OUStralopithecus: Overt User Simulation for Censorship Circumvention

Anna Harbluk Lorimer
University of Chicago
Chicago, IL, USA
annalorimer@uchicago.edu

Lindsey Tulloch
University of Waterloo
Waterloo, ON, Canada
ltulloch@uwaterloo.ca

Cecylia Bocovich
The Tor Project
Waterloo, ON, Canada
cecylia@torproject.org

Ian Goldberg
University of Waterloo
Waterloo, ON, Canada
iang@uwaterloo.ca

ABSTRACT

In many parts of the world, censors are continuously increasing their capacity to fingerprint, identify, and block censorship resistance tools to maintain control over what can and can not be accessed over the Internet. In response, traffic replacement, which involves co-opting a steady stream of uncensored *overt* traffic to serve as a perfect cover for censored *covert* content, has been developed in an effort to provide undetectable access to the open Internet for those in censored regions. Despite the promise of this technique, creating a suitable stream of uncensored overt traffic that is high throughput, fingerprint and identification resistant, and does not overburden the user to generate, is an underexplored area that is critical to traffic replacement's success.

To address this, we propose OUStralopithecus (OUStral for short), a web-based Overt User Simulator (OUS) that browses the web as a human would in order to avoid being detected by a censor. We implement OUStral as a Python library that can be added to an existing traffic-replacement system. To evaluate OUStral we connect it to an existing traffic replacement system, Slitheen, that replaces media data such as images. Additionally, we implement WebM video replacement for Slitheen to demonstrate the high throughput that OUStral is able to provide. We show that OUStral evades being detected as a bot by state-of-the-art bot detection software while providing a high-throughput overt data channel for covert data replacement.

CCS CONCEPTS

• **Social and professional topics → Technology and censorship**; • **Security and privacy → Privacy-preserving protocols**.

KEYWORDS

Obfuscation; censorship resistance; inter-connection shaping

ACM Reference Format:
Anna Harbluk Lorimer, Lindsey Tulloch, Cecylia Bocovich, and Ian Goldberg. 2021. OUStralopithecus: Overt User Simulation for Censorship Circumvention. In *Proceedings of the 20th Workshop on Privacy in the Electronic Society (WPES '21), November 15, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3463676.3485604

1 INTRODUCTION

There has been an increased effort by censors to quash dissent [56]. While tools to circumvent censorship do exist, they are not always effective or easily deployable, and in some cases, due to these shortcomings, the use of anti-censorship and privacy-preserving tools has resulted in severe punishment for users [12, 17].

Censors have responded to advances in censorship-circumvention techniques with their own increasingly powerful methods to control Internet access, such as DNS manipulation [4, 48, 51] and IP address filtering [2, 24]. Prior studies examining censors' behaviours have found such practices as systematically probing [23] and performing reconnaissance [27] to block censorship circumvention proxies, and using deep packet inspection to distinguish censorship circumvention traffic [63]. Iterating on techniques that disguise both user traffic itself and the endpoints used for censorship circumvention is directly motivated by such findings.

Tunneling-based censorship-circumvention systems are a promising way forward in the cat-and-mouse game between censors and censorship resistors. Tunnelling-based censorship circumvention tools [5, 10, 26, 40] leverage popular deployed protocols to tunnel censored *covert* traffic to the client within an uncensored *overt* protocol, making the technique resistant to adversaries monitoring lower-level traffic. However these systems still struggle to perfectly conceal the patterns of the underlying, covert traffic [31] and have likewise struggled to generate high-throughput overt traffic that is replaceable yet undetectable by a censor. Traffic replacement [7, 9, 10, 50] is a newer technique that *replaces* traffic generated through actual use of the overt protocol in a byte-for-byte manner. This creates a high-bandwidth channel that maintains the shape of streaming or video conferencing sessions.

In order for traffic replacement to work, there needs to be a method of generating replaceable, overt traffic in which to tunnel the covert traffic. Some existing techniques to generate replaceable overt content use live or pre-recorded audio and video calls [5, 7] from users at each endpoint. This technique puts the onus on the users to generate overt content, which is not an ideal user experience. Other systems use simple programmable scripts with headless browsers to automate browsing websites for replaceable content [8, 10, 50]. However, these implementations have not been evaluated for their ability to evade detection, meaning that it is unknown if a censor could simply detect and then block them to stop the flow of replaceable overt content.

