Obscura: Enabling Ephemeral Proxies for Traffic Encapsulation in WebRTC Media Streams Against Cost-Effective Censors

Afonso Vilalonga
Universidade NOVA de Lisboa & NOVA LINCS
Portugal
j.vilalonga@campus.fct.unl.pt

Kevin Gallagher
Universidade NOVA de Lisboa & NOVA LINCS
Portugal
k.gallagher@fct.unl.pt

João S. Resende
INESCTEC, Universidade do Porto
Portugal
jresende@fc.up.pt

Henrique Domingos
Universidade NOVA de Lisboa & NOVA LINCS
Portugal
hj@fct.unl.pt

Abstract

Recent research on online censorship has provided valuable insights into common censorship strategies and censors' tolerance for collateral damage. A consistent finding across these studies is that censors tend to favour cost-effective techniques such as proxy enumeration, active probing, and deep packet inspection (DPI), rather than more complex and non-deterministic methods such as deep learning-based traffic analysis. For example, a recent study on the Snowflake censorship evasion system reinforced this finding by demonstrating that authoritarian regimes primarily relied on DPI to target the system. However, as censorship techniques continue to evolve, two critical questions arise: (1) What future attack vectors are likely to emerge based on current research and observed censor capabilities? (2) How can these emerging threats, along with previously utilised censorship methods, be effectively mitigated?

In this paper, we present Obscura, a censorship evasion system designed to resist cost-effective, historically grounded censorship techniques while also defending against a class of plausible future attacks within a cost-effective threat model targeting WebRTC-based censorship evasion systems. Obscura is built upon four core features: (1) encapsulation of traffic within WebRTC media streams, (2) the use of a reliability layer, (3) support for both browser-based and Pion-based clients and proxy instances, and (4) the use of ephemeral proxies. Each feature is intended to mitigate either a known attack observed in the wild or a theoretically plausible attack consistent with the capabilities of a cost-effective censor. We provide a security analysis to justify our design choices and a performance evaluation to demonstrate that Obscura maintains reasonable throughput for typical online activities.

Keywords

Censorship Evasion, WebRTC, Ephemeral Proxies

Proceedings on Privacy Enhancing Technologies 2026(1), 583–603

https://doi.org/10.56553/popets-2026-0030

1 Introduction

Censorship evasion systems allow citizens in authoritarian regimes to exercise their freedom of expression and access uncensored information, a matter of particular importance in today's context, given the reported rise in online censorship [53]. Recent global data and research on online censorship [16–18, 28, 30, 31, 36, 42, 43, 45, 51, 61, 66, 77, 78, 80–82] have not only underscored the significance of these systems but also offered insights into the strategies used by censors to control information and enforce online censorship.

Censors typically use a two-pronged approach to target censorship evasion systems [62, 75]: (1) fingerprinting network flows to classify them as allowed or disallowed, and (2) blocking disallowed flows and their endpoints using techniques such as packet injection or firewall rules. Techniques for identifying disallowed connections include active probing [16, 17, 25, 77], deep packet inspection (DPI) [9, 21, 26, 58, 77], and traffic analysis via side-channel inference [3, 10, 39, 78]. Active probing tests suspicious hosts (e.g., proxies of censorship evasion systems) for irregular responses. DPI inspects unencrypted packet fields for distinctive byte sequences or fingerprints. Traffic analysis identifies and fingerprints censorship evasion systems by examining side-channel metadata, such as timing features across multiple network flows, using techniques including statistical methods, classifiers, and deep learning models.

Although censors have a wide range of techniques available to identify prohibited flows, a consistent finding in the literature is that they tend to favour simple, cost-effective, and deterministic methods over complex, resource-intensive, and non-deterministic ones [62]. To illustrate this finding, we focus on the Snowflake censorship evasion system. Snowflake [7] is an encapsulation-based Tor pluggable transport. In encapsulation-based censorship evasion systems, covert data is embedded within a carrier protocol, making traffic exchanged with a proxy appear indistinguishable from regular flows of the carrier protocol to external observers. For Snowflake, the carrier protocol is WebRTC [2], a widely used protocol stack for peer-to-peer real-time communication for the web. WebRTC supports arbitrary data connections (i.e., data channels), which Snowflake uses to encapsulate Tor traffic, as well as media streams. An important feature of Snowflake is the ephemeral nature of its proxies. If a user's connection to a proxy is interrupted, the system automatically reconnects them to a new one. Additionally, Snowflake can leverage volunteers' regular devices as proxies simply by having them open a specific webpage on their device, thereby

expanding the pool of available proxies. This ephemeral proxy model, combined with its ease of deployment, makes Snowflake a challenging target for proxy blocking (e.g., proxy enumeration attacks) and active probing. A recent study by Snowflake's authors [7] documented several attempts by authoritarian regimes to block Snowflake, primarily through DPI targeting fingerprints in the Snowflake DTLS handshake protocol or TLS fingerprints in complementary communication protocols within Snowflake, reinforcing the observation that censors have historically favoured simpler, cost-effective methods over more sophisticated approaches.

A logical follow-up question is: why was Snowflake vulnerable to such fingerprinting attacks? Unlike browser-to-browser WebRTC connections, which rely on the most prevalent browser-based WebRTC implementations, the Snowflake client implementation uses Pion [1], a Go-based WebRTC library. This choice has made the DTLS handshake between the client and the proxy particularly distinctive compared to typical browser-to-browser connections. As a result, censors have identified and exploited unique fingerprints in Pion's DTLS implementation, which are absent in browser-based WebRTC implementations. The underlying causes of TLS fingerprinting vulnerabilities are similar to those of DTLS fingerprinting, but mitigation strategies have been proposed and applied to address such attacks [26]. However, solutions to DTLS fingerprinting have remained scarce [38]. The typical strategy for countering DTLS fingerprinting has been to generate browser-like fingerprints within Snowflake and patch vulnerabilities as they arise, requiring ongoing maintenance and reflecting a reactive rather than proactive approach. Another concern for censorship evasion systems in general, and particularly for Snowflake, is identifying which cost-effective censorship methods censors might adopt next. Prior research has debated whether data channels constitute a distinctive fingerprint, given their presumed limited use in real-world WebRTC applications [7, 21]. Although data channels have thus far remained unblocked, a recent study proposed a simple and cost-effective technique for blocking them without disrupting major WebRTC applications [59], referring to it as a differential degradation attack (DDA). The same study also introduced a variant of this attack targeting censorship evasion systems that encapsulate traffic within WebRTC media streams. The core idea behind this variant is to disrupt these systems by inducing targeted frame loss while leaving regular WebRTC video traffic largely unaffected.

To counter current and future cost-effective attacks targeting WebRTC-based censorship evasion systems, we propose **Obscura**, a censorship evasion system that functions both as a standalone solution and as a pluggable transport. Obscura is built on four key principles: (1) traffic encapsulation within WebRTC media streams, (2) the use of a reliability layer, (3) support for both browser-based and Pion-based proxies and client instances, and (4) the use of ephemeral proxies. Together, these core features enable Obscura to defend against proxy enumeration, active probing, DPI of WebRTC-based protocols, and DDAs. However, within Obscura, we acknowledge an inherent trade-off between providing a reactive approach to DPI resistance and the level of resistance to the media-based encapsulation variant of DDAs, which we discuss in Section 5.3 and Appendix H. Additionally, we design the system to blend in with typical WebRTC usage, thereby mitigating potential fingerprints that could be exploited by passive lightweight traffic analysis (LTA) with low computational and storage overhead based on statistical or low-storage information collected over multiple packets. Techniques of this kind have been observed in the wild [78], in contrast to more complex methods such as those based on machine learning or deep learning.

To summarise, the main contributions of this work are as follows: (1) A novel architectural design for a censorship evasion system, based on the four identified key principles and targeting a specific cost-effective threat model; (2) novel insights into how video can be obtained for use in media-based traffic encapsulation systems for real-world deployments; (3) a strategy used within Obscura to defend against the DDAs identified in [59]; (4) a comprehensive security and performance analysis of Obscura; (5) an open-source implementation of our system [68].

2 Threat Model and Design Goals

In this section, we present the threat model of our system, outline the system goals, and discuss how these goals relate to both the threat model and the overall system design.

2.1 Threat Model

Our threat model is based on a common censorship evasion scenario [74], in which a user within a censor's jurisdiction attempts to access the uncensored internet. The censor allows general internet usage but seeks to block access to content it deems prohibited while minimising false positives. We assume that censors operate within a cost-effective framework, which we outline in this section. Since our system relies on WebRTC, we focus particularly on cost-effective attacks targeting WebRTC-based censorship evasion systems, as well as common, generic attacks on censorship evasion systems. Specifically, we assume that censors can use the following techniques: proxy enumeration, active probing, DPI targeting protocols within the WebRTC stack (e.g., targeting DTLS or WebRTC data channels), and packet manipulation (e.g., packet dropping). We also assume that censors may perform LTA relying on statistical information or low-storage data collected across multiple packets to fingerprint Obscura, as similar cost-effective attacks have been observed in practice [78]. The distinction between DPI and LTA is subtle. We define DPI as fingerprinting attacks that rely on inspecting unencrypted fields in packets, such as specific fields in the *ClientHello* message of the DTLS handshake. In contrast, we define LTA as requiring the examination of metadata (e.g., packet sizes) from multiple packets within a single flow. We classify DDAs as DPI-based attacks because, although their primary purpose is not to fingerprint censorship evasion systems, they rely on continuous analysis of unencrypted packet fields within a single flow to decide whether to drop them or not. Additionally, we classify techniques such as threshold-based blocking through packet counting (e.g., detecting an unusual number of control messages) and the use of packet sizes for traffic fingerprinting as LTA-based attacks that the censor can perform. However, we exclude attacks that fall outside this cost-effective framework. These include techniques that, in our view, are theoretical, complex, computationally expensive, prone to false positives, or non-deterministic, such as traffic analysis based on classifiers or deep learning. Throughout the remainder of this

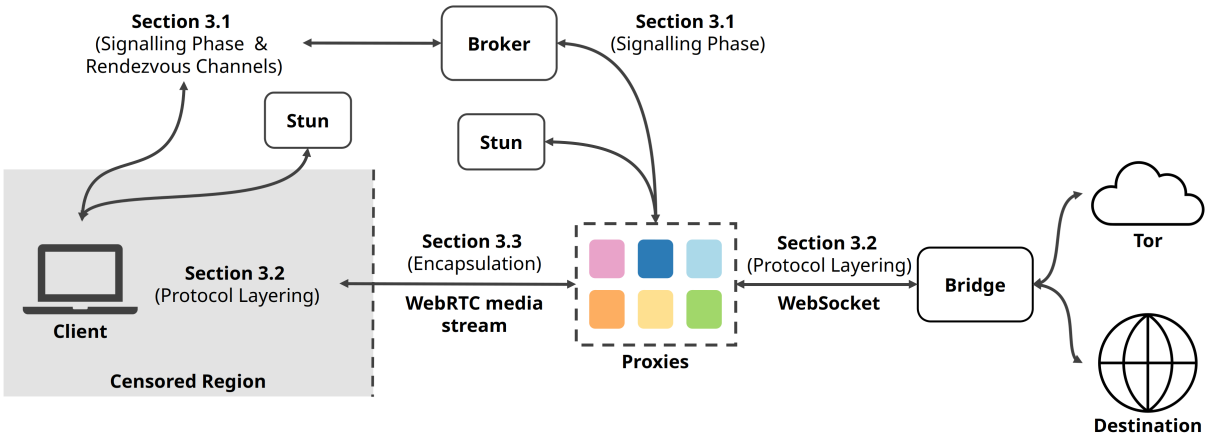


Figure 1: Obscura's system architecture overview.

paper, we refer to censors operating within the bounds of this threat model as cost-effective censors.

2.2 Design Goals

We identify three main design goals: resistance to baseline attacks, blending in, and utility and usability.

Resistance to Baseline Attacks: We implement strategies to resist five main generic attacks: (1) active probing and proxy enumeration, (2) packet manipulation, (3) DPI of protocols within the WebRTC stack, (4) data channel fingerprintability, and (5) the media streams variant of DDAs. To address (1), we utilise ephemeral proxies and allow a large pool of devices to function as proxies. For (2), we provide a reliability layer to mitigate packet dropping and use WebRTC's built-in cryptographic primitives to protect against packet modification and injection. To counter (3), we leverage the browser's native protocol fingerprints by enabling browser-to-browser connections between clients and proxies. To address (4), we use media channels as the carrier instead of data channels, and, finally, to address (5), we support Pion-based client and proxy instances and incorporate a reliability layer to tolerate packet loss.

Blending In: We implement strategies to minimise the effectiveness of LTA attacks. We also avoid introducing known or foreseeable major vulnerabilities, which we define as significant deviations from typical WebRTC behaviour, since knowingly introducing fingerprints that could be exploited by censors, regardless of whether their capabilities fall within our threat model, would be detrimental to the long-term robustness of the system. Thus, we identify the following as the most significant sources of fingerprintability: (1) ensuring that traffic structurally resembles authentic video content, and (2) avoiding unrealistic network patterns, such as reusing the same videos across all users or volunteers, or generating multiple consecutive Picture Loss Indication (PLI) requests (sent when missing or corrupted video frames are detected, see Section 3.3). To address (1), we use the available frame sizes and transmit them at the intended rate, regardless of whether there is data to encapsulate. To address (2), we ensure that the number of PLI requests remains within a typical range and that both clients and proxies exhibit WebRTC media stream traffic patterns consistent with those of diverse sources.

Utility and Usability: We provide reasonable performance, specifically reasonable throughput (i.e., utility) to support typical web browsing and online activities. The system's utility is demonstrated in Section 4. Regarding usability, the system is designed to be easy to use and configure, especially for users and volunteers. To this end, we minimise the number of actions and configurations required for deployment and use. For example, proxies can be deployed simply by launching a webpage, similar to Snowflake. Regarding resource consumption, Section 4 demonstrates that the system operates with low resource requirements, as confirmed by the tests performed on basic commodity hardware. However, concerning bandwidth, we discuss inherent trade-offs between minimising bandwidth usage and achieving our goal of blending in Section 3.4 and Section 5.

3 Obscura

Our system design is based on the Snowflake architecture, as illustrated in Figure 1. It incorporates the same five core components: the client (i.e., client-side software), broker, proxy, bridge, and STUN server [37]. Clients, proxies, and STUN servers are not assumed to be trusted; therefore, they can be deployed or used by anyone. However, STUN servers should be accessible to users and selected so that their blocking would inflict collateral damage on a censor. In contrast, we assume that the broker and bridge are trusted components managed by a trusted party. Obscura can operate as a standalone system, similar to a VPN, or as a pluggable transport. We use the terms "client" and "user" interchangeably to refer to either the individual operating the system or the client-side software itself. The intended meaning can be inferred from the context.

