Stegozoa: Enhancing WebRTC Covert Channels with Video Steganography for Internet Censorship Circumvention

Gabriel Figueira
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
gabriel.figueira@tecnico.ulisboa.pt

Diogo Barradas
University of Waterloo
diogo.barradas@uwaterloo.ca

Nuno Santos
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
nuno.m.santos@tecnico.ulisboa.pt

ABSTRACT

Several totalitarian states around the world deploy sophisticated censorship apparatuses to prevent citizens from freely accessing the Internet. To counter these restrictions, some censorship-circumvention tools establish covert channels through the media streams of popular conferencing applications. A recent tool named Protozoa allows for establishing high-performing, peer-to-peer covert channels over WebRTC media streams. However, Protozoa is vulnerable to potential man-in-the-middle attacks. This may occur in cases where WebRTC applications rely on WebRTC gateways to mediate users' connections. In such cases, an adversary that controls the WebRTC gateway can inspect the content of the media streams and trivially detect the transmission of covert payload.

This work proposes Stegozoa, a new censorship-circumvention tool that aims to foil the ability of adversaries in control of WebRTC gateways to detect covert data transmissions. Specifically, Stegozoa steganographically embeds covert data into the WebRTC video signal, preventing the detection of the covert payload through direct video content inspection. To this end, Stegozoa leverages state-of-the-art steganography techniques, applying them deep within WebRTC's video coding pipeline and fine-tuning them to efficiently use the available covert channel capacity while ensuring undetectability. We have fully implemented Stegozoa based on an instrumented Chromium codebase. Our evaluation reveals that Stegozoa can create secure WebRTC covert channels that are highly resistant to steganalysis and traffic analysis attacks. Despite the expected reduction in performance that was traded for stronger security, Stegozoa can deliver a reasonable throughput, allowing its users to run low-bandwidth message exchanging tasks.

CCS CONCEPTS

• **Security and privacy → Network security**; • **Social and professional topics → Technology and censorship**.

KEYWORDS

Censorship circumvention; Steganography; WebRTC

ASIA CCS '22, May 30–June 3, 2022, Nagasaki, Japan.

ACM ISBN 978-1-4503-9140-5/22/05…$15.00
https://doi.org/10.1145/3488932.3517419

ACM Reference Format:
Gabriel Figueira, Diogo Barradas, and Nuno Santos. 2022. Stegozoa: Enhancing WebRTC Covert Channels with Video Steganography for Internet Censorship Circumvention. In *Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security (ASIA CCS '22), May 30–June 3, 2022, Nagasaki, Japan.* ACM, New York, NY, USA, 14 pages. https://doi.org/https://doi.org/10.1145/3488932.3517419

1 INTRODUCTION

As of today, totalitarian states are known to deploy large-scale surveillance and censorship mechanisms in order to deter citizens from accessing or publishing selected kinds of information on the Internet [16]. As an example, countries like China [34], Turkey [1], or Russia [48], are known to deploy a sophisticated censorship apparatus with the ability to block the access to certain Internet destinations [45] and protocols [19], throttle the access to given websites [54], or to monitor social media platforms [35].

During the last few years, important efforts have been devoted to tackling the tight Internet control exerted by repressive state-level adversaries [33]. One of the latest advances in this space takes advantage of the unwillingness of censors to ban video conferencing applications to devise covert channels using as carrier signal the media streams generated by such applications [8, 10, 37]. Notably, Protozoa [10] is a new censorship-resistant tool that leverages WebRTC video calls to create high-speed covert channels between call endpoints. To this end, each Protozoa's endpoint instruments the WebRTC video pipeline of open-source browsers to fully replace the contents of compressed video frames by covert data payload. The resulting frames are then encrypted and transmitted over the network to the remote videocall peer. Following this process, the video streams generated by Protozoa preserve the traffic characteristics (statistical indicators like packet sizes and inter-arrival time distributions) of legitimate WebRTC streams. This severely undermines the effectiveness of modern traffic analysis techniques [9] to detect covert channels established over multimedia flows.

However, Protozoa operates under the strong assumption that the participants in a videocall exchange media data in a peer-to-peer fashion. Given that the peer-to-peer connections carrying the video streams are encrypted end-to-end, not even a state-level adversary with unrestricted access to the network infrastructure will be able to observe the raw video content of the WebRTC streams. Unfortunately, a myriad of WebRTC applications challenge this assumption by relying on *WebRTC gateways*: specialized servers that mediate the transmission of media streams between the participants engaged in a videocall for scalability and performance purposes [51]. The point in fact is that, other than blindly relaying media packets

between participants (as in the case of TURN servers [39]), WebRTC gateways have the ability to decrypt, validate, and perform arbitrary operations on incoming media streams [3]. Evidence suggests that popular WebRTC services, like Discord, do make use of WebRTC gateways to inspect audio data exchanged by users [50].

Unfortunately, this means that Protozoa users can be trivially detected if they make use of web conferencing services that leverage WebRTC gateways controlled by an adversary, e.g., when using a domestic video-conferencing infrastructure under the control of a censor. This risk exists because Protozoa encoded video frames can be clearly identified as they carry arbitrary covert data payload that fails to adhere to the specification of media codecs. While this threat has been previously identified in the literature [7], devising a tool that is able to achieve reasonable network performance while resisting both traffic analysis and the inspection of media content over WebRTC streams has remained an elusive goal.

This paper presents Stegozoa, a novel tool for circumventing Internet censorship that provides a messaging exchange service offering reasonable throughput for the surreptitious transmission of covert data over WebRTC video calls. It can withstand traffic analysis and media steganalysis attacks launched by state-level adversaries in control of WebRTC gateways. To meet this goal, Stegozoa creates covert channels by augmenting the WebRTC encoding pipeline with the ability to steganographically embed covert data into the compressed representation of WebRTC video frames. Specifically, Stegozoa leverages the combination of syndrome-trellis coding [24], a state-of-the art adaptive steganographic technique, with covert data encoding in the least significant bits of the quantized discrete cosine transform (QDCT) coefficients of residual video frames [53], i.e., a frame's representation in the frequency domain.

When building Stegozoa, we tackled two main technical challenges. The first challenge relates to finding a proper configuration for the chosen syndrome-trellis code, such that Stegozoa can achieve a reasonable throughput and strong resistance against steganalysis. The main difficulty of fine-tuning these parameters is that this task is tightly dependent on several implementation artifacts (e.g., the browser's VP8 encoding pipeline) and thus requires extensive empirical testing under different experimental settings.

The second challenge is to implement the above steganographic operations in real-time communication settings. While image/video steganography techniques do not typically operate under stringent time restrictions, steganographically embedding covert data within real-time constraints is fundamental in Stegozoa to prevent packet transmission delays that could cause deviations in traffic patterns and, consequently, make Stegozoa vulnerable to traffic analysis attacks. To tackle this challenge, we introduce several low-level optimizations in the WebRTC video encoder pipeline.

We evaluated Stegozoa's performance and resistance against detection under different network conditions and using different WebRTC carrier applications (Jitsi and Whereby). The results of our experimental evaluation revealed that a careful parameterization of Stegozoa's steganographic embedding mechanisms allows Stegozoa to resist traffic analysis and steganalysis attacks, while achieving enough throughput for performing realistic use case scenarios like consuming Twitter-style feeds and Wikipedia articles.

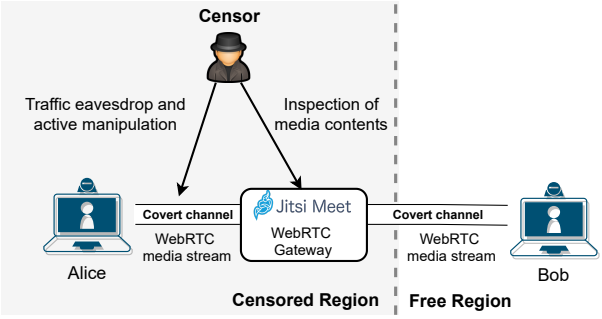


Figure 1: System model showing a multimedia covert streaming channel established over a WebRTC gateway.

2 THREAT MODEL

The threat model of Stegozoa is depicted in Figure 1. It represents two users, Alice and Bob, that intend to communicate some sensitive information away from the prying eyes of a state-level adversary, e.g., a censor. In this scenario, we assume the adversary to be interested in pinpointing and blocking WebRTC streams that may be used for establishing a covert channel between Alice, located within a censored region, and Bob, located in a free Internet region.

To detect a covert channel established over WebRTC media streams, the adversary can make use of a combination of network-level and application-level inspection techniques. First, the adversary can perform ML-based statistical traffic analysis on the traffic patterns generated by encrypted WebRTC media streams in an attempt to distinguish between legitimate WebRTC streams and streams that carry a covert channel [9]. To this end, the adversary can observe, store, actively manipulate, and analyze all the network flows exchanged between the parties that are engaged in communication. Second, the adversary can visually inspect the media payload exchanged between the call's participants, validate the composition of media data structures, and conduct video steganalysis. For enabling such capabilities, the adversary is assumed to have access to the unencrypted WebRTC media stream exchanged by Alice and Bob, which is made possible by controlling the WebRTC gateway that mediates the call participants' connection. Access to the WebRTC gateway can be granted through a variety of means. Specifically, the censor can a) run the WebRTC application and manage the WebRTC gateway, b) collude with a given WebRTC service that provides unencrypted media data to the censor, or c) exploit software vulnerabilities on WebRTC services to compromise WebRTC gateways controlled by third-party WebRTC services [43]. Note that, in the functional sense, controlling a WebRTC gateway also enables the adversary to perform deep packet inspection.

We assume that the adversary is unable to control the software installed on end-user computers and does not have the power to deploy rogue monitoring software on users' machines. Thus, the communication endpoints where Stegozoa is executed are deemed as trusted. In addition, we assume that the censor is not willing to block or downgrade the quality of WebRTC video calls unless there is strong evidence those are being used to convey some covert channel. Doing so indiscriminately may lead to collateral damage stemming from the disruption of important services to the social and economical tissue of the censored region [22].