We present OUStralopithecus (OUStral for short),[1] a new technique to generate replacable overt content that uses an Overt User Simulator (OUS), a term borrowed from the traffic-replacement system Slitheen [10], to automate the process of overt content generation while providing resistance to detection by a censor. OUStral is suited to browser-based tunnelling protocols, and provides a high-throughput covert channel. OUStral adopts a technique from the world of HCI known as an *impersonator bot* [67], which aims to browse the Internet exactly as a real human user would in order to evade bot-detection software. Since recent work in bot detection has shown that distinguishing bot-generated traffic from human-generated traffic is still a difficult task [32, 41], OUStral wields impersonator bots' ability to evade bot detectors as a defence against censors. Though it is unclear how closely a censor's behaviour would mirror that of a bot detector, being indistinguishable from a common human user is desirable for several reasons. Being deemed "human-like" increases the ease of browsing the Internet and decreases the probability that the OUS' traffic would be flagged as anomalous bot traffic. By using impersonator-bot-style browsing, OUStral both evades detection by a censor and generates sufficient replaceable overt traffic to make the user experience of accessing covert data with a traffic replacing censorship systems much better than in prior work. We implement OUStral as a Python library that can be imported into existing projects. We then connect OUStral to Slitheen, and tunnel covert traffic over WebM video to demonstrate the effectiveness of our design.

In this paper, we expand on previous work in the censorship circumvention space by designing techniques for web-browsing-based overt user simulation that make accesses to decoy sites look like real user behaviour. In addition to providing a novel approach for overt content generation, we also demonstrate that traffic tunnelling using video protocols provides high bandwidth and low overhead while not varying from the overt site's true traffic pattern. Our contributions can be summarized as follows:

- We design **OUStral, a web-based Overt User Simulator** to generate replaceable human-like traffic based on previous results from studying real user behaviour and evading bot detection.
- We **evaluate the performance of OUStral** by connecting it to Slitheen, a traffic-replacement-based censorship-circumvention system that is capable of replacing images. Additionally, we implement WebM video replacement in Slitheen to further demonstrate the throughput and low overhead of OUStral.
- We **evaluate OUStral's ability to evade detection** by running it against state-of-the-art bot detection software.

[1]OUStralopithicus is a pun of Australopithicus, a close ancestral relative of humans, harkening to the *human-like* behaviour of our system.

2 BACKGROUND

Though Internet censorship takes many forms, this work addresses a subset of censorship known as *Internet filtering*, in which access to content is selectively controlled by an adversary capable of monitoring and blocking traffic that passes through their area of influence. Through a combination of measurement studies, user reports, and metrics of the usage of censorship circumvention systems, we know that censors not only target websites and services with content deemed objectionable, but also censorship circumvention tools themselves [27, 59]. This happens through domain name or IP address blocking [2, 4, 24, 48, 51], but also through the use of more advanced deep-packet inspection techniques by looking for and blocking traffic with identifiable features of circumvention tools (i.e., fingerprints) [18, 25, 30, 59, 63].

2.1 Media-based Covert Channels

To reduce the threat of fingerprinting and blocking via traffic analysis, many censorship circumvention tools modify traffic patterns through a layer of traffic obfuscation. Different techniques include making traffic completely unidentifiable [3, 19, 64], mimicking allowed protocols [21, 47], and tunnelling traffic through allowed systems and protocols [5, 10, 26, 40]. While each technique has its own weaknesses and proposed attacks [31, 36, 62], we focus on methods that tunnel censorship resistance traffic through existing, unblocked protocols. Protocols that transport censorship resistance traffic are often referred to as **covert channels**, with the protocol itself termed **overt traffic** and the tunnelled data **covert traffic**.

Since many Internet protocols are too essential for a censor to block, they make ideal candidates for tunnelling censorship circumvention traffic. DNS, TLS, HTTP, and Voice-over-IP all lend themselves well to tunnelling because they are commonly used or can transmit a large amount of covert content. However, the key to success lies in producing tunnelled traffic that a censor can not detect, whether through identifying characteristics in packet size, traffic shape, and so on. Geddes et al. [31] found that mismatches between the traffic characteristics of the visible tunnel protocol and the censorship circumvention traffic can allow a censor to reliably detect the deviation from the intended use of the covert protocol.

Although tunnelling over DNS provides low throughput, its ubiquity, frequent usage, and structure make it difficult to filter as long as the tunnelled data is properly encoded as realistic responses to DNS requests. Akbar et al. [1] propose DNS-sly, which is capable of disguising a small amount of traffic in the responses of a cooperating DNS server without the use of an encrypted tunnel and is suitable for circumvention systems that require low-bandwidth registration channels for bootstrapping.

