Poking a Hole in the Wall: Efficient Censorship-Resistant Internet Communications by Parasitizing on WebRTC

Diogo Barradas, Nuno Santos, Luís Rodrigues, Vítor Nunes
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
{diogo.barradas,nuno.m.santos,ler,vitor.sobrinho.nunes}@tecnico.ulisboa.pt

ABSTRACT

Many censorship circumvention tools rely on trusted proxies that allow users within censored regions to access blocked Internet content by tunneling it through a covert channel (e.g,. piggybacking on Skype video calls). However, building tools that can simultaneously (i) provide good bandwidth capacity for accommodating the typical activities of Internet users, and (ii) be secure against traffic analysis attacks has remained an open problem and a stumbling block to the practical adoption of such tools for censorship evasion.

We present Protozoa, a censorship-resistant tunneling tool featuring both high-performing covert channels and strong traffic analysis resistance. To create a covert channel, a user only needs to make a video call with a trusted party located outside the censored region using a popular WebRTC streaming service, e.g., Whereby. Protozoa can then covertly tunnel all IP traffic from unmodified user applications (e.g., Firefox) through the WebRTC video stream. This is achieved by hooking into the WebRTC stack and replacing the encoded video frame data with IP packet payload, while ensuring that the payload of the WebRTC stream remains encrypted, and the stream's statistical properties remain in all identical to those of any common video call. This technique allows for sustaining enough throughput to enable common-use Internet applications, e.g., web browsing or bulk data transfer, and avoid detection by state-of-the-art traffic analysis attacks. We show that Protozoa is able to evade state-level censorship in China, Russia, and India.

CCS CONCEPTS

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

KEYWORDS

Censorship circumvention; Traffic analysis; WebRTC

ACM Reference Format:
Diogo Barradas, Nuno Santos, Luís Rodrigues, Vítor Nunes. 2020. Poking a Hole in the Wall: Efficient Censorship-Resistant Internet Communications by Parasitizing on WebRTC. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3372297.3417874

CCS '20, November 9–13, 2020, Virtual Event, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7089-9/20/11…$15.00
https://doi.org/10.1145/3372297.3417874

1 INTRODUCTION

State-level censors are known to apply techniques to prevent free access to information on the Internet. In fact, many countries have deployed a vast censorship apparatus to exercise control over available content, namely China [41], Russia [60], Iran [1], Bangladesh [52], India [82], Thailand [26], or Syria [11]. For instance, amidst the recent Coronavirus outbreak, the chinese government has shut down news websites [31] and instructed chinese social media platforms to censor references and keywords related to the infection [10, 64], in an attempt to handle the sharing of negative coverage within and outside the country. This control can be enforced through various techniques, such as keyword-based filters [80, 81], image filters [43], social platform monitoring [32, 42], and even the entire blocking of Internet destinations [57] or selected protocols [18].

To evade censorship, many circumvention tools have been proposed for enabling users to freely access/share information on the Internet [39, 69]. Typically, such tools rely on covert channels to allow for the stealthy transmission of sensitive data through an apparently innocuous carrier medium [39, 84], for instance a multimedia streaming carrier application like Skype. The key idea is to encode the covert data in such a way that an adversary capable of inspecting the full packet exchange cannot distinguish between a legitimate transmission and one that subliminally carries covert data. This general approach, which we call *multimedia covert streaming*, can typically be achieved in two ways: i) by entirely mimicking the carrier's network-level protocols [51] (*media protocol mimicking*), or ii) by embedding the covert data into the video (or audio signal) feed of the carrier application in the course of a regular video call [3, 36, 44, 46, 50] (*raw media tunneling*).

Given that all the carrier's traffic is encrypted, one way for an adversary to counter potential covert channels is to blatantly block all traffic generated by the carrier application. This method, however, can bring harmful side-effects even for a state-level adversary. In fact, considering how instrumental many media streaming applications are for the tissue of economic and social interactions within censored regions, the costs of shutting down popular applications can be overwhelming and erode even further the state's reputation in the eyes of its international peers. Leveraging on essential streaming applications can then serve as a strong deterrent for the enforcement of blocking policies, and constitutes the key insight that favors the effectiveness of multimedia covert streaming [21].

Nevertheless, an adversary can employ a second class of techniques based on traffic analysis. Essentially, it involves probing into the censored network region, inspecting the traffic generated by a presumable carrier application, and looking for discrepancies in the traffic (e.g., abnormal patterns) that might signal the presence of covert channels. Hence, it follows that an effective tool for multimedia covert streaming must be able to resist these kinds of attacks by exhibiting traffic patterns that will ideally be indistinguishable from legitimate traffic. In other words, when picking from two sampled flows – one legitimate flow and one crafted flow containing covert data – the adversary must not be able to distinguish them but by random guessing, i.e., with 50% chance of success.

Unfortunately, the existing tooling support for multimedia covert streaming is quite bleak. Several studies revealed that the presence of modulated covert traffic can be detected solely based on the analysis of traffic features such as packet sizes and packet inter-arrival times [4, 27]. In fact, machine learning (ML)-based traffic analysis techniques can effectively detect small changes in the packet frequency distributions caused by the embedding of covert data inside carrier video streams, namely deviations in the packet sizes and inter-packet arrival times when compared with legitimate traffic. Most existing tools fail this test and are prone to be detected with high accuracy rates [4]. While some tools like DeltaShaper [3] can tolerate detection to some degree, they do so at the expense of reducing the amount of covert data embedded into the cover video stream, severely limiting the covert channel bandwidth capacity that can be attained. For instance, in DeltaShaper, the maximum achievable throughput is only 7 Kbps, which is clearly insufficient for sustaining the traffic generated by common Internet users, e.g., interactive web browsing, media streaming, or bulk data transfers.

This paper presents Protozoa, a new multimedia covert streaming tool that provides good performance for the covert transmission of arbitrary IP traffic while featuring strong resistance to detection when subjected to ML-based traffic analysis by a state-level adversary. In particular, Protozoa allows an Internet user (*client*) located in a censored region to access blocked content by leveraging the help of a trusted user in the free region who will act as a *proxy* on the client's behalf. Protozoa enables then to create a bidirectional *covert tunnel* between both endpoints. Henceforth, the client can start browsing the web freely: the local IP traffic will be transparently redirected through the covert tunnel to its final destination host in the free region, e.g., YouTube. This local application is not restricted to a browser: Protozoa can tunnel IP traffic from arbitrary unmodified applications, e.g., email or BitTorrent clients.

Protozoa advances the state-of-the-art by incorporating two new ideas in its design. First, to enable the transmission of covert traffic, it uses web streaming applications based on WebRTC which are very popular and widely disseminated. Concretely, to create a covert tunnel, all that two users – client and proxy – need to do is to establish a video call using a WebRTC-enabled web streaming website, such as Whereby (`https://whereby.com`). Protozoa uses the video call's associated WebRTC media stream to tunnel covert IP traffic between both endpoints. Second, to encode the covert signal into the carrier stream, Protozoa introduces a technique named *encoded media tunneling*, which allows for boosting the capacity of covert channels while offering strong resistance to traffic analysis attacks. It consists of embedding the covert data into *encoded video frames*, i.e., right after the lossy compression has been applied by the video codec. This mechanism is implemented by modifying the WebRTC stack of Protozoa's Chromium browser component.

We extensively evaluated our Protozoa prototype both through a set of microbenchmarks resorting to media sessions established over Whereby, and by testing it in various realistic usage scenarios and workloads. Our results showed that, under normal network

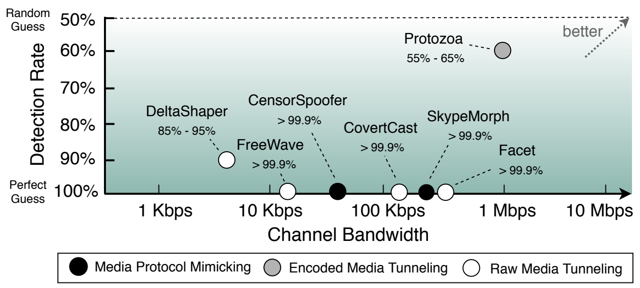

Figure 1: Design space of multimedia covert streaming tools along two dimensions: covert channel capacity (X-axis), and traffic analysis resistance (Y-axis). Darker shades indicate increasing chances of detection, i.e., tools are more insecure. Protozoa outperforms the existing systems in both dimensions. A detailed analysis of this plot is found in Section 9.1.

conditions, Protozoa can deliver covert channel bandwidth capacities in the order of 1.4Mbps and channel efficiency of 98.8%, while providing strong resistance to traffic analysis using state-of-the-art ML-based techniques [4]. As illustrated in Figure 1, these results represent a significant departure over existing media covert streaming techniques, dramatically improving the detection rate of the best performing tool, i.e., Facet, from >99.0% to 55%-65%, i.e., from nearly perfect guess to very close to random guess, while simultaneously improving covert channel bandwidth by 3×. Additional experiments show that Protozoa can withstand a number of active network perturbations without jeopardizing its resistance against traffic analysis or breaking the covert channel connection.

To assess the portability of Protozoa, we also tested our system on alternative WebRTC services: `appr.tc` and `coderpad.io`. Our results showed that it can consistently achieve similar throughput and traffic analysis resistance properties when used over different applications, which makes it useful in scenarios where specific applications are blocked (e.g., `appr.tc` is blocked in China). Lastly, we tested Protozoa in the wild by deploying Protozoa in three regions that are known to enforce Internet censorship through several means: the Great Firewall of China (GFW) apparatus in China, and ISP censorship in Russia and India. We performed several experiments from servers deployed in each of these regions. First, we accessed blacklisted content without using Protozoa, and checked that it was indeed inaccessible. Then, using Protozoa, we were able to access this content, showing that our system can successfully breach through existing censorship mechanisms deployed in these regions, and provide free access to blocked Internet content.