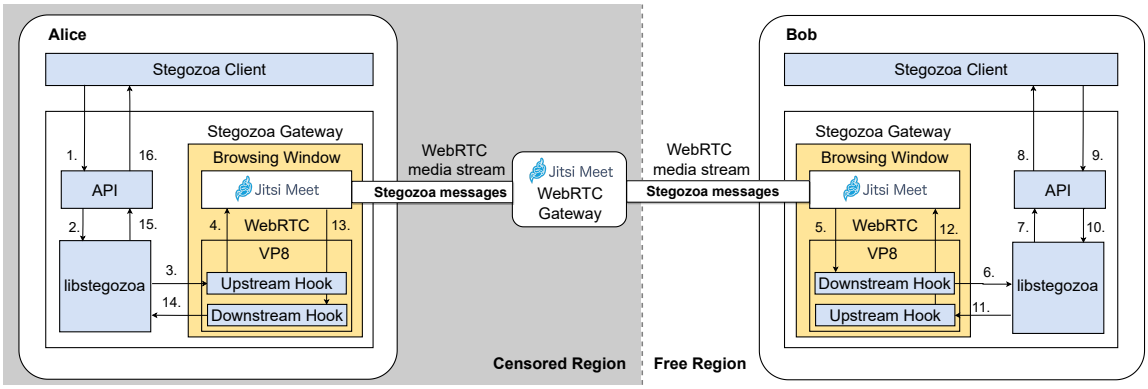


Figure 2: Architecture of Stegozoa: The components of our system are highlighted in blue.

3 STEGOZOA

This section presents Stegozoa, a novel censorship circumvention system that tunnels covert data over steganographically-marked WebRTC video streams. During our exposition, we describe Stegozoa's architecture, illustrate the covert channel establishment process, and detail how Stegozoa overcomes three main technical challenges tied to the i) creation of an efficient covert channel over real-time video streams, ii) efficient utilization of the covert channel's bandwidth, and iii) preservation of traffic analysis resistance.

3.1 Architecture

Figure 2 depicts the architecture of Stegozoa. The system itself consists of a gateway service composed of three main elements: an API, the `libstegozoa` library, and a set of media interception hooks located inside a modified Web browser. The coordinated interaction of these elements, acting at different abstraction layers, allows Stegozoa to establish a covert channel between two users – Alice and Bob – and stealthily exchange messages through this channel. A client application running executed at each communication endpoint offers a user-friendly interface for sending and receiving messages through the API exposed by the Stegozoa gateway.

For establishing covert channels, Stegozoa instruments the media pipeline of WebRTC video conferencing applications. These applications, for instance Jitsi Meet, rely on JavaScript code that executes in users' browsers as part of the WebRTC framework for performing peer discovery, signalling, and calling establishment operations. Users' browsers are also responsible for encoding, transmitting, and decoding media data relying on WebRTC's native C++ implementation, enabling call participants to exchange media streams. In the context of Stegozoa, these media streams (video streams in particular) will be forwarded between call participants via a WebRTC gateway while acting as a carrier for steganographically-marked covert data. Next, we describe the components of the Stegozoa gateway, detailing their operation in a top-down approach.

1. Stegozoa API: The API exposed by Stegozoa embodies Stegozoa's *high-level layer*. Stegozoa makes multiple API calls available so that external applications can leverage Stegozoa as a covert communication library, effectively using its covert channels as a surreptitious transport layer. Hence, we refer to such applications, e.g., a software that allows for covert access to sensitive Wikipedia pages, as *Stegozoa clients*. Stegozoa exposes the following API calls:

- **`initialize()`**: This function initializes `libstegozoa` and Stegozoa's media interception hooks' data structures, including a message queue. The local Stegozoa endpoint can then receive messages from another endpoint. It launches a periodic participant discovery operation that tracks the presence of other participants in a video call (see Section 3.2). Each endpoint is attributed a randomly-generated ID.
- **`getPeers()`**: Returns to the caller a list of IDs of known peers in the video call. The output of this function depends on the periodic discovery operation triggered by `initialize`.
- **`send(byte[]:message, int:recipientID)`** This function receives a message of arbitrary size and a recipient participant ID. The message is sent to the intended destination, while possibly being fragmented into several packets of fixed size.
- **`receive()`**: Returns the first message on top of the message queue that is addressed to the local Stegozoa endpoint. The local endpoint drops messages addressed to other participants if a message's recipient ID fails to match the local participant ID. The operation is blocked if the message queue is empty.
- **`shutdown()`** This function tears down the Stegozoa transmission and removes associated data structures.

2. Stegozoa library (`libstegozoa`): The Stegozoa library can be considered as Stegozoa's *intermediate-level layer*, bridging Stegozoa's data encoding mechanisms within users' browsers WebRTC stack with the rich functionality offered by Stegozoa's API. `libstegozoa`'s main responsibility is that of managing *packets*, Stegozoa's fundamental communication unit, to ensure the efficient usage of the covert communication channel. Specifically, `libstegozoa` is responsible for creating packets and for managing the fragmentation and reassembly of covert messages that can potentially span across multiple packets. In addition, by interacting with Stegozoa's media interception hooks, `libstegozoa` implements the low-level logic that will assess which participants in a given video call are using Stegozoa, and for establishing sessions between them.

3. Media interception hooks: The media interception hooks compose Stegozoa's *low-level layer*. These hooks, located deep within WebRTC's native codebase (e.g., in a modified WebRTC-enabled browser like Chromium), enable Stegozoa to perform several functions that require interaction with different WebRTC resources. In particular, media interception hooks are responsible for i) accessing WebRTC call metadata and forward such data to `libstegozoa`

for additional processing, and ii) manipulating one of WebRTC's default video codec (VP8) encoding/decoding pipeline to perform the steganographic embedding/extraction of Stegozoa packets from upstream/downstream WebRTC video streams using an efficient and high-bandwidth video steganography technique. Communication between Stegozoa's low-level and intermediate-level layers is performed through named pipes.

Implementation: We developed a prototype [23] of Stegozoa by writing approximately 600 lines of Python3 code for implementing `libstegozoa`, and 1 600 lines of C code for instrumenting the native WebRTC and `libvpx` (VP8's implementation) codebase of Chromium v89.0.4389.82, a stable release from March 2021. We chose to implement our prototype for the VP8 codec since this codec was the default choice for popular WebRTC applications we experimented with. However, we note that the close similarity between the internal behaviour of VP8 and H.264 [20] does not preclude our covert data encoding scheme from being applied to this alternative codec. For scaling up our experiments during the evaluation of our prototype, we make use of the `v4l2loopback` camera emulator and `ffmpeg` video utilities for establishing video calls using pre-recorded video feeds.

3.2 Establishing Stegozoa Covert Channels

Figure 3 illustrates how two users, Alice and Bob, can establish a covert communication channel using Stegozoa to exchange some sensitive content, e.g., news articles. First, Alice accesses a given WebRTC video-conferencing service, like `meet.jit.si`, creates a chatroom (A1), and shares the chatroom's URL (A2) with Bob (B1), using some out-of-band channel, e.g., e-mail or steganographically marked cryptocurrency transactions [41]. Likewise, Alice and Bob should also exchange a shared secret – this secret will be used to bootstrap Stegozoa's steganographic encoder (see Section 3.3). Then, both Alice and Bob join the chatroom (A3 and B2) and start transmitting video content. Note that while access to the chatroom may also be protected by a password, the adversary described in Section 2 is already assumed to possess the ability to visually and programmatically inspect the media exchanged by Alice and Bob.

Once the media session is in place, Alice and Bob invoke Stegozoa's `initialize` API call (A4 and B3) to bootstrap their local Stegozoa endpoints. As part of Stegozoa's bootstrapping process, `libstegozoa` will generate and broadcast *discovery packets*, special control packets that will be steganographically marked in video frames and which will act as a participant's presence beacon. Other Stegozoa participants who listen to this beacon will then reply with their participant ID in a response packet. Then, the sender of the discovery packet will collect the observed replies and register the current Stegozoa peers' IDs. In our example, after peer discovery is finished, Bob can list his current peers' IDs by issuing the `getPeers` API call (B4). Armed with the information that Alice's ID is equal to 1, Bob can now send a message to Alice by invoking the `send` API call (B5). Finally, Alice will be able to receive Bob's message after automatically invoking Stegozoa's `receive` API call (A5).

Thus far, we have described Stegozoa's architecture and general operational workflow. In the next sections, we shed light on Stegozoa's steganographic embedding mechanism, detail how Stegozoa can make an efficient use of the steganographic covert channel,

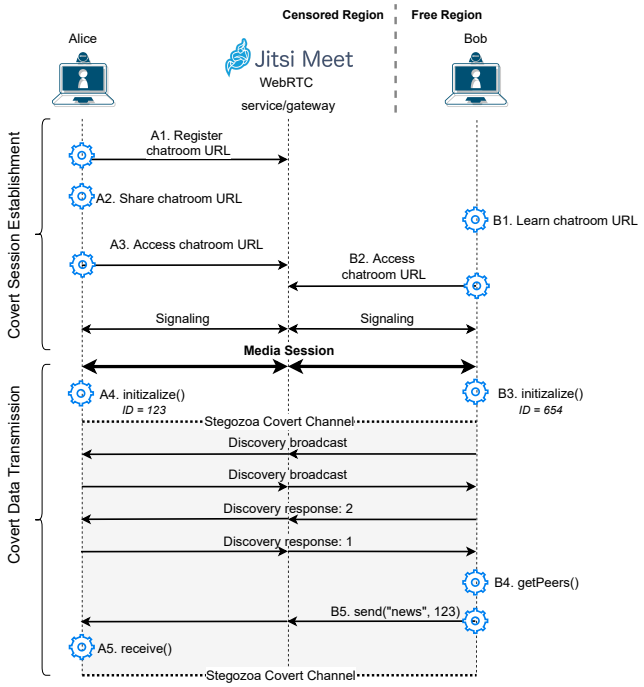


Figure 3: Stegozoa's covert channel establishment.

and describe a set of critical optimizations required for hardening Stegozoa against traffic analysis attacks.