Web-based video and streaming services such as Voice-over-IP (VoIP) and YouTube, on the other hand, offer a highly flexible, high-throughput tunnel for circumvention traffic. FreeWave [38], CovertCast [46], DeltaShaper [5], and Protozoa [7] all tunnel traffic over video and audio channels. However, suppressing the identifying patterns of the tunnelled traffic in these systems is difficult [31]. While FreeWave and CovertCast do little to hide the patterns of tunnelled traffic, DeltaShaper offers users a tunable trade-off between throughput and censorship resistance by embedding tunnelled traffic in carrier frames obtained from pre-recorded Skype calls or the webcam as the user browses [5]. In subsequent work, Barradas et al. [6] show that both DeltaShaper and CovertCast are vulnerable to identification by a machine learning capable adversary able to distinguish traffic based on packet lengths among other features. Protozoa, the most recent proposal for a multi-media covert channel, replaces collected ambient video and audio data from a client and proxy user with covert traffic to maintain the shape of normal WebRTC traffic and thwart these state-of-the-art ML attacks.

Traffic replacement, the method of replacing generated overt traffic byte-by-byte with covert traffic, is a stegonagraphic technique [43] that has roots in image steganography; for example, the least significant bits of pixels in an image are replaced with a covert message in a way that does not affect the visual look of an image, passing without detection by an unsuspecting adversary. While steganographic methods can be detected by an adversary knowledgeable about the technique, recent work in the censorship resistance space that replaces traffic in encrypted channels can resist detection even by adversaries that actively look for it. Slitheen [9–11] first proposed the byte-for-byte replacement of images and later video and audio frames of web-browsing sessions. Waterfall [50] expanded on this technique with asymmetric deployment improvements and throughput optimizations. More recently, Protozoa [7] used this technique to replace the media data of WebRTC steams [7]. Protozoa reports the highest throughput yet for a traffic replacement system and even outperforms less secure media-based covert channels.

2.2 Replacable Traffic Generation and Bot Detection

Though traffic replacement systems require a constant stream of replaceable overt traffic, the designs of these systems tend to focus exclusively on covert traffic concealment rather than the method of overt traffic generation. Ensuring that the generated overt traffic is *also* resistant to blocking by a censor is a largely unexplored and often overlooked topic. Ideally, replaceable overt content generation requires minimal involvement from the user, acting as a background process that the user need not interact with. Additionally, care must be taken to avoid implementations that may create network traffic patterns that can be flagged and blocked by a censor looking to choke the flow of replaceable overt content to a censorship-circumvention system.

Traffic replacement systems in prior work have generated overt traffic in a variety of ways. DeltaShaper [5] and Protozoa [7] rely on video calls made on popular communication platforms. This traffic provides a reasonable bandwidth but decreases usability somwhat as the process is not automated and requires a user to either save recorded video calls or use the system while engaged in a video call. A user may reasonably be wary of using real-time video while using censorship circumvention systems, despite these systems replacing generated video data with covert data.

Waterfall's solution to user-friendly overt content generation is to use PhantomJS, a scriptable headless browser, to load a website every second [50]. While this approach addresses the need for overt content generation to not involve the user, the traffic patterns generated are easily recognizable and therefore risk being blocked by a censor.

Slitheen [10] and a recent iteration Slitheen++ [8] both use an Overt User Simulator (OUS) to generate replaceable content while browsing the Internet. Though the term OUS was introduced in Slitheen, the functionality of the OUS itself was not particularly concerned with simulating a user. Instead it uses a headless browser (PhantomJS) to make multiple simultaneous overt requests that can be tagged to indicate requests for covert connections and traffic replacement. Identifying that Slitheen's OUS may be recognizable by censors, Birtel and Rossow [8] aimed to make Slitheen's OUS more realistic as one of Slitheen++'s improvements on the original Slitheen design. To do this, they introduce a webcrawler into the OUS that browses within a domain and adds thinking time between requests, which they note negatively impacts the throughput. Both the original Slitheen OUS and the Slitheen++ OUS meet the requirements of minimal user involvement and high bandwith. Though Slitheen++ does indeed improve upon Slitheen in terms of recognizability, neither system is capable of avoiding being detected by state-of-the-art bot detection software as we show in Section 5.2.1.

As Houmansadr et al. [36] point out, minor inconsistencies in imitations of common protocols are observable by censors, making imitation a fundamentally flawed approach to unobservability. When designing an overt traffic generating tool, it is important to consider several details along the network stack, as well as typical human browsing behaviour, to prevent being distinguishable from other traffic. As an example, PhantomJS, which was originally developed for automating web page interaction [35], was used to drive overt traffic generation for several prior censorship circumvention systems [8, 10, 50]. However, the use of PhantomJS can be easily distinguished from browsers like Firefox or Chrome that a typical human users might use; moreover, active development on the browser has been suspended as of 2018 and users have moved away from it entirely.

Guo et al. [32] make some early observations on bot detection at lower network layers (network and transport) that replicate the capabilities of a censor in our scheme; that is, the application data is hidden by encryption and only lower level traffic is accessible for analysis. Guo et al. find that the most effective features for identifying bots involve differentiating Linux users from other OS users along with packet features (sizes and timings). Therefore, an implementation of an OUS that crawls the Internet to generate replaceable overt content should be flexible enough to run on different operating systems.