2 THREAT MODEL

The general system model of a *multimedia covert streaming* (MCS) tool is illustrated in Figure 2. It represents two users, one acting as *client* (Alice), and a second acting as *proxy* (Bob). The client is located in a censored region controlled by a state-level adversary, and the proxy is based in a free Internet region. The adversary is able to observe, store, interfere with, and analyze all the network flows within its jurisdiction, and block the generalized access to remote Internet services, such as CNN, Facebook, or Twitter, by the residents in the censored region. The censorship policies can be based on the IP address or the domain name of the target destination,

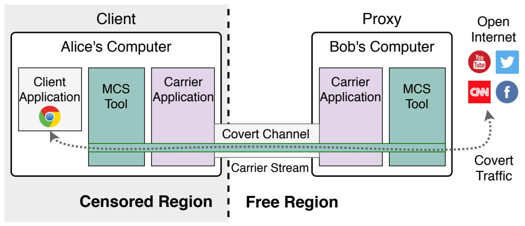

Figure 2: System model of a multimedia covert streaming tool for censorship-resistant Internet communications.

the protocol used in the communication (e.g., BitTorrent or Tor), or blacklisted content (e.g., through keyword and image filtering).

An MCS tool aims at enabling the client to overcome the communication restrictions enforced by the adversary by leveraging (i) the cooperation of a proxy located in the free region operated by a trusted Internet user (Bob), and (ii) a carrier application consisting of an encrypted video-streaming service (e.g., Skype) whose traffic the adversary authorizes to cross the boundaries of the censored region. Both users – client and proxy – must run the MCS software on their local computers to create a covert tunnel through the media stream managed by the carrier application. This tunnel will allow the client to contact remote hosts on the open Internet.

To defeat an MCS tool, the adversary can use deep packet inspection for pinpointing traffic indicators that lead to the detection of a covert channel. To increase its chances for successful detection, it may apply statistical traffic analysis techniques over collected network traces [4]. The adversary may also launch indiscriminate active network attacks aimed at perturbing the correct behavior of covert channels lurking under seemingly legitimate flows while ensuring that legitimate flows maintain a reasonable quality.

However, the adversary will seek only to rapidly disrupt and tear down those flows which are suspected of carrying covert channels, and it will refrain from blocking the carrier application altogether if such an application is reckoned to provide an important service to the population. The adversary is also deemed to be computationally bounded, and unable to decrypt encrypted traffic generated by the carrier application. The adversary's control is also limited to the network: it has no access to the persistent or volatile state of clients, proxies, or carrier application provider, and enjoys no privileges over the software that can be executed by each party.

3 PARASITIZING ON WEBRTC STREAMS

In designing Protozoa, we elected WebRTC media streams as the carrier medium for deploying practical, efficient, and secure covert channels. WebRTC [47] is a W3C standardization initiative for protocols and APIs for enabling secure real-time communication between web browsers. All major browsers have built-in WebRTC implementations enabling the generalized use of this technology.

WebRTC creates new opportunities for building MCS services that can simultaneously be widely available and easy to use. By acting at this layer, any WebRTC-powered application can be transparently used for covert data transmission. WebRTC has been currently adopted by numerous services that integrate real-time communication capabilities[1]. This integration has been greatly facilitated by the simplicity of the JavaScript WebRTC API [28]. The generalized usage of web conferencing for professional dealings, in particular, will make it very deterring for a state-level adversary to block all WebRTC traffic due to the extensive collateral damage to the country's own sustainability [21]. This profusion of WebRTC services also gives Protozoa users a great deal of flexibility and options to choose from when it comes to picking the carrier medium for covert transmission. Creating covert sessions is in itself a user-friendly operation since establishing a Protozoa connection is in no way different from making a video call on a web streaming application (including the process of joining a chatroom URL).

Given that a widely-used WebRTC stack is openly available (in the Chromium web browser), we have full access to the WebRTC video streaming pipelines which allows us to develop a new efficient and secure covert data encoding technique named *encoded media tunneling*. In existing raw media tunneling tools, the covert data is encoded as pixels of the carrier video frames. Unfortunately, given that the raw input signal undergoes lossy compression by the video codec of the carrier application, a significant amount of redundancy must be included to enable the covert data recovery by the receiver which degrades the utilization efficiency of the covert channel. Moreover, using up all possible pixel area of the carrier frames for encoding covert data reduces the video codec's compression efficiency, which increases the network packet sizes and deforms the packet size frequency distribution when compared to that of legitimate flows. Because this discrepancy can be detected by an adversary, the covert channel tools must be parsimonious at using up the video stream's pixel area thereby throttling even further the bandwidth capacity of the covert channel. Encoded media tunneling overcomes these limitations and allows for boosting the capacity of covert channels while offering strong resistance to traffic analysis attacks. Next, we describe our technique as we present Protozoa.

4 PROTOZOA

This section presents Protozoa, a system that provides covert channels over WebRTC media sessions[2]. Next, we present its architecture and then describe its most relevant technical details.

4.1 Architecture

Figure 3 depicts the architecture of Protozoa. It follows the system model of a general multimedia covert streaming (MCS) tool presented in Figure 2. In Protozoa, the carrier application consists of a web application that uses the WebRTC framework for providing a point-to-point live streaming service between its users, e.g., Whereby. Typically, such an application consists of a backend that handles the signaling and session establishment of video calls between participants, and client-side JavaScript & HTML code that initiates video calls and manages the video transmission via the WebRTC API provided by the browser. The resulting media stream will be used by Protozoa as the carrier for a covert channel.

The MCS tool (see Figure 2) is materialized by the Protozoa software bundle which targets the Linux platform, and it is set up differently to work as client or proxy by two participating parties.

[1]There are many different WebRTC-based web streaming applications, such as social applications like Whereby or Facebook Messenger, the gaming-focused chat Discord, professional video conferencing services like Amazon Chime and Slack, remote coding interview software like Coderpad, or even health monitoring through Vidyo.

[2]"Protozoa" alludes to parasitic biological organisms which feed on other organisms.

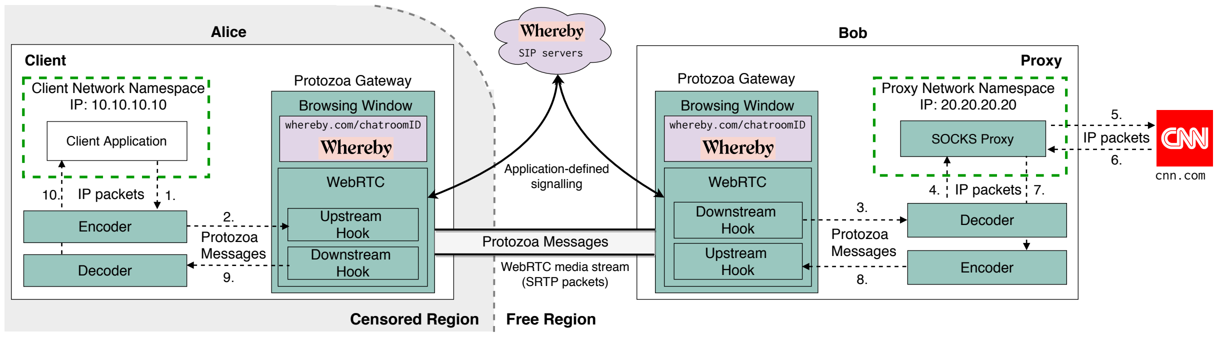

Figure 3: Architecture of Protozoa: The components of our system are highlighted in a darker shade.

The client will be able to execute unmodified IP applications whose traffic can seamlessly be routed through the WebRTC covert channel by the proxy to destination hosts anywhere in the free Internet region. Figure 3 shows Alice (client) executing a local application for accessing cnn.com using Bob's computer as proxy.

The Protozoa software bundle comprises four main components: gateway server, encoder service, decoder service, and SOCKS proxy server. Both client and proxy run the gateway server and the encoder and decoder services. The proxy also runs the SOCKS proxy server. Both endpoints leverage network namespaces for transparent interception and manipulation of IP packets. All these components cooperate in forwarding covert IP traffic by implementing three cross-cutting functional layers. Next, we explain how these layers operate by introducing them in a bottom-up fashion.

1. WebRTC layer: This layer is responsible for the setup and management of a point-to-point WebRTC covert channel between two parties, and it is implemented by the gateway servers running on each communication endpoint. The covert channel is piggybacked on a WebRTC media stream instantiated by a carrier web streaming application. This channel supports full-duplex bidirectional communication and exchanges Protozoa messages defined in a specific format. These messages can contain arbitrary IP payload. The gateway server is built out of a modified Chromium browser, and instrumented with the placement of two hooks – *upstream* and *downstream* – in the WebRTC stack. We leverage Chromium's functionality to provide a web browsing UI and runtime environment which will allow for the execution of the client-side WebRTC application code. The hooks intercept the WebRTC streams so as to replace the payload of the WebRTC video frames with covert Protozoa messages. The gateway server opens two pipes for receiving upstream and downstream messages from the codec layer.

2. Codec layer: This layer performs two complementary encoding and decoding operations. The former is responsible for encoding streams of IP packets generated by local networked applications. Packets are read from a libnetfilter queue [56], and encapsulated into Protozoa messages, which are forwarded to the local gateway server and then delivered to the remote endpoint. Decoding performs the reverse operation, i.e., reads incoming Protozoa messages from the local gateway server, extracts the enclosed IP packets, and writes them to a raw socket to be routed to their final destination. These operations are coordinated by the encoder and decoder at both endpoints to sustain simultaneously two IP packet flows, i.e., upstream and downstream. Internally, these components maintain packet and message queues, and implement packet fragmentation and reassembly so as to efficiently use the covert channel capacity.

3. SOCKS layer: This layer enables the exchange of IP packets between the networked applications running on the client, and a remote Internet host through a SOCKS v5 proxy server running on the proxy. This is achieved by the use of Linux's network namespaces and configuration of iptables. Namespaces are implemented by the Linux kernel and allow for the creation of virtual network interfaces. In our context, we use namespaces for creating a virtual network environment for the client application and a second one for the SOCKS proxy server. Each environment features a virtual network interface that is exposed to the local processes with a specific IP address, e.g., 10.10.10.10, or 20.20.20.20, respectively (see Figure 3). Protozoa then configures the local iptables so as to route all (upstream) IP packets with destination address 20.20.20.20 to the namespace of the proxy, and all (downstream) IP packets with destination address 10.10.10.10 to the namespace of the client. Thus, by configuring a client application to use the IP address 20.20.20.20 as SOCKS proxy, all its IP connections will be transparently delivered to the SOCKS server proxy, which in turn will deliver the packets to its remote destination. So, for instance, the web request to cnn.com depicted in Figure 3 can be performed by running the curl command on a Linux terminal as follows:

```
$ ip netns exec PROTOZOA_ENV_CLIENT
  curl -x socks5h://20.20.20.20:1080 https://cnn.com
```

This means that, in order to use Protozoa's covert tunnels, the user must configure the client application to use a SOCKS proxy. For client applications that do not natively support the use of SOCKS proxy servers, the user can use an additional tool, proxychains [62], which provides the client application with SOCKS proxy support.