3.3 Steganographic Channel on WebRTC Video

Finding an adequate steganography primitive that can be used for embedding covert information in real-time video streams is a critical piece required for the construction of Stegozoa. Next, we describe how WebRTC performs real-time video encoding and detail the rationale behind our choice of Stegozoa's steganography technique.

Real-time video encoding in WebRTC: For efficiently achieving high image quality, real-time video codecs attempt to reduce the need for exchanging data over the network. To this end, codecs like VP8 and H.264 (the default video codecs on WebRTC) provide sophisticated image coding operations that aim at removing redundant information between frames (inter-prediction), and within frames themselves (intra-prediction). For instance, motion compensation is an inter-prediction technique where the encoder attempts to record the translation of moving objects between adjacent frames, only needing to communicate the direction of translation and a few additional data pertaining to new elements on the frame.

Figure 4 shows the operations performed to encode a video frame. For a given frame, the encoder first divides a raw frame into macroblocks, squares of pixels containing three components: luminance (Y), and chroma (Cr and Cb). Macroblocks have a dimension of 16x16 pixels for the Y component while having a dimension of 8x8 pixels for the chroma components [5]. Then, the encoder performs the intra- and inter-prediction phases (boxes in light shade). It then transforms the residual frame (the frame obtained from the difference between the original frame and the predicted one). Typically, the transformation applied to this residual frame involves

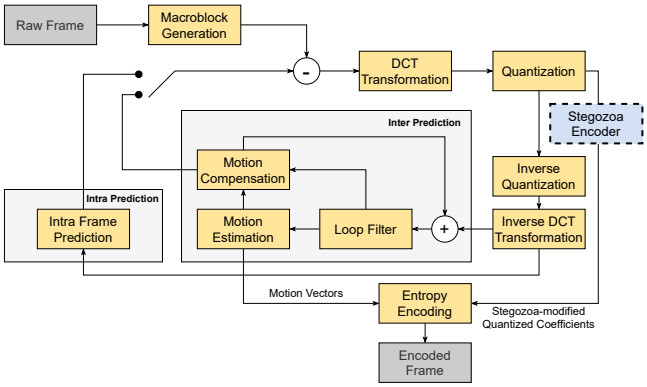


Figure 4: Coding phases of the VP8 video encoder. Stegozoa's additional covert data encoding phase is highlighted in blue.

a scaled approximate discrete cosine transform (DCT) [6] over 4x4 pixel blocks obtained from a subdivision of the frame's macroblocks. The result of this transformation is a set of DCT coefficients (also laid out in a 4x4 block) which encode the representation of the block in the frequency domain. Each DCT block includes a DC coefficient (the one with the lowest frequency, placed in the top-left corner), while the remaining coefficients are called the AC coefficients. Then, the resulting coefficients are quantized in the quantization phase by a scalar value. The final step is the entropy encoding phase, where the quantized discrete cosine transform (QDCT) coefficients are losslessly compressed into a binary series of boolean values.

Stegozoa's steganographic scheme: We intend to create a covert channel by embedding covert data into WebRTC video streams using steganography. The video stream is a sequence of frames, with each frame having a potential embedding capacity. To avoid detection by an adversary that has access to the video payload, we need to devise a data embedding scheme that avoids the distortion of the video frames' contents and prevents video decoding corruptions to occur. At the same time, our embedding scheme should increase the number of covert bits that can be embedded per frame to increase the bandwidth of the covert channel. To achieve these goals, we employ least significant bit (LSB) substitution on the QDCT coefficients in combination with two additional steps:

(1) **QDCT coefficient filtering:** Changes on the DC coefficients are more impactful than changes on the AC coefficients; additionally, there are fewer blue chroma and red chroma coefficients than luminance coefficients (there are 2 luminance coefficients for each chroma coefficient), making changes to chroma coefficients more visible. Therefore, to reduce video distortion, we select a subset of QDCT coefficients for embedding the message payload, namely: non-DC, luminance coefficients [44]. We also exclude coefficients equal to 0 or 1.
(2) **Syndrome-trellis coding (STC):** From the pre-filtered subset of QDCT coefficients, we further apply syndrome-trellis coding (STC) [24] on top of LSB substitution for minimizing the video distortion. The objective of STC is to generate a stego vector y that minimizes the cost of changing the original cover vector (i.e., the pre-selected coefficients) to y. To measure the impact of changing a given bit, the STC algorithm uses a configurable *distortion function*. For instance, a function that returns a constant value indicates that every bit change has the same impact on the resulting stego vector (which, in our case, translates to the perceived image quality of the steganographically-marked frame). The STC algorithm is controlled by a parameter α, which determines the fraction of the embedding capacity used for embedding the message, and a parameter h. h and w (where $w = 1/\alpha$) represent the height and width of a sliding matrix $\hat{H}$, used to compose an *embedding matrix* H[1]. The algorithm then seeks to find y such that $Hy = x$, where x is the message (in bits, 0 or 1) to be embedded of size m, and y is a vector where the message is embedded, of size $m \times w$. The y is the resulting stego vector to be sent to the receiver, who will then extract m using the shared H. In Stegozoa, we set $h = 7$ according to the syndrome-trellis code proposed by Filler et al. [24], and select the value of α experimentally (see Section 5.1).

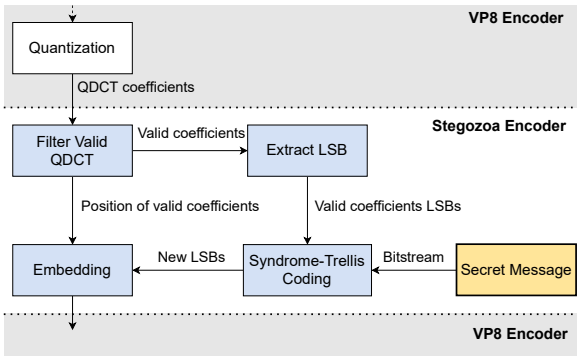


Figure 5: Stegozoa' covert data encoding phases.

Stegozoa's encoding stages: Putting it all together, Figure 5 shows the steps performed by the Stegozoa encoder for applying LSB substitution with coefficient filtering and STC. The inputs are the QDCT coefficients generated by the built-in quantization phase of the VP8 codec (see Figure 4). In the coefficient filtering stage, the encoder computes the number of available valid coefficients to discover a frame's embedding capacity. It then extracts the LSBs of the valid coefficients which are given as input (along with the payload message) to the STC algorithm. In turn, the STC algorithm returns the final LSB vector reflecting the payload message to be embedded into the video frame in the last phase. With the embedding operation concluded, the VP8 encoding pipeline can then proceed and finish encoding the frame using the newly changed QDCT coefficients.

3.4 Efficient Bandwidth Utilization

The steganographic scheme described above will make it very difficult for an adversary to detect the presence of covert messages in the resulting video stream. However, this enhanced security comes at the price of a dramatic reduction of the covert channel bandwidth, i.e., the number of bits that can safely be used for sending payload data per video frame. Some of these bits must be further

[1] H works as a shared secret between sender and receiver preventing a third party from trivially extracting the embedded message from the video frames. Matrices $\hat{H}$ with greater height and width can enhance the cryptographic secrecy of the covert communication at the expense of a computationally heavier steganographic embedding process [18]. We defer the evaluation of different $\hat{H}$ to future work.

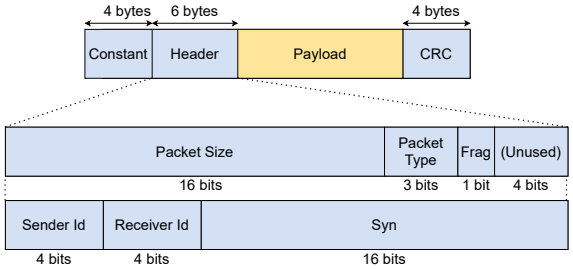


Figure 6: Format of Stegozoa packets. The packet type field identifies five types of packets: discovery packet (0), discovery response packet (1), regular packet (2), retransmission request packet (3), and retransmitted packet (4).

reserved for the transmission of meta-data and control messages, e.g., to deal with message fragmentation, error recovery, and session management. Thus, it is fundamental to efficiently use the available bandwidth to increase the performance of the channel. Next, we present the most interesting design aspects of the Stegozoa protocols affecting the bandwidth utilization of covert channels.

Message fragmentation and reassembly: To efficiently fit the content of the client-generated messages into the available covert bit space of the video streams, Stegozoa endpoints implement a 2-step message fragmentation and reassembly protocol:

(1) **Fragmentation of messages into packets:** Given that the API call `send(message, receiver)` allows a client application to send a message of arbitrary size, `libstegozoa` first splits the outgoing message into chunks of bounded size. For processing efficiency reasons, we set the maximum size of each chunk to 242 bytes (i.e., 256 bytes minus 14 bytes for additional meta-data fields). The library then encloses each chunk inside a *packet* that contains three additional fields — a 4-byte constant, a 6-byte header, and a 4-byte CRC value – as shown in Figure 6. Each resulting packet is then forwarded to the Stegozoa hooks inside the WebRTC stack to be further fragmented before being transmitted.
(2) **Fragmentation of packets into frames:** Since the available space in any given frame is variable, it may be necessary to further fragment single packets into smaller pieces, fitting each piece into the full covert bit capacity of the frame. Thus, to send a given packet, all the sender needs to do is: (1) wait for the next available frame, (2) determine the amount of free space in that frame, (3) read the next piece of the packet to be sent, and (4) encode that piece into the frame.

Since the video frames typically arrive in order, the reassembly process is relatively simple. First, the receiver's WebRTC hook collects the fragments embedded in each frame and reassembles them into individual packets. To determine the packet boundaries, Stegozoa scans the recovered bit stream looking for a constant 4-byte value (see Figure 6) which signals the beginning of a packet. The packet size provided in the header field indicates the packet's ending. In a second step, `libstegozoa` collects the reassembled packets inside a packet queue until the last packet arrives. A fragmentation field in the header identifies the trailer packet composing the message. The fully recovered message is then placed in a different queue, waiting to be fetched through the `receive()` API call.