Consider a user residing in a heavily censored region attempting to access the open internet through Obscura. The user first launches the Obscura client, which connects to a broker to request an available proxy. If a proxy is available, the necessary information for establishing a WebRTC media connection between the client and the proxy is exchanged through the broker. Using the Interactive Connectivity Establishment (ICE) protocol [33], both peers obtain their public IP addresses (i.e., server-reflexive candidates) via STUN

servers and exchange these candidates through the broker to form candidate pairs. Each peer then tests these pairs to identify viable network paths. However, ICE alone is insufficient to establish a peer-to-peer connection. The client and proxy must also negotiate session parameters, such as codecs and cryptographic keys, using the Session Description Protocol (SDP) [5]. The proxy sends an SDP offer, and the client responds with an SDP answer. Once the connection is established, the client and proxy communicate directly with each other. This phase, in which the WebRTC connection is established, is referred to as the signalling phase, during which all messages are exchanged through the broker. Additionally, proxies may go offline; when this occurs, the client contacts the broker to request a new proxy and restarts the signalling process. A reliability layer ensures that the user session resumes once the client establishes a connection with the new proxy. Covert traffic is routed through proxies to a bridge and encapsulated within video frames, which are transmitted over a WebRTC media stream between the client and the proxy. Media streams use the Secure Real-Time Transport Protocol (SRTP) [8] for encryption, integrity, and authentication of RTP streams [55], with encryption keys negotiated via DTLS. The Real-Time Control Protocol (RTCP) [55] is used in conjunction with SRTP to monitor connection quality.

In the remainder of this section, we describe the system's operation by highlighting the features that support our design goals. We begin with the signalling phase, which is necessary for establishing the WebRTC connection on which the system relies (Section 3.1). Next, we discuss the architectural layering of our protocol, explaining how it ensures reliability and provides the foundation for proxy ephemerality. Both aspects are central to achieving our first design goal (Section 3.2). We then describe our data encapsulation methodologies, which directly support the first and second design goals (Section 3.3). Finally, we discuss how the video content used for encapsulation is obtained and how this design supports the second design goal (Section 3.4). Where appropriate, we consider how each feature may involve trade-offs related to our third goal. For implementation details, we refer to Appendix A.

3.1 Signalling and Rendezvous Channels

We illustrate our signalling protocol in Figure 2. Proxies connect to the broker via a secure WebSocket and send periodic pings at regular intervals of X time units. If the broker does not receive a ping within this timeframe, it marks the proxy as invalid and discards it. Proxies also inform the broker of their type (i.e., browser-based or Pion-based). The broker maintains the state of all active proxies and pairs a client with one when the client sends a request to establish a connection. If no proxies are available, the broker waits until a proxy becomes available for the client. Once the broker establishes a client-proxy pair, the client sends a packet to the proxy via the broker containing the bridge address that the client will use for the session, along with its client type. The proxy and client exchange their types so the system can determine which encapsulation methodology to use (see Section 3.3). The proxy then connects to the bridge via a secure WebSocket. If the connection between the proxy and the bridge is successful, the proxy sends an SDP offer to the client and begins transmitting the ICE candidates as they arrive. The client receives the SDP offer, responds with its

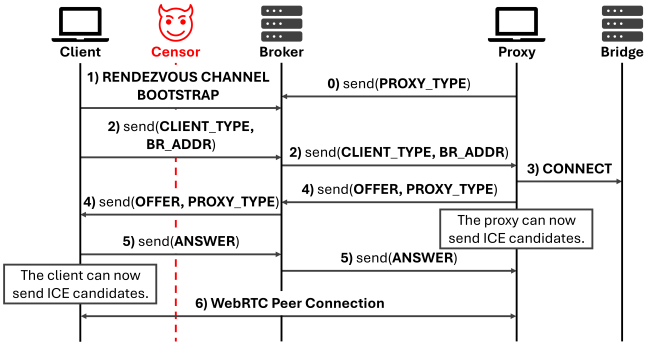


Figure 2: Obscura's signaling protocol.

own SDP answer, and transmits its ICE candidates. We use Trickle ICE [2] to transmit ICE candidates as they are gathered, rather than waiting to collect all candidates first, thereby reducing connection establishment time. The proxy and client communicate directly once they establish the WebRTC peer-to-peer connection.

We use a first-in, first-out (FIFO) method for client–proxy allocation. However, the modularity of the signalling protocol allows it to be replaced entirely or extended with other heuristics, such as geographic restrictions, NAT types, or instance types (e.g., browser-based or Pion-based). Since proxies are ephemeral, they may become unavailable. When this occurs, the client initiates the signalling process by contacting the broker to request a new proxy after detecting that the WebRTC peer-to-peer connection with the previous proxy has failed. The new proxy and the client then repeat the signalling phase to establish a new WebRTC connection. Since the client maintains the configured bridge address independently of the proxy, every proxy to which the client connects will be instructed to establish a connection with that bridge unless this configuration is modified on the client side. Currently, the bridge address is stored in a configuration file, but could alternatively be provided by the broker.

Proxies can contact the broker directly, as they are assumed to operate outside the censored region and are therefore immune to being blocked from accessing the broker. In contrast, users within the censored region cannot communicate directly with the broker, as this would expose it to blocking efforts and render the system unusable. To address this, we use censorship-resistant channels to exchange information between the client and broker during the signalling phase. These channels, often referred to as rendezvous channels [71, 72], must remain functional even if the censor is aware of their operational details. Although typically more costly and offering lower throughput than conventional censorship evasion techniques, rendezvous channels provide greater resilience to blocking, as their disruption would cause significant collateral damage. Although existing approaches, such as domain fronting [23] and cloud-based rendezvous channels [50], could be integrated into Obscura, we developed two custom rendezvous channels: one based on Google Pub/Sub [71] and another leveraging TURN servers [69]. Further implementation details are provided in Appendix B.

3.2 Protocol Layering

We adopt the Turbo Tunnel architecture [20], which structures transmitted data in censorship evasion systems into three hierarchical layers: the obfuscation layer, the session layer, and the application layer. Each layer serves a distinct function, and the encapsulation follows a specific order: application-layer traffic is encapsulated within session-layer traffic, which in turn is encapsulated within obfuscation-layer traffic.

The obfuscation layer is the network visible layer and is responsible for obfuscating covert traffic. In Obscura, obfuscation is achieved by encapsulating session layer packets within the video frame payloads of WebRTC media streams. The session layer ensures resilience to packet loss and ordering by encapsulating application layer traffic within a reliable network protocol. These features help mitigate packet manipulation attacks and enable proxy switching without restarting ongoing client sessions. For the session layer, Obscura uses the KCP protocol [57], a reliable UDP-based transport protocol, together with the smux library [79] to multiplex streams over a single connection and to clean up timed out streams caused by client disconnections, thereby preventing unnecessary resource consumption on the bridge. Additionally, we use the encryption capabilities of the KCP protocol to ensure that the covert content remains encrypted between the client and the bridge. The application layer traffic corresponds to the client application traffic that Obscura will covertly transmit. In pluggable transport mode, the application traffic is Tor traffic. In standalone mode, it may originate from any client application that supports a SOCKS proxy, which serves as the mechanism through which Obscura in standalone mode expects client application traffic to be routed.

To enable proxy ephemerality, the session-layer state must be maintained between the client and the bridge rather than the client and the proxy. Managing the state at the proxy would lead to session loss when switching proxies. To address this, the client software integrates the client-side implementation of the KCP/smux protocol while the bridge runs the server-side counterpart. When the client or bridge receives application-layer packets, they are processed by the KCP/smux stack, which encapsulates them into KCP/smux packets. The KCP/smux packets are then delivered to Obscura's control for obfuscation before being transmitted to the other endpoint. On the receiving end, when obfuscated packets are received, Obscura removes the obfuscation layer and forwards them to the KCP/smux stack, which extracts and delivers the decapsulated client application-layer packets back to Obscura. Obscura then forwards this data to the client application. Since the bridge handles traffic from multiple proxies representing different clients, it must dynamically map delivered KCP/smux session-layer packets from the KCP/smux stack to the correct proxy associated with each packet. We use a custom eight-byte ID appended to each KCP/smux packet to map received packets from the KCP/smux stack to a specific proxy. The bridge maintains a mapping table linking session identifiers to WebSocket connections. When a new WebSocket connection is established between a proxy and the bridge, the bridge uses the custom ID to either update an existing mapping (if the client has switched proxies) or create a new one. During a proxy switch, any packets lost during the transition are retransmitted due to the reliability properties of the KCP/smux protocol.

3.3 Data Encapsulation

Three main components are involved in transmitting and receiving a media stream between two peers in a WebRTC connection: (1) the media source/media sink, (2) the encoder/decoder, and (3) the packetiser/depacketiser. First, the media source generates the content to be streamed. Second, the encoder (i.e., the video codec) processes and compresses the audio and video frames. Third, the packetiser segments the encoded frames into smaller payloads, which are then encrypted and transmitted over the network as SRTP packets. On the receiving end, the reverse process occurs: the depacketiser reconstructs the decrypted RTP payloads into complete encoded frames, the decoder converts them into raw media content, and the media sink plays the final decoded output.

We base our browser-based WebRTC application on the sample implementations provided in [54] and use the Insertable Streams API [73] to manipulate video frames without modifying the browser code. This API operates between the encoder and packetiser on the sending side and between the depacketiser and decoder on the receiving side. It provides access to frames after encoding but before packetisation for encapsulation and after depacketisation but before decoding for decapsulation. Pion-based instances differ from browser-based instances because Pion's WebRTC implementation provides access to individual RTP packets at the receiving side rather than fully reconstructed encoded frames. Although Pion's SampleBuilder interface [49] can theoretically reassemble full frames, we have not evaluated this functionality and have found limited practical examples. More importantly, relying on direct RTP packet access with Pion provides a mechanism to defend against DDAs. Therefore, we base our Pion-to-Pion encapsulation and decapsulation methodology on direct RTP packet access. As a result, our encapsulation and decapsulation strategies are tailored to the instance types in use, each offering distinct advantages: Browser-to-browser (B-B) connections generate fingerprints that match those of standard browsers when using WebRTC protocols. Pion-to-Pion (P-P) connections allow defence mechanisms against DDAs and offer greater usability for clients in both the pluggable transport and standalone versions, as they do not require the client to use a browser with WebRTC capabilities enabled (a particularly important feature for the pluggable transport version); Pion-to-browser (P-B) and browser-to-Pion (B-P) connections, while not explicitly designed to mitigate any specific attack in our threat model, enhance the system usability and interoperability, similar to how Snowflake client Pion instances support both browser-based and Pion-based proxy connections.

For all encapsulation methodologies, we maximise the encoded data space within each video frame allocated for data encapsulation. However, doing so might sometimes require the fragmentation of KCP/smux packets (i.e., session-layer packets). To enable the reassembly of fragmented KCP/smux packets, we prepend a custom 11-byte header to each block of data we encapsulate inside the payload of a frame, whether it is a complete KCP/smux packet or a fragment. We refer to these data blocks, along with the header, as Obscura packets. Each Obscura packet header consists of five fields: (1) a 1-byte marker flag indicating the start of an Obscura packet, (2) a 4-byte sequence number used for ordering and reassembly, (3) a 1-byte segment number to track the order of each fragment,

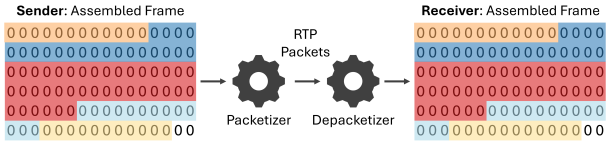


Figure 3: Workflow for B-B connections. Each color (different shade) represents an Obscura packet (sizes not to scale). For simplicity, the VP8 payload header is omitted.

(4) a 4-byte payload length indicating the size of the payload, and (5) a 1-byte flag indicating whether the packet contains the final segment of the original KCP/smux packet.

We develop our encapsulation methodologies for VP8-encoded video, though adapting them to support other codecs should be feasible with minimal code changes. To achieve this, we consider both the RTP header format [55] and the RTP payload format for VP8 video [76]. An RTP header has a minimum size of 12 bytes but may include optional extensions. In the case of VP8, an additional field known as the VP8 payload descriptor adds 1 to 6 bytes to the RTP header. This descriptor includes metadata such as fragmentation information and frame type. Within the RTP payload for VP8, two key components are present: the VP8 payload header and the encoded frame data. The VP8 payload header marks the beginning of the encoded frame data and provides information, such as whether the frame is a key frame or an interframe (i.e., a delta frame). The size of this payload header varies between 10 bytes for key frames and 3 bytes for delta frames.

Browser-to-Browser: B-B connections rely on the WebRTC implementations of browsers, supporting both Firefox and Chromium-based browsers. The encapsulation algorithm for B-B connections generates Obscura packets whenever the available space in the encoded video frame data is at least 22 bytes: 11 bytes for the Obscura packet header, 10 bytes for the maximum VP8 payload header, and at least 1 byte for the Obscura packet payload, which carries the covert data. Frames whose encoded data size falls below this 22-byte threshold are marked as containing no Obscura packets. If the available space in a frame exceeds the size of the data to be encapsulated, the remaining space is replaced with zeros. Each received encoded frame is examined to detect and extract any embedded Obscura packets during decapsulation. We illustrate the B-B encapsulation and decapsulation methodology in Figure 3.

Replacing the encoded frame content with covert data without proper handling can trigger multiple RTCP PLI requests in browser-based instances. These requests are issued when the decoder fails to process the modified frames, interpreting them as corrupted and continuously requesting key frames to restore proper playback. While occasional PLI requests during a connection may not be considered abnormal, the repeated transmission of requests in rapid succession could be exploited by a censor as a fingerprinting vector, as such behaviour is uncommon in typical WebRTC applications. To mitigate this risk, we use a browser canvas to generate a synthetic image that is encoded as a valid VP8 frame. We then replace the payload of the received frame with that of the synthetic frame. Additionally, we reuse the original RTCEncodedVideoFrame object [14] (i.e., the object representation of an encoded frame in the WebRTC sender and receiver pipeline) received because (1) it is the object type accepted by the decoder, and (2) it can only be created through the WebRTC sender or receiver pipeline. Attempting to pass the synthetic frame's object type directly to the decoder results in an exception due to type incompatibility. By modifying the payload of the existing RTCEncodedVideoFrame, we ensure compatibility with the WebRTC pipeline and avoid decoding errors. When encapsulating data, the first 10 bytes of the frame are skipped to preserve the VP8 payload header, which has a maximum size of 10 bytes (i.e., corresponding to a key frame). We do this because these bytes populate a specific read-only field in the RTCEncodedVideoFrame object, the frame type field. This approach ensures that the frame retains a valid VP8 payload header and, consequently, a valid type field in the RTCEncodedVideoFrame on the receiving side, allowing the frame to be reused by simply replacing the encapsulated data with the synthetic frame. We note that, particularly in Firefox, PLI requests are occasionally sent sporadically throughout a session in our test environment. This behaviour is consistent and occurs both when using our application with the Insertable Streams API enabled, regardless of whether encapsulation or decapsulation is performed, and when the API is disabled.

