Hiding Content in Video Calls to Resist Censorship

Raúl Aguilar Arroyo, 429631

17/01/2025

Abstract

Internet censorship impacts nearly 25% of the global population, restricting freedom of expression and access to information. This paper evaluates six WebRTC-based protocols—SkypeMorph, CensorSpoofer, Protozoa, Stegozoa, TorKameleon, and Snowflake—designed for censorship circumvention. It analyzes their evolution from protocol mimicry to advanced steganographic methods, emphasizing the trade-offs between security, usability, and performance. While modern systems like Snowflake and TorKameleon improve on accessibility and resistance to detection, architectural vulnerabilities persist. This study highlights critical insights and future directions for developing robust censorship circumvention tools.

1 Introduction

Authoritarian governments increasingly use sophisticated methods like deep packet inspection and protocol disruption to suppress dissent, affecting nearly 25% of the global population[2]. Censored content often includes political speech, anonymity tools, and information-sharing platforms[2, 16]. To combat these controls, researchers are developing new censorship circumvention tools, with WebRTC-based video calls offering a promising approach. These tools leverage encrypted peer-to-peer connections to obscure traffic, making it difficult for censors to detect and block.

This paper evaluates six notable protocols (3) The objectives are to: (1) provide a comprehensive description of these methods, (2) assess their security, usability, and real-world applicability, based on sections 2.2 objectives, and (3) offer insights for future research and tool development. The analysis reveals a clear evolution from simple protocol mimicry to more sophisticated steganographic techniques, highlighting trade-offs between security, performance, and usability. Despite advancements, no current system fully optimizes all aspects, and future developments should focus on bridging the gap between security and usability.

2 Background

2.1 Key Concepts

- **Real-Time Communication and Secure Transport**
  - **WebRTC (Web Real-Time Communication)**: A real-time communication framework enabling secure, peer-to-peer multimedia exchanges without additional plugins. Its widespread adoption in applications like Discord makes it challenging for censors to block without collateral damage[12, p. 15].
  - **Tunneling:** Encapsulating one network protocol within another to create a secure and private communication channel over a public network, also it makes the traffic appear to be legitimate[4, 15].
- **Information Hiding Techniques**
  - **Steganography:** Embedding hidden information within multimedia content (e.g. videos) to conceal the existence of the data.[12, p. 8-14].
  - **Protocol Obfuscation:** Modifies the behavior of protocols to make them harder for censors to detect. Examples include the use of non-standard ports, data fragmentation, or the introduction of noise[11, p. 98].
- **Censorship Analysis and Countermeasures**
  - **Correlation Attacks:** These techniques aim to de-anonymize users by analyzing traffic patterns. **Passive correlation attacks** involve monitoring traffic without direct interference, while **active correlation attacks** inject identifiable marks into traffic to trace its origin. Both approaches exploit traffic analysis vulnerabilities to compromise anonymity[14, p. 1491].
  - **Deep Packet Inspection**: DPI is a traffic analysis technique that examines the content of data packets along with their headers, allowing censors to identify traffic based on its content[9].

2.2 Key Traits of an Ideal Censorship Circumvention System

An ideal censorship-resistant system must meet three main objectives:

2.2.1 Undetectability

Traffic should **blend seamlessly with legitimate network activity**, resisting detection by censors. It must **withstand deep packet inspection**, ensuring intercepted data cannot be distinguished from regular traffic. Furthermore, **avoidance of static or predictable communicationsignatures** is crucial to prevent identification and blocking.

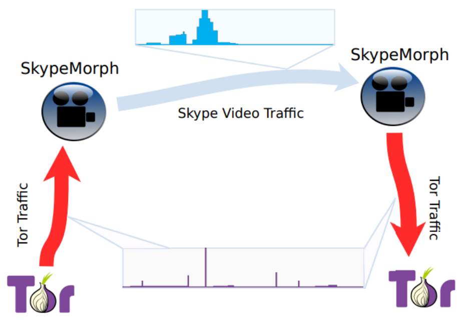

Figure 1: Overview of the SkypeMorph Architecture. The histograms show the distribution of packet sizes in Tor (at the bottom) and Skype video (at the top).[11, p. 102]

2.2.2 User and Communication Security

Strong **anonymity** should disconnect user identities from their network activity, ensuring user and communication security. **Data confidentiality** must guarantee that messages remain unintelligible to unauthorized parties. Furthermore, **resistance to both active and passive correlation** attacks is essential to protect users and their communications.

2.2.3 Accessibility, scalability, and resilience

Systems must support diverse user bases while maintaining **low latency and high throughput**. **Decentralization** is vital to eliminate single points of failure and ensure continued operation even under adverse network conditions. Ease of use is a priority, enabling users with **minimal technical expertise** to install, configure, and operate the system. **Compatibility** across platforms and devices ensures broad accessibility.