Recovery from errors: As described above, Stegozoa transmits covert packets within WebRTC-based video streams. However, packet loss or corruption may occur as a result of *simulcast* video stream switches. Simulcast [26] is a technique that enables WebRTC gateways to efficiently adapt the resolution of the video streams transmitted to the participants of a video call. When simulcast is used, each participant sends to the gateway multiple video camera feeds encoded in different resolutions. Then, depending on the downlink bandwidth of each receiver, the gateway automatically selects the most suitable video stream to each individual receiver's network conditions. When simulcast is active, Stegozoa replicates the same packet in all outgoing video stream resolutions. The receiver retrieves the packet by recovering it from the video stream selected by the WebRTC gateway. However, packet fragmentation boundaries are expected to depend on frames' resolution, since these will have different embedding capacities. Thus, if the gateway decides to switch the video stream sent to the receiver, a received packet may end up being lost or corrupted. A second reason that may cause the loss/corruption of Stegozoa packets is the unsynchronized invocation of `initialize(ID)`, where a sender transmits packets to other participants before these packets are ready to be received at the destination. To tackle the problems of packet loss/corruption, we employ different solutions for packets sent in different stages of the covert data transmission (see Figure 3):

(1) **Error recovery with discovery packets:** To recover from the loss or corruption of discovery packets, we rely on the liveliness property of the system – i.e., eventually, we assume that a packet will be successfully received. Our solution is then to broadcast the discovery packet periodically (every 5 seconds). Since the discovery and response packets are relatively lightweight, the resulting overheads of broadcasting these packets are marginal. This solution also solves the problem of unsynchronized `initialize(ID)` calls.
(2) **Error recovery with regular packets:** To recover from errors involving the transmission of regular data packets, Stegozoa relies on a retransmission mechanism. Given that the available bandwidth of the channel is scarce and its latency is high, we strive to reduce the number of control messages required for coordinating the packet retransmission process. Our solution relies on a dedicated field in the packet header (*syn*) which numbers all packets transmitted by the sender during a covert session. The syn field is set to 0 for the first sent packet, and then it is incremented for every packet transmitted subsequently. Since video streams typically arrive in order, the receiver only needs to keep a local syn counter for tracking the next expected packet, and request a packet retransmission if some packet was skipped.

To speed up the error recovery process, Stegozoa gives priority to retransmission requests and retransmitted packets.

3.5 Preserving Traffic Analysis Resistance

To ensure that the video streams modified by Stegozoa are secure against advanced traffic analysis techniques, it is essential that the generated network traffic preserves the original features of the carrier video stream. However, since WebRTC video frames must

be encoded in real-time, any significant overhead caused by Stegozoa may lead to the generation of abnormal traffic patterns that would make Stegozoa connections prone to detection. To prevent this problem and reduce the delay in encoding a frame caused by Stegozoa's operations, we developed the following two essential implementation-dependent optimizations for the Stegozoa encoder. Section 5.2 demonstrates the positive impact of these optimizations in making Stegozoa robust against traffic analysis attacks.

1. Trailing null-coefficient optimization: Before embedding covert data in a given frame, the Stegozoa encoder needs to compute the number of valid QDCT coefficients of the frame. To this end, the encoder performs a nested loop where it first visits all the macroblocks of the frame, and then visits the blocks of each macroblock and loops over the coefficients of each block – in total, 240 coefficients per macroblock[2]. Depending on the video resolution, a frame can have hundreds or even thousands of macroblocks. As a result, the number of coefficients that need to be scanned can grow very large potentially introducing detectable video encoding slowdowns. To speed up this process, we leverage the following insights. After the transformation and quantization phases of VP8 (see Figure 4), most of the high-frequency coefficients are set to zero. Given that null coefficients are not suitable for covert bit embedding (see Section 3.3), we can save precious looping time by skipping the analysis of these coefficients. Luckily, the VP8 video encoder keeps track of the "end of block", which is a direct pointer to the last non-zero coefficient of each block (when the block is processed in raster scan order). Thus, our optimization is to loop over the macroblock's blocks first and then check the coefficients in raster scan order until we find the last non-zero coefficient. This way, we avoid having to test most of the undesirable coefficients, thereby reducing the overhead caused by the above operation.

2. Hardware-accelerated memcopy optimization: For the embedding operation to work (see Figure 5), the QDCT coefficients must be copied to a new buffer so that the STC algorithm can process them independently, which requires copying 400 coefficients in total per macroblock (i.e., 25 blocks with 16 coefficients each). At first, we performed this operation using the `memcpy` function from the `libc` library. However, just like in the case mentioned above, we observed that `memcpy`'s software implementation introduced noticeably delays in the resulting inter-packet time distributions. To tackle this problem, we used advanced vector extensions (AVX) [29] instructions for copying data. The AVX are extensions to the x86 instruction set architecture introduced in 2011. We used instructions that copy 256-bits blocks at a time (e.g., `_mm256_stream_si256`). We have also optimized the code to unroll time-consuming loops.

4 EVALUATION METHODOLOGY

This section describes our evaluation methodology. First, we describe the goals and approach of our evaluation. Then, we detail the testbed we designed for performing our experiments and the metrics used to assess the quality of our solution.

[2]Each macroblock has 24 blocks – 16 luminance blocks, 4 red chroma blocks, and 4 blue chroma blocks (depending on the type of prediction, a 25th virtual block can exist, composed of the DC coefficients of the luminance blocks), with each block containing 16 coefficients. Since we only consider the luminance coefficients for embedding, only the luminance blocks are checked, making the number of checked coefficients 256 – as we do not consider the DC coefficients for embedding, this number goes down to 240.

4.1 Evaluation Goals and Approach

Our evaluation goals are twofold: (i) assess the network performance of our Stegozoa prototype, and (ii) measure its security against detection by a powerful adversary. Related to the latter, we wish to evaluate two different dimensions of Stegozoa's resistance against detection. Firstly, we investigate whether it is possible for a network adversary to leverage *traffic analysis* techniques to distinguish between the traffic generated by Stegozoa and the traffic of unmodified WebRTC media streams. Secondly, we are interested in understanding whether adversaries in control of WebRTC gateways can detect the operation of Stegozoa through *video steganalysis*.

To perform traffic analysis we employ state-of-the-art machine learning-based classifiers. We use XGBoost [9] in combination with a comprehensive set of features based on multiple summary statistics computed over the packet length and inter arrival times, such as burst behaviors and high-order statistics (e.g., kurtosis).

To perform video steganalysis, we leverage the available implementations of the SUPERB [14] and IDFB [58] video steganalysis techniques in combination with a XGBoost classifier. Both SUPERB and IDFB target the steganalysis of embedding methods that rely on the manipulation of QDCT coefficients. Since these techniques output a set of features per frame, we compute a set of summary statistics that represent a whole video stream, based on its individual frame features, to classify WebRTC streams on a per-call basis. Specifically, for each feature, we computed the mean, median, variance, standard deviation, maximum, minimum, and percentiles across each of the video frames. We leverage 10-fold cross validation to evaluate both the traffic analysis and steganalysis classifiers.

We performed a set of experiments where two Stegozoa endpoints uninterruptedly send messages in each direction so as to use the covert channel at full capacity. We do so to stress Stegozoa's covert data encoding mechanisms. Then, based on the collected data, we are able to (i) assess its performance and (ii) assess its resistance against detection when triggering possible network traffic and image distortions caused by the operation of our prototype.

4.2 Experimental Testbed

Our laboratory testbed, illustrated in Figure 7, is composed of three Ubuntu 18.04.5 LTS virtual machines (VMs) with four virtual 2.00 GHz Intel Xeon Gold 6138 cores and 8GB of RAM each. While VM1 and VM3 both execute an instance of Stegozoa, VM2 acts as a gateway and router for the Stegozoa endpoints. VM2 also mimics the behavior of a middlebox controlled by the adversary, allowing the collection of network traces required for conducting traffic analysis. Lastly, videoconferencing calls are established resorting to Jitsi (`meet.jit.si`), a well-known WebRTC application that makes use of WebRTC gateways to mediate the exchange of media streams.

To generate representative packet traces and unencrypted media recordings of legitimate and Stegozoa media streams, we borrow 492 videos from Protozoa's "Chat" video library – these videos were collected from YouTube and represent typical videoconferencing interactions [10]. In our experiments, we partition the dataset in two halves and establish 246 legitimate WebRTC media streams and 246 Stegozoa media streams while transmitting the same video on each side of the WebRTC stream. The transmission of the same video in both sides of the stream allows us to avoid contamination

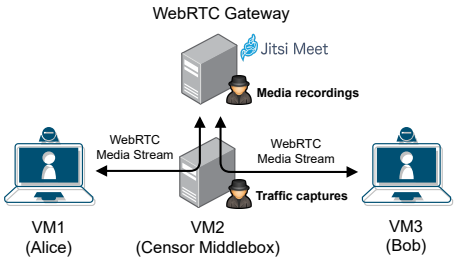


Figure 7: Laboratory setup.

during training, stemming from the inclusion of the same video samples when generating Stegozoa and legitimate streams. Packet traces are collected by VM2 for a duration of 30 seconds, a sufficient period for enabling the accurate detection of covert channels within multimedia streams using ML-based traffic analysis techniques [9]. Accordingly, we obtain media recordings from Jitsi's call recording feature so as to emulate the access to unencrypted media data by WebRTC gateway-controlling adversary. Video calls are automatically recorded for a duration of 30 seconds. The resulting video files are fed to steganalysis feature extractors [14, 58], whose output can be used to classify video samples as legitimate or Stegozoa.

4.3 Metrics

We employed the following metrics when evaluating Stegozoa's performance and resistance against detection:

Performance metrics: We assess our prototype's performance with respect to its achievable *throughput* and *latency*. In addition, we measure the necessary resources (CPU and RAM) for executing Stegozoa at each WebRTC call endpoint.