Pion-to-Pion: The P-P encapsulation methodology operates at the granularity of RTP packets on the decapsulation side. However, the full encoded frames are accessible on the encapsulation side. Therefore, the encapsulation algorithm must account for the differing access patterns between the sender and receiver. When encapsulating data in a P-P connection, we ensure that individual Obscura packets are not fragmented across multiple RTP packets, as this would make reassembly extremely difficult. This is enforced by ensuring that no Obscura packet exceeds the maximum RTP payload size and preventing any Obscura packet from being positioned within the frame payload in a way that causes the packetiser to split it across two RTP packets. We use an auxiliary formula (Equation 1) to calculate the current **m**aximum RTP **p**ayload **s**ize (*MPS*) in bytes, for each frame.

$$MPS = M\_RTP\_S - (RTP\_H\_S + VP8\_PD\_S) \tag{1}$$

In Equation 1, M_RTP_S denotes the maximum size for an RTP packet, which is set to 1200 bytes in the Pion implementation [48]. The RTP_H_S represents the size of the RTP header, with a default and minimum value of 12 bytes. The $VP8\_PD\_S$ corresponds to the size of the VP8 payload descriptor, which varies depending on the length of the pictureID field. The pictureID is a 7- or 15-bit field that increments with each frame as a frame index. The VP8 payload descriptor is included in every RTP packet and has the following sizes in Pion: 1 byte for the initial RTP packet(s) of a frame, 3 bytes if the pictureID is less than 128, and 4 bytes if it is 128 or greater. When the pictureID reaches 128, it can either return to 0 or be extended to a 15-bit field. A 1-bit marker indicates whether the pictureID is using 7 or 15 bits. By default, Pion initialises the pictureID to 0, extends it to a 15-bit field when it reaches 128, and wraps it back to 0 once the full 15-bit range is exhausted [46]. Based on this behaviour, we calculate the Pion *MPS* value as follows:

- $1187 = 1200 - (12 + 1)\ bytes$: For the first frame.
- $1185 = 1200 - (12 + 3)\ bytes$: For frames with a pictureID between 1 and 127.

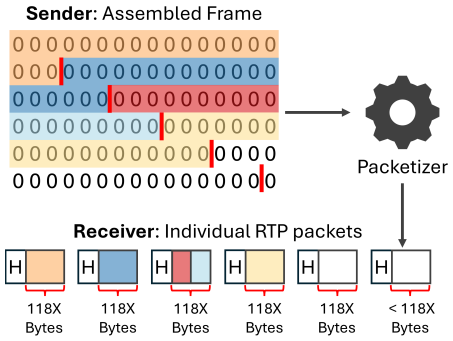


Figure 4: P-P Encapsulation. The numbers beneath the packets indicate the RTP payload sizes, where the value of X can be 4, 5, or 7. The regions between two red lines represent a frame segment with a maximum size of MPS. For illustration purposes, we assume that the maximum size of each RTP packet payload is 19 zeros, corresponding to $118X$ bytes.

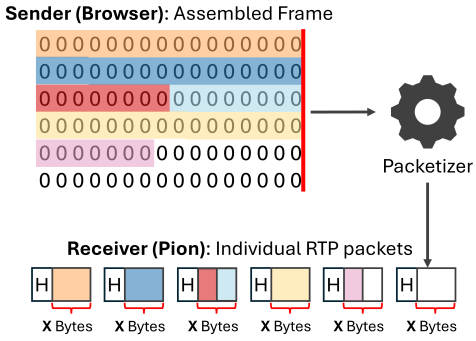


Figure 5: Encapsulation methodology for B-P connections. $X = \left\lfloor \frac{FS}{\lceil \frac{FS}{1174} \rceil} \right\rfloor + 1$ byte from the last packet to first (right to left) for $FS \bmod \lceil \frac{FS}{1174} \rceil$ remaing bytes; FS is the frame size.

- $1184 = 1200 - (12 + 4)$ *bytes*: For frames with a pictureID greater than or equal to 128.

Obscura maintains a variable representing the current value of the pictureID field in order to calculate the MPS for the current frame. Then, for each frame, until either the available space within the frame is exhausted, or there is no more data to encapsulate, Obscura creates Obscura **p**ackets of **s**ize PS using Equation 2.

$$PS = min(len(D),\ max(RS - HS,\ 0),\ max(FS - HS,\ 0)) \quad (2)$$

In equation 2, D is the array of bytes representing the covert data to be transmitted; RS denotes the remaining space available for encapsulation within the current frame segment (i.e., a frame chunk that fits within a single RTP packet) and, for each frame, is initially set to MPS; FS is the total frame size; and HS (11) is the size of the Obscura packet header. The value of RS is updated each time a new Obscura packet is encapsulated by decrementing it by the size of the encapsulated packet. Once the current frame segment is fully occupied with Obscura packets, or when there is no remaining space available or data to encapsulate, the value of RS is reset to MPS. This design ensures that each Obscura packet is encapsulated within a frame segment of maximum size MPS. Since Obscura packets are never allowed to span multiple frame segments and are always contained within a single RTP packet, any received RTP packet can be used to fully recover the Obscura packet(s) it contains without requiring reassembly across packets. The P-P encapsulation and decapsulation methodology is illustrated in Figure 4. For decapsulation, we analyse the VP8 payload descriptor in each received RTP packet to compute the offset within the RTP packet payload where Obscura packets begin. PLI requests caused by corrupted frames are not a concern in P-P connections, as Pion does not include built-in support for real-time video decoding.

Pion-to-Browser & Browser-to-Pion: Obscura ensures full interoperability across all instances and supported browsers. For P-B connections, where Pion functions as the sender and the browser as the receiver, Obscura uses the same techniques as in B-B connections. This is possible because Pion has access to fully encoded frames before transmission, and the browser provides access to fully reassembled received frames. For B-P connections, Obscura uses a modified version of the techniques used in P-P connections.

Testing on Chromium-based browsers and Firefox revealed that Obscura's maximum RTP payload size is 1174 bytes. The RTP header remains fixed at 12 bytes, and the pictureID is always used in its extended 15-bit format, initialised with a random 15-bit value and not reset to 0 upon reaching its maximum. Referencing Equation 1, there is a 10-byte discrepancy in the payload size calculation: $(1200 - (12 + 4) = 1184; 1184 - 1174 = 10)$. Feedback from the WebRTC API community suggests that this discrepancy may result from the packetiser's configuration, which deliberately produces slightly smaller RTP payload sizes to account for possible header extensions and to minimise the need for frequent reconfigurations [67]. It may also stem from negotiated RTP header extensions between the browser and the Pion instance. Additionally, the WebRTC implementations in the tested browsers differ from Pion in the way their packetisers operate. Specifically, in our test environment, and for B–P connections, the tested browsers distribute frame data almost evenly across packets [27], rather than filling each packet before allocating new ones, as Pion does. To accommodate this behaviour, our algorithm first determines the number of packets required for a given frame. Based on the number of packets and the frame size, it then calculates how many bytes of the frame each packet will evenly receive. Finally, any remaining bytes are redistributed, beginning with the last packet and proceeding toward the first. An example of this process is illustrated in Figure 5.

3.4 Obtaining the Video

Most media-tunneling censorship evasion systems face a common challenge: how can users and volunteers obtain suitable video content for streaming? A typical solution proposed in the literature is to use the device's camera feed or screen sharing. However, this approach raises significant privacy concerns, as users and volunteers may be unwilling to share personal video feeds, even if the content is ultimately replaced. An alternative is to bundle the system with preselected videos or allow users to provide their own. Yet, enabling user selection can reduce overall usability, as it requires

users and volunteers to acquire video content continually. On the other hand, relying on a single bundled video or a limited set of bundled videos introduces the risk of creating a recognisable fingerprint, as multiple users and volunteers would stream identical content. To mitigate this risk while maintaining usability, we propose two strategies: (1) streaming from a canvas animation and (2) implementing an efficient video distribution scheme.

Canvas Animations: Using a canvas animation as the video source improves usability by eliminating the need for users, volunteers, or system operators to manage and download video files. Additionally, the use of random animations, along with their ease of deployment and development, enables multiple users to utilise the system with visually distinct animations simultaneously. However, using animations may result in lower throughput compared to video sources, as they are typically less dynamic and transmit fewer bytes per frame (see Appendix C for further evaluation). Integrating this approach with Pion instances presents additional challenges, as there is no straightforward method for capturing a canvas animation as a stream outside of browser environments. Instead, a third-party tool, such as FFmpeg [19], must encode the animation frames into a live feed that can then be streamed to the receiving peer.

Video Distribution Scheme: The primary challenge of using multiple videos without requiring users or volunteers to obtain them manually lies in distributing them efficiently, given that video files are typically large in size. Although this remains an open problem, we propose a solution: enabling the broker to distribute metadata files upon request. These files contain metadata about the videos (e.g., number of frames, frame rate, frame sizes), allowing users and volunteers to generate synthetic frames replicating the original content's structural characteristics. While this approach still requires the broker to distribute metadata files, it significantly reduces overhead compared to distributing full video files. An additional advantage is the flexibility to select source videos for metadata generation based on specific criteria, such as regional popularity (e.g., using popular videos in a given area) or throughput (e.g., prioritising visually dynamic videos).

4 Experimental Evaluation

In this section, we evaluate Obscura's performance.

4.1 Evaluation Goals and Metrics

The primary objective of our performance evaluation was to assess Obscura's throughput under varying network conditions and system configurations, thereby developing a comprehensive understanding of the system's behaviour across diverse network environments, video content profiles, and operational setups. We conducted tests using three types of connections between the client and the proxy in standalone mode: Pion-to-Pion (P–P), Chrome-to-Chrome (C–C), and Firefox-to-Firefox (F–F). We also conducted tests on asymmetric connections between the client and the proxy, as well as on the system operating in pluggable transport mode. However, due to space constraints and because these tests did not yield significant new insights beyond those discussed in this section, the results are presented in Appendix E and Appendix F, respectively. We adopted a testing methodology similar to that used by the Tor Project to collect performance metrics [60]. Specifically, we deployed an HTTP server hosting text files, performed repeated downloads of a 1 MB file, and calculated throughput based on the time required to complete each download. All throughput values reported in the subsequent sections were obtained using this approach. For each measurement, we conducted at least 100 downloads and generated box plots from the resulting throughput samples.

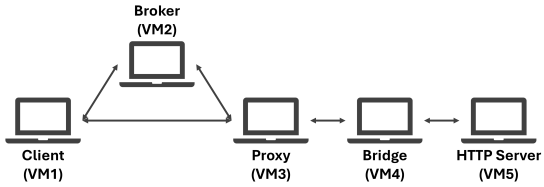


Figure 6: Setup for Obscura's experimental evaluation

We note that some of the box plots, particularly those in Section 9, exhibit a large number of outliers or a wide range between minimum and maximum values. This variation arises from the nature of the videos used, as well as from differences across different portions of the same video. For example, two videos classified under the same profile (e.g., Coding) may produce significantly different throughput values if one is primarily static and the other features more dynamic visual content. Even within a single video, abrupt transitions can cause temporary spikes in throughput. For example, a shift from a dark-mode IDE to a bright web browser page with more visual information can trigger a momentary increase in throughput due to the change in visual content. We refer the reader to Appendix D for illustrative examples of variations in video frame sizes across different video profiles and connection types.

4.2 Test bench

Our test bench, illustrated in Figure 6, consists of five 64-bit Ubuntu 22.04 LTS virtual machines (VMs), each configured with 4 virtual cores and 8 GB of RAM. These VMs run on a dedicated cloud server equipped with an AMD Epyc 7313 CPU (16 cores, 32 threads, 3.0/3.7 GHz), 128 GB of RAM, and 1 Gbps bandwidth. The limited resources allocated to each VM demonstrate that the system components can be deployed on modest hardware. VM1 and VM3 serve as the client and proxy, respectively; VM2 functions as the broker; VM4 acts as a bridge; and VM5 hosts an HTTP server that provides files for download. Browsers are used in headless mode with autoplay enabled, and its automation is managed through Selenium [56].

Regarding the video, we used a WebM file as a video element to enable access and streaming in browser-based instances. For Pion instances, we used the same video in IVF file format (a format readable by Pion). Unless otherwise specified, the video used in the tests was Big Buck Bunny [6], a publicly available short animated film encoded with VP8, with a resolution of 1280×720 and a target bitrate of 2 Mbps. Video encoding was performed using FFmpeg [19]. The 100 measurements for each throughput value exceeded the duration of any video used in our experiments. Therefore, each video was looped continuously during experiments until at least 100 throughput measurements were collected, ensuring a consistent basis for comparison across all tests. Finally, network conditions between VMs were enforced using the Linux *tc* command.

4.3 Varying Network Conditions

This section presents the system's throughput results under varying network conditions. We evaluate P-P, F-F, and C-C connections across round-trip times (RTTs) of 80 ms, 115 ms, and 165 ms; bandwidth constraints of 250 Kbps, 750 Kbps, and 1500 Kbps; and packet loss rates of 2%, 5%, and 10%. The selected bandwidth and packet loss values are based on prior studies of WebRTC performance [32] and evaluations of WebRTC-based censorship evasion systems [4]. For RTT configuration, the RTT between VM4 and VM5 is set to 15 ms, representing typical latency between clients and CDN edge servers co-located with a bridge [44, 64]. Given the widespread deployment of CDN infrastructure, this is considered a reasonable assumption. The RTT between VM3 and VM4 is fixed at 50 ms to model intra-continental latency [44, 65]. We do not specify RTT values between VM1 and VM2 or VM2 and VM3, as measurements begin only after the proxy and client establish a connection, rendering VM2 irrelevant to performance evaluation. RTTs between VM1 and VM3 are varied at 15 ms, 50 ms, and 100 ms to simulate regional, intra-continental, and intercontinental conditions, respectively [44, 65]. The total RTTs reported reflect the combined RTT across all links. Both packet loss and bandwidth constraints are applied on the VM1–VM3 connection (i.e., client-proxy connection).