Recent work on web bot detection at the application layer utilizes machine learning techniques to distinguish bots from regular users based on significant features like click behaviour, time on pages, and the depth of a requested page [33, 41, 53, 54, 58]. However, using impersonator bots, introduced by Yang et al. [67], that leverage human web-browsing characteristics such as dwell time and navigation within a site to impersonate human behaviour, make distinguishing such bots from human users difficult at the application layer—Iliou et al. [41] find that the more web traffic looks like a normal user, the more difficult it is to detect.

This demonstrates the importance of considering human-like behaviour all along the network stack when developing a tool to generate human-like traffic and avoid detection. Both the typical usage and current state of tools must be taken into account and commonly used versions of tools (as opposed to close approximations or imitations) should be used wherever possible.

2.3 Deployment architectures

An overt-content generation scheme needs to be supported by a system that is capable of replacing the traffic that is generated. As we describe in Section 3, OUStral is decoupled from whatever traffic replacement system it is connected to; however, deployment strategies for both OUStral (or OUStral-like systems) and the traffic generation system must be considered.

Perhaps the most important consideration when deploying an overt content-generation system is that it must be deployed in such a way that it can support a variety of protocols that can be used to tunnel traffic. If the overt content generation system is deployed too far down in the stack, it risks excluding perfectly viable protocols and hindering itself in the event that a censor decides to block a certain protocol. Another aspect to consider is that it should be decoupled from the traffic-replacement system for ease of maintenance and so that it can be easily run without user interference. With these considerations in mind, generating overt content in a web browser addresses both the need to support many protocols while also limiting user interaction since browsers can be easily automated. The ease of connecting a traffic-replacement system to a web-browser based overt content generator depends on the architecture of the traffic-replacement system. However, as we show in Section 3, it can be as simple as a one-line change to connect an overt content generator to an existing traffic-replacement system.

In addition to the inclusion of an OUS, the traffic replacement system itself must itself be deployed. Traffic replacement requires two vantage points at the covert channel, to insert and extract tunnelled data in each direction. There are two main architectures seen in existing work: an end-to-end deployment architecture that uses hooks in the overt protocol software to replace generated overt data before it is encrypted on each end, and an end-to-middle deployment that intercepts the encrypted covert channel en route between the client and an unsuspecting third party.

Protozoa [7] uses an end-to-end deployment architecture with hooks in the WebRTC software implementation that replace video and audio data after they are packetized into frames, but before they are encrypted and sent on the wire. As a result, the traffic maintains the shape of a regular video call but contains covert traffic in the place of captured video and audio frames. An advantage of this deployment model is the simplicity granted by control over the ecosystem. Both endpoints run the same software with the same hooks, eliminating the need for state machines. End-to-end censorship resistance systems are also quite easy to deploy. In the case of Protozoa, WebRTC makes deployment even easier by allowing proxies to be run on home networks behind NATs and firewalls. The deployment requirements are very similar to Snowflake [34], a WebRTC-based censorship resistance system that has grown to over 9000 participant proxies in 2021 run by volunteers through a WebExtension in their browser.[2] The downside of an end-to-end deployment is that users need to be told which endpoints to connect to, and any information that a regular user of a censorship resistance system can find out can also be discovered by a censor. Censors can then block these endpoints and prevent or significantly degrade the use of the system. However, with the potential to support millions of proxies, the limits of enumeration have yet to be tested.

[2]https://metrics.torproject.org/collector.html#snowflake-stats

Slitheen and Waterfall are both decoy routing (also known as refraction networking) systems [8, 10, 22, 37, 44, 50, 57, 65, 66] that envision an end-to-middle deployment where clients make steganographically tagged connections to non-participant websites that are intercepted by routers in the middle of the network. These decoy routing "relay stations" perform an intentional man-in-the-middle and process traffic packet-by-packet, decrypting TLS records and replacing their contents with covert traffic. The decoy routing model has a theoretical advantage to censorship resistance over end-to-end systems by forcing censors to block large swaths of the Internet, as they have to block all websites that *route through* the deployed relay station. Work on optimizing the placement of decoy routers aims to increase this collateral damage to a high enough percentage of the Internet to make the blockage of decoy routing infeasible, while also minimizing the required number of deployments [13, 39, 49].