4.2 Execution Workflow

This section describes the execution workflow involved in a complete communication using Protozoa covert tunnels. Using the example depicted in Figure 3, we describe the full message exchange sequence that takes place in order for Alice to fetch a web page from cnn.com through a WebRTC covert tunnel facilitated by Bob, who is an individual volunteer trusted by Alice. This tunnel is created through a WebRTC video call between Alice and Bob using Whereby in the course of a Protozoa *covert session*, which is divided into two stages: covert session establishment, and covert data transmission. Figure 4 represents the messages exchanged.

1. Covert session establishment: The covert tunnel is set up between client and proxy, requiring both participants to agree on a common rendezvous point for a WebRTC media connection. In our example, Alice and Bob use the web browsing interface of their Protozoa gateway to join a common video chatroom. They begin by bootstrapping the Protozoa software: Alice in client mode (A1), and Bob in proxy mode (B1). Then, Alice accesses `whereby.com`, creates a password-protected chatroom, and obtains the chatroom URL (A2). Similarly to using alternative MCS tools [3, 46], Alice uses an out-of-band channel, e.g., email, social network web site, or mobile app (e.g., Whatsapp) to share the chatroom URL and password with Bob (A3 and B2). Both users can now join the chatroom (A4 and B3) and initiate a video call by feeding a carrier video stream from their local cameras or (optionally) from a prerecorded video; this video will be replaced by covert payload. As the WebRTC video stream is initiated, Protozoa hooks into it, and sets up the covert tunnel.

2. Covert data transmission: Once the covert tunnel is ready, Alice can access remote Internet services. For instance, to access `cnn.com`, Alice can simply run the `curl` command listed in the section above to issue an HTTP GET request to `cnn.com`. The IP traffic generated from this request will be transparently tunneled through the covert channel. Protozoa will continuously stream video until the termination of the covert session, even when there is no covert traffic to be transmitted; in this case, dummy payload (chaff) is sent.

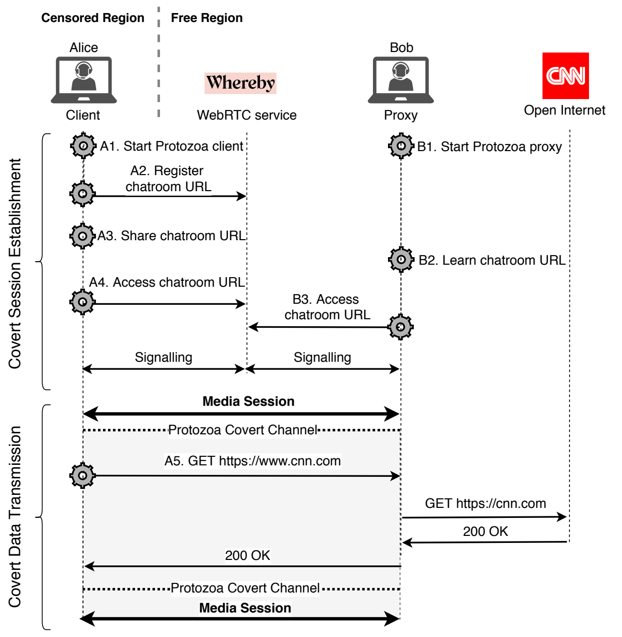

Figure 4: Covert session: gear symbol denotes user actions.

4.3 Network-level Security of Covert Sessions

At covert data transmission, standard WebRTC ensures that all exchanged packets are integrity-protected and the message payload containing sensitive video data is encrypted. Thus, an adversary cannot read its content, or modify it without detection. Nevertheless, we must ensure that the covert session has been securely established. In particular, an adversary may attempt a man-in-the-middle or an impersonation attack during the session negotiation phase (see Figure 4) enabling it to decrypt the message payload and inspect the covert data. To prevent these attacks, Protozoa leverages the security mechanisms implemented by WebRTC and by the carrier WebRTC web streaming application, namely the following ones:

a) HTTPS: Client and proxy run client-side code of the WebRTC web application which connects to its backend servers through HTTPS. This means that all messages involving interactions with the backend (i.e., A2, A4, B3) will be exchanged over TLS-enabled secure channels. In particular, this prevents an adversary from obtaining the URL that would allow it to join the chatroom, or to mount a MITM by advertising different URLs to client and proxy.

b) SIP / DTLS-SRTP: To establish a media session, WebRTC leverages the Session Initiation Protocol (SIP) to signal one endpoint's intention (e.g., the client's) to connect to its corresponding peer (e.g., the proxy). This protocol involves the communication between each endpoint (client/proxy) and a SIP server run by the WebRTC application provider (see Figure 3). This server is used to exchange media session parameters between endpoints, and it is combined with the DTLS-SRTP protocol [48, 83] to perform an initial key exchange so as to offer protection against man-in-the-middle attacks. The WebRTC application provider also runs a STUN server which helps the endpoints located behind a NAT to determine their respective public (NAT'ed) IP addresses, and share them with their peers. To ensure that the media sessions between endpoints are not hijacked and pointed to different IP locations, the connection attempts to the IP addresses of target endpoints are secured by a MAC, which is computed using the key exchanged in the signaling channel [63]. Once a WebRTC session has been established, WebRTC leverages the Secure Real-time Transport Protocol (SRTP) [5, 83] for encrypting and authenticating the content of the media in transit.

4.4 Encoded Media Tunneling

Protozoa uses the video streams generated by client and proxy as a medium for carrying covert IP packet data in both directions. To this end, we employ a new approach named *encoded media tunneling*. Similar to existing raw media tunneling techniques [3, 46], our method replaces carrier video information with a covert message. However, instead of replacing the pixels of the raw input video, it replaces the bits of the encoded video signal, i.e., *after* the input video has been compressed by the WebRTC video codec. This technique helps increase not only the capacity of the channel but also its resistance to traffic analysis. In our system, it is implemented by instrumenting the WebRTC stack of the Protozoa gateway.

Upstream and downstream hooks: Figure 5 illustrates the WebRTC stack as it is implemented in the Protozoa gateway. It is based on the WebRTC stack bundled into the Chromium browser. The WebRTC stack contains a built-in codec (VP8), which processes the video signal of local web applications that use the WebRTC API. To access the video frames generated by the WebRTC application and implement encoded media tunneling, the WebRTC stack includes two hooks that can intercept the processing of the media stream in different directions, i.e., upstream or downstream. The *upstream hook* intercepts outgoing frame data, i.e., from a local camera device to the network. It is placed after the raw video signal has been processed by the video engine, and right before the frame data is passed

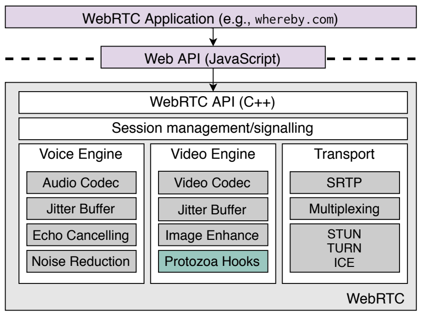

Figure 5: WebRTC software stack and Protozoa's hooks.

over to the transport layer where SRTP packets are created, and sent to the network. The *downstream hook* intercepts incoming frame data, i.e., from the network to the local screen. It is placed right after the transport layer has finished reconstructing an encoded frame sent in multiple network packets, and right before handing it over to the video engine to be decoded and rendered on screen. We strategically placed hooks in the WebRTC stack in order to manipulate a special data structure, named *encoded frame bitstream partitions* (EFBP), where we can embed Protozoa messages.

Using EFBP as a covert data mule: To help understand how the covert data is embedded into the carrier frame data, Figure 6 depicts the format of the SRTP packets, which is the means through which video data is exchanged. The bulk of SRTP packet space is reserved for the transmission of media payload in the field named *encoded frame bitstream* (EFB). This field contains the bits of an encoded (compressed) video frame as it is generated by VP8, the default WebRTC codec. An encoded frame contains a small (3-10 bytes) uncompressed header, and two partitions which carry compressed bitstreams containing actual carrier video data. We call EFBP the contiguous space occupied by these partitions. The EFBP has five extremely interesting properties for our problem domain:

(1) The EFBP consists of a blob of bits that contains the actual video data of the carrier stream. Since this information is irrelevant for us, we can effectively use this field for carrying covert data by overwriting it with covert data.
(2) The EFBP, once it is generated by VP8, is no longer modified by the WebRTC downstream pipeline. This means that the covert data bits placed in this field are not going to be corrupted, e.g., due to compression or other destructive operations, before being sent to the network.
(3) The EFBP will be used as payload of the SRTP packet, and contains no relevant metadata that influences the transport layer logic, hence, modifying this field will not disturb the normal functioning of packet transfer over the network.
(4) The EFBP, prior to being assembled into SRTP packets, will be encrypted, and protected with authentication markers. This means that covert data placed inside the EFBP will be encrypted and integrity-protected for free.
(5) The EFBP will be encrypted resorting to a stream cipher that preserves the plaintext size, therefore, embedded covert messages do not change the size of encrypted EFBPs.

For all these reasons, we use the EFBP as a free storage space for transmitting covert data in the form of Protozoa messages.