3 Summary of most relevant methods

In order to better understand the current methods of censorship circumvention, we will summarise those considered most relevant, ordered according to their complexity and evolution. The selected methods for the analysis are SkypeMorph, CensorSpoofer, Protozoa, Stegozoa, TorKameleon, and Snowflake. These methods have been ordered chronologically, with earlier systems presented first, and also by complexity, starting with simpler approaches and progressing to more advanced techniques.

3.1 SkypeMorph

3.1.1 Obfuscation Technique

SkypeMorph disguises Tor traffic by mimicking Skype video calls. It modifies traffic patterns, such as packet sizes and timing intervals, to emulate legitimate Skype behavior.[11]

3.1.2 Internal Operation

The SkypeMorph process has two phases: Setup and Traffic Modeling [11, p. 102].

Setup Phase: The client and bridge[1] log into the Skype API with unique credentials. The bridge listens for calls, while the client prepares a connection. After exchanging public keys and UDP ports via Skype text messages, both generate a shared secret key, verified by hash exchange. If successful, the client starts a brief video call, leaving the UDP connection active. The bridge dynamically selects new UDP ports to emulate Skype behavior[11, p. 102].

Traffic Modeling Phase: After setup, the video call ends but the UDP connection is used to transmit Tor traffic disguised as Skype traffic. A traffic shaping oracle modifies Tor packets to mimic Skype patterns. SkypeMorph also adds AES encryption and HMAC authentication atop Tor's existing encryption[11, p. 103].

3.1.3 Performance

SkypeMorph incurs a bandwidth overhead as it transmits data volumes similar to a legitimate Skype video call, even when Tor traffic is insufficient to fill the packets. Tests reveal an average download speed of 34 KB/s with a 28% overhead[11, 4], which should be sufficient for standard internet browsing.

3.1.4 Censor Resistance and Vulnerabilities

The main vulnerability of SkypeMorph lies in the fact that, despite its efforts to mimic Skype traffic, it does not actually run the Skype application. This makes it susceptible to a number of active and passive attacks that can distinguish it from genuine Skype traffic[10].

It is vulnerable to pattern detection, inability to interact with Skype's supernode network and inability to fully mimic Skype's behavior. Being distinguishable even with passive attacks[10, p. 70-72].

3.1.5 Usability

Using SkypeMorph can be challenging for average users due to its relatively complex deployment and configuration requirement [5].

3.1.6 Conclusion

In conclusion, SkypeMorph was a significant milestone in traffic obfuscation and censorship resistance but has limited relevance for modern use. Its partial imitation of Skype traffic makes it detectable, even by less sophisticated censors, and its complex setup and reliance on pre-recorded traces pose practical challenges. While innovative for its time, SkypeMorph is neither safe nor reliable against today's advanced censorship methods.

[1]The bridge in SkypeMorph facilitates anonymous Tor access by relaying client connections to the Tor network[11].

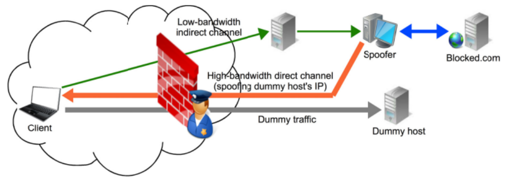
Low-bandwidth indirect channel
Spoofer
Blocked.com
High-bandwidth direct channel (spoofing dummy host's IP)
Client
Dummy traffic
Dummy host

Figure 2: The CensorSpoofer framework. The user pretends to communicate with a fictitious external host, sending URLs to the spoofer through an indirect low-bandwidth channel (such as email or steganographic instant messaging). The spoofer retrieves the blocked pages and injects the censored data downstream to the user, spoofing the IP of the fictitious host.[15]

3.2 CensorSpoofer

3.2.1 Obfuscation Technique

CensorSpoofer is a censorship evasion system that exploits the asymmetry between upstream and downstream web traffic. It uses indirect channels, like instant messaging or email, for upstream traffic, where the user sends URLs, employing steganography to hide requests within normal communications [15].

3.2.2 Internal Operation

CensorSpoofer uses IP spoofing[2] and a "dummy host" to deliver censored content over downstream traffic. It simulates a VoIP communication between the user and the dummy host, while the proxy server injects the censored web content, making it appear as if it comes from the dummy host [15, p. 123].

An invitation system creates a trusted network, requiring new users to be endorsed by existing members. Each user receives unique identifiers for secure communication with the spoofer, preventing compromised users from identifying others [15, p. 126].

The selection of decoy (or dummy) hosts involves a sophisticated verification process, including port scanning to confirm the availability of necessary services (SIP[3], RTP, RTCP) and validation of network paths to ensure communication plausibility. The system also continuously monitors the status of these hosts to maintain robust communication. These hosts are carefully selected to appear as legitimate VoIP clients by verifying that the correct ports, such as SIP for voice traffic, are open [15, p. 127].