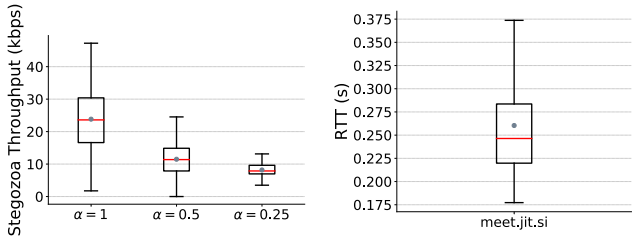
Detection metrics: We use the same set of metrics for evaluating Stegozoa's resistance against traffic analysis and steganalysis. As in prior work [9, 10], we use the true positive rate (TPR), false positive rate (FPR), and the area under the Receiver Operating Characteristic (ROC) curve (AUC) to assess the resistance of our prototype against detection. The TPR measures the fraction of Stegozoa streams that are identified correctly, while the FPR measures the fraction of legitimate streams that are wrongly classified as Stegozoa. The ROC curve plots a classifier's TPR against its FPR for different configurable internal thresholds. This trade-off can be summarized using the AUC – an AUC of 0.5 is equivalent to random guessing while a perfect classifier achieves an AUC of 1.

5 EVALUATION RESULTS

This section presents the evaluation of our Stegozoa prototype. We start by studying Stegozoa's performance and security while varying the steganographic embedding capacity. Then, we study the impacts of changes on network conditions on Stegozoa. Lastly, we compare Stegozoa with related covert channel-generating tools.

5.1 Covert Channel Performance

We evaluate different dimensions of Stegozoa's covert channel performance and motivate the use of our system in two usage scenarios. During our experiments, the VMs executing Stegozoa consumed 50% of their total CPU and 900MB of RAM. Stegozoa can therefore run on common, off-the-shelf consumer hardware. In the below



(a) Throughput (b) Latency

Figure 8: Stegozoa's throughput and messages' round-trip-time when establishing a covert channel over Jitsi.

Stegozoa Config.	# of Tweets			# of Wikipedia articles		
	10 min	30 min	60min	10 min	30 min	60 min
α=1	54 091	162 273	324 546	472	1 416	2 832
α=0.50	25 909	77 727	155 454	226	678	1 356
α=0.25	18 636	55 908	111 816	163	489	978

Table 1: Average number of downloaded tweet and Wikipedia-sized messages for increasing Stegozoa call times.

experiments, we test Stegozoa's performance for different configurations of α, which acts as Stegozoa's tuning knob between security and achievable covert channel throughput (see Section 3.3). While our traffic analysis (see Section 5.2) and steganalysis (see Section 5.3) experiments reveal α=0.25 to be a rather secure configuration, we also show our results for larger α values, enabling Stegozoa users to select different security/performance trade-offs. We do not show results for experiments using α values under 0.25 since we this value provides compelling resistance against traffic analysis (Section 5.2) and video steganalysis (Section 5.3). While smaller α values can potentially offer increased resistance traffic analysis and steganalysis, the expected trade-offs in covert channel performance makes these a less compelling option for parameterizing our scheme.

Throughput averages 8.2 Kbps: Figure 8 a) depicts a boxplot of Stegozoa's throughput for decreasing values of α. The box shown in the plot is drawn from the first quartile to the third quartile of obtained values. The whiskers represent the variation between the minimum and maximum achieved throughput, excluding outliers. The horizontal red line depicts the median of the obtained values, whereas the grey dot represents the mean. For instance, the most steganalysis-resistant Stegozoa configuration (α=0.25) averages a throughput of 8.2 Kbps, while the less steganalysis-resistant configuration (α=1) averages a throughput of 23.8 Kbps. The plot also shows that the variability of achieved throughput is larger as α increases. This may be explained due to the heterogeneity of the video source used to embed Stegozoa's secret messages.

RTT of "ping" varies between 261-374 ms: Figure 8 b) depicts a boxplot of Stegozoa's latency measured as the time it takes for a sender to "ping" the receiver, i.e., transmit a small Stegozoa packet in an outbound video frame and receive an acknowledgement packet in an inbound video frame. Since these packets are very small (we send the headers only), they can always be transmitted on a video frame regardless of the chosen α. Stegozoa averages a round-trip-time of 0.261 seconds, and a maximum of 0.374 seconds. This can make Stegozoa an attractive and suitable tool for particular kinds of interactive traffic like censorship-resistant instant messaging.

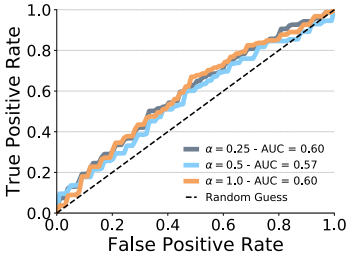
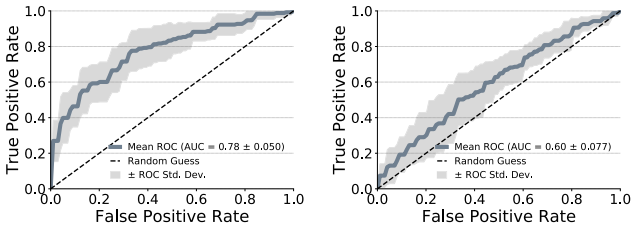
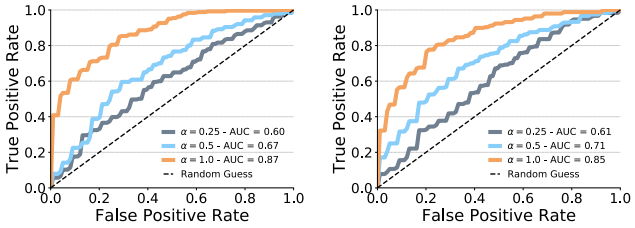


Figure 9: ROC AUC obtained by the traffic analysis classifier when distinguishing Stegozoa from legitimate streams.



(a) Before optimizations (b) After optimizations

Figure 10: Results of the traffic analysis classifier using summary statistics before and after optimizations ($\alpha = 0.25$).



(a) IDFB features (b) SUPERB features

Figure 11: ROC AUC obtained by the steganalysis classifiers when distinguishing Stegozoa from legitimate videos.

Viable use-case scenarios: Despite its reasonable performance, Stegozoa is not amenable to the transmission of bulk data, nor for streaming live video through its covert channel. Nevertheless, it can be used for exchanging commonly censored small-sized content like Twitter-style feeds [54], or Wikipedia articles [47]. Table 1 depicts the amount of average-sized tweets (33 characters × 1 byte ≈33 bytes [46]) and Wikipedia articles (623 words × 6 characters × 1 byte ≈3 783 bytes [52]) that can be transferred in Stegozoa calls ranging from 10 to 60 minutes, for decreasing values of α. We can observe from the table that Stegozoa's most steganalysis-resistant configuration ($\alpha = 0.25$) can, for instance, download about 18636 tweet-sized messages during a 10 minute videocall or download 978 Wikipedia articles during a 60 minute videocall. While this experiment has focused the transmission of uncompressed text, it is likely that the number of transmitted tweet or Wikipedia article-sized messages can be increased by taking advantage of data compression techniques (e.g., by compressing text into `.zip` files).

5.2 Resistance Against Traffic Analysis

Keeping our focus on the set of α configurations studied above, we now evaluate Stegozoa's resistance against state-of-the-art traffic analysis attacks geared at identifying the operation of covert channels established over media streams.

Figure 9 depicts the ROC curves of the XGBoost [9] classifier when attempting to identify Stegozoa streams amongst legitimate WebRTC streams, for different values of α, through the analysis of summary statistics collected from network traces. We see that an adversary is able to obtain a maximum AUC of only 0.6 for $\alpha = 1$. Intuitively, our experiments show that *Stegozoa is robust against traffic analysis attacks.* As an example, a censor willing to block 50% of all Stegozoa streams (TPR = 0.5) would erroneously block about 40% of legitimate WebRTC streams (FPR = 0.4). While different censors may tolerate different TPR/FPR trade-offs, Figure 9 shows that distinguishing Stegozoa from legitimate WebRTC streams is rather similar to random guessing as AUC remains close to 0.5.

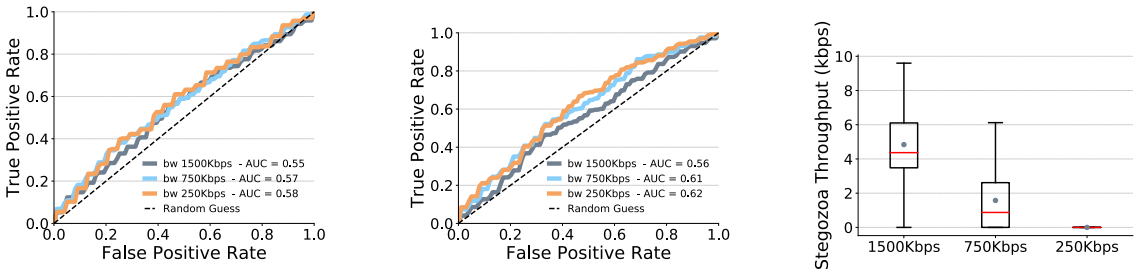
Since the above summary statistics contain information about packet timing (recall Section 4.1), these are particularly sensitive to even slight increases in inter-packet delays. In fact, we observed that the relatively small overheads imposed by Stegozoa's vanilla steganographic encoding scheme, i.e., without optimizations, has a detrimental effect on the ability of Stegozoa to resist against traffic analysis. Figure 10 shows the results obtained by the classifier before and after introducing the optimizations mentioned in Section 3.5. The figure shows that the ability of an adversary to detect Stegozoa (α=0.25) through traffic analysis decreases significantly, with the AUC dropping from 0.78 to 0.60 (similar to random guessing).