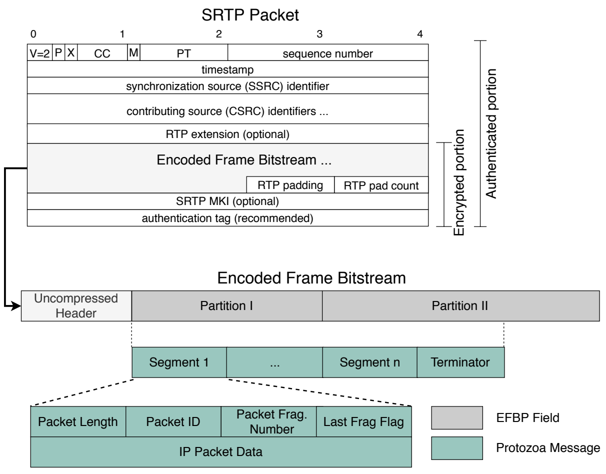

Figure 6: Format of SRTP packets: Protozoa replaces the EFBP payload containing carrier video bits with covert data.

Protozoa message transmission: To embed covert IP packets inside an EFBP, Protozoa uses the message format depicted in Figure 6. A single message consists of multiple segments followed by a Terminator (i.e., a zero-length segment) which delimits the EFBP area occupied by covert data. Each segment carries an entire IP packet or a packet fragment. IP packet fragmentation may be required at the sender's endpoint shall the next available IP packet be larger than the available EFBP space on the frame; this process will help to use the covert channel in its maximal capacity. As a result, a covert IP packet can be transmitted in a single segment or span across multiple segments. Each segment has a small header that allows the receiving endpoint to reassemble IP packet fragments.

Message transmission works as follows. For every new frame generated by the video engine, there is an opportunity for sending a new message, which causes the upstream hook to be executed and given access to the encoded frame. The hook accesses the EFBP data structure, checks its size, and tells the encoder service how much free space exists in the frame for sending covert data. The hook waits for the encoder to assemble a new message containing locally queued IP packets. Then, the hook copies it into the EFBP and returns, letting the WebRTC pipeline to proceed with its normal execution until the resulting SRTP packets are transmitted. At the receiving endpoint, the reverse operation is performed. Whenever an encoded frame is reassembled, the downstream hook is executed and extracts the Protozoa message from the EFBP field. This message is sent to the local decoder service, which reassembles the ingress IP packets and forwards them to their rightful destination.

4.5 Prevent Video Decoding Malfunction

While it is possible to fully replace the content of the EFBP field, the undisciplined corruption of a frame bitstream can prevent the video decoder in the WebRTC downstream pipeline from correctly decoding video frame data at the receiver's endpoint. In fact, we empirically verified that in such situations, WebRTC triggers congestion control mechanisms in the downstream pipeline for ensuring the reception of video. In particular, it advertises a Picture Loss Indication (PLI) in the accompanying RTCP control channel [61], aimed at requesting the retransmission of a key frame upon being unable to decode the corrupted frame data. In particular, VP8 produces two different types of encoded frames. Key frames can be decoded without any reference to previous frames and provide seeking points within a video stream. Delta frames are encoded with reference to the prior key frame and ensuing frames. By advertising PLIs and sending key frames upon detecting corrupted frames, the resulting traffic patterns produced by WebRTC applications would make Protozoa vulnerable to traffic analysis.

To overcome this problem, the downstream hook feeds the WebRTC video decoder with a pre-recorded sequence of valid encoded frames instead of the corrupted frames received over the network. Since the encoded frames may either be key frames or delta frames, the downstream hook uses the uncompressed header information kept intact after covert data embedding (see Figure 6) to decide which type of frame and corresponding resolution (e.g., 640x480) should be provided to the native WebRTC video decoder. Then, it restores the corrupted bitstream with a bitstream of a valid frame. This allows us to establish a covert channel where the size of egress frames on the upstream pipeline is maintained, and to deliver the decoder valid data so that it does not trigger congestion control.

4.6 Implementation and Optimizations

We developed a Protozoa prototype [2] by writing approximately 3,000 lines of C++ code. This includes the instrumentation of the native WebRTC codebase of the Chromium browser v79.0.3945.117, a stable release from January 2020. Protozoa requires the proper establishment of a WebRTC video session for embedding data into encoded frames sent over the wire. To this end, WebRTC must be able to access a video feed that can be directly obtained from the physical camera device available in the system. Alternatively, it is possible to set up a camera emulator by using the `v4l2loopback` kernel module [70] and feed recorded video with the help of the `ffmpeg` video library [20]. In Section 6, we leverage the latter method for evaluating Protozoa in light of different video profiles.

Fine-tuning of IP packet queues: We performed an important optimization related to the size of the packet queues maintained internally by Protozoa's encoding service. Specifically, we refer to the queue that holds intercepted IP packets generated in the upstream pipeline network namespace. A typical rule-of-thumb for managing packet queues suggests the parameterization of a buffer size according to the following formula: *Buffer Size ≥ RTT * Channel Bandwidth* [49]. According to our experimental results, we conservatively assume that each packet in the queue has a size equal to the MTU. Additionally, we empirically verify that the round-trip-time experienced by our system is ≈200ms when connected in a LAN network, i.e., when the latency between WebRTC hosts is sub-millisecond. Configuring our packet queue with the above parameters yields a queue size of 24 packets for the 200ms round-trip-time experienced by Protozoa and a bandwidth of approximately 1.4Mbps achieved when sending video at 640x480 resolution.

5 EVALUATION METHODOLOGY

This section describes our evaluation methodology for assessing the quality and performance of our Protozoa prototype.

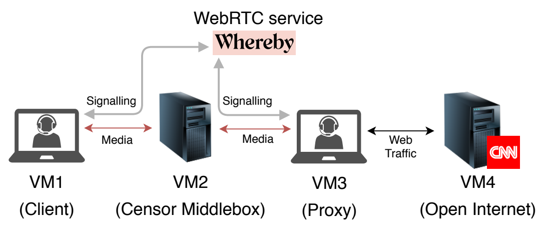

Figure 7: Laboratory setup.

5.1 Evaluation Goals and Approach

The goal of our experiments is twofold: i) evaluate the performance of Protozoa's covert channel in face of different network conditions, and ii) assess the ability of our system to resist against detection from an adversary able to perform statistical traffic analysis attacks.

To measure the performance of the covert IP flows tunneled by Protozoa covert WebRTC session, we leverage `iPerf`. This enables us to stress the covert channel capacity.

When testing our system's ability to resist traffic analysis attacks, we aim to reproduce the ideal conditions for the adversary. Essentially, the attacker's aim is to analyze the statistical properties of WebRTC's media (SRTP) and control (RTCP) packet flows so as to identify Protozoa traffic among legitimate WebRTC media sessions. To this end, we apply a state-of-the-art traffic classifier [4], which leverages two different sets of features: i) quantized packet size distributions, and ii) summary statistics computed from packet size and inter-arrival time distributions. Then, we collect a balanced dataset composed of legitimate and Protozoa WebRTC packet traces, and measure the AUC achieved by the above classifier when performing binary classification using 10-fold cross-validation. Note that, in the wild, class imbalance is expected to be skewed towards the abundance of legitimate streams and would likely make the adversary's task harder than in a controlled lab environment [12]. To ensure that the collected traces reflect realistic convert traffic transmissions, we keep the channel busy by injecting artificial chaff into the covert tunnel (using `iPerf`) while collecting these traces.

5.2 Experimental Testbed and Datasets

Our laboratory testbed, illustrated in Figure 7, is composed of four 64-bit Ubuntu 18.04.5 LTS virtual machines (VMs) provisioned with two virtual 2,3 GHz Intel Core i5 CPU cores and 16GB of RAM. VM1 and VM3 execute an instance of our prototype, operating as a Protozoa client and proxy, respectively. VM2 acts as the gateway and router for the two Protozoa VMs, and mimics the operation of a censor middlebox by collecting packet traces required for conducting statistical traffic analysis. Finally, VM4 is used to pose as a server in the open Internet which receives requests from the Protozoa proxy in VM3 acting on behalf of the client in VM1.

To conduct our experiments, we collected a total of 2000 YouTube video samples from four different categories (500 videos each) labeled by hand. These categories focus different video profiles assumed to be common in WebRTC services, and which we identify as *Chat*, *Coding*, *Gaming*, and *Sports*. For generating packet traces pertaining to legitimate and Protozoa media sessions, we split each of the four datasets (one for each video profile) in half. Then, we establish 250 legitimate WebRTC connections and 250 Protozoa connections while mirroring the video transmitted on each side of the connection. This allows us to avoid the contamination of the training data by mixing the same video samples in both legitimate and Protozoa connections. Video is set to be transmitted at 30fps and at a 640x480 resolution over the `whereby.com` WebRTC service, unless stated otherwise. Packet traces are collected for a duration of 30 seconds, a time interval shown in prior work to be sufficient for accurate detection of MCS streams using state-of-the-art statistical traffic analysis [4]. As described in Section 6, we validate that the use of longer traces did not significantly affect our results.

5.3 Metrics

We adopt a set of metrics for evaluating Protozoa's covert channel performance and resistance against statistical traffic analysis:

Performance metrics: In order to be able to compare Protozoa to existing work, we leverage *throughput* as the metric of performance of the covert channel. Additionally, we are interested in measuring Protozoa's covert channel *efficiency*, which provides the ratio between the total amount of data transmitted in the covert channel and the total available space in encoded video frame bitstreams.

Security metrics: Akin to earlier studies on the resistance of MCS systems to traffic analysis attacks, we use the following metrics to evaluate Protozoa's traffic analysis resistance capability: *true positive rate* (TPR), *false positive rate* (FPR), and the *area under the ROC curve* (AUC). The TPR measures the fraction of Protozoa flows that are correctly identified as such, while the FPR measures the proportion of legitimate flows erroneously classified as Protozoa flows. An adversary aims at obtaining a high TPR and a low FPR when performing covert traffic classification. The Receiver Operating Characteristic (ROC) curve plots the TPR against the FPR for the different possible cutout points for classifiers possessing adjustable internal thresholds. The AUC [19] summarizes this trade-off. Note that an AUC of 0.5 is equivalent to random guessing.

In the following sections, we evaluate our prototype by resorting to a set of microbenchmarks and by conducting a number of experiments when deploying Protozoa in real-world scenarios.

6 EVALUATION USING MICROBENCHMARKS

In this section, we evaluate Protozoa using a series of microbenchmarks. We test our system on a baseline scenario and then study the effects of varying the network and carrier conditions.