To illustrate how this works, imagine a user wanting to access a censored website. The user initiates a normal VoIP call to their SIP identifier. The spoofer selects a dummy host and responds with a manipulated OK message, establishing a fake VoIP session. The user sends encrypted audio packets (random content) to the decoy host while transmitting the actual URLs through the steganographic upstream channel. The spoofer retrieves the requested content and sends it back disguised as VoIP traffic, appearing to come from the decoy host.

[2]IP spoofing involves creating packets with a false source address.

[3]SIP is a communication protocol used primarily to establish, maintain and terminate real-time communication sessions.

3.2.3 Performance

Experimental results show that CensorSpoofer enabled clients to download a blocked Wikipedia page in China. Downloading the full 160 KB page took 27 seconds, while the HTML file alone took 6 seconds [15]. Although CensorSpoofer takes longer for larger pages due to the need to reshape traffic into VoIP, download times for smaller content, like HTML files, remain acceptable for basic internet navigation.

3.2.4 Vulnerabilities

Over time, several vulnerabilities in the system have been demonstrated, particularly related to SIP probing, as discussed in Houmansadr's work [10]. For example, sending invalid SIP messages causes a genuine SIP client to respond with a "400 BadRequest," while CensorSpoofer fails to respond, revealing its nature. This allows a censor to break the system's unobservability without affecting legitimate traffic [10].

3.2.5 Usability

CensorSpoofer's complex setup, including the need for an invitation from an existing user, makes it challenging for non-technical users. Additionally, it has never been deployed for real-world use, remaining a proof of concept rather than an accessible tool.

3.2.6 Conclusion

CensorSpoofer was an innovative censorship circumvention system that introduced obfuscation techniques and asymmetric architecture to hide web traffic and circumvent censorship controls. However, it is not secure by modern standards and should not be used.

3.3 Protozoa

3.3.1 Obfuscation Technique

Protozoa conceals transmitted information by replacing data in encoded video frames with the payload of IP packets, occurring after video compression, and modifying the WebRTC stack of the Chromium browser [4, p. 36].

3.3.2 Internal Operation

Protozoa creates a covert tunnel within a WebRTC video call between a client in a censored region and a proxy in a free-access region. Its architecture consists of four main components[4]:

- **Gateway Server**: Runs on both client and proxy, managing the covert tunnel[4].
- **Encoder Service**: Encapsulates the client's IP packets into Protozoa messages, which are inserted into the video stream[4].

- **Decoder Service**: Extracts IP packets from Protozoa messages at the proxy[4].
- **SOCKS proxy server**: Redirects client traffic to its final destination on the Internet[4].

Protozoa works in two stages. In the first stage, the client and proxy establish a WebRTC connection by sharing a URL and password via an out-of-band channel (e.g., email). Once both join the chat room, they initiate the video call (a P2P connection)[4].

In the second stage, Protozoa replaces the video data with IP packets. The client's IP packets are encapsulated into Protozoa messages and sent through the WebRTC call. The proxy extracts the IP packets from the messages, which are then forwarded to the SOCKS proxy server, directing the data to its final destination[4].

3.3.3 Performance

Protozoa achieved a covert channel throughput of 1.4 Mbps with 98.8% efficiency under normal network conditions, supporting common Internet applications like web browsing and bulk data transfer. Lab tests with a 640x480 video resolution reported an average throughput of 1422 Kbps [4, p. 42-43].

3.3.4 Vulnerabilities

Protozoa is designed to resist machine-learning-based traffic analysis by state-level adversaries [4, p. 36], but it has critical vulnerabilities [7, 4]. Its reliance on peer-to-peer (P2P) transmission is problematic, as many WebRTC applications use relay servers, allowing adversaries controlling these gateways to detect Protozoa traffic [12].

Additionally, Protozoa lacks anonymity mechanisms like Tor, exposing users when detected and undermining its security objectives.

3.3.5 Usability

Deploying Protozoa can pose challenges, necessitating users to compile a modified version of the Chromium browser, and it lacks compatibility with Tor [14, p. 1491].

3.3.6 Conclusion

Protozoa demonstrates innovative strategies for evading censorship but is hampered by vulnerabilities make it less secure for long-term use in modern, highly monitored environments.

3.4 Stegozoa

3.4.1 Obfuscation Technique

Stegozoa improves Protozoa's security by embedding transmitted information within video frames of WebRTC calls. It uses Syndrome-Trellis Coding (STC), an advanced adaptive steganographic method, combined with data encoding in the least significant bits of Quantized Discrete Cosine Transform (QDCT) coefficients of residual video frames. This technique modifies QDCT coefficients minimally, hiding data without visible distortion, leveraging compression properties of the VP8 codec used by WebRTC [12, p. 18-20].

3.4.2 Internal Operation

Implemented as a library (*libstegozoa*) for WebRTC applications, Stegozoa intercepts video frames before encoding, embedding covert data using the chosen steganographic method. On the receiving side, the library extracts the hidden data from video frames[7].