Overall, P–P and B–B connections exhibit reduced performance as network conditions deteriorate. However, under less severe conditions, specifically, in the absence of bandwidth constraints and with packet loss rates at or below 2%, the system maintains reasonable throughput. The baseline configuration used throughout the remainder of this section is defined as 0% packet loss, no bandwidth constraints, and a system RTT of 115 ms. Under this baseline, the measured averaged throughput is 1.79 Mbps for F–F, 1.49 Mbps for C–C, and 1.32 Mbps for P–P connections.

Insight #1: Tolerance of P–P connections to bandwidth constraints: When using Pion to stream video from a file, the absence of real-time video encoding mechanisms significantly limits the system's adaptability to varying network conditions, particularly under bandwidth constraints [47]. As illustrated in the bandwidth constraints figure (middle graph) of Figure 7, Obscura is unable to sustain a P–P connection when the video bitrate exceeds the available bandwidth. Given that the video has a target bitrate of 2 Mbps and the lowest bandwidth constraint is 1500 Kbps, the covert P–P connection becomes unstable, eventually tearing down or experiencing severe degradation. Additional tests presented in Appendix G further corroborate this behaviour.

Insight #2: Adaptability of B–B connections to network conditions: Due to built-in congestion control mechanisms and live video encoding, the system naturally adapts to network conditions in B–B connections. Both F-F and C-C connections perform well under bandwidth constraints. However, F-F connections appear more sensitive to low bandwidth constraints, rendering the system unusable at 250 Kbps. In contrast, under ideal conditions (i.e., no bandwidth constraints or packet loss), F-F connections achieve higher throughput than C-C connections across all RTT values. Regarding packet loss, even at a rate of 10%, C–C connections remain usable, achieving an average throughput value of 460 Kbps. In contrast, F–F connections become unstable, exhibiting high variability of throughput values with an average of 160 Kbps. One reason F-F connections outperform C-C connections in our baseline conditions is the difference in the bitrate value used by Firefox and Chrome. Although we use a video with a fixed bitrate, browsers adjust the bitrate in real time. Thus, we measured the bitrate for each connection type every second over five minutes, repeating this process five times. The results showed an average bitrate of 4.6 Mbps for F-F connections and 2.4 Mbps for C-C connections. This also helps explain why B–B connections outperform P–P connections, as we used video encoded with a fixed target bitrate of 2 Mbps in FFmpeg, and Pion lacks built-in real-time encoding capabilities. However, as will be discussed in Section 4.4, bitrate is likely not be the sole factor explaining why F-F connections outperform C-C connections. Furthermore, Appendix C shows that these differences may also depend on the video content.

Insight #3: Comparative packet loss response of P–P and B–B connections: We speculate that P–P connections outperform B–B connections under high packet loss conditions for two reasons: (1) Pion instances on the receiving end process individual RTP packets directly, rather than relying on fully reassembled encoded frames for decapsulation. As a result, they do not need to wait for all packets to arrive to reconstruct a complete frame, unlike B–B connections, which require full frame reassembly to extract covert content; (2) because P–P connections use pre-encoded video and Pion lacks support for real-time video encoding, P-P connections are less susceptible to the video quality degradation typically caused by packet loss, such as reduced bitrate and frame size.

4.4 Varying Video Parameters

This section focuses on analysing the impact of varying video bitrates and resolution on throughput. To assess the effect of bitrate changes, we fixed the video resolution at 1280×720 and varied the bitrate across 500 Kbps, 1 Mbps, 2 Mbps, and 5 Mbps. For each bitrate, the video was encoded using the specified value as the target bitrate in FFmpeg. In B–B connections, the specified bitrate was enforced by setting the connection's maximum bitrate parameter, which serves as the target bitrate. For resolution testing, we did not specify the bitrate; instead, we allowed the codec to determine an appropriate value. For P–P connections, no target bitrate was set when encoding the video using FFmpeg. However, since FFmpeg encodes video at a significantly lower bitrate than B–B connections when no target bitrate is specified, we also present resolution testing results for P–P connections using a fixed target bitrate of 2 Mbps across all tested resolutions for comparison. We note that even when a target bitrate of 2 Mbps is specified, FFmpeg adjusts the actual bitrate based on the content and resolution during the encoding process. Baseline network conditions, as defined in Section 4.3, were applied for all tests. The tested resolutions were 426×240, 854×480, 1280×720, and 1920×1080. The results are presented in Figure 8.

Insight #4: Video resolution vs. video bitrate: The bitrate significantly impacts Obscura's throughput across all connection types. This behaviour is expected, as bitrate determines the amount of data used per unit of time to encode video. Regarding video resolution, the impact is primarily driven by frame size and the bitrate requirements associated with each specific resolution. In B-B connections, which use browser-based real-time variable bitrate encoding, lower resolutions are encoded with smaller average bitrates than higher

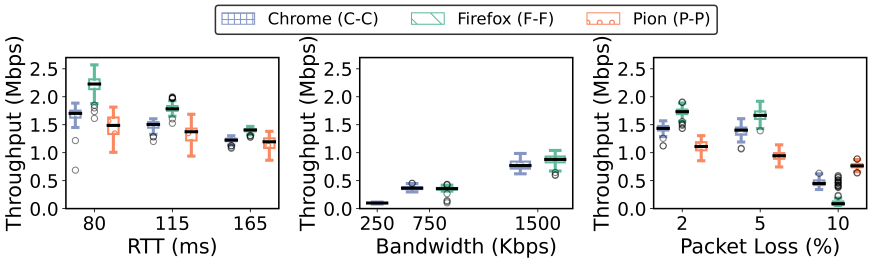


Figure 7: Results under varying network conditions. Leftmost figure: Varying system RTT; Middle figure: Varying bandwidth constraints; Rightmost figure: Varying packet loss. Box plots follow the order shown in the legend.

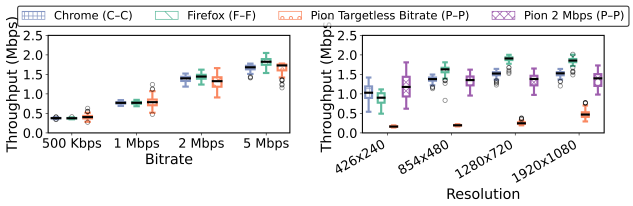


Figure 8: Results with varying video parameters. Leftmost figure: Varying bitrate; Rightmost figure: Varying resolution. Box plots follow the order shown in the legend.

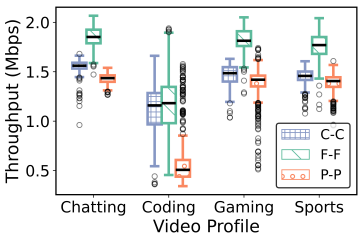


Figure 9: Box plots of throughput values for all video profiles and connection types.

resolutions for the same video content. A similar trend is observed in P-P connections, where FFmpeg adjusts the bitrate based on resolution, even when a target bitrate is specified during encoding. However, this adjustment occurs during the "offline" encoding process, before transmission, rather than in real time, as in B-B connections. This behaviour explains the upward trend in throughput as resolution increases for both targeted and non-targeted bitrate configurations in P–P connections. In the targeted bitrate scenario, however, throughput values at higher resolutions become more closely aligned, as FFmpeg attempts to maintain the specified target bitrate across resolutions, causing differences in actual bitrate to diminish with increasing resolution.

Insight #5: Bitrate and bottlenecks: With a target bitrate of 5 Mbps, F-F connections exhibit average throughput values closely matching those of the baseline scenario described in Section 4.3. This similarity arises because, under our testbench conditions, the baseline F–F connections naturally achieve an average bitrate close to 5 Mbps, as shown in Section 4.3. In contrast, C–C connections did not naturally reach this bitrate by default in our tests. When a 5 Mbps bitrate is enforced, the average bitrate of C–C connections increases accordingly, explaining the observed rise in throughput compared to the baseline values. However, even when C-C and F-F connections operate at the same higher bitrate values (above 2 Mbps), F-F connections outperform C-C in our testbench, suggesting that factors beyond bitrate contribute to the performance disparity. In addition, the throughput gains for B-B and P-P connections diminish between 2 Mbps and 5 Mbps, with this range exhibiting the smallest increase in throughput. On average, the throughput rises by 0.34 Mbps between 2 Mbps and 5 Mbps, compared to 0.38 Mbps between 500 Kbps and 1 Mbps, and 0.6 Mbps between 1 Mbps and 2 Mbps across all connection types. This pattern suggests that at higher bitrates, factors other than the video bitrate, such as system-level constraints, might become the primary limitations on throughput.

4.5 Varying Video Profiles

This section analyses the impact of different video profiles across all connection types. We selected four video profiles—Gaming, Chatting, Coding, and Sports—based on those used in prior related work [4]. For each profile, five distinct videos were used, and at least 100 throughput measurements were collected per video. Figure 9 presents the box plots of the 500 throughput measurements for each video profile per connection type. All videos were encoded with a target bitrate of 2 Mbps. However, as previously discussed, B–B connections enforce their bitrate due to the use of variable real-time bitrate encoding.

Insight #6: Impact of video profile on throughput: As expected, video profiles have a clear impact on throughput, as the content of a video determines its level of visual dynamism and, consequently, the required bitrate and frame sizes. The coding profile exhibits the lowest average throughput across all connection types, primarily due to its relatively static nature compared to other profiles. However, this does not imply that every segment of a coding video is static or results in low throughput. Some coding videos or specific segments may be as dynamic as those in other profiles, which explains the high maximum values and outliers observed in the throughput measurements for the coding profile. For example, in P–P connections, within the coding profile, the difference between the highest and lowest average bitrate reported by FFmpeg across the five coding videos was 1,069 Kbps, illustrating the significant

variability in content dynamism even within the same profile. Additionally, P-P connections exhibit multiple low outliers for the gaming profile. This discrepancy is likely due to one of the selected videos featuring a strategy game with relatively static content, leading to periods of lower throughput compared to the average gaming video. B–B connections exhibit fewer outliers in the gaming video profile, likely because the real-time bitrate encoding and the browser's tendency to maintain a target bitrate lead to more stable throughput than P–P connections.

5 Security Analysis

This section examines Obscura's resilience to attacks by cost-effective censors, as defined in our threat model, and discusses potential fingerprinting vectors that could be exploited within Obscura.

5.1 Proxy- and Packet Manipulation-Based Attacks

Proxy enumeration and active probing: The ephemeral nature of proxies, the heterogeneous pool of volunteers, and the vast number of available IP addresses that Obscura can leverage contribute to a resistance to proxy enumeration and IP blocking comparable to Snowflake [7, 40]. However, this resistance is contingent upon two factors: (1) the number of volunteers recruited and (2) the unwillingness of the censor to risk significant collateral damage by blindly blocking IP addresses or performing highly manual tasks to block proxies. For (1), given the relative ease of proxy deployment, we assume that Obscura can access a proxy pool of a scale similar to Snowflake's. Additionally, gamification and other strategies to expand the volunteer pool can be explored and implemented. Concerning (2), these efforts have not been observed against Snowflake, leading us to assume that censors appear reluctant to carry out such blind or manual attacks, resulting in a similar outcome for Obscura. However, as with data channel fingerprinting, we do not rule it out as a potential future attack, though we assume that the possible high proxy rejuvenation rates and the potential collateral damage may present challenges for censors attempting to carry it out. Regarding active probing, identifying Obscura proxies should be challenging for censors, as the volunteer pool can comprise a diverse range of devices with varying configurations. Furthermore, Obscura proxies do not expose ports or accept incoming connections.

Packet manipulation: Altering or adding packets in any way would be detected by the authentication and integrity mechanisms of the SRTP protocol. Adversaries may attempt to drop packets from the WebRTC media stream to disrupt the system's functionality. However, as demonstrated in Section 4.3, the reliability layer incorporated in our system, together with the WebRTC retransmission mechanisms, allows Obscura to tolerate a high percentage of dropped packets without compromising the covert connection, while maintaining reasonable throughput.

5.2 System Fingerprintibility

STUN Servers: Similar to Snowflake, we utilise a public pool of STUN servers instead of deploying our own, aiming to cause collateral damage to the censor if they attempt to block them. Additionally, the STUN message format for B-B connections is identical to that used by the browser. In contrast, P-P connections utilise the STUN message format of Pion. Should the detection of P-P connections based on the format of STUN messages become a concern in the future, disguising strategies could be implemented reactively.

Rendezvous Methods: As previously discussed, the resistance of rendezvous channels to fingerprinting depends on the specific method used. For rendezvous channels that use TLS and may be vulnerable to TLS fingerprinting [23], B-B connections will inherit the browser's TLS fingerprint if the methods are compatible with browser environments. In contrast, for rendezvous methods that cannot be used within browsers or are used in P-P connections, TLS fingerprinting can be mitigated by utilising the uTLS package [26].

Media Streams: We propose solutions that use multiple video sources rather than relying on a single static source for all users, thereby reducing the likelihood that the video can be exploited as a system fingerprint. For a more detailed discussion on the feasibility of video-based fingerprinting, see Appendix D. Additionally, we do not alter the size of frames or create new ones. Instead, we modify only the encrypted payload of the SRTP packets and ensure the intended transmission rate is maintained, regardless of whether there is data to encapsulate. While we do not currently consider machine learning or deep learning-based traffic analysis attacks, Obscura's traffic could still be vulnerable to such techniques, particularly through packet-based timing analysis [3]. However, recent research has shown that the accuracy claims of previous studies on detecting censorship evasion systems do not hold, particularly given the low base rates of circumvention traffic in practice [75]. Instead, the authors propose a methodology based on host detection and suggest that proxy ephemerality is an effective mitigation strategy. Nevertheless, a traffic analysis study against Obscura, and if necessary, the development of traffic shaping techniques or optimisations, should be considered should this threat become more realistic. Since Pion does not support real-time encoding, the video quality does not adapt to changing network conditions. This limitation may serve as a distinctive fingerprint of Pion itself, which censors might consider depending on the potential collateral damage and cost of detection.