6.1 Baseline Deployment

Protozoa can be evaluated in multiple scenarios that depend on many factors (e.g., carrier video, carrier WebRTC application, or network conditions). Since validating all these dimensions is a hard endeavor, we first present an analysis of Protozoa based on a *baseline* deployment scenario which gathers a set of conditions expected to be found in a real-world deployment of Protozoa.

Our baseline deployment encompasses the following configuration. First, we select Whereby, a popular WebRTC application, as the carrier application for the Protozoa covert channel. Second, we select the videos comprising the *Chat* dataset as carrier media. Third, we assume that the round-trip-time (RTT) between Protozoa endpoints (VM1 - VM3) is in the order of 50ms, a typical value for connections established within the same continent [58, 72]. Lastly,

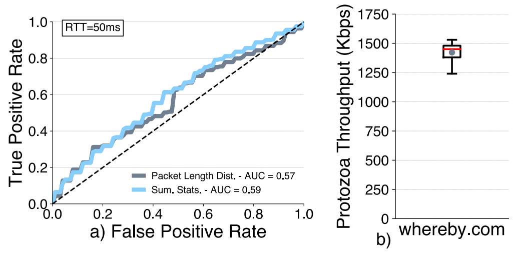

Figure 8: Baseline traffic analysis and performance results.

Duration (s)	10	20	30	40	50	60
AUC	0.56	0.60	0.59	0.58	0.58	0.61

Table 1: Classifier's AUC for varying trace durations.

we assume a 15ms RTT from the Protozoa proxy to an open Internet service (VM3 - VM4). This value is reasonable even when accessing foreign services due to the proliferation of CDN edge servers which may be regionally co-located with a Protozoa proxy [58, 71].

6.2 Baseline Performance Results

We now evaluate Protozoa's resistance against traffic analysis and assess the throughput and efficiency of the covert channel in the baseline deployment settings presented in the previous section.

Traffic analysis resistance: Figure 8a) depicts the ROC curve of the classifier when attempting to identify Protozoa connections resorting to two sets of features: quantized packet size distributions, and summary statistics. Firstly, we see that summary statistics provide a better overall detection rate, enabling the classifier to obtain an AUC of 0.59 (for the remainder of our evaluation, we will limit ourselves to present the results corresponding to the use of summary statistics). Secondly, the ROC curve shows that a censor would incur in a large FPR when blocking Protozoa flows resorting to the state-of-the-art classifier. Essentially, the FPR represents the collateral damage that results from setting the TPR to a specific cutoff value. As an example, if we assume that the censor would like to block 80% of all Protozoa flows (TPR = 0.8), it would erroneously flag approximately 60% of all legitimate flows as covert channels (FPR = 0.6). Although the cutoff FPR value is determined in a discretionary fashion by each censor (i.e., different censors can possibly withstand different TPR/FPR tradeoffs), the figure shows that distinguishing between Protozoa streams and legitimate media streams is close to random guessing.

To assess the robustness of Protozoa for packet traces of different durations, we repeated the same set of experiments using trace lengths up to 60 seconds as depicted in Table 1. These results suggest that the size of the traces has no meaningful impact on the AUC, given that the measured AUCs exhibit small fluctuations between 0.56 and 0.61. Thus, in the interest of scaling up our experiments, we conducted our remaining evaluation resorting to 30s traces.

Performance: Figure 8b) depicts a boxplot showing the throughput achieved by Protozoa's covert channels. We can observe that, under the baseline deployment conditions, Protozoa achieves an average throughput of 1422 Kbps, while the 90th percentile sits at 1510 Kbps, and the 75th percentile at 1480Kbps. This amounts to a throughput increase of 3× when compared to Facet, and a 3-fold order of magnitude increase when compared to DeltaShaper.

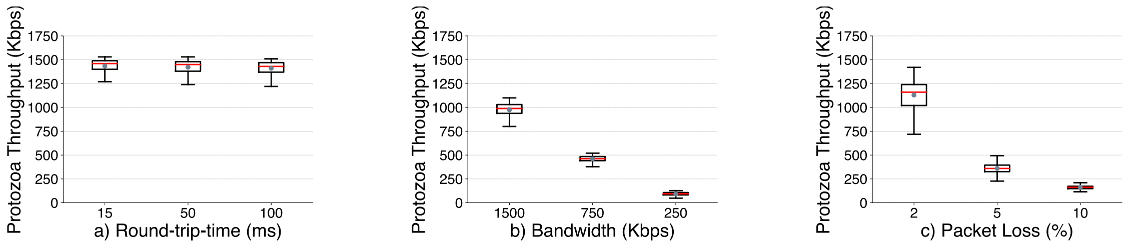

Figure 9: Throughput of Protozoa's covert channel when established over different network conditions.

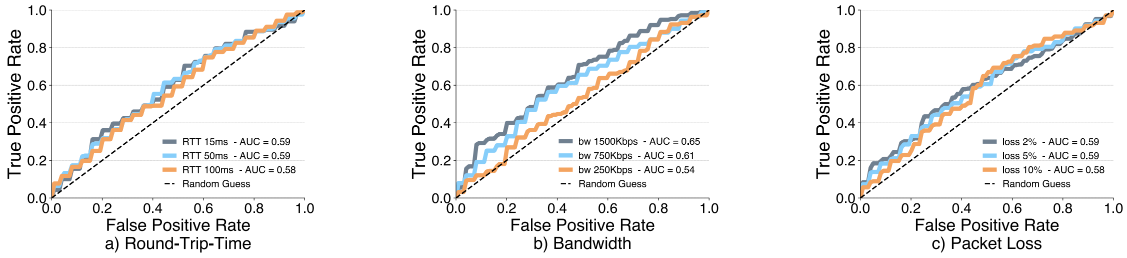

Figure 10: ROC AUC obtained by the classifier when detecting Protozoa flows under different network conditions.

Additionally, we analyzed the efficiency of Protozoa's covert channel by measuring the ratio between the data embedded in each outgoing frame and the size of the frame. When using iPerf to stress the upstream covert channel link, we observed that Protozoa used 98.8% of the available frame space to transmit covert data. This suggests that our packet encoding scheme can use the majority of the encoded frame bitstream to transfer covert data.

Lastly, regarding resource consumption, the client and proxy VMs peaked at a 21.6% usage of their total CPU and at 596MB of memory usage. These numbers suggest that Protozoa can be executed on various commodity hardware platforms.

In the next sections, we evaluate our system beyond our baseline setup across multiple other network deployment scenarios.

6.3 Varying Network Conditions

Assessing the security of our prototype in face of different network conditions is paramount i) to understand whether Protozoa can remain undetectable in practical deployment scenarios, and ii) to ascertain whether our system can withstand active network perturbations introduced by a network adversary, aimed at disclosing the operation of the system or at breaking the covert channel connection. To manipulate network conditions, we leveraged the traffic control facility Linux NetEm [30] and varied the network conditions in the following dimensions: i) *latency*, ii) *bandwidth*, and iii) *packet loss*. A recent study [37] reported that WebRTC connections can withstand a limited range of network perturbations before being torn down due to QoS constraints. We bound the network perturbations to inject in the network according to the ranges suggested by this study. Next, we detail the results of our experiments.

Latency variation of the covert channel: We delay packets so as to achieve a round-trip-time (RTT) of 15ms, 50ms, and 100ms between Protozoa endpoints (VM1 - VM3). These values emulate regional, intra-continental, and inter-continental RTTs [58, 72], while being kept within the bounds of 300ms, recommended for establishing real-time multimedia sessions with acceptable quality [67].

Our results are as follows. Figure 9 a) illustrates the breakdown of throughput achieved by our prototype as the RTT between Protozoa endpoints increases. It shows that the latency introduced between endpoints does not impact the throughput obtained by Protozoa. In particular, the throughput remains at an average of about 1420Kbps in the three configurations tested. Figure 10 a) presents the ROC curves obtained by the classifier when attempting to detect Protozoa in a network with different RTT configurations. We see that the classifier obtains a maximum AUC of 0.59. Thus, irrespective of the latency introduced between endpoints, the adversary does not obtain an advantage at distinguishing Protozoa flows.

Bandwidth variation of the covert channel: We symmetrically limit the bandwidth of the link to 1500Kbps, 750Kbps, and 250Kbps, beyond the unrestricted bandwidth conditions assumed in our baseline case. In these conditions, WebRTC streams use approximately 80% of the available bandwidth, agreeing with other studies [37].

Figure 9 b) shows that the achievable throughput tends to decrease as bandwidth is more scarce. For instance, while Protozoa's throughput averages 975Kbps when bandwidth is capped at 1500Kbps, it drops to 460Kbps at 750Kbps, and attains an average 91Kbps when bandwidth is only 250Kbps. This effect is expected since the constrained amount of bandwidth leads to the decrease of frame rates and forces the downgrade of video resolution and encoded frame size, thus reducing the available space for embedding covert data. For instance, a 250Kbps bandwidth cap only allows a WebRTC stream to obtain ≈25 FPS at a low 480x270 resolution [37].

As for resistance to traffic analysis, the AUC in Figure 10 b) reveals that the bandwidth variation does not provide sufficient information for the classifier to accurately distinguish between legitimate and Protozoa connections, peaking at 0.65 AUC when the bandwidth is limited to 1500Kbps.

Packet loss rate variation of the covert channel: We assess the properties of Protozoa when the media channel is subjected to packet losses. Following the experiments of Jansen et al. [37], we drop 2%, 5%, and 10% of the packets pertaining to WebRTC

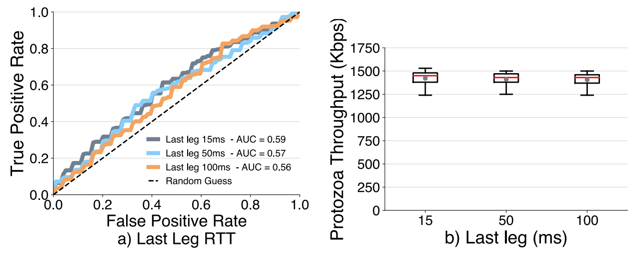

Figure 11: Throughput and traffic analysis resistance obtained while varying the RTT between VM3 and VM4.