To ensure efficient and reliable communication, Stegozoa incorporates a message fragmentation and reassembly protocol for transmitting smaller packets through the covert channel. It also includes a retransmission mechanism to mitigate data loss [7].

3.4.3 Performance

Stegozoa's performance is determined by its steganographic embedding capacity. In lab tests, it achieved a throughput of 11.4 Kbps with a high embedding capacity ($\alpha = 0.50$), sufficient for small data exchanges like Twitter feeds or Wikipedia articles, but unsuitable for bulk data transfer or live streaming [7].

The parameter α controls the fraction of embedding capacity used, balancing performance and security. Higher α increases throughput but introduces more video distortion, making the system more vulnerable to detection. Lower α enhances security by reducing distortion and detection risks, albeit at the cost of reduced throughput [7].

3.4.4 Vulnerabilities

Stegozoa's primary vulnerability lies in its lack of robustness to encrypted media manipulation [12]. Re-encoding the video stream, whether intentional by adversaries or incidental via WebRTC gateways, can alter the DCT coefficients used for embedding covert data, rendering messages unrecoverable [12].

Despite this, Stegozoa demonstrates strong resistance to traffic analysis and steganalysis. Experimental results show detection classifiers struggled to differentiate Stegozoa transmissions from legitimate WebRTC traffic, achieving an AUC near 0.5—indicating detection was essentially random [12, 7].

Thus, while Stegozoa's reliability is limited under specific conditions, it remains largely unobservable.

3.4.5 Usability

Stegozoa integrates as a library within WebRTC applications, simplifying usage for end users. Users only need to **initiate a video call** with a trusted contact outside the censored region, and Stegozoa will **tunnel traffic** through the call. It has also been tested with various calling services [7].

3.4.6 Conclusion

Stegozoa enhances Protozoa with advanced steganographic techniques, improving resistance to traffic analysis and ste-

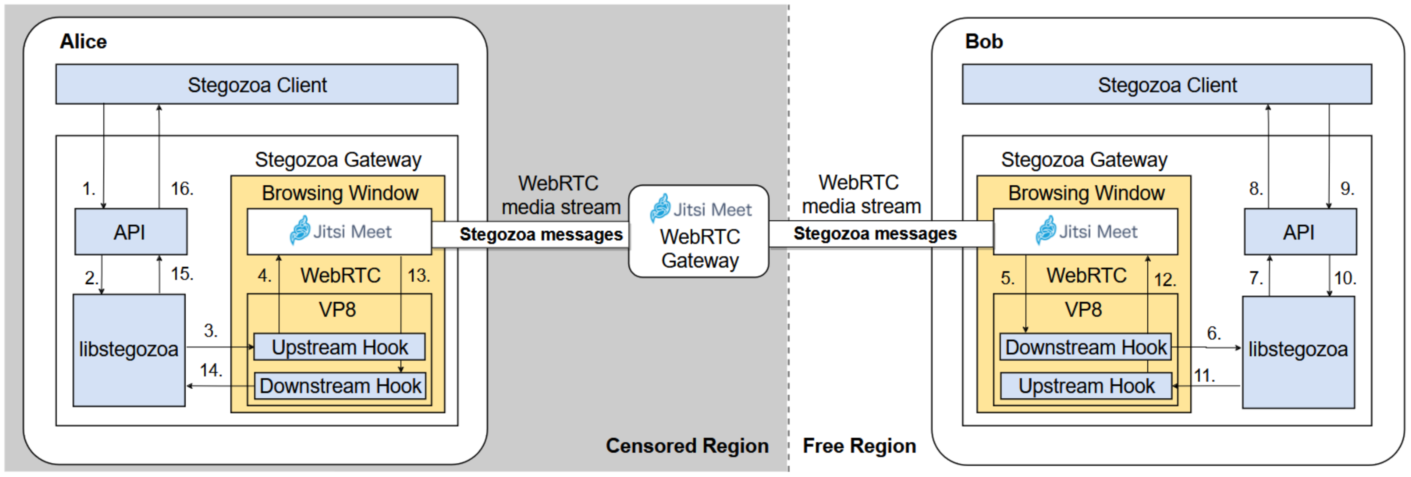

Figure 3: Architecture of Stegozoa: The components of the system are highlighted in blue[7]

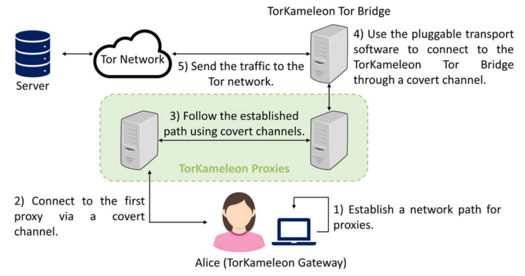
TorKameleon Tor Bridge
Server
Tor Network
5) Send the traffic to the Tor network.
4) Use the pluggable transport software to connect to the TorKameleon Tor Bridge through a covert channel.
3) Follow the established path using covert channels.
TorKameleon Proxies
2) Connect to the first proxy via a covert channel.
1) Establish a network path for proxies.
Alice (TorKameleon Gateway)

Figure 4: System Model and Workflow of the TorKameleon Ecosystem. When using the pluggable transport without proxies, the user establishes a direct connection to the TorKameleon Tor Bridge through the TorKameleon pluggable transport client-side, which operates on the user's local device.[14]

ganalysis while simplifying usability. However, vulnerabilities to video manipulation and recoding remain. Adopting solutions proposed by Cruzat[12] could strengthen its robustness, ensuring better security and effectiveness in modern censorship evasion.