A known disadvantage of decoy routing systems is that they are difficult to deploy. They require deployment at ISP-owned routers that exist between censored users and popular third-party websites. The only two currently deployed decoy routing systems were designed specifically for maximum deployability, and have features that preclude the traffic replacement techniques used by Waterfall and Slitheen. TapDance [65] was the first widely deployed decoy routing system [28, 60], and has supported up to 559,000 users as a transport for the popular VPN service Psiphon [52]. TapDance does not perform in-line blocking to avoid violating the Terms of Service of most ISPs; that is, traffic that passes from a client to a third-party site through a TapDance relay station is not diverted, dropped, or held for processing but rather allowed to pass unmodified to its destination. Covert traffic is *copied* to the covert destination and tricks are used to delay the third party site from terminating the connection. Conjure [29] further improves the deployability of decoy routing by directing clients to unused IP address space as their overt destination rather than making connections to unsuspecting third parties.

3 OUSTRAL

Censorship circumvention systems that rely on embedding, tunnelling, or traffic replacement face the problem of generating realistic overt traffic that, from a censor's perspective, looks believably like a real user. The generation method must be undetectable by an adversary who has access to network-level monitoring capabilities; otherwise it risks being blocked or used as evidence to persecute users. To address this problem we created OUStral, an automated overt user simulator and overt content generator. OUStral is designed to be part of the client-side software in a censorship circumvention system and is available as an open-source library that can be incorporated into existing censorship circumvention systems.[3] OUStral is implemented as a bot that instructs a browser to create many connections using the tunnel protocol (in this case HTTPS).

[3]Oustral is available at https://gitlab.com/oustral/oustral.

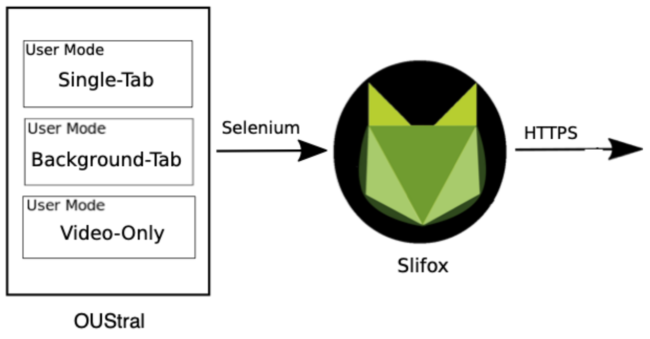

Figure 1: When OUStral is run, a user mode is selected via the command line. OUStral uses Selenium to connect to Slifox, a modified version of Firefox. OUStral can then use Slifox to browse the Internet in human-like manner to generate replaceable overt content.

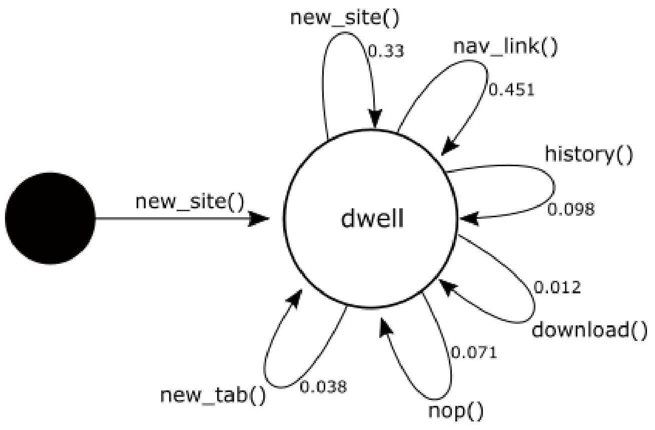

Figure 2: The OUStral state machine.

It navigates the web by mimicking human browsing behaviour, therefore evading detection by a censor looking for identifying traffic patterns. While previous work [10, 11, 50] showed how to perform *per-connection shaping* of traffic flows corresponding to individual HTTPS connections, we additionally address the question of *inter-connection shaping*: producing realistic *patterns* of HTTPS connections.

We design OUStral to simulate human-like web browsing while bearing in mind that 1) its behaviour should be sufficiently human-like so as not to be flagged by bot-detection software, and 2) it must generate enough replaceable overt content for the user to be able to read covert content. To inform the design and implementation of the "human-ness" of OUStral, we look to studies of how users browse the Internet and *impersonator bots* [67] that imitate human browsing behaviour.

Human browsing behaviours have been examined in a number of studies. Von der Weth and Hauswirth [61] find that human users perform a variety of different actions (*browsing actions*) while browsing the Internet. They measured the frequencies of each of these *browsing actions* and found that 45.1% of actions were clicks on links and 33% were navigations to new websites. Liu et al. [45] found that user dwell time on sites (i.e., a user reading a web page or watching a video) followed a Weibull distribution. A study of Firefox users by Dubroy and Balakrishnan [20] found that per 100 actions, users had on average about four tabs open, that 45% of tabs were only selected once, and 77.7% of switches away from the current tab were to tabs that had already been viewed. Impersonator bots [67], developed by Yang et al., leverage the two main characteristics of web browsing: dwell time (the time a user spends looking at a page without clicking on a link), and navigation between and within sites. Since we want OUStral to evade detection by appearing to be human, we make use of these previous works indicating that human-generated web traffic tends to follow predictable distributions [45, 67].