5.3 Resistance Against Video Steganalysis

In addition to Stegozoa's robustness against traffic analysis, a second pillar underpinning the security of our system is its resistance against video steganalysis. In this section, we evaluate Stegozoa's resistance against steganalysis attacks launched by an adversary with the capability to inspect unencrypted WebRTC media streams and employ the IDFB and SUPERB algorithms. Our analysis shows that Stegozoa offers the best steganalysis resistance for $\alpha = 0.25$.

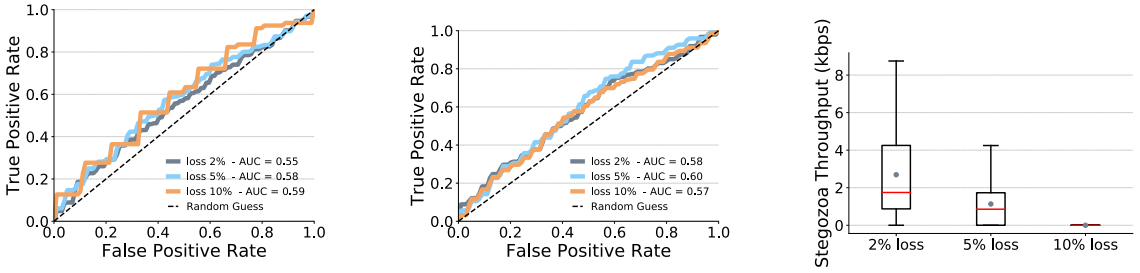
Steganalysis based on IDFB: Figure 11 a) depicts the ROC curves of the IDFB-based steganalysis classifier when attempting to distinguish legitimate videos from Stegozoa videos that embed covert data, for different values of α. The plot shows that the classifier is able to correctly detect a large fraction Stegozoa videos when α=1, obtaining an AUC of 0.87. This is unsurprising since, in this configuration, Stegozoa's steganographic output is expected to exhibit a high distortion rate. For α values of 0.5 and 0.25, the classifier is only able to obtain an AUC of 0.67 and 0.60, respectively. Indeed, this provides Stegozoa users to choose a configurable trade-off between covert channel performance and efficiency, e.g., conservatively choosing α=0.5 for achieving a reasonable resistance against detection and an average throughput of 11.4 Kbps (cf. Section 5.1).

Steganalysis based on SUPERB: Figure 11 b) depicts the ROC curves of the SUPERB-based classifier when attempting to pinpoint Stegozoa videos that embed covert data, for different values of α. The figure shows that this steganalysis classifier has a similar performance to the IDFB-based classifier, achieving a high AUC when steganographic output video distortion is higher (α=1), and an AUC close to random guessing for conservative values of α.



(a) Traffic analysis (b) Steganalysis - SUPERB (c) Throughput

Figure 12: Stegozoa's ($\alpha = 0.25$) performance and resistance against detection when subject to bandwidth impairment.



(a) Traffic analysis (b) Steganalysis - SUPERB (c) Throughput

Figure 13: Stegozoa's ($\alpha = 0.25$) performance and resistance against detection when subject to packet loss.

5.4 Effects of Adverse Network Conditions

In this section, we evaluate our prototype's performance and resistance against detection when establishing covert channels over networks that exhibit adverse network conditions. Network conditions can be adverse in two cases. Firstly, the experiments we describe below mimic realistic scenarios where *poor network conditions* may be experienced. Secondly, and most importantly, these experiments also shed light on whether Stegozoa can resist against powerful adversaries that can launch *active attacks* by artificially manipulating the network conditions (e.g., through throttling of the video streams) making Stegozoa covert channels easier to detect.

Akin to previous work [10], we artificially introduce network impairments in Stegozoa streams by using the `netem` network emulation functionality in Linux. We introduce network perturbations in two different dimensions, namely bandwidth and packet loss, since these dimensions induce the larger performance fluctuations in Protozoa [10], the non-steganalysis resistant sibling of Stegozoa. Likewise, we bound our perturbations to the typical range of adverse network conditions that can be sustained by WebRTC media connections, previously studied by Jansen et al. [30], and experimentally confirmed by ourselves. The following paragraphs describe the results of the above experiments.

Analysis of bandwidth restrictions: Figure 12 depicts Stegozoa's resistance to traffic analysis, steganalysis (against SUPERB, the overall better performing classifier), and achieved throughput when restricting bandwidth from 1 500 Kbps down to 250 Kbps. Figure 12 a) shows that Stegozoa's resistance against traffic analysis is maintained when network bandwidth is impaired, since the traffic analysis classifier obtains a maximum AUC of 0.58 when bandwidth is limited to 250 Kbps. In turn, Figure 12 b) shows that Stegozoa is also able to resist steganalysis when bandwidth restrictions are in place. In effect, the steganalysis classifier is only able to achieve an AUC of 0.62, which remains rather close to random guessing. Lastly, Figure 12 c) depicts the throughput obtained by Stegozoa when subjected to bandwidth restrictions. We can observe that average throughput decreases from 8.2Kbps (in unconstrained bandwidth conditions) to 4.8 Kbps, 1.6 Kbps, and 0 Kbps when bandwidth is constrained to 1 500, 750, and 250 Kbps, respectively. This throughput degradation is expected as bandwidth constriction causes WebRTC's congestion control mechanisms to reduce frame rate and downgrade video resolution, leading to inferior per-frame steganographic capacity. While pronounced bandwidth impairments significantly hamper Stegozoa's performance, we posit that state-level adversaries would avoid to enforce such restrictions since they would also plummet the quality of legitimate calls [30].

Analysis of packet loss percentage: Figure 13 depicts Stegozoa's resistance to traffic analysis, steganalysis, and achieved throughput when imposing packet loss rates from 2% up to 10% on the network. Figure 13 a) shows that Stegozoa's resistance against traffic analysis is upheld for different packet loss rates. We can see that the traffic analysis classifier achieves a maximum AUC of 0.59 for a packet loss rate of 10%, only 0.01 larger than the achieved AUC when restricting bandwidth to 250 Kbps (Figure 12 a)). Figure 13 b) reveals that Stegozoa resists an adversary's steganalysis efforts when packet loss is applied to Stegozoa's WebRTC streams. As shown in the plot, the steganalysis classifier achieves a maximum AUC of 0.60 for a packet loss rate of 10%. Finally, Figure 13 c) shows that Stegozoa's throughput is negatively affected by packet loss. The boxplot shows that the average throughput of the covert channel decreases from 8.2Kbps (in a network experiencing no packet

System	Traffic analysis resistance	Steganalysis resistance	Throughput
Protozoa	✓	–	1 400.000 Kbps
SkypeLine	✓	✓	0.064 Kbps
Stegozoa (Whereby, $\alpha = 0.25$)	✓	✓	2.600 Kbps
Stegozoa (Jitsi, $\alpha = 0.25$)	✓	✓	8.200 Kbps
Stegozoa (Jitsi, $\alpha = 0.50$)	✓	✓	11.400 Kbps

Table 2: Comparison of Stegozoa and existing systems that encode covert data over media-conferencing streams.

loss) to 2.6 Kbps and 1.1 Kbps for packet loss rates of 2% and 5%, respectively. This can be explained by the fact that lost packets will prevent the reassembly of certain video frames, which will trigger the retransmission of Stegozoa packets. In addition, and similarly to bandwidth restrictions, large packet loss rates are also known to reduce the quality of WebRTC streams. In particular, both WebRTC streams are torn down for packet loss rates of 10% [30].

5.5 Portability Across WebRTC Applications

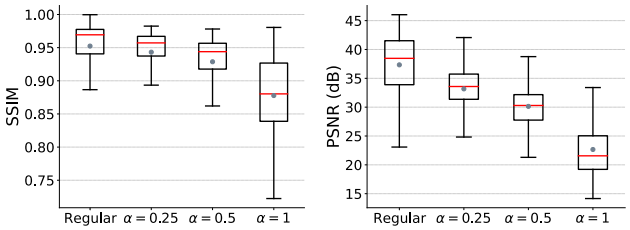
To test Stegozoa covert channels over alternative WebRTC applications, we replicated our experiments using Whereby (`whereby.com`), a commercial WebRTC application. For our experiments, we used the Whereby *Large Room* which explicitly leverages a WebRTC gateway to forward traffic between a call's participants. Akin to the setup illustrated in Figure 7, we use Stegozoa (α=0.25) to establish a covert channel between Alice and Bob, collect the traffic flowing between these endpoints, and resort to Whereby's call recording feature to obtain video samples for further steganalysis.

Our results reveal that Stegozoa's resistance against detection is also preserved on Whereby. Under normal network conditions, the traffic analysis classifier obtained an AUC of 0.62 while the steganalysis classifier obtained an AUC of 0.58 when using the SUPERB extractor (which provided the best results). When subjecting Stegozoa over Whereby streams to the same network perturbations described in Section 5.4, the traffic analysis classifier obtained a maximum AUC of 0.61 for a bandwidth restriction of 250 Kbps.

Stegozoa over Whereby streams (in unperturbed network conditions) achieves an average throughput of 2.6 Kbps, which is 5.6 Kbps shorter than covert channels established over Jitsi. A closer inspection of video metadata revealed that, in steady state, Jitsi exchanges videos with higher resolution than Whereby (1280x720 vs. 480x360, resp.). Consequently, Jitsi videos are comprised of a larger amount of macroblocks and provide a larger embedding capacity for Stegozoa messages. Interestingly, when bandwidth is restricted to 750 Kbps, Stegozoa over Whereby achieves a throughput of 2.5 Kbps (only 0.1 Kbps less than without throttling), suggesting that Whereby video requires only a bitrate of 750 Kbps or lower. Our remaining experiments show that the throughput is severely affected by more significant bandwidth restrictions or packet loss.

5.6 Visual Quality Analysis on Whereby

Since Stegozoa relies on the replacement of DCT coefficients' LSBs to encode covert data, one may wonder whether these changes cause perceptible image artifacts or quality degradation that may be used by a human analyst (in control of a WebRTC gateway) to detect Stegozoa streams through visual inspection. To understand if Stegozoa streams can be detected through such analysis, we conducted an experiment for computing two perceptual quality metrics



(a) SSIM comparison (b) PSNR comparison

Figure 14: Stegozoa's streams video quality assessment using SSIM and PSNR over Whereby (for different α settings).