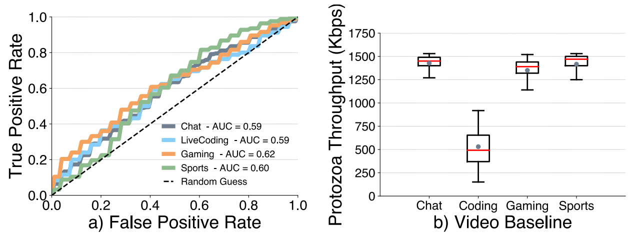

Figure 12: Throughput and traffic analysis resistance obtained while using different video profiles.

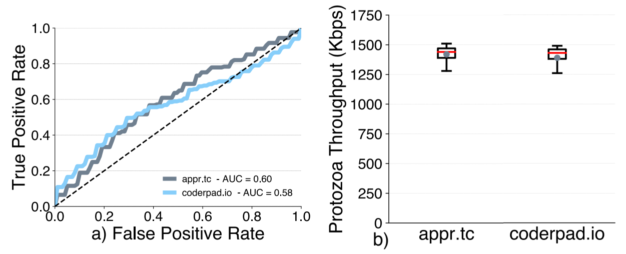

Figure 13: Throughput and traffic analysis resistance obtained while using different WebRTC services.

connections. Each of these loss rates causes WebRTC's congestion control to increase sending rate (2%), slowly increase sending rate (5%), and converging data rate to values leading to the tear-down of the video stream (10%). Typical recommendations for real-time media traffic sit at no more than 1% packet loss [67].

Figure 9 c) shows that while Protozoa's throughput is negatively affected by increasing packet loss rates, our prototype is still able to sustain an average throughput of 1130Kbps for packet loss rates of 2% and of 360Kbps for a loss rate of 5%. While a 10% packet loss substantially decreases the throughput to 160Kbps, we note that Protozoa's covert channel connections remained active and did not break for the duration of the experiment.

Lastly, the results in Figure 10 c) show that Protozoa preserves high-levels of traffic analysis resistance when the network link between Protozoa endpoints is subject to variable packet loss rates.

Latency variation at the last mile: We now focus on the impact of the RTT between Protozoa's proxy (VM3) and open Internet services (VM4) to the network performance. Figure 11 b) shows the breakdown of throughput achieved by Protozoa when the RTT between VM1 and VM2 endpoints is set to 50ms and the last leg of the connection to the Internet service ranges from 15ms to 100ms.

Similarly to our earlier experiments when varying the latency between Protozoa endpoints, the throughput is not compromised when latency increases between the Protozoa proxy and the Internet service. The average throughput of our system is rather stable, at around 1410Kbps, for the three tested configurations.

6.4 Varying Carrier Conditions

We also evaluate our system varying two carrier-specific conditions:

Varying video profiles: This experiment aims to test whether different video profiles used as cover media affect the throughput of our system. Such a question arises since variable bitrate video encoders, such as the ones used in WebRTC (e.g. VP8), adjust the amount of output data according to the complexity of encoded video segments. To answer this question, we evaluate the performance of the covert channel when established through the *Chat*, *Coding*, *Gaming*, and *Sports* video profiles in our baseline deployment.

Performance-wise, Figure 12b) depicts the throughput achieved by Protozoa when using different video profiles as cover media. We see that our system achieves a similar average throughput of approximately 1400Kbps for *Chat*, *Gaming*, and *Sports* media flows, while reaching an average throughput of 530Kbps when transmitting *Coding* media. These results concur to the observation that a large portion of video frames remain static in live coding videos. Additionally, these numbers suggest that the throughput is consistent within each baseline, achieving a maximum standard deviation of 157Kbps across the *Chat*, *Gaming*, and *Sports* baselines.

In turn, Figure 12a) shows the ROC curves for the classifier when attempting to distinguish Protozoa connections conducted over the different video profiles. The classifier achieves a similar AUC for all profiles (≈0.6 AUC), suggesting that the resistance against traffic analysis is preserved irrespective of the video profile used as cover.

Varying WebRTC services: To assess whether the security and performance properties of our system hold when media calls are established over multiple WebRTC applications, we conducted further experiments over two additional WebRTC services: a) `coderpad.io` – a live coding interview application – and b) `appr.tc` – Google's bare-bones demo application based on the simple WebRTC WebAPI.

Figure 13a) depicts the ROC curves for the classifier when inspecting Protozoa streams established through `coderpad.io` and `appr.tc`. The results show that the classifier obtains an AUC of 0.58 for streams established over `coderpad.io` and an AUC of 0.60 for streams established over `appr.tc`. In both cases, we see that Protozoa remains undetectable by a network eavesdropper.

As for the throughput when establishing a covert channel resorting to the two alternative WebRTC applications, Figure 13b) shows that Protozoa achieves an average throughput of 1420Kbps for `appr.tc` and of 1388Kbps for `coderpad.io`, similar to what was obtained with `whereby.com`, as presented previously in Section 6.2.

7 TESTING IN THE WILD

In this section, we test our system in multiple real-world settings. First, we present the results of an experiment comprising the execution of a set of typical workloads conducted by Internet users, over Protozoa. Then, as proof of concept, we show that Protozoa is able to evade the censorship apparatus of real-world adversaries by using it to access censored content in China, Russia, and India.

Application	Protocol	Workload
A. Curl	HTTP, FTP	Page transfer (16 MB) in 89s
B. Transmission	BitTorrent	File transfer (2GB) in 3h5m
C. Mutt	SMTP	Send email (1KB) in 5.5ms
D. Irssi	IRC	Send text message (80B) in 0.44ms
E. VLC	HTTP	Video streaming at 20/14 fps in 240p/480p
F. Firefox	HTTP	Web-surfing session

Table 2: Real application workloads over Protozoa.

7.1 Testing with Real Application Workloads

We tested Protozoa with multiple networked applications as depicted in Table 2. We used the baseline setup presented in Section 6.2. To tunnel the traffic of applications that do not natively support a SOCKS proxy, such as `mutt`, we leverage `proxychains` [62], a SOCKS proxy wrapper, to redirect such traffic through Protozoa.

First, we used Curl (A) over the covert channel to download files with sizes ranging from 1KB to 256MB using both the HTTP and FTP protocols. We also verified that Protozoa is able to uniformly distribute the bandwidth of the covert channel among simultaneous Curl connections. In order to test Protozoa's covert channel on a different transport protocol, we configured the Transmission BitTorrent client (B) to download a popular Linux distribution ISO. We also found Protozoa to be successful when operating email – Mutt (C) – and instant-messaging – Irssi IRC chat client (D) – applications. Additionally, we tested the ability to stream video content over VLC (E). Lastly, we ran Firefox (F) over Protozoa to navigate different web pages and to stream videos from YouTube, confirming that Protozoa enables interactive web-surfing tasks.

Overall, the results of our experiments show that Protozoa is able to accommodate a number of common Internet applications, including web-browsing, video streaming, or bulk data transfer.

7.2 Evading State-Level Adversaries

To test Protozoa's ability to circumvent real-world censors, we ran Protozoa in three geographical locations known to be active targets of Internet censorship – China [6], Russia [60], and India [82].

First, we identified sets of web pages that are blocked for each country [59, 68, 85]. We selected web pages in several categories: gambling, pornography, news/politics, drug sale, and circumvention tools. For each set, we verified that the web pages could not be directly accessed (i.e., without using Protozoa) using Firefox running on a server physically deployed in the respective country. Then, we repeated this access after setting up Protozoa covert sessions. In each server, we configured a Protozoa client to establish a covert channel towards a Protozoa proxy located in LA, USA, and used this channel to access the blocked web pages. To run our experiments in said countries, we resorted to virtual private servers (VPSs) in Shanghai, Moscow, and Mumbai, respectively, and deployed the Protozoa bundle amounting to a total of 150 MB.

Blocking policies: We observed that browsing blocked websites in Russia and India resulted in ISP blockpages, whereas in China they would simply not load properly. A closer look at the traffic traces produced when trying to access blocked web pages revealed that the GFW performs packet drops on connections aimed at blacklisted hosts. This observation is consistent with the behavior of the GFW [6]. Browsing a blocked website from within Russia and India showed that blocking policies implemented within datacenters are less restrictive from those applied on typical ISP connections, i.e., we could access websites which would trigger the return of ISP blockpages when browsed over a VPN; therefore, given that our VPSes are located inside datacenters, for ensuring a reliable Protozoa testing, in Russia and India we route all VPS traffic through a VPN server hosted in the same country, where we obtain blockpages when visiting forbidden websites. This differentiation, however, did not occur on the VPS within China, where the pages found to be blocked when browsing over a VPN inside the country were also blocked when accessing from our VPS in a datacenter.

WebRTC Application	Reachability		
	China	Russia	India
`appr.tc`	-	✓	✓
`aws.amazon.com/chime`	✓	✓	✓
`codassium.com`	✓	✓	✓
`coderpad.io`	✓	✓	✓
`discordapp.com`	-	✓	✓
`gotomeeting.com`	✓	✓	✓
`hangouts.google.com`	-	✓	✓
`messenger.com`	-	✓	✓
`slack.com`	✓	✓	✓
`whereby.com`	✓	✓	✓

Table 3: Reachability tests on popular WebRTC services.

Availability of WebRTC services: Paramount to the functioning of Protozoa is the ability to connect to a foreign WebRTC service. Since Protozoa makes no assumption over the WebRTC application used as a vehicle for the covert channel, it is only necessary to find one unblocked application within the censored region. Table 3 shows that multiple WebRTC applications are available in the countries focused in our evaluation. Importantly, the table shows that, despite several WebRTC applications being blocked in China, a user still has plenty of alternative WebRTC media applications that can be used as a carrier for Protozoa covert channels.

Reaching censored content: To reach our blocked page sets, we leveraged `whereby.com` to establish Protozoa connections. We were able to access all such blocked websites in China, Russia, and India.