We implemented OUStral using Selenium [55], a browser automation tool that has all the necessary capabilities to mimic how a human would use a web browser. We used the Python version of Selenium 4.3 and configured Selenium to have a common user-agent (the most recent version of Firefox at the time of the experiments) so that a censor would not immediately know it was a bot. OUStral connects to a censorship circumvention system via Selenium's WebDriver module that allows for the specification of which browser binary to use (this setup is a one-line change) while browsing the web. OUStral navigates between a list of overt websites (such as news and social sites) that are not filtered by the censor, and alternates between making a browsing action (opening a tab, clicking on a link, navigating to a new site, etc.) and dwelling on a page to simulate a user reading the page. In Figure 1 we show the interaction between OUStral and Slifox, a modified version of Firefox (see Section 4.2) that we used for our implementation and testing, though Selenium can be easily configured to use a wide variety of browsers. Were OUStral to be widely adopted and used in real deployments, it should of course allow overt sites to opt out of being navigated in the manner of TapDance [65]. Sites opting out would add an entry to their robots.txt file, which would be periodically checked by the censorship resistance system software maintainers (but not by OUStral itself, lest that check reveal it is itself an OUS rather than a human user).

Since OUStral is implemented using Selenium and a browser, it does not limit the protocols that can be used to tunnel covert traffic, meaning that it is compatible with any browser-based censorship circumvention system that uses traffic replacement.

3.1 Behaviour

OUStral's behaviour has been carefully designed to implement findings from prior work on human browsing behaviour. The first action OUStral takes is to randomly select an overt site and navigate to the home page of that site as a starting point, as shown in Figure 2. Once the first request is completed, OUStral alternates between dwelling and one of many *browsing actions* based on those studied by Von der Weth and Hauswirth [61]. The possible browsing actions OUStral can select are: clicking on an internal link, navigating to a new site via the URL bar, returning to a previously visited page, waiting for a download to complete, and not taking an action. We also added the ability to open a new tab per a user study by Dubroy

Table 1: A sample from the OUStral logs showing the browsing actions made by OUStral during a browsing session using the single-tab user mode.

```
Next action: new_addr
Navigating to https://...
Dwelling for 7.077051533814314
Navigating within site ...
Dwelling for 3.6133869493285653
```

and Balakrishnan [20] that investigated tab usage patterns among Mozilla Firefox users and found that tab switching was among the most frequent navigation mechanisms. OUStral determines which *browsing action* to take probabilistically using the random.choice method in Python's numpy library; this method has an optional parameter that sets the relative probabilities that each item in the list is chosen. The probability of each action is based on the work of Von der Weth and Hauswirth as discussed above. The probability that a new tab is opened was implemented in the same way, using the findings in Dubroy and Balakrishnan's [20] study of Mozilla Firefox users. The probabilities of each *browsing action* are displayed in Figure 2.

When OUStral's next selected action is to click on an internal link, OUStral compiles a list of all the links on the site with the same domain as the current site, then randomly selects a link in that list to click on. Sample logs from a browsing session are shown in Table 1.

Prior work by Liu et al. [45] found that dwell times in browsing behaviour followed a Weibull distribution. Using these results, we implemented our dwell time action to follow a Weibull distribution by choosing a scale (λ) of 30 seconds and a shape (k) of 0.75.

OUStral does not simulate the *duration* of a browsing session. This is because OUStral is run simultaneously as a user browses covert content in another browser, meaning that the overt content browsing session length is the same as the covert content browsing session length. Since the covert content browsing session is already in the control of a human user, the overt browsing session length already meets the requirement of being human-like.

3.2 User Modes

Evidently, how people browse the Internet varies widely from person to person and we do not claim that our implementation of an OUS can perfectly mimic all the different ways people browse. Therefore, we implemented OUStral to use user modes that are easily tunable and customizable to variations in browsing behaviour. We implemented three user modes (shown in Figure 1): single-tab, background-tab, and video-only.

Both the single-tab and the background-tab modes use the same base behaviour implementation, that is, dwelling and performing one of several *browing actions* as described in the previous section. The difference between these two modes is that the background-tab mode also opens an additional tab with a YouTube video running in the background at startup. That video continues to play while, in another tab, OUStral continues to browse.

The video-only mode is distinct from both the single-tab and background-tab modes as it only browses YouTube to watch videos rather than following the same dwelling and browsing actions behaviour to navigate to new sites and follow links within a domain. Instead in the video-only mode OUStral begins by navigating to the YouTube home page, then selects a video to watch. OUStral polls the page to determine when the video has finished playing by determining if the "end-mode" css class is present on the video div. When the video is finished playing, OUStral dwells, and then chooses a link to another video in the recommended bar, then clicks on that link. The video-only mode has a repeating cycle of: watching a video, dwelling, selecting a recommended video, watching the new video, etc., until the browsing session is complete.