Session Duration Behaviour: Maintaining an always-on video stream can increase bandwidth usage for users and volunteers, potentially affecting Obscura's usability. This raises an important question: when should the stream start and stop? We argue that maintaining the stream, even during user inactivity, is the most viable approach for two reasons. First, the system cannot reliably detect user idleness unless the user explicitly stops Obscura, and frequent stream connection interruptions may cause anomalous traffic patterns. Second, the system is never completely idle; the session layer connection must be maintained through control packets even without user interaction. Terminating the connection would result in loss of session state, forcing the user to restart the session. We acknowledge the trade-off between blending in, based on our understanding of typical WebRTC usage patterns, and ensuring system usability for users in censored regions. However, bandwidth constraints may also lead to unforeseen usage behaviours, particularly when connections are maintained until users actively disconnect, such as repeatedly connecting briefly to perform a task and then disconnecting immediately afterwards. Nevertheless, we propose a mitigation strategy that, while not ideal, may reduce bandwidth consumption while maintaining active connections: dynamically

lowering the video bitrate or resolution during assumed idle periods. Although this approach may introduce its own fingerprints, it more closely resembles standard browser-based WebRTC behaviour, which adapts video quality to network conditions, than repeatedly tearing down and reestablishing connections during idle periods.

Insertable Streams API: The Insertable Streams API is not negotiated during the establishment of a WebRTC connection, making it impossible to detect its use by observing the connection setup alone. While more sophisticated attacks, such as timing-based traffic analysis, could allow censors to infer its use, we consider this issue analogous to that discussed in the media stream paragraph.

DTLS Protocol: B-B connections provide proactive resistance to DTLS fingerprinting by ensuring that the fingerprints of the *ClientHello* and *ServerHello* messages match those of genuine browsers. To evaluate our resistance to DTLS fingerprinting, we captured 100 DTLS handshakes for each of the following WebRTC-based applications: Facebook Messenger, Google Meet, Discord, and a sample WebRTC application [54]. For each application, 50 handshakes were collected using Chrome and 50 using Firefox. We also captured 100 handshakes for Obscura, consisting of 50 from F-F connections and 50 from C-C connections. We used the dfind tool [38] to automatically identify unique identifiers in Obscura's DTLS handshakes by analysing the following *ClientHello* fields: length, fragment offset, DTLS version, cookie length, cipher suite length, cipher suites, extension length, and extensions. For the *ServerHello* messages, the same fields were analysed except for cipher suites and cipher suite length fields, which were replaced by the chosen cipher field. We observed no unique identifiers in either C–C or F–F connections, except for variations in the order of *ClientHello* extensions, particularly in C–C connections, while the set of extensions remained constant. However, the ordering varied across handshakes in Chrome, even for the same application, indicating that relying solely on this feature could lead to false positives. When the sample WebRTC application was excluded from the dataset, C-C connections still exhibited no unique identifiers. However, F-F connections showed unique identifiers in the *ServerHello* message, including a length of 86 bytes, a chosen cipher suite of 0x1301 (*TLS_AES_128_GCM_SHA256*), and an extension field length of 46 bytes We argue that this fingerprint is not unique to Obscura, but rather corresponds to the default WebRTC fingerprint in Firefox. Therefore, blocking such handshakes would also result in the blocking of legitimate Firefox-based WebRTC connections. For Pion-based clients or proxies, DTLS handshakes are implemented by Pion and, similar to Snowflake, may produce identifiable fingerprints. Thus, achieving resistance to DTLS fingerprinting requires a reactive strategy, involving either manual or automated updates to mitigate potential fingerprints [38].

5.3 Differential Degradation Attacks

Differential degradation attacks [59] aim to render censorship evasion systems unusable while allowing the carrier application to remain functional. The original paper introduced these attacks in the context of WebRTC-based systems, specifically targeting Snowflake and Protozoa [4]. Protozoa, like Obscura, is a WebRTC-based censorship evasion system that uses media streams for encapsulation. In this section, we focus on the DDA against Protozoa, as we argue that the attack on Snowflake can be mitigated by simply switching the carrier protocol from data channels to media channels. The attack proposed against Protozoa aims to degrade covert channel throughput while preserving acceptable video quality for regular WebRTC connections. Instead of causing indiscriminate packet loss, the idea is to selectively drop RTP packets, leveraging the original authors' observation that the loss of a single packet can result in the entire video frame being discarded. The authors demonstrate that even low frame loss rates can reduce throughput to single-digit KBps, as each lost frame may result in the loss of multiple covert packets. Considering that Protozoa lacks a reliability layer, requires complete frames to recover covert data, and relies on browser-based WebRTC connections, which degrade video quality in the presence of packet loss, we hypothesise that this attack could be mitigated by addressing each of these limitations, all of which are handled by P-P connections.

We perform the attack as described by the authors to test our hypothesis. First, we determine whether the connection is a WebRTC connection. If it is, we inspect all outgoing SRTP packets associated with the connection, examining their payload type and marker bit fields [8]. The payload type field indicates whether the packet contains video or audio content, while the marker bit field identifies the last packet of a frame. We use the Netfilter Linux library [41] to intercept packets at both the client and the proxy, leveraging the setup described in Section 4.2, and randomly sample a percentage of frames to drop. However, since we were unsure how the authors enforced packet loss—whether only a single packet from a frame was dropped, all packets were dropped, or some intermediate number of packets were dropped—we simulated two attack scenarios: the best-case and the worst-case scenarios. In the best-case, a single packet from a frame is dropped, while in the worst-case, every packet of a frame is dropped. Specifically, for the worst-case, we dropped all packets with a marker bit of 0 that occurred between two packets with a marker bit of 1, including the final packet with a marker bit of 1, for frames marked for dropping. For the best-case, we dropped only packets with a marker bit value of 1, for frames marked for dropping. This frame loss is applied bidirectionally to the stream, and we conduct tests at 1920×1080 and 426×240 resolutions for P-P connections, for both video encoded with a target bitrate of 2 Mbps and without any specified target bitrate. For each resolution, we evaluate different frame loss percentages. The results are presented in Figure 10. For context, at a resolution of 1920×1080 with 15% frame loss, the authors report that Protozoa's throughput drops to single-digit KBps, while the video stream maintains acceptable quality. A similar effect is observed at 426×240 with 50% frame loss.

Our results, shown in Figure 10, indicate that Obscura's throughput degradation in the best-case scenario is substantially lower than that reported for Protozoa in [59]. In the worst-case scenario (Figure 11), although throughput is significantly lower than in the best-case, it remains reasonably higher than the values reported in [59] for equivalent frame loss percentages. The average throughput drops below 100 Kbps (12.5 KBps) only for 426×240 at 2 Mbps with 50% and 70% frame loss, and for 1920×1080 at 2 Mbps with 45% frame loss. However, the authors did not evaluate a 45% frame loss for the 1920×1080 resolution, and their results indicate noticeable video quality degradation at 70% frame loss for the 426×240 resolution. We hypothesise that bitrate affects the rate of throughput

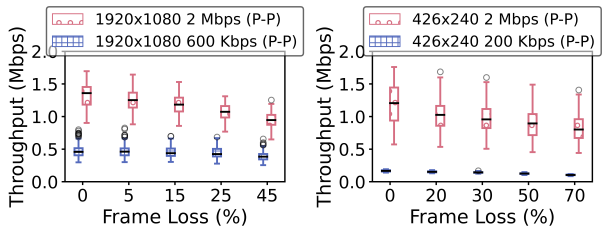


Figure 10: Results of DDAs on P-P connections for the best-case scenario.

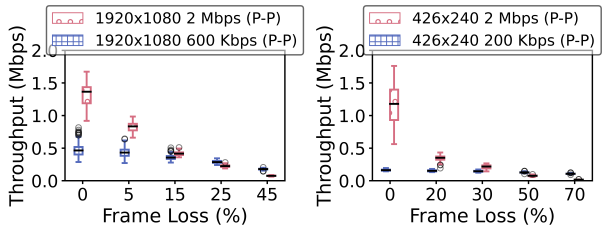


Figure 11: Results of DDAs on P-P connections for the worst-case scenario.

degradation under this attack. Higher bitrates, even at lower resolutions, increase frame size and the number of packets per frame, causing Obscura to embed more data per frame. Consequently, losing a single frame results in greater data loss, and more dropped packets require additional retransmissions. We also speculate that as the attack transitions from best to worst-case scenarios, the importance of decapsulation granularity decreases, while the reliability layer and Pion's lack of video quality adaptation in response to packet loss become more significant. However, even in the worst-case scenario, performing decapsulation at the RTP packet level may still be beneficial because it allows faster access to individual RTP packets containing covert Obscura packets, rather than waiting for the reassembly of complete frames. Since the features described above are present in P–P connections but absent in B–B connections, the latter are more susceptible to DDAs. This creates a trade-off between enhanced resilience to the media stream variant of DDAs and proactive resistance to DTLS fingerprinting within our system. If P–P connections are used, reactive patches must be applied to potential fingerprints to maintain DPI resistance, as in Snowflake. Further details on DDAs are provided in Appendix H.

6 Related Work

WebRTC-Based Censorship Evasion Systems: TorKameleon [70] is a preliminary experimental Tor pluggable transport designed to evaluate the resilience of WebRTC-based traffic encapsulation against active correlation attacks. Obscura builds upon TorKameleon by redesigning its architecture, trust model, and encapsulation techniques. It incorporates ephemeral proxies, a reliability layer, new client and proxy instance types, interoperability mechanisms, novel encapsulation algorithms, and a distinct threat model, accompanied by a novel security and performance analyses. While we have discussed Snowflake [7] throughout this paper, its primary distinction from Obscura lies in its use of data channels rather than media streams. Protozoa [4], another WebRTC media stream encapsulation-based system, modifies the Chromium browser by embedding hooks to intercept encoded frames. However, this design poses significant challenges for real-world deployment, requiring compiling a customised version of Chromium. Protozoa also lacks a reliability layer and, like Snowflake, responds reactively to DTLS fingerprinting. Users must repeatedly download and recompile updated Chromium builds with the necessary hooks, a process significantly more complex than simply updating a browser, as is the case with Obscura's B–B connections. Additionally, Protozoa lacks proxy ephemerality and relies on a trust model built around trusted volunteers. Stegozoa [24], an iteration over Protozoa, uses steganography to conceal data within video frames. However, it experiences reduced throughput due to a stricter threat model and inherits the limitations of Protozoa. Finally, uProxy [63], similar to Snowflake, uses WebRTC data channels, but its proxies are static.

Ephemeral Proxy-Based Censorship Evasion Systems: In addition to Snowflake, several other systems implement proxy ephemerality. Flash Proxy [22], the precursor to Snowflake, enables clients to switch between browser-based proxies in real time, although it relies on WebSocket connections rather than WebRTC. SpotProxy [35] automates the migration of proxies from censorship evasion systems across lower-cost cloud virtual machines. Other systems, including NetShuffle [34] and MassBrowser [40], adopt similar approaches to proxy ephemerality. For instance, NetShuffle obfuscates edge network domain-to-IP mappings: clients are assigned a proxy identifier (e.g., a domain name) and NetShuffle's DNS resolver maps this to a temporary, client-facing IP address that differs from the proxy's actual internal IP. Finally, although MassBrowser is not explicitly designed for ephemeral proxies, it pairs users with volunteer proxies selected from a dynamic pool, some of which may be untrusted by the user.

7 Conclusion

This paper introduced a novel encapsulation-based censorship evasion system designed to defend against a threat model characterised as the cost-effective censor. To address this adversary, we proposed a unified approach that integrates techniques not previously combined within a single system, specifically WebRTC media-based encapsulation and proxy ephemerality, while supporting both Pion and browser-based client and proxy instances. Within this threat model, we also presented a mitigation strategy for a new class of attacks, called differential degradation attacks, targeting WebRTC-based censorship evasion systems. We believe that the system can be deployed in real-world settings and that its core techniques could be incorporated into other WebRTC-based censorship evasion systems, such as Snowflake. However, we acknowledge that our design reflects explicit trade-offs between our intended goals and the practical constraints of real-world usability. While our design has allowed us to achieve our objectives, we acknowledge that alternative decisions may be necessary in more constrained deployment scenarios, such as those with limited bandwidth. We argue that such considerations can only be fully understood through real-world deployment and observing the system in operation.

Acknowledgments

We thank the anonymous reviewers and our revision editor for their valuable feedback, suggestions, and discussions. This work was supported by the FCT Ph.D. scholarship grant Ref. (PRT/BD/154787/2023), awarded by the CMU Portugal Affiliated Ph.D. program, by UID/04-516/NOVA Laboratory for Computer Science and Informatics (NOVA LINCS) with the financial support of FCT.IP (Fundação para a Ciência e a Tecnologia) and by national funds through FCT.IP, under the support UID/50014/2023 (https://doi.org/10.54499/UID/50014/2023). This work was developed within the scope of the project HOSKY, with reference 2024.07347.IACDC, co-funded by Component 5 - Capitalization and Business Innovation, integrated in the Resilience Dimension of the Recovery and Resilience Plan within the scope of the Recovery and Resilience Mechanism (MRR) of the European Union (EU), framed in the Next Generation EU, for the period 2021-2026, measure RE-C05-i08.M04 (to support the launch of a programme of R&D projects geared towards the development and implementation of advanced cybersecurity, artificial intelligence and data science systems in public administration, as well as a scientific training programme), as part of the funding contract signed between the Recovering Portugal Mission Structure (EMRP) and the FCT.IP, as intermediary beneficiary (https://doi.org/10.54499/2024.07347.IACDC). Finally, we acknowledge our use of ChatGPT4 to revise the text throughout the writing of the paper, correcting typos, grammatical errors, and awkward phrasing.