3.5 TorKameleon

3.5.1 Obfuscation Technique

TorKameleon is a tool designed to enhance Tor's resistance to correlation attacks. It operates as a Tor pluggable transport, using multipath routing and traffic encapsulation to hide user communication [14].

3.5.2 Internal Operation

TorKameleon implements K-anonymization by splitting user traffic across multiple proxies in a pre-Tor network, mixing it with the traffic of K additional users to reduce correlation risks[14]. It operates in three modes:

- **Pluggable Transport Mode:** Acts as a pluggable transport for Tor, hiding traffic in WebRTC or TLS tunnels[14].
- **Standalone Mode:** Deploys a network of proxies to route fragmented traffic, making correlation harder[14].
- **Combined Mode:** Combines both previous modes, routing traffic through proxies before reaching Tor.[14].

In all modes, TorKameleon follows this process[14]:

1. **Route Establishment:** The user configures a proxy route, or the TorKameleon gateway software determines it automatically.
2. **Proxy Connection:** The user connects to the first proxy in the route.
3. **Encapsulation and Forwarding:** Traffic is encapsulated and sent through proxies until final proxy.
4. **Connection to Tor:** Final proxy sends traffic to Tor via the TorKameleon bridge.

3.5.3 Performance

In the scenario where security is to be maximized, TorKameleon has latencies between 530-655 ms, and a througput of about 1500 Kbps. This is sufficient for surfing the Internet[14].

3.5.4 Vulnerabilities

TorKameleon has been tested against passive and active correlation attacks. It resists passive attacks, making its traffic hard to identify, and also shows resistance to active attacks with small data blocks. However, using larger data blocks can create vulnerabilities[14]. With proper configuration, TorKameleon is secure and unobservable, but its recent nature means **there may still be undetected vulnerabilities**.

3.5.5 Usability

The current TorKameleon implementation is a prototype, designed as a proof of concept rather than a user-friendly tool. It lacks integration with web browsers and requires manual configuration of components like WebRTC parameters, STUN/TURN servers, and the TorKameleon proxy. The setup also involves tools like Docker and network configurations, making it unsuitable for non-technical users[1, 14].

3.5.6 Conclusion

TorKameleon demonstrates strong resistance against passive and active correlation attacks in specific configurations, but its security relies on proper setup. Due to its novelty and limited peer review, further research and testing are needed to uncover potential weaknesses.

3.6 Snowflake

3.6.1 Obfuscation Method

Snowflake does not rely on traditional information-hiding techniques such as steganography or packet manipulation to mask transmitted data. Its censorship resistance derives from the use of a **vast, constantly changing network of temporary proxies** known as "snowflakes." These proxies, run by volunteers, act as intermediaries, relaying traffic from censored clients via peer-to-peer WebRTC protocols to a centralized bridge[4]. The information itself is not hidden; instead, Snowflake's strategy focuses on making communication harder to block by avoiding fixed censorship targets [6].

3.6.2 Internal Operation

The connection process through Snowflake is divided into three phases: rendezvous, peer-to-peer connection establishment and data transfer[6].

The rendezvous phase begins when a client seeking to circumvent censorship sends a message to the broker, a central server that facilitates connections between clients and proxies. To avoid being blocked, the client uses an indirect communication channel resistant to censorship, such as domain fronting, AMP cache, or SQS.[5]. The broker manages proxies that periodically check for clients requiring service. Upon connection, the broker matches the client with an available proxy, considering factors like NAT compatibility. It sends the client's SDP (Session Description Protocol) offer to the proxy, which responds with its own SDP, completing the match[6].

In the peer-to-peer connection establishment, the client and proxy exchange information to form a direct connection via WebRTC, overcoming obstacles such as firewalls and NAT. This uses ICE (Interactive Connectivity Establishment), which tests address combinations to find a working pair. Snowflake minimizes dependency on potentially blockable TURN servers by implementing a NAT classification system. Clients and proxies self-report their NAT types, and the broker avoids pairing devices that require TURN. Simultaneously, the proxy establishes a connection to the bridge using WebSocket[6].

