddam, B. Li, M. Derakhshani, and I. Goldberg, "Skypemorph: Protocol obfuscation for Tor bridges," in *Proc. of CCS*, Raleigh, North Carolina, USA, 2012.

[12] K. P. Dyer, S. E. Coull, and T. Shrimpton, "Marionette: A programmable network-traffic obfuscation system," in *Proc. of USENIX Security*, Washington, D.C., USA, 2015.

[13] W. Zhou, A. Houmansadr, M. Caesar, and N. Borisov, "Sweet: Serving the web by exploiting email tunnels," in *Proc. of HotPETS*, 2013.

[14] 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.

[15] D. Fifield, C. Lan, R. Hynes, P. Wegmann, and V. Paxson, "Blocking-resistant communication through domain fronting," in *Proc. of PETS*, Philadelphia, PA, USA, 2015.

[16] B. Hahn, R. Nithyanand, P. Gill, and R. Johnson, "Games without frontiers: Investigating video games as a covert channel," in *Proc. of IEEE Euro S&P*, 2016.

[17] P. Vines and T. Kohno, "Rook: Using Video Games As a Low-Bandwidth Censorship Resistant Communication Platform," in *Proc. of WPES*, 2015.

[18] 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.

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