References

[1] Pion. Pion WebRTC. https://github.com/pion/webrtc
[2] Harald T. Alvestrand. 2021. Overview: Real-Time Protocols for Browser-Based Applications. RFC 8825. https://doi.org/10.17487/RFC8825
[3] Diogo Barradas, Nuno Santos, and Luís Rodrigues. 2018. Effective Detection of Multimedia Protocol Tunneling using Machine Learning. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association.
[4] Diogo Barradas, Nuno Santos, Luís Rodrigues, and 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*. Association for Computing Machinery.
[5] Ali C. Begen, Paul Kyzivat, Colin Perkins, and Mark J. Handley. 2021. SDP: Session Description Protocol. RFC 8866. https://doi.org/10.17487/RFC8866
[6] Big Buck Bunny. 2008. Big Buck Bunny. https://peach.blender.org/ Accessed: 2025-02-25.
[7] Cecylia Bocovich, Arlo Breault, David Fifield, Serene, and Xiaokang Wang. 2024. Snowflake, a censorship circumvention system using temporary WebRTC proxies. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association.
[8] Elisabetta Carrara, Karl Norrman, David McGrew, Mats Naslund, and Mark Baugher. 2004. The Secure Real-time Transport Protocol (SRTP). RFC 3711. https://doi.org/10.17487/RFC3711
[9] Junqiang Chen, Guang Cheng, and Hantao Mei. 2023. F-ACCUMUL: A Protocol Fingerprint and Accumulative Payload Length Sample-Based Tor-Snowflake Traffic-Identifying Framework. *Applied Sciences* (2023).
[10] Giovanni Cherubin, Rob Jansen, and Carmela Troncoso. 2022. Online Website Fingerprinting: Evaluating Website Fingerprinting Attacks on Tor in the Real World. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association.
[11] Google Cloud. 2025. *Google Cloud Pub/Sub*. https://cloud.google.com/pubsub Accessed: 2025-02-25.
[12] Cloudflare. 2025. *TURN Service*. https://developers.cloudflare.com/calls/turn/ Accessed: 2025-02-25.
[13] dmbb. 2020. *Protozoa repo*. https://github.com/dmbb/Protozoa/blob/master/machine_setup/Vagrantfile Accessed: 2025-02-25.
[14] MDN Web Docs. 2025. RTCEncodedVideoFrame. https://developer.mozilla.org/en-US/docs/Web/API/RTCEncodedVideoFrame Accessed: 2025-02-25.
[15] MDN Web Docs. 2025. Web Workers API - Web APIs. https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API Accessed: 2025-02-25.
[16] Arun Dunna, Ciaran O'Brien, and Phillipa Gill. 2018. Analyzing China's Blocking of Unpublished Tor Bridges. In *8th USENIX Workshop on Free and Open Communications on the Internet (FOCI 18)*. USENIX Association.
[17] Roya Ensafi, David Fifield, Philipp Winter, Nick Feamster, Nicholas Weaver, and Vern Paxson. 2015. Examining How the Great Firewall Discovers Hidden Circumvention Servers. In *Proceedings of the 2015 Internet Measurement Conference*. Association for Computing Machinery.
[18] Yuzhou Feng, Ruyu Zhai, Radu Sion, and Bogdan Carbunar. 2023. A Study of China's Censorship and Its Evasion Through the Lens of Online Gaming. In *USENIX Security Symposium*. USENIX. https://www.usenix.org/system/files/usenixsecurity23-feng.pdf
[19] FFmpeg. 2025. FFmpeg. https://www.ffmpeg.org/ Accessed: 2025-02-25.
[20] David Fifield. 2020. Turbo Tunnel, a good way to design censorship circumvention protocols. In *Free and Open Communications on the Internet*. USENIX.
[21] David Fifield and Mia Gil Epner. 2016. Fingerprintability of WebRTC. arXiv:1605.08805
[22] David Fifield, Nate Hardison, Jonathan Ellithorpe, Emily Stark, Roger Dingledine, Phil Porras, and Dan Boneh. 2012. Evading Censorship with Browser-Based Proxies. In *Privacy Enhancing Technologies Symposium*. Springer.
[23] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. 2015. Blocking-resistant communication through domain fronting. *Privacy Enhancing Technologies* 2015, 2 (2015).
[24] 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 on Asia Conference on Computer and Communications Security*. Association for Computing Machinery.
[25] Sergey Frolov, Jack Wampler, and Eric Wustrow. 2020. Detecting Probe-resistant Proxies. In *Network and Distributed System Security*. The Internet Society.
[26] Sergey Frolov and Eric Wustrow. 2019. The use of TLS in Censorship Circumvention. In *Network and Distributed System Security*. The Internet Society.
[27] Google. 2025. RTP VP8 Packetization Format Source Code. https://chromium.googlesource.com/external/webrtc/+/lkgr/modules/rtp_rtcp/source/rtp_format_vp8.cc Accessed: 2025-02-27.
[28] Devashish Gosain, Kartikey Singh, Rishi Sharma, Jithin S, and Sambuddho Chakravaty. 2024. Out in the Open: On the Implementation of Mobile App Filtering in India. In *Passive and Active Measurement Conference*. Springer.
[29] David Hasselquist, Ethan Witwer, August Carlson, Niklas Johansson, and Niklas Carlsson. 2024. Raising the Bar: Improved Fingerprinting Attacks and Defenses for Video Streaming Traffic. *Privacy Enhancing Technologies* 2024, 4.
[30] Nguyen Phong Hoang, Jakub Dalek, Masashi Crete-Nishihata, Nicolas Christin, Vinod Yegneswaran, Michalis Polychronakis, and Nick Feamster. 2024. GFWeb: Measuring the Great Firewall's Web Censorship at Scale. In *USENIX Security Symposium*. USENIX.
[31] Nguyen Phong Hoang, Arian Akhavan Niaki, Jakub Dalek, Jeffrey Knockel, Pellaeon Lin, Bill Marczak, Masashi Crete-Nishihata, Phillipa Gill, and Michalis Polychronakis. 2021. How Great is the Great Firewall? Measuring China's DNS Censorship. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association.
[32] Bart Jansen, Timothy Goodwin, Varun Gupta, Fernando Kuipers, and Gil Zussman. 2018. Performance Evaluation of WebRTC-based Video Conferencing. *SIGMETRICS Perform. Eval. Rev.* (2018). https://doi.org/10.1145/3199524.3199534
[33] Ari Keränen, Christer Holmberg, and Jonathan Rosenberg. 2018. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal. RFC 8445. https://doi.org/10.17487/RFC8445
[34] Patrick Tser Jern Kon, Aniket Gattani, Dhiraj Saharia, Tianyu Cao, Diogo Barradas, Ang Chen, Micah Sherr, and Benjamin E. Ujcich. 2024. NetShuffle: Circumventing Censorship with Shuffle Proxies at the Edge. In *Symposium on Security & Privacy*. IEEE.
[35] Patrick Tser Jern Kon, Sina Kamali, Jinyu Pei, Diogo Barradas, Ang Chen, Micah Sherr, and Moti Yung. 2024. SpotProxy: Rediscovering the Cloud for Censorship Circumvention. In *USENIX Security Symposium*. USENIX.
[36] Alexander Master and Christina Garman. 2023. A Worldwide View of Nation-state Internet Censorship. In *Free and Open Communications on the Internet*.
[37] Philip Matthews, Jonathan Rosenberg, Dan Wing, and Rohan Mahy. 2008. Session Traversal Utilities for NAT (STUN). RFC 5389. https://doi.org/10.17487/RFC5389
[38] Theodor Signebøen Midtlien. 2024. *Reducing distinguishability of DTLS for usage in Snowflake*. Master's thesis. Norwegian University of Science and Technology (NTNU).
[39] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. 2018. DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery.
[40] Milad Nasr, Hadi Zolfaghari, Amir Houmansadr, and Amirhossein Ghafari. 2020. MassBrowser: Unblocking the Censored Web for the Masses, by the Masses. In *Network and Distributed System Security*. The Internet Society.
[41] Netfilter. 2025. Netfilter. https://www.netfilter.org/ Accessed: 2025-02-25.
[42] Arian Akhavan Niaki, Shinyoung Cho, Zachary Weinberg, Nguyen Phong Hoang, Abbas Razaghpanah, Nicolas Christin, and Phillipa Gill. 2020. ICLab: A Global, Longitudinal Internet Censorship Measurement Platform. In *2020 IEEE Symposium on Security and Privacy (SP)*.

[43] Sadia Nourin, Van Tran, Xi Jiang, Kevin Bock, Nick Feamster, Nguyen Phong Hoang, and Dave Levin. 2023. Measuring and Evading Turkmenistan's Internet Censorship. In *The International World Wide Web Conference*. ACM.
[44] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. 2010. The Akamai network: a platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.* (2010). https://doi.org/10.1145/1842733.1842736
[45] Ramakrishna Padmanabhan, Arturo Filastò, Maria Xynou, Ram Sundara Raman, Kennedy Middleton, Mingwei Zhang, Doug Madory, Molly Roberts, and Alberto Dainotti. 2021. A multi-perspective view of Internet censorship in Myanmar. In *Free and Open Communications on the Internet*. ACM.
[46] Pion. 2024. vp8_packet.go. https://github.com/pion/rtp/blob/a21194ecfb5362261a0dc4af1f68e4a8944df345/codecs/vp8_packet.go#L6 Accessed: 2025-02-25.
[47] Pion. 2025. bandwidth-estimation-from-disk. https://github.com/pion/webrtc/tree/master/examples/bandwidth-estimation-from-disk Accessed: 2025-02-25.
[48] Pion. 2025. constants.go. https://github.com/pion/webrtc/blob/44062a7a78006d7e6b2f97d991ad3c12a648150c/constants.go#L28 Accessed: 2025-02-25.
[49] Pion. 2025. rtp-to-webrtc. https://github.com/pion/webrtc/tree/master/examples/rtp-to-webrtc Accessed: 2025-02-25.
[50] Michael Pu, Andrew Wang, Anthony Chang, Kieran Quan, and Yi Wei Zhou. 2024. Exploring Amazon Simple Queue Service *SQS* for Censorship Circumvention. In *Free and Open Communications on the Internet*.
[51] Raymond Rambert, Zachary Weinberg, Diogo Barradas, and Nicolas Christin. 2021. Chinese Wall or Swiss Cheese? Keyword filtering in the Great Firewall of China. In *WWW*. ACM.
[52] Marc B. Rosen, James Parker, and Alex J. Malozemoff. 2021. Balboa: Bobbing and Weaving around Network Censorship. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association.
[53] Zach Rosson, Felicia, and Carolyn Tackett. 2024. The most violent year: internet shutdowns in 2023. https://www.accessnow.org/internet-shutdowns-2023/. Accessed: 2025-02-12.
[54] WebRTC samples. 2023. Stream from a video to a peer connection. https://webrtc.github.io/samples/src/content/capture/video-pc/ Accessed: 2025-02-25.
[55] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. 2003. RTP: A Transport Protocol for Real-Time Applications. RFC 3550. https://doi.org/10.17487/RFC3550
[56] Selenium. 2025. Selenium. https://www.selenium.dev/ Accessed: 2025-02-25.
[57] skywind3000. 2020. KCP - A Fast and Reliable ARQ Protocol. https://github.com/skywind3000/kcp/blob/1.7/README.en.md Accessed: 2025-02-25.
[58] Sophie Stalla-Bourdillon, Evangelia Papadaki, and Tim Chown. 2014. From porn to cybersecurity passing by copyright: How mass surveillance technologies are gaining legitimacy … The case of deep packet inspection technologies. *Computer Law & Security Review* (2014).
[59] Zhen Sun and Vitaly Shmatikov. 2024. Differential Degradation Vulnerabilities in Censorship Circumvention Systems. arXiv:2409.06247 [cs.CR]
[60] The Tor Project. 2025. Performance. https://metrics.torproject.org/torperf.html Accessed: 2025-02-25.
[61] A. Troianovski and V. Safronova. 2022. Russia takes censorship to new extremes, stifling war coverage. *The New York Times* (2022). https://www.nytimes.com/2022/03/04/world/europe/russia-censorship-media-crackdown.html Accessed: 2025-02-12.
[62] Michael Carl Tschantz, Sadia Afroz, Anonymous, and Vern Paxson. 2016. SoK: Towards Grounding Censorship Circumvention in Empiricism. In *Symposium on Security & Privacy*. IEEE. https://www.eecs.berkeley.edu/~sa499/papers/oakland2016.pdf
[63] uProxy. -. uProxy. https://www.uproxy.org/ Accessed: 2025-02-12.
[64] A. Vakali and G. Pallis. 2003. Content delivery networks: status and trends. *IEEE Internet Computing* (2003). https://doi.org/10.1109/MIC.2003.1250586
[65] Verizon. 2025. IP Latency Statistics. https://www.verizon.com/business/terms/latency/ Accessed: 2025-02-25.
[66] Vasilis Ververis, Lucas Lasota, Tatiana Ermakova, and Benjamin Fabian. 2023. Website blocking in the European Union: Network interference from the perspective of Open Internet. *Policy & Internet* (2023).
[67] Afonso Vilalonga. 2025. GitHub Issue #233: Frame packetization. https://github.com/w3c/webrtc-encoded-transform/issues/233 Accessed: 2025-02-25.
[68] Afonso Vilalonga. 2025. Obscura—Artifact. https://github.com/AfonsoVilalonga/Obscura---Artifact
[69] Afonso Vilalonga, Kevin Gallagher, Osman Yağan, João S. Resende, and Henrique Domingos. 2025. Extended Abstract: Using TURN Servers for Censorship Evasion. In *Free and Open Communications on the Internet*.
[70] Afonso Vilalonga, Joao S. Resende, and Henrique Domingos. 2023. TorKameleon: Improving Tor's Censorship Resistance with K-anonymization and Media-based Covert Channels . In *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE Computer Society.
[71] Afonso Vilalonga, João S. Resende, and Henrique Domingos. 2024. Looking at the Clouds: Leveraging Pub/Sub Cloud Services for Censorship-Resistant Rendezvous Channels. In *Free and Open Communications on the Internet*.
[72] Paul Vines, Samuel McKay, Jesse Jenter, and Suresh Krishnaswamy. 2024. Communication Breakdown: Modularizing Application Tunneling for Signaling Around Censorship. *Privacy Enhancing Technologies* 2024, 1 (2024).
[73] W3C. 2025. WebRTC Encoded Transform. https://github.com/w3c/webrtc-encoded-transform/blob/main/explainer.md Accessed: 2025-02-25.
[74] Ryan Wails, Andrew Stange, Eliana Troper, Aylin Caliskan, Roger Dingledine, Rob Jansen, and Micah Sherr. 2022. Learning to Behave: Improving Covert Channel Security with Behavior-Based Designs. *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2022, 3 (2022). https://doi.org/10.56553/popets-2022-0068
[75] Ryan Wails, George Arnold Sullivan, Micah Sherr, and Rob Jansen. 2024. On Precisely Detecting Censorship Circumvention in Real-World Networks. In *Network and Distributed System Security*. The Internet Society.
[76] Patrik Westin, Henrik Lundin, Michael Glover, Justin Uberti, and Frank Galligan. 2016. RTP Payload Format for VP8 Video. RFC 7741. https://doi.org/10.17487/RFC7741
[77] Philipp Winter and Stefan Lindskog. 2012. How the Great Firewall of China is Blocking Tor. In *Free and Open Communications on the Internet*. USENIX.
[78] Mingshi Wu, Jackson Sippe, Danesh Sivakumar, Jack Burg, Peter Anderson, Xiaokang Wang, Kevin Bock, Amir Houmansadr, Dave Levin, and Eric Wustrow. 2023. How the Great Firewall of China Detects and Blocks Fully Encrypted Traffic. In *USENIX Security Symposium*. USENIX.
[79] xtaci. 2023. smux. https://github.com/xtaci/smux Accessed: 2025-02-25.
[80] Diwen Xue, Benjamin Mixon-Baca, ValdikSS, Anna Ablove, Beau Kujath, Jedidiah R. Crandall, and Roya Ensafi. 2022. TSPU: Russia's Decentralized Censorship System. In *Internet Measurement Conference*. ACM.
[81] Diwen Xue, Reethika Ramesh, ValdikSS, Leonid Evdokimov, Andrey Viktorov, Arham Jain, Eric Wustrow, Simone Basso, and Roya Ensafi. 2021. Throttling Twitter: an emerging censorship technique in Russia. In *Internet Measurement Conference*. ACM.
[82] Tony Huiquan Zhang, Jianhua Xu, and Jinjin Liu. 2024. How Do Toothless Tigers Bite? Extra-institutional Governance and Internet Censorship by Local Governments in China. *The China Quarterly* (2024).