In the data transfer phase, the proxy relays information between the client and the bridge without altering its content. A protocol stack secures the data: WebRTC provides confidentiality via DTLS (Datagram Transport Layer Security), while Turbo Tunnel ensures session reliability. If a proxy fails, the client reconnects through a new rendezvous without noticeable interruption. Finally, the bridge routes the client's traffic through the Tor network for additional anonymity[6].

3.6.3 Performance

Although the authors do not provide precise details on Snowflake's resource consumption and performance, it is described as an **ultralightweight system**[6]. Snowflake has multiple implementations to suit different needs and platforms such as a Web browser extension, Web Badge, a command-line implementation and through the mobile app, Orbot.

3.6.4 Vulnerabilities

Snowflake's main vulnerability is its protocol fingerprint, which allows censors to identify and block its connections. During the rendezvous phase, patterns in TLS traffic can reveal its use, and in the WebRTC connection, STUN messages and DTLS fingerprint exhibit distinct characteristics. This can condition the system, as censors can block access to the broker or disrupt WebRTC connections[6].

These vulnerabilities condition the system by allowing censors to block access to the broker, disrupting the initial pairing between clients and proxies, or identify and stop WebRTC connections, interrupting the communication tunnel[6]. It also puts the identity of users at risk, however snowflake is a modern protocol, under constant development and the developers have already mentioned that they are working on these problems[6, 13].

3.6.5 Usability

Snowflake is designed for a seamless user experience, requiring no special configurations or technical knowledge from the client. The client software automatically and transparently handles bridge connections, proxy selection, and data transfer, offering an experience similar to other web applications [6]. It is also the only method introduced that supports mobile devices (through orbot)[6]. This is important to increase its reach to as many users as possible.

3.6.6 Conclusion

Snowflake demonstrates a robust approach to censorship resistance through its distributed and adaptive proxy network. However, the protocol's reliance on WebRTC and

[4]The bridge in Snowflake is a centralized component that receives traffic from proxies and forwards it to the final destination on the Internet, acting as an intermediary between censored clients and the outside world.[6]

[5]AMP cache and SQS act as hidden communication channels that allow Snowflake clients to connect to the broker without being detected by censors. Both methods are based on legitimate and widely used services, making it difficult to selectively block them without causing a negative impact on other services[6, p. 2638]

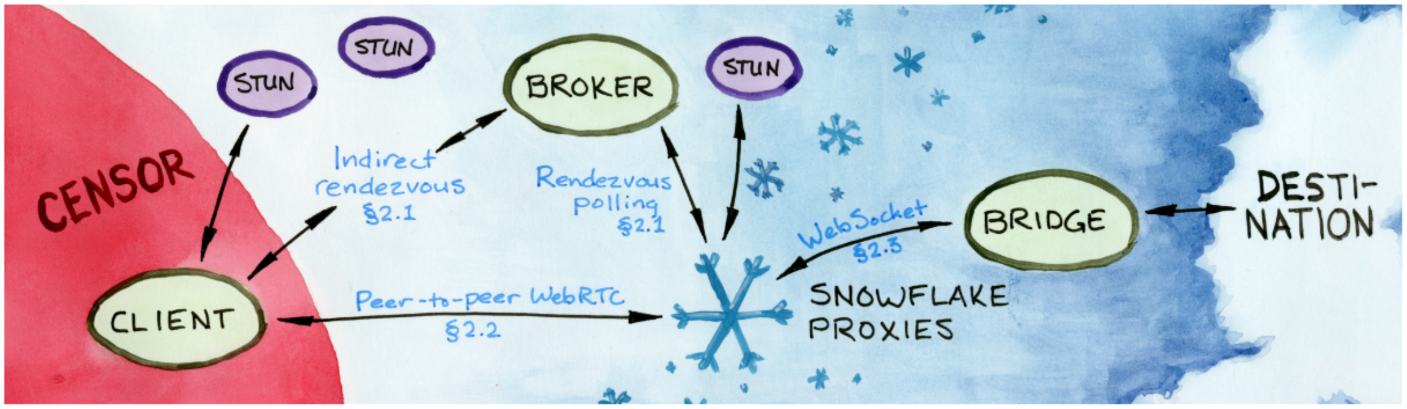

Figure 5: Architecture of Snowflake. The client contacts the broker through an indirect rendezvous channel with high blocking resistance. The broker matches the client with one of the proxies that are currently polling. The client and proxy connect to one another using WebRTC. The proxy connects to the bridge, then begins copying traffic in both directions. If the proxy disappears, the client does another rendezvous and resumes its session with a new proxy.[6]

the specific vulnerabilities it introduces, specially fingerprint patterns, highlight areas requiring ongoing attention. Addressing these vulnerabilities with enhanced obfuscation techniques and dynamic fingerprint management is essential to maintaining Snowflake's efficacy in circumventing censorship.