In order to diversify and meet the needs of different censored regions, more user modes tailored to specific circumstances can easily be added.

3.3 Evading Detection

Detecting bots is of interest to many different actors on the Internet, and so many server-side machine learning and rule-based techniques have been developed for larger-scale detection of bot traffic [67]. Since bot detection is a well-established field, it is not unreasonable to assume that a censor has many different tools at their disposal to detect bot-based overt traffic generation techniques and our implementation must take that into account.

OUStral is an open-source project, meaning that it must be assumed that a censor has perfect knowledge of its behaviour. However, since OUStral's behaviour is probabilistic and is configured to use probabilities in line with a human's browsing behaviour, should a censor attempt to detect OUStral by looking for those probabilities in traffic patterns, OUStral's traffic will be lost in a sea of legitimately human traffic. We also note that a censor would have to observe OUStral's traffic for an amount of time significantly longer than a browsing session to accurately determine the probabilities at which actions are occurring.

OUStral's design is flexible, with several tunable parameters meaning that the similarities across deployments can be minimized. We also leave the design sufficiently open such that other user modes may easily be added to suit many different situations. The distribution used to determine dwell time, the probabilities of each browsing action, and the user mode used to browse, can all be tuned either in the code itself or by command-line arguments. It is easy to change which browser OUStral uses by either using the drivers that come with Selenium or to specify which binary Selenium should use, meaning that the OUS can be easily configured to use browsers such as Slifox (our modified version of Firefox).

It is important to note that there are considerations to be taken when deploying OUStral to ensure that it does not easily tip off a censor. For example, in an updated report by cybersecurity firm Imperva [42] that was referenced by Guo et al. [32], Amazon was found to be the most common bot-originating ISP in 2020, with Chrome being the most common browser used by bots. Additionally, Imperva recommended blocking traffic from easily accessible hosting and proxy services such as Host Europe GMBH and Digital Ocean and for versions of Chrome, Firefox, and Safari browsers that are three years beyond End of Life. For browser versions two years beyond End of Life, CAPTCHAs are recommended. OUStral should therefore be used with recent versions of popular browsers.

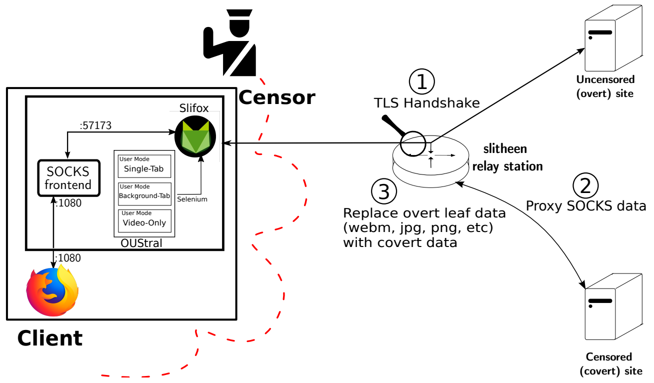

Figure 3: Our specific implementation of OUStral using Slitheen [10]. OUStral navigates to an overt site using our Slifox browser to establish a TLS session between Slifox and the overt site. The Slitheen relay monitors encrypted traffic to and from the client, looking for TLS sessions steganographically tagged to its own public key. The Slitheen relay station receives upstream proxy data from the client in an X-Slitheen header of a valid HTTP GET request to the overt site. Once the station has relayed the upstream data to the covert site, it stores the downstream responses in a queue. The received overt data from video or audio streams is then replaced with the queued data. The now concealed covert data is returned to the client and then forwarded by Slifox to the SOCKS frontend. The censor only sees the TLS handshake and encrypted traffic to and from the overt site.

OUStral also has the advantage over the bots considered in this report that it naturally runs on end-users' personal machines, and not in cloud-based hosting environments.

Finally, we note that future work on OUStral should refrain from implementing interaction with forms, logins, or credentials as this risks getting OUStral flagged for credential stuffing or other malicious activity.

Work by Guo et al. [32] focuses on distinguishing malicious CloudBots, such as those that Imperva identify as making up the bulk of malicious bots on the Internet, from human users using Machine Learning to analyze raw traffic. Lower-level traffic features that Guo et al. found to be most useful for distinction are primarily features that distinguish Linux operating systems from others such as maximum segment size, TCP window size, and time to live. OUStral is able to avoid many of the features Guo et al. [32] identify by simply using a personal machine with an operating system other than Linux.

We provide an overview of the details of our specific implementation, indicating the process by which Slitheen replaces overt content generated by OUStral, in Figure 3.