A Implementation

Our Pion-based instances were developed in Go (version 1.21.2) using the Pion framework (version V4), while the browser-based instances were implemented in JavaScript and HTML. For testing, we used Node.js (version 18.18) to serve the WebRTC web application. The Go codebase comprises approximately 5,000 lines of code, with some overlap across components, while the JavaScript and HTML codebase consists of around 4,000 lines, including overlapping sections for different browser implementations.

Browser-based proxies and clients use the Web Workers API [15] to handle tasks such as encapsulation, decapsulation, and synthetic frame generation in separate threads, preventing performance bottlenecks. Each browser-based proxy supports only one client to avoid overloading volunteers' devices, although it can be configured to support multiple clients. Additionally, we developed a Go-based intermediary layer to enable communication between the client software and the browser-based WebRTC application. This layer processes incoming client packets, delivers them to the KCP/smux stack, and forwards them to the browser via a local WebSocket. The Go layer is used exclusively for browser-based client instances.

B Rendezvous Channels

In this section, we present the rendezvous channels developed parallel with Obscura in more detail. While we provide a brief overview of both, detailed explanations and justifications for each are beyond the scope of this paper and are left to their respective publications. Additionally, it is worth noting that the TURN rendezvous system is still under development.

***Google Pub/Sub Service* [71]:** The Google Pub/Sub service rendezvous channel utilises the Google Pub/Sub service [11] as a relay for information exchange. The client initially contacts the broker by sending an initialisation message to a designated topic, which is

subscribed to exclusively by brokers. The client establishes a symmetric key with the broker, which is then used to encrypt messages sent to the user through a topic that is exclusively subscribed to by users. Since multiple users can access the same topic simultaneously, a unique symmetric key must be established between each user and the broker. Once the client and the broker have established the symmetric key, Obscura's WebRTC signalling phase proceeds, as shown in Figure 2. The Google Pub/Sub service functions as the rendezvous channel, routing all messages between the broker and the client through the Google Pub/Sub servers, ensuring that direct communication between the client and the broker does not occur. The rendezvous nature of this channel arises from the fact that to block the system, a censor would need to block access to the Google Pub/Sub API, which would also cause collateral damage by affecting all other regular applications using it.

***TURN Servers* [69]:** The rationale for using TURN servers as a signalling channel is to leverage the infrastructure provided by TURN service providers to facilitate information exchange between the broker and the client. WebRTC is a widely adopted technology stack censors are often reluctant to block. TURN servers enable communication between two peers when a direct WebRTC peer-to-peer connection is impossible due to restrictive NAT configurations. Consequently, some of these services (e.g., Cloudflare TURN servers [12]) may remain accessible in certain censored regions, allowing them to function as relays between users and brokers. The broker and client establish a symmetric key to ensure message confidentiality, preventing TURN servers from accessing the content of the messages. Once this symmetric key is established, the remaining message exchange follows the signalling phase protocol outlined in Figure 2.

C Evaluation of Animations as Video

In this section, we evaluate the effectiveness of using animations, rather than actual video, as the carrier for our covert channel. For both P-P and B-B connections, only the video component is rendered. However, audio can also be generated automatically, as discussed in related work [52].

For B-B connections, we implemented a simple deterministic bouncing ball animation at a resolution of 1280×720. This animation is rendered on a web canvas and captured directly from it. Modern browsers typically throttle background animations by reducing rendering frequency when a page is not in focus, requiring the browser window to remain active throughout the session. Consequently, operating a browser-based client or proxy instance that utilises animations becomes impractical for users or volunteers, thereby hindering the system's usability. To address this issue, both Firefox and Chrome must be configured to turn off background throttling. This can be done by modifying specific browser settings either manually through the GUI or via Selenium.

For P-P connections, we replicated the same bouncing ball animation at a resolution of 1280×720. However, unlike browser-based animations, generating frames in Pion is more complex. In our setup, each animation frame is generated individually and sent to a locally running FFmpeg process, which encodes the frames in real time and transmits them back to the Pion instance. These encoded frames are then transmitted as RTP packets over a WebRTC connection. All of this is handled automatically on the client side. We configure FFmpeg for real-time encoding with a target bitrate of 2 Mbps. In practice, the actual bitrate of the encoded animation we developed is significantly lower, in the low hundreds of Kbps, likely due to the animation's low visual complexity.

We present the results for all connection types under the network conditions described in Section 4.3 in Figure 12. P–P connections consistently outperform B–B connections across all configurations in this test, likely due to differences in the methods used to generate the animations, which may affect the animations themselves, as well as possibly differing handling of low-complexity visual content by FFmpeg compared to browser-based encoders. As in Section 4.3, both F–F and P–P connections fail to operate under a 250 kbps bandwidth constraint in our test environment, even when streaming animations. However, under 10% packet loss, F–F connections exhibit greater stability than that reported in Section 4.3.

One of the main advantages of using animations is the wide variety available, with many examples readily implemented online, particularly for browsers, and the relative ease of developing new ones. We demonstrated the use of animations using a simpler, less visually dynamic example, featuring a small ball bouncing on a static background. However, more dynamic animations can be developed and used to achieve higher bitrates. Obscura can be bundled with a diverse set of deterministic and randomized animations tailored to different network conditions and usage scenarios, providing a large pool of potential carriers for covert traffic.

D Video-based Fingerprints

In this section, we assess the feasibility of fingerprinting the system based on three factors: (1) the consistent use of identical video content across users or proxies, (2) the repeated looping of video content, and (3) the profiling of streamed content in relation to typical usage patterns of WebRTC-based web applications.

Recent studies have explored the use of machine learning and related techniques to infer the nature of streamed content, particularly in the context of HTTP-based streaming [29]. Although we believe such methods could be adapted to detect whether a connection is streaming specific video content, even when only encrypted traffic is observable (as in the case of WebRTC), we consider them to be complex machine-learning-based attacks that may be less feasible than other types of attacks encountered in the wild. This approach not only falls outside the current threat model but may also be impractical for censors to implement due to its complexity. However, when only a single identical video or a fixed set of videos is distributed to all users, a censor could potentially identify a fingerprint in the video or set of videos based on metadata, such as the size of the first X frames, and use this information to block connections through a possible LTA attack. To mitigate this risk, we develop solutions in Section 3.4 that ensure that different users and volunteers can use distinct videos without relying on the same content.

To illustrate how it might be possible to identify whether a specific video is being streamed using only traffic traces, even if the video content itself is unknown, we present the frame sizes of one five-minute video from each of the four video profiles (sports,

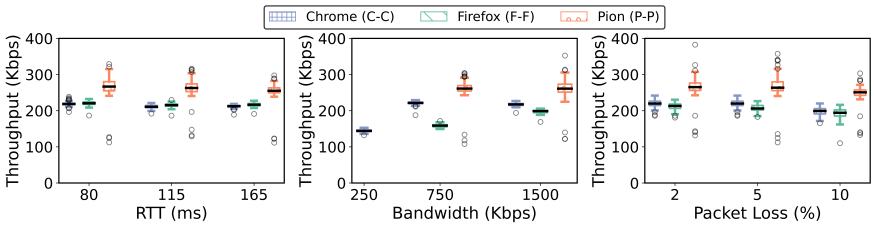


Figure 12: Performance under varying network conditions using an animation as a carrier for the covert channel. Leftmost figure: Varying system RTT; Middle figure: Varying bandwidth constraints; Rightmost figure: Varying packet loss. Box plots follow the order shown in the legend.

gaming, chatting, and coding) introduced in Section 4.5, for F-F, C-C, and P-P connections in Figures 13, 14, and 15, respectively. These connections were established using a default WebRTC web application, rather than Obscura, with two peers of the same type (i.e., Firefox, Chrome, or Pion). Each five-minute video was looped three times, with dotted vertical red lines marking the start of each loop, resulting in a total capture duration of 15 minutes per graph. Streaming was unidirectional, with one browser tab or instance functioning solely as the sender and the other as the receiver. All connections were established in a localhost environment. Frame sizes were reconstructed by capturing SRTP traffic and summing the payload sizes of consecutive SRTP packets until a packet with the marker bit set was encountered. Although this method may not precisely replicate the original frame size distribution, it provides a sufficiently accurate approximation for our analysis. Additionally, we do not claim that this would be the method used be the censor to try and fingerprint our system or any other system. Our aim with these results is to demonstrate that different types of video content produce distinct network fingerprints based on their traffic patterns, with differences sometimes discernible even to the naked eye. In theory, such fingerprints could be used to identify network connections that are using a particular video. We do not claim that censors are currently conducting this type of analysis, nor that it could be deployed in a real-world scenario; rather, we discuss its potential feasibility and practicality, as well as the mitigation strategies that we have developed or could develop to address it.

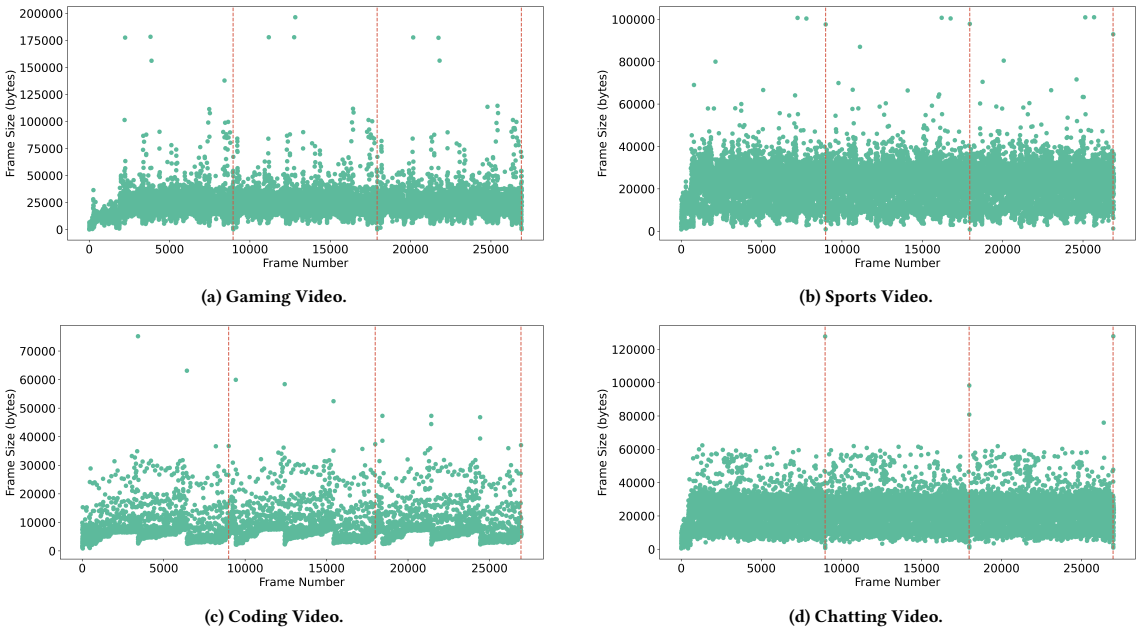
The plots indicate that the four videos exhibit distinct and recognisable traffic patterns, which are consistently preserved across different browser implementations. For example, similar traffic patterns are observed when the same video is streamed via Chrome and Firefox. We hypothesise that the differences between browser-based and Pion-based traffic traces stem from the different encoders used, as well as the lack of real-time video adaptability in P-P connections compared to the real-time video quality adaptability in B-B connections.

Looping video content also produces observable recurring traffic patterns. This looping behaviour is more identifiable in Pion, which maintains fixed encoding parameters and does not adapt video quality based on network conditions. In contrast, browser-based connections dynamically adjust video quality, resulting in varying frame sizes across loops. Nevertheless, the looping pattern remains visually recognisable in the packet traces. However, we argue that while looping may offer a fingerprinting vector when the video duration is known, it poses challenges in a black-box setting. These challenges include accurately detecting loop occurrences when the video duration is unknown to the censor, as well as mitigating false positives caused by naturally looping videos or intentional user behaviours (e.g., repeatedly playing a music video during a screen-sharing session). Consequently, the effectiveness and practicality of this approach in real-world scenarios remain uncertain. In some cases, where only a small set of videos is used, a censor may be able to learn the traffic characteristics of the videos within that set and block connections after a certain number of loops. However, if users and volunteers use a sufficiently large and diverse set of videos (e.g., by using different animations and video files or shaping the frames to match the metadata characteristics of varying content, as described in Section 3.4), we hypothesise that detecting looping behaviour and reliably blocking connections becomes significantly more difficult for the censor. Additionally, if only a small set of videos is used, we assume it would be easier for the censor to fingerprint the system based on which videos are included in that set, rather than relying solely on looping behaviour.

Finally, it could be argued that streamed content or the duration of streaming sessions may reflect user behaviour that deviates from "regular" WebRTC behaviour and could serve as a fingerprinting vector. However, we argue that these factors do not offer a reliable or practical basis for fingerprinting. While this view may be subject to debate, it is supported by the vast diversity of WebRTC applications, each characterised by distinct usage patterns. Establishing a standard for "normal" usage is inherently challenging, as these applications serve varied user demographics. For instance, Discord targets a different audience than traditional, business-oriented video conferencing platforms. As a result, both the nature of shared content and the duration of sessions can vary substantially across applications. Even within the same application, different users may exhibit distinct usage patterns. In light of this variability, we hypothesise that it would be difficult for a censor to reliably classify streamed content or session duration as anomalous without causing significant collateral damage.