4 Answers to Key Questions

1. **What factors should end users consider when selecting a censorship-resistant system, given their technical proficiency and available resources?**

   When selecting a censorship-resistant system, users should consider their technical skills, the capabilities of the censor, and their specific needs. The system should be easy to use, resistant to blocking, and meet performance requirements. The following factors are crucial:

   - **Ease of use**: For non-technical users, simplicity is key. Systems that require minimal setup and function like common web applications are ideal. **`Snowflake` (also maybe Stegozoa) is the only modern system that offers true accessibility for end users**.
   - **Resilience to blocking**: The system should resist common censorship techniques such as IP blocking, DNS tampering, and deep packet inspection. Techniques like traffic obfuscation and dynamic proxy rotation enhance resilience. **`Snowflake` uses temporary, ever-changing proxies, making it difficult for censors to block.**
   - **Performance**: Users should consider whether the system meets their speed and latency needs. Systems that use steganography or tunneling may have slower performance. For basic web browsing, `Stegozoa` and `Snowflake` offer reliable performance for low to moderate bandwidth tasks.
   - **Availability**: A reliable system should offer redundancy and adapt to evolving censorship tactics. Dynamic networks and backup options improve availability.
   - **Security**: The system must protect user privacy and resist interception. Encryption and protection against man-in-the-middle attacks are essential. Systems like `Stegozoa`, `Snowflake`, and `TorKameleon` prioritize strong security features.

   While `Snowflake` is currently the most accessible option for end users, it would be valuable to develop and deploy an accessible implementation of `TorKameleon` to provide more robust alternatives in the future.

2. **How does the usability of different censorship circumvention tools impact their adoption by non-technical users?**

   The usability of censorship circumvention tools plays a crucial role in their adoption by non-technical users. Snowflake is highly accessible, requiring no configuration and operating seamlessly through browser extensions. Stegozoa simplifies adoption by integrating as a library within WebRTC applications, making it intuitive for those familiar with basic app installations. TorKameleon, while promising, needs a more user-friendly implementation to lower barriers to entry, highlighting the importance of simplicity and accessibility in driving widespread adoption.

3. **Under what circumstances might decentralized architectures provide better censorship resistance compared to more static solutions?**

   Decentralized architectures and adaptable systems like TorKameleon or Snowflake offer stronger censorship resistance in dynamic environments. These systems excel in scenarios such as when censors target known nodes—Snowflake's temporary, frequently changing proxies make blocking difficult; when facing sophisticated traffic analysis—TorKameleon uses adaptive traffic patterns and covert channels to evade detection; when censorship is intermittent or localized—decentralized systems reroute traffic seamlessly through alternative nodes; and when user resources are limited—Snowflake's lightweight proxies eliminate the need for complex configurations. These features make them more resilient compared to static solutions, which are easier to detect and block.

4. **What challenges arise in balancing security and performance when implementing techniques like protocol obfuscation and steganography in real-world applications?**

   - **Security** often demands complex obfuscation, which can reduce **performance** (e.g., `Stegozoa`'s low throughput).
   - High-performance systems like `Protozoa` may sacrifice security by being vulnerable to traffic analysis or correlation attacks.
   - Striking a balance requires adaptive techniques that optimize both aspects based on user needs and network conditions.

5. **How can developers of censorship circumvention tools ensure adaptability to evolving censorship techniques and network conditions?**

   Developers can ensure adaptability by adopting a multi-faceted approach that emphasizes flexibility, resilience, and continuous learning. Key strategies include:

   - **Modular Design**: Implement modular architectures for easy updates and integration of new techniques.
   - **Decentralization**: Use decentralized networks, to distribute communication and avoid single points of failure.
   - **Obfuscation Techniques**: Employ multiple methods of traffic obfuscation and dynamically switch between them based on network conditions.
   - **Continuous Monitoring and Feedback**: Regularly monitor censorship tactics and gather user feedback to identify new challenges and refine tools accordingly.
   - **Collaboration**: Foster collaboration between developers, researchers, and users to share knowledge, identify vulnerabilities, and develop effective circumvention strategies.
   - **Ease of Use**: Prioritize user-friendly interfaces and minimal configuration requirements to ensure broad adoption, even by non-technical users.

By incorporating these principles, developers can create adaptable and resilient tools that respond effectively to evolving censorship techniques and network conditions.

5 Future Trends and Challenges

The future of censorship circumvention tools will be shaped by the dynamic interplay of advancing technology, evolving censorship techniques, and the shifting needs of users. Below are key trends and challenges anticipated in this domain:

Advancing Censorship Techniques

Censors are likely to adopt more sophisticated technologies, including machine learning (as expòsed on Barrada's work [3])and big data analytics, to detect and block circumvention efforts. Techniques such as deep packet inspection (DPI) and advanced traffic analysis will present significant hurdles. Developers must innovate to counteract these measures, employing obfuscation, mimicry, and other advanced techniques to evade detection.