7.3 Ethical Considerations

The experiments conducted in this section involve the access to censored content from a number of vantage points within countries known to experience Internet censorship. These accesses raise important ethical concerns since they risk triggering reprisals from local authorities. We followed the best practices described in the Menlo report [14] to guide three major decisions of our experimental design. First, we did not recruit volunteers for our experiments. Instead, we rented VPSes from commercial VPS providers which understand the legal implications of offering network and computing services in each country they operate. Second, albeit using the signalling infrastructure of existing WebRTC applications, Protozoa does not compromise in any way the integrity of such applications. Covert traffic is exclusively forwarded by replacing user-generated video content. Lastly, we did not collect any sensitive user data.

8 SECURITY DISCUSSION

We now discuss some potential attacks to Protozoa and defenses:

Packet dropping: An adversary may instrumentally drop a small number of selected packets of WebRTC media streams in an attempt to dramatically slow down the covert data transmission or disrupt the functioning of Protozoa protocols causing, in either case, a denial of service. In contrast to other systems [27, 34], Protozoa is robust against these attacks since it does not rely on specific packets for managing covert channels. Moreover, Section 6.3 shows that applications that use Protozoa's covert channels are able to tolerate a large percentage of dropped packets without terminating.

Active probing: Active probing attacks aim at identifying Protozoa proxies, e.g., by attempting to join some active chatroom and identify the transmission of corrupted video streams which telltale the presence of covert channels. By selecting WebRTC chatrooms that implement member admission controls, e.g., using passwords or contact list checks, Protozoa users can evade this attack.

Fingerprinting of cover videos: If Protozoa is set up to stream a pre-recorded cover video, an adversary may attempt to identify a particular user by using that video for fingerprinting Protozoa covert channels. This threat can be countered by: i) rotating the pre-recorded video, ii) or feeding a live video from the local camera.

Long-term user profiling: An adversary may keep track of a user's interactions with WebRTC services so as to build a profile of interactions with multimedia applications. An accurate profile may enable an adversary to indirectly detect the usage of Protozoa through connections with out-of-ordinary duration or by detecting the placement of calls at unusual times of the day. Assessing the feasibility of this threat is an interesting direction for future work.

9 RELATED WORK

We now describe past approaches aimed at evading Internet censorship and locate Protozoa in the spectrum of existing techniques.

9.1 Comparison with Similar Systems

Protozoa fits in the family of multimedia covert streaming systems. It stands out by introducing a new technique – encoded media tunneling. Next, we compare our system against two other branches of this family (Figure 1 puts all these systems in perspective.)

Media protocol mimicking: Previous systems have introduced traffic morphing [77] techniques for the transmission of covert data by imitating multimedia protocols. For instance, by entirely replacing the payload of media packets by encoded data, SkypeMorph [51] and CensorSpoofer [74] deliver a reasonable throughput of 344Kbps [51] and 64Kbps [74], respectively. However, due to the difficulty in mimicking the complete behavior of multimedia protocols, these systems are prone to be detected with 100% accuracy through a combination of passive and active attacks [34]. In contrast, Protozoa provides not only strong resistance against traffic analysis, but also higher throughput (around 1.4Mbps).

Raw media tunneling: Systems like FreeWave [36], Facet [46], DeltaShaper [3], and CovertCast [50] modulate covert data in the audio/video input of multimedia applications. Some of these systems can sustain a reasonable throughput. For instance, Facet can reach 471Kbps [46] and CovertCast 168Kbps [50]. However, these systems are vulnerable to statistical traffic analysis techniques [4, 27]: FreeWave, Facet, CovertCast are detected with over 99% accuracy, while DeltaShaper between 85%-95% [4, 27]. Protozoa outperforms these systems both performance and security wise.

9.2 Beyond Multimedia Covert Streaming

Protocol mimicking is a general technique for carrying covert data by imitating the behavior of a carrier protocol. However, most solutions [15, 16, 76] suffer from the same limitations as their multimedia protocol siblings and are prone to network attacks [34, 73].

Protocol tunneling has been used in other contexts. SWEET [86], CloudTransport [9], and Castle [29] tunnel covert data through steganographically marked email, cloud storage services, and real-time strategy games, respectively; *meek* [23, 66] leverages domain fronting to hide Tor traffic inside HTTPS connections to allowed hosts. However, unlike Protozoa, some of these systems have not been evaluated against state-of-the-art traffic analysis attacks, and others have already been shown to be vulnerable to detection [73].

There are many other related techniques. Ephemeral proxies like Snowflake [21, 22] (which uses WebRTC connections) redirect traffic through short-lived proxies provided by volunteers; however, unlike Protozoa, the covert traffic is fingerprintable and the presence of secret messages can be detected through traffic analysis. Protocol randomization [13] transforms traffic into random bytes to evade protocol blacklists, but it fails in the presence of protocol whitelisting and is vulnerable to entropy analysis [73]. Refraction networking [7, 8, 17, 24, 25, 35, 38, 78, 79] incorporates special traffic redirection routers inside cooperative ISPs which need to be carefully placed, otherwise a censor can avoid network paths containing such routers [53, 54, 65]. In contrast, Protozoa relies on individual trusted users located outside the censored region. Packet manipulation strategies [6, 40, 45, 75] aim at invalidating the state of censors' firewalls; Protozoa's covert channels can breach through such firewalls provided that WebRTC traffic is not blocked.

Lastly, some systems provide access to censored content cached in CDNs [33, 87]. Protozoa provides access to any publicly available content accessible to the Protozoa proxies. MassBrowser [55] leverages cache browsing [33, 87] and volunteer proxies to reach censored content. However, since the connections between clients and proxies are protected with a variant of Obfsproxy [13], they are also affected by the limitations of protocol randomization.

10 CONCLUSIONS

This paper introduced Protozoa, the first multimedia-based censorship circumvention tool which generates secure covert channels by instrumenting the innards of the WebRTC multimedia framework. Our evaluation shows that Protozoa traffic cannot be distinguished from typical WebRTC flows by state-of-the-art traffic analysis techniques. Further, the results of our evaluation show that Protozoa enables an increase in throughput of up to three orders of magnitude when compared against similar (and less secure) tunneling tools. Currently, Protozoa requires active user support at the proxy's end and demands users to find trusted proxies for exchanging covert content. Devising a scalable solution for finding trusted proxies is an interesting direction for future work.

Acknowledgments: We thank our shepherd, Nick Feamster, and the anonymous reviewers for their comments. This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) via the SFRH/BD/136967/2018 grant, and the PTDC/EEI-COM/29271/2017 and UIDB/ 50021/ 2020 projects.

REFERENCES

[1] Aryan, S., Aryan, H., and Halderman, J. A. Internet censorship in Iran : A first look. In *Proceedings of the 3rd USENIX Workshop on Free and Open Communications on the Internet* (Washington, DC, USA, 2013).

[2] Barradas, D. Protozoa code repository. https://github.com/dmbb/Protozoa, 2020. Accessed: 2020-08-20.

[3] Barradas, D., Santos, N., and Rodrigues, L. Deltashaper: Enabling unobservable censorship-resistant tcp tunneling over videoconferencing streams. In *Proceedings on Privacy Enhancing Technologies* (Minneapolis, MN, USA, 2017), vol. 2017(4), pp. 5–22.

[4] Barradas, D., Santos, N., and Rodrigues, L. Effective detection of multimedia protocol tunneling using machine learning. In *Proceedings of the 27th USENIX Security Symposium* (Baltimore, MD, USA, 2018).

[5] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and Norrman, K. The secure real-time transport protocol (srtp). RFC 3711, March 2004.

[6] 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* (London, UK, 2019), pp. 2199–2214.

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

[8] Bocovich, C., and Goldberg, I. Secure asymmetry and deployability for decoy routing systems. In *Proceedings on Privacy Enhancing Technologies* (Barcelona, Spain, 2018), vol. 2018 (3), pp. 43–62.

[9] Brubaker, C., Houmansadr, A., and Shmatikov, V. Cloudtransport: Using cloud storage for censorship-resistant networking. In *Privacy Enhancing Technologies*, vol. 8555 of *Lecture Notes in Computer Science*. Springer International Publishing, 2014, pp. 1–20.

[10] Cate Cadell - Reuters. Report says China internet firms censored coronavirus terms, criticism early in outbreak. https://www.reuters.com/article/us-health-coronavirus-china-censorship/report-says-china-internet-firms-censored-coronavirus-terms-criticism-early-in-outbreak-idUSKBN20Q1VS, 2020. Accessed: 2020-08-20.

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

[12] Chandola, V., Banerjee, A., and Kumar, V. Anomaly detection: A survey. *ACM computing surveys (CSUR) 41*, 3 (2009).

[13] Dingledine, R. Obfsproxy: the next step in the censorship arms race. https://blog.torproject.org/blog/obfsproxy-next-step-censorship-arms-race, 2012. Accessed: 2020-08-20.

[14] Dittrich, D., and Kenneally, E. The Menlo report: Ethical principles guiding information and communication technology research. In *U.S.Department of Homeland Security, Tech. Rep.* (2012).

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

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

[17] Ellard, D., Jones, C., Manfredi, V., Strayer, W. T., Thapa, B., Van Welie, M., and Jackson, A. Rebound: Decoy routing on asymmetric routes via error messages. In *Proceedings of the 2015 IEEE 40th Conference on Local Computer Networks (LCN)* (Clearwater Beach, FL, USA, 2015), pp. 91–99.

[18] 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 2015 ACM Conference on Internet Measurement Conference* (Tokyo, Japan, 2015), pp. 445–458.

[19] Fawcett, T. Roc graphs: Notes and practical considerations for researchers. *Machine Learning 31* (01 2004), 1–38.

[20] FFmpeg. https://ffmpeg.org, 2000. Accessed: 2020-08-20.

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

[22] Fifield, D., Hardison, N., Ellithorpe, J., Stark, E., Boneh, D., Dingledine, R., and Porras, P. Evading censorship with browser-based proxies. In *Proceedings of the 12th International Conference on Privacy Enhancing Technologies* (Vigo, Spain, 2012), pp. 239–258.

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

[24] Frolov, S., Douglas, F., Scott, W., McDonald, A., VanderSloot, B., Hynes, R., Kruger, A., Kallitsis, M., Robinson, D. G., Schultze, S., Borisov, N., Halderman, A., and Wustrow, E. An ISP-Scale Deployment of TapDance . In *Proceedings of the 7th USENIX Workshop on Free and Open Communications on the Internet* (Vancouver, BC, 2017).