– the peak signal-to-noise ratio (PSNR) [28] and the structural similarity index (SSIM) [49] – over the images exchanged in Stegozoa video streams and those exchanged in legitimate video streams.

We further note that an adversary would be unable to compute SSIM and PSNR for arbitrary streams crossing the WebRTC gateway, and to use the output of such computations to extract features that would allow it to distinguish between legitimate and Stegozoa streams. This is due to the fact that the adversary would require access to both the original video frames (before compression) and the encoded video frames produced by Stegozoa, since both metrics measure image quality by comparing the same video frame before and after a frame is processed by the video encoder. However, as we will show next, the computation of PSNR and SSIM provides quantifiable evidence that the visual quality of WebRTC video streams is not significantly degraded for most Stegozoa configurations.

Figure 14 a) shows the results of an experiment where we compute and compare the SSIM of regular WebRTC video streams with that of Stegozoa video streams. The boxplots show that the average SSIM of Stegozoa configurations using an α of 0.25 and 0.5 are within the range of typical SSIM values found in legitimate WebRTC streams. In turn, while the use of $\alpha = 1$ places the SSIM outside the typical values of legitimate streams, it still achieves an average SSIM of 0.87 and over 0.92 at the 75th percentile. This suggests that the use of α equal to 0.25 or 0.5 will generate covert data-carrying images that are very similar in quality to the encoded images generated by the original WebRTC pipeline.

In addition, Figure 14 b) depicts the results of a similar experiment where we compute and compare the PSNR of regular WebRTC video streams with that of Stegozoa streams. The use of α values of 0.25 and 0.5 results in images with a quality within that expected to be observed in legitimate WebRTC video streams, and which is within reference PSNR measurements (>30dB) found in WebRTC video quality assessments conducted with the VP8 codec [27]. In contrast, images generated with $\alpha = 1$ exhibit poor quality (obtaining an average PSNR of ≈20dB). These results are aligned with our analysis in Section 5.3, which reveals that $\alpha = 1$ is less resistant to steganalysis than the remaining α configurations under test.

5.7 Comparison to Related Systems

In Table 2, we compare the performance of Stegozoa with that of Protozoa [10] and SkypeLine [36], two state-of-the-art systems that build covert channels over multimedia conferencing applications. Protozoa allows for the creation of high-speed and traffic analysis-resistant covert channels over peer-to-peer WebRTC media streams.

While Protozoa largely outperforms Stegozoa w.r.t. throughput, Protozoa is trivially detected by an adversary that can inspect the media stream. In turn, analogously to Stegozoa, SkypeLine assumes a stronger threat model where the adversary is able to inspect unencrypted Skype audio streams, and proposes a steganographic data embedding mechanism over audio calls. However, Stegozoa delivers a throughput of two to three orders of magnitude larger than SkypeLine's. Such an increase in throughput stems from the fact that Stegozoa uses a carrier signal for covert data with higher bandwidth capacity than SkypeLine, i.e., video instead of audio. While Stegozoa could potentially adapt SkypeLine's audio-based steganography mechanism to further increase its covert throughput, Table 2 also reveals that this procedure would not provide tangible benefits. Thus, for simplicity, we refrained from implementing this audio steganography technique in Stegozoa's prototype.

6 DISCUSSION

Tolerating media transcoding: Stegozoa cannot tolerate live video transcoding, i.e., re-encoding live media to different codecs, or dynamic video rescaling performed in Multipoint Conferencing Unit (MCU)-style WebRTC gateways [51]. However, due to large server-side computing demands, MCUs are infrequently adopted, being mostly used for integration with legacy systems [3]. Steganography techniques like TRIST [15] can survive image resizing and upscaling, but could still be defeated by an MCU-controlling adversary that indiscriminately reduces the quality of media streams. We will explore transcoding-resistant STC [21] in future work.

Syndrome-Trellis codes' distortion function: When developing Stegozoa, we take a simplified approach and use a unitary cost distortion function that assumes that changes to any QDCT LSB will imply the same cost, irrespective of the particular cover video frame being modified. Hence, this may lead to sub-optimal covert data embedding rates and to the generation of lower quality steganographic outputs. The exploration of an efficient distortion function that can measure the impact of data embedding operations under real-time requirements is an interesting direction for future work.

Insertable streams: Insertable streams [31] are recently proposed WebRTC extensions to further strengthen the end-to-end security of WebRTC calls. They allow call participants to encrypt their media streams with a secret key (possibly exchanged out-of-band), prior to applying WebRTC's default DTLS-SRTP-based encryption at the network layer. Hence, the use of insertable streams could prevent WebRTC gateways from inspecting the media being relayed. However, at the moment, the use of insertable streams in WebRTC calls is optional and remains undeployed in the majority of WebRTC services. In addition, we expect adversaries to deploy generic WebRTC gateway implementations like Janus [3], which require access to the exchanged media streams for additional processing.

7 RELATED WORK

Internet censorship-circumvention: There are many approaches for censorship-resistant communication [33], such as refraction networking [12]), censorship-resistant CDNs [59], ephemeral proxy-based tools [22], traffic randomization tools [17], or packet manipulators [11]). Our work follows a specific line of proposals for the creation of covert channels leveraging multimedia streams. For instance, SkypeMorph [42] shapes covert traffic to mimic legitimate multimedia streams. In turn, other systems [8, 37, 40] modulate covert data into artificial media content exchanged by media streaming applications. A recent censorship-resistant tool, named Protozoa [10], instruments the WebRTC media coding stack to replace encoded video content with covert data. Yet, none of the above systems operate under a threat model that assumes adversaries to be able to inspect unencrypted media streams. In contrast to the above systems, and closer to our work, SkypeLine [36] assumes that an adversary can inspect unencrypted media streams, and leverages an audio steganography scheme for creating covert channels over audio calls. However, SkypeLine is restricted to a single application (Skype) and achieves an inferior covert channel throughput when compared to Stegozoa (between 2 to 3 orders of magnitude).

Video steganography: Video steganography techniques can be distinguished in three main categories [38]. First, *intra-embedding techniques* embed the secret message in different elements of the compressed domain, such as motion vectors [56], intra-frame prediction values [13], or QDCT coefficients [53], as in the case of Stegozoa. Second, *pre-embedding techniques* embed the secret message into the raw domain, i.e., into non-compressed video stream elements like a frame's DCT coefficients [2]. Third, *post-embedding techniques* embed the secret message into the bitstream that is generated after a frame's encoding process is finished [4]. Stegozoa is the first system that adopts an intra-embedding technique based on QDCT coefficients to steganographically secure the transmission of covert data within WebRTC media streams.

Video steganalysis: The state-of-the-art steganalysis techniques focus on detecting particular steganographic schemes (e.g., motion vector-based, QDCT coefficients-based) through a combination of feature-based steganalysis and machine learning (ML) techniques [32]. ML steganalysis typically assumes that the steganalyst has full knowledge of the embedding method [25, 57]. While other proposals exist for "universal" steganalysis techniques, where the steganographic scheme being used is unknown, these are typically less effective [55]. In our work, we evaluate Stegozoa leveraging two state-of-the-art ML-based steganalysis techniques targeting QDCT-based steganography: IDFB [58] and SUPERB [14].

8 CONCLUSIONS

This paper presents Stegozoa, a novel Internet censorship evasion system that employs video steganography over WebRTC-mediated videoconferencing streams. The results of our experimental evaluation show that a careful selection of covert data embedding parameters allows Stegozoa to achieve reasonable throughput while resisting state-of-the-art traffic analysis and steganalysis attacks launched by an adversary with the ability to inspect the media content flowing through Stegozoa endpoints.

Acknowledgments: We thank the anonymous reviewers for their comments and insightful feedback. This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under projects UIDB/50021/2020 and (DAnon) CMU/TIC/0044/2021.

REFERENCES

[1] Akgül, M., and Kirlidoğ, M. Internet censorship in Turkey. *Internet Policy Review* (2015).

[2] Alavianmehr, M. A., Rezaei, M., Helfroush, M. S., and Tashk, A. A lossless data hiding scheme on video raw data robust against H.264/AVC compression. In *Proceedings of the 2nd International eConference on Computer and Knowledge Engineering (ICCKE)* (2012).

[3] Amirante, A., Castaldi, T., Miniero, L., and Romano, S. P. Janus: A General Purpose WebRTC Gateway. In *Proceedings of the Conference on Principles, Systems and Applications of IP Telecommunications* (2014).

[4] An, K., Lu, Z., Yu, F., and Luo, X. A Fast Data Hiding Algorithm for H.264/AVC Video in Bitstream Domain. In *Proceedings of the International Conference on Genetic and Evolutionary Computing* (2019).

[5] Bankoski, J., Wilkins, P., and Xu, Y. Technical overview of VP8, an open source video codec for the web. In *Proceedings of the IEEE International Conference on Multimedia and Expo* (2011).

[6] Bankoski, J., Wilkins, P., and Xu, Y. *VP8 Data Format and Decoding Guide.* IETF Internet-Draft, 2011.

[7] Barradas, D., and Santos, N. Towards a Scalable Censorship-Resistant Overlay Network Based on WebRTC Covert Channels. In *Proceedings of the 1st International Workshop on Distributed Infrastructure for Common Good* (2020).

[8] Barradas, D., Santos, N., and Rodrigues, L. DeltaShaper: Enabling Unobservable Censorship-resistant TCP Tunneling over Videoconferencing Streams. *Proceedings on Privacy Enhancing Technologies* (2017).

[9] Barradas, D., Santos, N., and Rodrigues, L. Effective Detection of Multimedia Protocol Tunneling using Machine Learning. In *Proceedings of the 27th USENIX Security Symposium* (2018).