E Evaluation of Asymmetric Connection Types

In this section, we present the results for the asymmetric connection types, as shown in Figure 16. By "asymmetric", we refer to connections with different client and proxy instances, specifically Pion–Firefox (P–F), Pion–Chrome (P–C), Firefox–Pion (F–P),



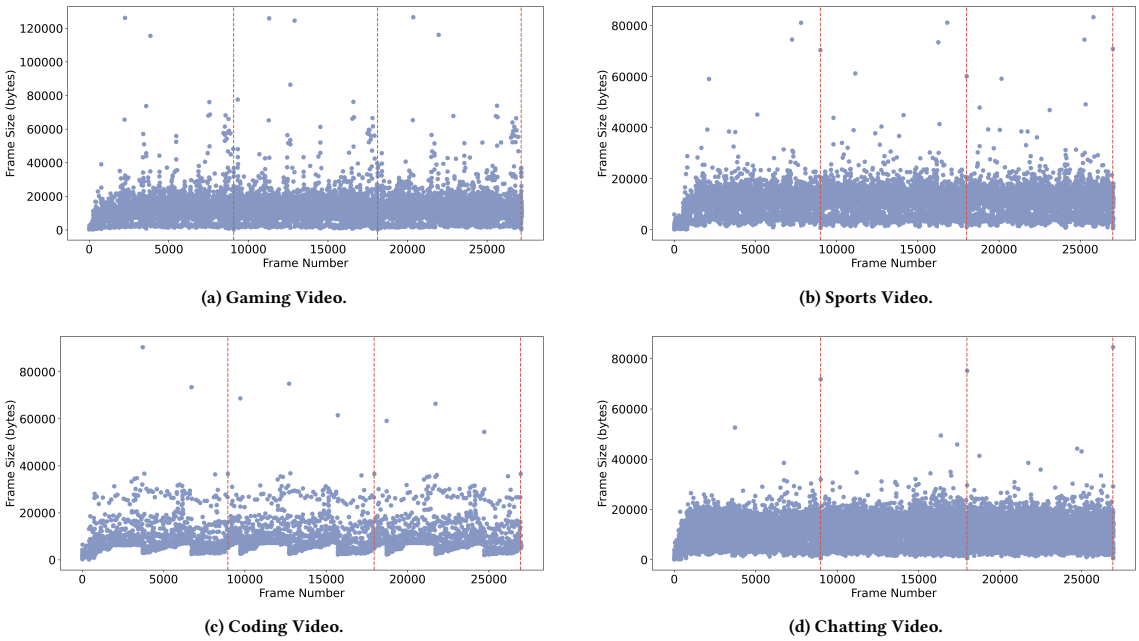
(a) Gaming Video.

(b) Sports Video.

(c) Coding Video.

(d) Chatting Video.

Figure 13: Traffic patterns based on frame sizes for a single video from each of four video profiles, streamed over a Firefox-to-Firefox connection.



(a) Gaming Video.

(b) Sports Video.

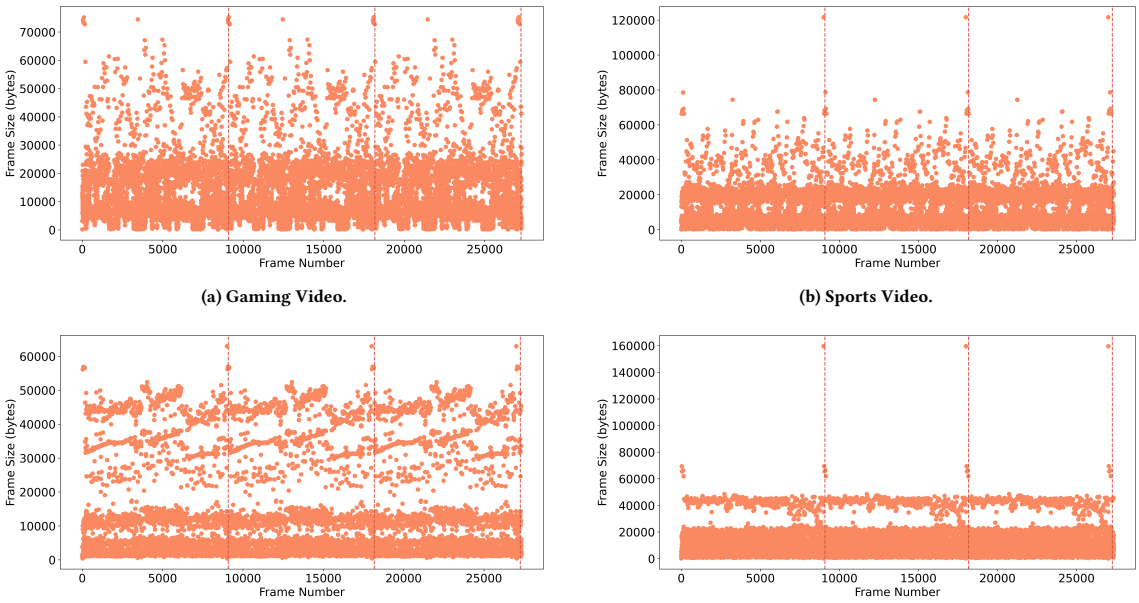
(c) Coding Video.

(d) Chatting Video.

Figure 14: Traffic patterns based on frame sizes for a single video from each of four video profiles, streamed over a Chrome-to-Chrome connection.

Firefox–Chrome (F–C), Chrome–Pion (C–P), and Chrome–Firefox (C–F).

Based on the results obtained using the testing methodology described in Section 4.1, which involved downloading a file, we hypothesise that the proxy component has a significant impact on the overall system performance in asymmetric connections. This outcome is expected because, in this experimental setup, the proxy is primarily responsible for transmitting a substantially larger volume of encapsulated traffic to the client, specifically the file content.



(a) Gaming Video.

(b) Sports Video.

(c) Coding Video.

(d) Chatting Video.

Figure 15: Traffic patterns based on frame sizes for a single video from each of four video profiles, streamed over a Pion-to-Pion connection.

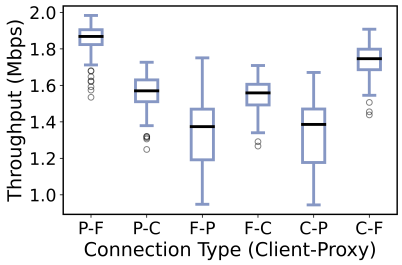


Figure 16: Box plots of throughput values for all asymmetric connections.

Consequently, connections such as P–F and C–F achieve the highest average throughput, followed by P–C and F–C, while F–P and C–P exhibit the lowest average throughput. In other words, within our testbench and for this test, Firefox proxy instances consistently outperform Chrome proxy instances, which, in turn, outperform Pion proxy instances. This hierarchy aligns with the performance trends observed across most tests presented in this paper.

F Pluggable Transport

In this section, we evaluate the performance of Obscura as a Tor pluggable transport. To ensure realistic conditions, no relays are explicitly specified in the circuit configuration, except for a Tor bridge under our control, which operates as an Obscura Tor bridge (VM4 in the setup described in Section 4.2). The HTTP server is hosted directly on the dedicated server, without the use of a virtual machine, to ensure public accessibility via the dedicated server's public IP address. This configuration enables the Tor exit relay to reach the server. Other than that, the test bench follows the same setup discussed in Section 4.2. We conduct tests with RTTs between VM1 (the client) and VM3 (the proxy), set to 15 ms, 50 ms, and 100 ms, while maintaining a constant RTT of 50 ms between VM3 and VM4. The RTT between the Tor exit node and the HTTP server is not measured and is therefore not reflected in the x-axis values of the graph, although it does contribute to the total RTT. Additionally, we perform tests under varying packet loss rates of 2%, 5%, and 10%.

We present the results in Figure 17. These findings are consistent with those presented in Section 4.3, reflecting similar trends across the tested connection types and indicating that the Tor network has only a minor impact on system throughput.

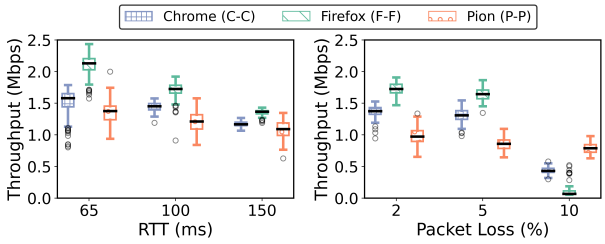


Figure 17: Results under varying network conditions for Obscura in pluggable transport mode. Leftmost figure: varying system RTT; Rightmost figure: varying packet loss. Box plots follow the order shown in the legend.

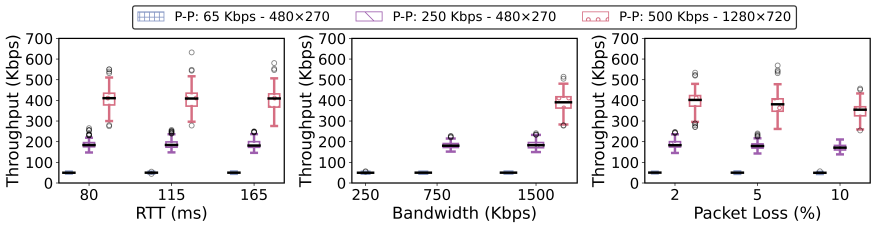


Figure 18: Results for varying network conditions in P-P connections with different video encodings. Leftmost figure: varying system RTT; Middle figure: varying bandwidth constraints; Rightmost figure: varying packet loss. Box plots follow the order shown in the legend.

G Pion's Tolerance to Bandwidth Constraints

To confirm that Pion's lack of real-time variable bitrate and resolution encoding limits its usability in bandwidth-constrained networks, we evaluated P-P connections under the varying network conditions described in Section 4.3, using low bitrate and low resolution encoded videos. The results are presented in Figure 18. We use the same video encoded with three different parameter settings: 1280×720 at 500 Kbps, 480×270 at 250 Kbps, and 480×270 at 65 Kbps, corresponding to the respective bitrates and resolutions. The results show that as video encoding quality decreases (i.e., lower bitrates and resolutions), the system becomes better suited for bandwidth-constrained environments. However, they also emphasise the importance of selecting appropriate encoding parameters for both video and audio based on network conditions, particularly bandwidth constraints, when using P-P connections.

H Differential Degradation Attacks

While testing our hypothesis on mitigating DDAs, we also conducted preliminary tests on F–F and C–C connections under the best-case and worst-case scenarios described in Section 5.3. Our goal was to verify whether we would obtain the same results as those reported in the referenced paper, considering that our system includes a reliability layer, whereas Protozoa does not. However, at the time of writing, our results showed higher throughput values than those reported in the referenced paper for the best-case scenario, while for the worst-case scenario we got similar values. The main differences observed in the best-case scenario pertained to the level of degradation in system throughput compared to that of Protozoa. For C-C connections at high resolution (1920×1080), throughput remained relatively high up to 25% frame loss, comparable to our baseline. However, at frame loss rates of 45% or higher, Obscura became unstable. For F-F connections, throughput remained high up to 15% frame loss but became unstable at 25%, exhibiting highly variable throughput and becoming unusable at 45%. At lower resolutions (426×240), C–C connections remained usable, with throughput values relatively close to the baseline up to 30% frame loss. At a 50% frame loss rate, the connection began to show instability, with throughput fluctuating between high and low values and at 70%, the throughput dropped to approximately 2 KB/s. For F-F connections, at 426×240 resolution, the connection was usable with relatively high throughput up to 20% frame loss but became unstable at 30%, exhibiting high throughput variability. It degraded further at 50% and was unusable at 70%. To contextualise, at the time of writing, the authors reported that Protozoa achieved an average throughput of 7 KBps for 15% frame loss using Chrome at a resolution of 1920×1080. For a resolution of 426×240 and 30% frame loss, Protozoa achieved a throughput of 10.4 KBps.

In the worst-case scenario, the results observed for both C–C and F–F connections were similar to those reported in the original paper and considerably worse than in the best-case scenario. This is expected, as the packet drop rate is higher. Since all packets from X% of the frames are dropped (where X denotes the percentage of frame loss), rather than only a single packet per frame, more covert data is lost, resulting in lower throughput values compared to the best-case scenario. For example, in a video with 10 frames where each frame contains an average of 5 packets, dropping 50% of the frames in the worst-case scenario would result in an average of 25 dropped packets out of 50, excluding retransmissions, whereas in the best-case scenario, it would result in an average of 5 dropped packets.

We hypothesise that the discrepancies between our best-case scenario results and those reported by the authors in [59] may stem from three primary factors. First, our system includes a reliability layer, which is absent in Protozoa. This layer is configured for faster retransmissions, potentially improving resilience to attacks by enabling quicker detection and recovery from packet loss. Second, differences in the experimental setup may also account for the observed variation. Specifically, we use a different censorship evasion system, a custom-developed WebRTC application based on the default WebRTC implementations [54] (in contrast to the authors' use of AppRTC in [59]), and a distinct test environment. Additionally, based on our interpretation of the paper, the authors appear to use a specific, unmodified commit of Protozoa [59]. Inspection of that commit revealed that Protozoa depends on a Chromium version from 2020 [13], suggesting that discrepancies in browser versions, given the substantial version gap, may have influenced the results, since we used the most recent versions of both Chrome and Firefox available at the time of writing. Third, since the attack code is not publicly available, our implementation may differ from that of the original authors. Consequently, what we define as the best-case scenario may not correspond to the attack the authors actually performed. Instead, the authors may have considered the worst-case scenario, or something closer to it, which is why we also tested the worst-case scenario in our experimental evaluation. We also observed the WebRTC retransmission mechanism activate, which we believe helped the system maintain higher functionality at lower

frame loss rates, particularly in the best-case scenario. However, it is unclear whether the authors also marked retransmitted packets for dropping during the DDAs, as we did in our implementation of the attack.

In conclusion, P–P connections demonstrate greater resilience to DDAs than B–B connections, particularly under the worst-case scenario. This trend persists, though less pronounced, in the best-case scenario, where B–B connections require higher frame loss rates for the attack to significantly degrade throughput or induce instability, whereas lower frame loss rates are enough to cause instability or render the system unusable in the worst-case scenario. Thus, we conclude that P–P connections are the most viable option to resist DDAs across all tested scenarios; however, their use shifts Obscura's defence against DTLS fingerprinting from a proactive to a reactive strategy. This shift occurs because Pion instances do not leverage native browser fingerprints, requiring frequent updates, patches, or alternative methods to replicate them whenever vulnerabilities or new fingerprints are identified in Pion, similar to Snowflake. Nonetheless, we argue that, depending on the attack implementation and system configuration, B–B connections may still be a usable option against DDAs, particularly in the best-case scenario.