Mobile-Centric Solutions

The proliferation of mobile devices as primary internet access points, with 85% of internet users accessing the web via mobile devices[8], particularly in heavily censored regions, necessitates the development of lightweight, resource-efficient tools optimized for mobile platforms. These tools must operate effectively on networks with low bandwidth and high latency.

Improving Usability

To achieve widespread adoption, tools must be accessible to non-technical users. This involves simplifying configuration, offering intuitive interfaces, and minimizing the need for user intervention. Usability-focused design will enhance the impact of these tools, particularly in regions with low technological literacy.

Policy and Advocacy Efforts

In addition to technical innovation, advocating for policies that support a free and open internet is crucial. Efforts should focus on promoting digital rights, opposing restrictive legislation, and raising awareness of censorship issues globally.

6 Conclusion

This comprehensive survey has examined the evolution of video call-based censorship circumvention methods, from early protocol mimicry attempts like SkypeMorph to modern steganographic approaches like Stegozoa and hybrid systems like Snowflake. This analysis reveals several key findings about the state and trajectory of this field.

The progression in circumvention techniques shows a clear trend toward more sophisticated approaches, moving from simple protocol mimicry to advanced steganographic methods and hybrid architectures. However, this analysis consistently reveals an inherent trade-off between security and performance across these systems. High-throughput solutions like Protozoa (1.4 Mbps)[4] prove more vulnerable to detection, while more secure systems like Stegozoa (11.4 Kbps)[7] operate at significantly lower speeds.

The examination of system vulnerabilities demonstrates that modern circumvention tools must address multiple threat vectors simultaneously, from traffic analysis to correlation attacks. While recent systems like Snowflake and TorKameleon show more comprehensive approaches to security, their reliance on centralized components introduces potential bottlenecks and points of failure[6, 14].

Perhaps most significantly, this survey highlights a persistent gap between theoretical capabilities and practical usability. Despite the technical sophistication of systems like TorKameleon, only Snowflake achieves sufficient accessibility for non-technical users, though this comes with its own security trade-offs. This disparity between security and usability remains a fundamental challenge that future developments in the field must address.

References

[1] Afonso Vilalonga . Torkameleon. `https://github.com/AfonsoVilalonga/TorKameleon`. Accessed: 2024-12-09.

[2] Giuseppe Aceto and Antonio Pescapé. Internet censorship detection: A survey. *Computer Networks*, 83:381–421, 2015. ID: 271990.

[3] Diogo Barradas, Nuno Santos, and Luís Rodrigues. Effective detection of multimedia protocol tunneling using machine learning. In *27th USENIX Security Symposium (USENIX Security 18)*, page 169–185, 2018.

[4] Diogo Barradas, Nuno Santos, Luís Rodrigues, and Vítor Nunes. 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*, page 35–48, 2020.

[5] BlizzardPlus. Code talker tunnel. `https://github.com/blizzardplus/Code-Talker-Tunnel/tree/master/source`. Accessed: 2024-12-08.

[6] John Fifield. Analyzing snowflake protocols. In *Proceedings of the 33rd USENIX Security Symposium*, Philadelphia, PA, USA, August 14–16 2024. USENIX Association. Accessed: 2024-12-08.

[7] Gabriel Figueira, Diogo Barradas, and Nuno Santos. 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*, page 1154–1167, 2022.

[8] thinkwithgoogle.com. `https://www.thinkwithgoogle.com/intl/es-es/insights/tendencias-de-consumo/el-m%C3%B3vil-l%C3%ADder-en-el-consumo-de-internet/`, 2017. [Accessed 13-01-2025].

[9] Bridger Hahn, Rishab Nithyanand, Phillipa Gill, and Rob Johnson. Games without frontiers: Investigating video games as a covert channel. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, page 63–77. IEEE, 2016.

[10] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. The parrot is dead: Observing unobservable network communications. In *2013 IEEE Symposium on Security and Privacy*, page 65–79. IEEE, 2013.

[11] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. Skypemorph: Protocol obfuscation for tor bridges. In *Proceedings of the 2012 ACM conference on Computer and communications security*, page 97–108, 2012.

[12] Adrian Cruzat La Rosa. No title. *Re-encoding Resistance: Towards Robust Covert Channels over WebRTC Video Streaming*, 2024.

[13] the tor project. Snowflake repositorie. `https://gitlab.torproject.org/tpo/anti-censorship/pluggable-transports/snowflake`. Accessed: 2025-01-10.

[14] Afonso Vilalonga, João S. Resende, and Henrique Domingos. 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)*, page 1490–1495. IEEE, 2023.

[15] Qiyan Wang, Xun Gong, Giang TK Nguyen, Amir Houmansadr, and Nikita Borisov. Censorspoofer: asymmetric communication using ip spoofing for censorship-resistant web browsing. In *Proceedings of the 2012 ACM conference on Computer and communications security*, page 121–132, 2012.

[16] Barney Warf. Geographies of global internet censorship. *GeoJournal*, 76:1–23, 2011.