[10] Barradas, D., Santos, N., Rodrigues, L., and Nunes, V. Poking a Hole in the Wall: Efficient Censorship-Resistant Internet Communications by Parasitizing on WebRTC. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security* (2020).

[11] Bock, K., Hughey, G., Qiang, X., and Levin, D. Geneva: Evolving Censorship Evasion Strategies. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security* (2019).

[12] Bocovich, C., and Goldberg, I. Slitheen: Perfectly imitated decoy routing through traffic replacement. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security* (2016).

[13] Cao, M., Tian, L., and Li, C. A Secure Video Steganography Based on the Intra-Prediction Mode (IPM) for H264. *Sensors* (2020).

[14] Cao, Y., Zhang, H., Zhao, X., and He, X. Steganalysis of H.264/AVC Videos Exploiting Subtractive Prediction Error Blocks. *IEEE Transactions on Information Forensics and Security* (2021).

[15] Connolly, C., Lincoln, P., Mason, I., and Yegneswaran, V. TRIST: Circumventing Censorship with Transcoding-Resistant Image Steganography. In *Proceedigns of the USENIX Workshop on Free and Open Communications on the Internet* (2014).

[16] Deibert, R., Palfrey, J., Rohozinski, R., and Zittrain, J. *Access controlled: The Shaping of Power, Rights, and Rule in Cyberspace.* the MIT Press, 2010.

[17] Dingledine, R. Obfsproxy: the next step in the censorship arms race. https://blog.torproject.org/blog/obfsproxy-next-step-censorship-arms-race, 2012. Accessed: 2021-11-19.

[18] Du, H., Liu, J., Tian, Y., and Luo, X. Cryptographic secrecy analysis of adaptive steganographic syndrome-trellis codes. *Security and Communication Networks 2021* (2021).

[19] Ensafi, R., Fifield, D., Winter, P., Feamster, N., Weaver, N., and Paxson, V. Examining How the Great Firewall Discovers Hidden Circumvention Servers. In *Proceedings of the Internet Measurement Conference* (2015).

[20] Feller, C., Wuenschmann, J., Roll, T., and Rothermel, A. The VP8 video codec - overview and comparison to H.264/AVC. In *Proceedings of the 2011 IEEE International Conference on Consumer Electronics* (2011).

[21] Feng, B., Liu, Z., Wu, X., and Lin, Y. Robust syndrome-trellis codes for fault-tolerant steganography. In *Security with Intelligent Computing and Big-Data Services* (2020).

[22] Fifield, D. *Threat modeling and circumvention of Internet censorship.* PhD thesis, EECS Department, University of California, Berkeley, 2017.

[23] Figueira, G. Stegozoa code repository. https://github.com/GabrielCFigueira/stegozoa-video, 2022.

[24] Filler, T., Judas, J., and Fridrich, J. Minimizing embedding impact in steganography using trellis-coded quantization. In *Media Forensics and Security II* (2010).

[25] Ghamsarian, N., Schoeffmann, K., and Khademi, M. Blind MV-based video steganalysis based on joint inter-frame and intra-frame statistics. *Multimedia Tools and Applications* (2020).

[26] Grozev, B., Politis, G., Ivov, E., Noel, T., and Singh, V. Experimental Evaluation of Simulcast for WebRTC. *IEEE Communications Standards Magazine* (2017).

[27] Grozev, B., Politis, G., Ivov, E., Noel, T., and Singh, V. Experimental evaluation of simulcast for webrtc. *IEEE Communications Standards Magazine 1*, 2 (2017), 52–59.

[28] Huynh-Thu, Q., and Ghanbari, M. The accuracy of psnr in predicting video quality for different video scenes and frame rates. *Telecommunication Systems 49*, 1 (2012), 35–48.

[29] Intel Corp. *Intel® Advanced Vector Extensions Programming Reference*, 2011.

[30] Jansen, B., Goodwin, T., Gupta, V., Kuipers, F., and Zussman, G. Performance evaluation of webrtc-based video conferencing. *ACM SIGMETRICS Performance Evaluation Review 45*, 2 (2018).

[31] Jitsi. This is what end-to-end encryption should look like! https://jitsi.org/blog/e2ee/, 2020. Accessed: 2021-11-19.

[32] Ker, A. D., Bas, P., Böhme, R., Cogranne, R., Craver, S., Filler, T., Fridrich, J., and Pevný, T. Moving Steganography and Steganalysis from the Laboratory into the Real World. In *Proceedings of the First ACM Workshop on Information Hiding and Multimedia Security* (2013).

[33] Khattak, S., Elahi, T., Simon, L., Swanson, C. M., Murdoch, S. J., and Goldberg, I. Sok: Making sense of censorship resistance systems. *Proceedings on Privacy Enhancing Technologies 2016*, 4 (2016).

[34] King, G., Pan, J., and Roberts, M. E. Reverse-engineering censorship in China: Randomized experimentation and participant observation. *Science* (2014).

[35] Knockel, J., Crete-Nishihata, M., Ng, J. Q., Senft, A., and Crandall, J. R. Every Rose Has Its Thorn: Censorship and Surveillance on Social Video Platforms in China. In *Proceedings of the 5th USENIX Workshop on Free and Open Communications on the Internet* (2015).

[36] Kohls, K., Holz, T., Kolossa, D., and Pöpper, C. Skypeline: Robust Hidden Data Transmission for VoIP. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security* (2016).

[37] Li, S., Schliep, M., and Hopper, N. Facet: Streaming over Videoconferencing for Censorship Circumvention. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society* (2014).

[38] Liu, Y., Liu, S., Wang, Y., Zhao, H., and Liu, S. Video steganography: A review. *Neurocomputing* (2019).

[39] Mahy, R., Matthews, P., and Rosenberg, J. Traversal using relays around NAT (TURN): Relay extensions to session traversal utilities for NAT (STUN) - rfc 5766. https://tools.ietf.org/html/rfc5766, 2010.

[40] McPherson, R., Houmansadr, A., and Shmatikov, V. CovertCast: Using Live Streaming to Evade Internet Censorship. *Proceedings on Privacy Enhancing Technologies* (2016).

[41] Minaei, M., Moreno-Sanchez, P., and Kate, A. MoneyMorph: Censorship Resistant Rendezvous using Permissionless Cryptocurrencies. *Proceedings on Privacy Enhancing Technologies 2020*, 3 (2020), 404–424.

[42] Mohajeri Moghaddam, H., Li, B., Derakhshani, M., and Goldberg, I. Skypemorph: Protocol Obfuscation for Tor Bridges. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security* (2012).

[43] National Vulnerability Database. CVE-2020-13901 Detail. https://nvd.nist.gov/vuln/detail/CVE-2020-13901, 2020. Accessed: 2021-11-19.

[44] Neufeld, A., and Ker, A. D. A Study of Embedding Operations and Locations for Steganography in H.264 Video. In *Media Watermarking, Security, and Forensics 2013* (2013).

[45] Niaki, A. A., Cho, S., Weinberg, Z., Hoang, N. P., Razaghpanah, A., Christin, N., and Gill, P. ICLab: A Global, Longitudinal Internet Censorship Measurement Platform. In *Proceedings of the IEEE Symposium on Security and Privacy* (2020).

[46] Perez, S. Twitter's doubling of character count from 140 to 280 had little impact on length of tweets. https://techcrunch.com/2018/10/30/twitters-doubling-of-character-count-from-140-to-280-had-little-impact-on-length-of-tweets/, 2021. Accessed: 2021-11-19.

[47] Rambert, R., Weinberg, Z., Barradas, D., and Christin, N. Chinese wall or swiss cheese? Keyword filtering in the great firewall of china. In *Proceedings of the 30th The Web Conference* (2021).

[48] Ramesh, R., Raman, R. S., Bernhard, M., Ongkowijaya, V., Evdokimov, L., Edmundson, A., Sprecher, S., Ikram, M., and Ensafi, R. Decentralized Control: A Case Study of Russia. In *Proceedings of the Network and Distributed Systems Security Symposium* (2020).

[49] Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (2004), 600–612.

[50] Wells, D., and J.Bingham. Let's Reverse Engineer Discord. https://medium.com/tenable-techblog/lets-reverse-engineer-discord-1976773f4626, 2020. Accessed: 2021-11-19.

[51] Westerlund, M., and Wenger, S. RTP Topologies - RFC 7667. https://datatracker.ietf.org/doc/html/rfc7667, 2015.

[52] Wikipedia, the free encyclopedia. Wikipedia:Size in volumes. https://en.wikipedia.org/wiki/Wikipedia:Size_in_volumes, 2021. Accessed: 2021-11-19.

[53] Xie, P., Zhang, H., You, W., Zhao, X., Yu, J., and Ma, Y. Adaptive VP8 Steganography Based on Deblocking Filtering. In *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security* (2019).

[54] Xue, D., Ramesh, R., S, V. S., Evdokimov, L., Viktorov, A., Jain, A., Wustrow, E., Basso, S., and Ensafi, R. Throttling Twitter: An emerging censorship technique in Russia. In *Proceedings of the 21st ACM Internet Measurement Conference* (2021).

[55] Zhai, L., Wang, L., and Ren, Y. Universal Detection of Video Steganography in Multiple Domains Based on the Consistency of Motion Vectors. *IEEE Transactions on Information Forensics and Security* (2020).
[56] Zhang, H., Cao, Y., and Zhao, X. Motion vector-based video steganography with preserved local optimality. *Multimedia Tools and Applications* (2016).
[57] Zhang, H., Cao, Y., and Zhao, X. A Steganalytic Approach to Detect Motion Vector Modification Using Near-Perfect Estimation for Local Optimality. *IEEE Transactions on Information Forensics and Security* (2017).
[58] Zhang, H., You, W., and Zhao, X. A Video Steganalytic Approach Against Quantized Transform Coefficient-Based H.264 Steganography by Exploiting In-Loop Deblocking Filtering. *IEEE Access* (2020).
[59] Zolfaghari, H., and Houmansadr, A. Practical censorship evasion leveraging content delivery networks. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security* (2016).