[25] Frolov, S., Wampler, J., Tan, S. C., Halderman, J. A., Borisov, N., and Wustrow, E. Conjure: Summoning proxies from unused address space. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom, 2019), p. 2215–2229.

[26] Gebhart, G., Author, A., and Kohno, T. Internet censorship in Thailand: User practices and potential threats. In *Proceedings of the 2nd IEEE European Symposium on Security & Privacy* (Paris, France, 2017).

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

[28] Google Developers - WebRTC. Getting started with WebRTC. https://webrtc.org/getting-started/overview, 2019. Accessed: 2020-08-20.

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

[30] Hemminger, S. Network emulation with netem. *6th Linux and open source conference for Australia and New Zealand* - `linux.conf.au` (2005).

[31] Hernández, J. C. As China Cracks Down on Coronavirus Coverage, Journalists Fight Back. https://www.nytimes.com/2020/03/14/business/media/coronavirus-china-journalists.html, 2020. Accessed: 2020-08-20.

[32] Hiruncharoenvate, C., Lin, Z., and Gilbert, E. Algorithmically bypassing censorship on Sina Weibo with nondeterministic homophone substitutions. In *Proceedings of the 9th International Conference on Web and Social Media* (Oxford, UK, 2015), AAAI.

[33] Holowczak, J., and Houmansadr, A. Cachebrowser: Bypassing chinese censorship without proxies using cached content. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), pp. 70–83.

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

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

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

[37] 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), 56–68.

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

[39] Khattak, S., Elahi, T., Simon, L., Swanson, C. M., Murdoch, S. J., and Goldberg, I. Sok: Making sense of censorship resistance systems. In *Proceedings on Privacy Enhancing Technologies* (Darmstadt, Germany, 2016), vol. 2016, pp. 37–61.

[40] Khattak, S., Javed, M., Anderson, P. D., and Paxson, V. Towards illuminating a censorship monitor's model to facilitate evasion. In *Proceedings of the 3rd USENIX Workshop on Free and Open Communications on the Internet* (Washington, D.C., USA, 2013).

[41] King, G., Pan, J., and Roberts, M. E. Reverse-engineering censorship in china: Randomized experimentation and participant observation. *Science 345*, 6199 (2014).

[42] 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* (Washington, D.C., USA, 2015).

[43] Knockel, J., Ruan, L., and Crete-Nishihata, M. An analysis of automatic image filtering on wechat moments. In *Proceedings of the 8th USENIX Workshop on Free and Open Communications on the Internet* (Baltimore, MD, USA, 2018).

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

[45] Li, F., Razaghpanah, A., Kakhki, A. M., Niaki, A. A., Choffnes, D., Gill, P., and Mislove, A. lib• erate,(n) a library for exposing (traffic-classification) rules and avoiding them efficiently. In *Proceedings of the Internet Measurement Conference* (London, UK, 2017), pp. 128–141.

[46] 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* (Scottsdale, AZ, USA, 2014), pp. 163–172.

[47] Loreto, S., and Romano, S. P. Real-time communications in the web: Issues, achievements, and ongoing standardization efforts. *IEEE Internet Computing 16*, 5 (2012), 68–73.

[48] McGrew, D., and Rescorla, E. Datagram transport layer security (dtls) extension to establish keys for the secure real-time transport protocol (srtp). RFC 5764, May 2010.

[49] McKeown, N., Appenzeller, G., and Keslassy, I. Sizing router buffers (redux). *SIGCOMM Computer Communication Review 49*, 5 (Nov. 2019), 69–74.

[50] McPherson, R., Houmansadr, A., and Shmatikov, V. CovertCast: Using live streaming to evade internet censorship. In *Proceedings on Privacy Enhancing Technologies* (Darmstadt, Germany, 2016), vol. 2016(3), pp. 212–225.

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

[52] Morshed, M. B., Dye, M., Ahmed, S. I., and Kumar, N. When the Internet goes down in Bangladesh. In *Proceedings of the 20th ACM Conference on Computer-Supported Cooperative Work and Social Computing* (Portland, Oregon, USA, 2017).

[53] Nasr, M., and Houmansadr, A. Game of decoys: Optimal decoy routing through game theory. In *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria, 2016), pp. 1727–1738.

[54] Nasr, M., Zolfaghari, H., and Houmansadr, A. The waterfall of liberty: Decoy routing circumvention that resists routing attacks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), pp. 2037–2052.

[55] Nasr, M., Zolfaghari, H., and Houmansadr, A. MassBrowser: Unblocking the censored web for the masses, by the masses. In *Proceedings of the* 27^{th} *Annual Network and Distributed System Security Symposium* (San Diego, CA, USA, 2020).

[56] Netfilter Framework. http://www.netfilter.org/, 1998. Accessed: 2020-08-20.

[57] 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 41st IEEE Symposium on Security and Privacy* (San Francisco, CA, USA, 2020).

[58] Nygren, E., Sitaraman, R. K., and Sun, J. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review 44*, 3 (2010), 2–19.

[59] Paige Leskin - Business Insider. Here are all the major US tech companies blocked behind China's 'Great Firewall'. https://www.businessinsider.com/major-us-tech-companies-blocked-from-operating-in-china-2019-5, 2019. Accessed: 2020-08-20.

[60] 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* 27^{th} *Network and Distributed Systems Security Symposium* (San Diego, CA, USA, 2020).

[61] RFC 1928 - SOCKS Protocol Version 5. https://tools.ietf.org/html/rfc1928. Last Accessed: 2020-08-20.

[62] rofl0r - GitHub. Proxychains-ng. https://github.com/rofl0r/proxychains-ng, 2011. Accessed: 2020-08-20.

[63] Roy, R. R. *Handbook of SDP for Multimedia Session Negotiations: SIP and WebRTC IP Telephony.* CRC Press, 2018.

[64] Ruan, L., Knockel, J., and Crete-Nishihata, M. Censored Contagion: How Information on the Coronavirus is Managed on Chinese Social Media. https://citizenlab.ca/2020/03/censored-contagion-how-information-on-the-coronavirus-is-managed-on-chinese-social-media/, 2020. Accessed: 2020-08-20.

[65] Schuchard, M., Geddes, J., Thompson, C., and Hopper, N. Routing around decoys. In *Proceedings of the ACM Conference on Computer and Communications Security* (Raleigh, North Carolina, USA, 2012), pp. 85–96.

[66] Sheffey, S. R., and Aderholdt, F. Improving meek with adversarial techniques. In *Proceedings of the 9th USENIX Workshop on Free and Open Communications on the Internet* (Santa Clara, CA, USA, 2019).

[67] Szigeti, T., and Hattingh, C. *End-to-End QoS Network Design: Quality of Service in LANs, WANs, and VPNs.* Cisco Press, 2004.

[68] Tarun Kumar Yadav - GitHub. Analyzing-Web-Censorship-Mechanisms-in-India. https://github.com/tarun14110/Analyzing-Web-Censorship-Mechanisms-in-India/blob/master/possibly_blocked_websites.txt, 2018. Accessed: 2020-08-20.

[69] Tschantz, M. C., Afroz, S., Anonymous, and Paxson, V. Sok: Towards grounding censorship circumvention in empiricism. In *Proceedings of the IEEE Symposium on Security and Privacy* (2016), pp. 914–933.

[70] v4l2loopback. https://github.com/umlaeute/v4l2loopback, 2005. Last Accessed: 2020-08-20.

[71] Vakali, A., and Pallis, G. Content delivery networks: status and trends. *IEEE Internet Computing 7*, 6 (2003), 68–74.

[72] Verizon Business. IP Latency Statistics. https://enterprise.verizon.com/terms/latency/, 2015. Accessed: 2020-08-20.

[73] Wang, L., Dyer, K. P., Akella, A., Ristenpart, T., and Shrimpton, T. Seeing through network-protocol obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, CO, USA, 2015), pp. 57–69.

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

[75] Wang, Z., Cao, Y., Qian, Z., Song, C., and Krishnamurthy, S. V. Your state is not mine: a closer look at evading stateful internet censorship. In *Proceedings of the Internet Measurement Conference* (London, UK, 2017), pp. 114–127.

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

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

[78] Wustrow, E., Swanson, C. M., and Halderman, J. A. Tapdance: End-to-middle anticensorship without flow blocking. In *Proceedings of the 23rd USENIX Security Symposium* (San Diego, CA, 2014), pp. 159–174.

[79] Wustrow, E., Wolchok, S., Goldberg, I., and Halderman, J. A. Telex: Anticensorship in the network infrastructure. In *Proceedings of the 20th USENIX Security Symposium* (San Francisco, CA, USA, 2011).

[80] Xiong, R., and Knockel, J. An efficient method to determine which combination of keywords triggered automatic filtering of a message. In *Proceedings of the 9th USENIX Workshop on Free and Open Communications on the Internet* (Santa Clara, CA, USA, 2019).

[81] Xu, X., Mao, Z. M., and Halderman, J. A. Internet censorship in china: Where does the filtering occur? In *Proceedings of the 12th International Conference on Passive and Active Network Measurement* (Vienna, Austria, 2011), pp. 133–142.

[82] Yadav, T. K., Sinha, A., Gosain, D., Sharma, P. K., and Chakravarty, S. Where the light gets in: Analyzing web censorship mechanisms in India. In *Proceedings of the Internet Measurement Conference* (Boston, MA, USA, 2018).

[83] Yoshimasa Iwase - NTT Communications. A Study of WebRTC Security. https://webrtc-security.github.io/, 2015. Accessed: 2020-08-20.

[84] Zander, S., Armitage, G., and Branch, P. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys & Tutorials 9*, 3 (2007), 44–57.

[85] zapret-info. Register of Internet Addresses filtered in Russian Federation. https://github.com/zapret-info/z-i, 2020. Accessed: 2020-08-20.

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

[87] Zolfaghari, H., and Houmansadr, A. Practical censorship evasion leveraging content delivery networks. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria, 2016), pp. 1715–1726.