4 WEB VIDEO STREAMING AS A COVERT CHANNEL

In this section, we discuss the traffic replacement method we used for testing our video-based OUStral user modes. We adopt a decoy routing architecture for this work, but overt user simulators and video-based covert channels can also be used in end-to-end deployment scenarios, as recently demonstrated by Protozoa [7]. We detail our implementation of WebM traffic replacement for Slitheen that was necessary to turn web-based video and audio streams into the covert channel used by OUStral. Our method is based on the extension of Slitheen that applies a more generalized traffic replacement method to the protocols and container formats embedded in HTTP responses [9].

4.1 Traffic Replacement Overview

The original Slitheen [10] design uses a traffic replacement module that parses HTTP and replaces the contents of leaf resources, as determined by the `Content-Type` header of the HTTP response, with covert data. A leaf resource is any HTTP resource that can be replaced with covert data without affecting the pattern of network traffic that a browser loading the decoy site would normally produce. For example, HTML resources typically contain references to embedded images, style sheets, and scripts that the browser loads by making additional HTTP requests. If this resource is replaced with covert data, the browser would not make those additional network requests and would produce different network traffic patterns than a regular visit to the site. HTML is not a leaf resource, but resources like images can be safely replaced because they will not cause the browser parsing it to produce new network traffic.

The state machine of the replacement module in the original Slitheen was relatively simple and only parsed the HTTP and TLS data of decrypted packets. However, video and audio streams are more complex than that paper suggests. In a typical web streaming session, the client does not download the entire video in one request. Instead, the client loads and plays the first part of the video before requesting additional video data. Data that has been received by the client but not yet played is colloquially referred to as the *buffer* and buffering behaviour varies depending on the video playback engine. If the video resource were replaced in its entirety, most video playback engines would fail, preventing requests for future parts of the video. These video resources contain data that the browser needs to produce further network traffic in order to remain true to the traffic patterns of a regular video streaming session.

Video resources still contain a large amount of *leaf data*: video and audio frames that would normally just be output in the form of graphics and sound to the user, which potentially provides us with a high-bandwidth channel. Bocovich [9] describes a more general method for finding and replacing leaf data; we can treat the container formats used to transmit video and audio resources as protocols, find the parts of these protocols that can be safely replaced, and update the state machine at the relay station to parse the container format and replace only leaf data with covert traffic.

4.2 Replacing WebM Resources

Our implementation of traffic replacement for video and audio resources involves three parts:

(1) identifying the leaf data in the resource type we would like to use as a covert channel,
(2) extending the TLS and HTTP state machine to parse and identify the leaf content in these resources as they pass through the relay station, and

(3) modifying the client browser software we use to load decoy sites to extract the covert content and pass it to the user through a local SOCKS connection.

There are multiple different container formats for transmitting video and audio content. We decided to focus on WebM[4] due to its open-source specification and increased use by popular video streaming sites. The WebM container protocol is a sub-specification of Matroshka[5] in which containers are composed of various levels of sub-containers, the elements of which are defined in no particular order. Container elements begin with a header that indicates the element ID and size, followed by the element data. Some elements contain meta information about the resource such as track numbers, timestamps, and seeking pointers. The leaf data in the WebM container format are elements that contain encoded video and audio frames. These are located in what are called Simple Block elements (element ID `0xa3`), inside the main Cluster container of the WebM resource.

Figure 4 shows the state machine we used to replace leaf WebM data. Because WebM exists inside of HTTP resources which are encrypted with TLS, the final state machine we use is a composition of the machine shown and the TLS/HTTP state machine given in the original Slitheen paper [10]. Because Slitheen is restricted to only being able to parse and operate on data within a single packet boundary, we sometimes miss resources due to packets arriving out of order at the relay station. To signal which resources have been replaced, we modify the element ID to `0xef`, an ID not used by existing Matroshka container elements.

On the client side, we modify Slitheen to use a version of Firefox we call Slifox[6] as the browser it uses to load overt content. The decision to modify Firefox rather than another browser was motivated by ease of maintenance given that Firefox is an open source project. In addition to the necessary changes to NSS (Mozilla's TLS implementation) to implement decoy routing capabilities, Slifox has a modified version of WebM's processing code that extracts covert data from container elements with ID `0xef`. However, we found that more work was needed to allow the playblack engine to continue processing and requesting more resources. Popular video streaming sites (such as YouTube) often ship their own video playback engines with the video resource as JavaScript resources. The JavaScript handles some processing of the video and audio frames and the requests for more buffered data. Feeding our mangled, covert-data filled frames to the engine caused it fail and stop loading more video content. To solve this, we instead performed another replacement at the client to feed blank stub keyframes to playback engine to prevent errors. This is a technique also used by Barradas et al. [7].

The original PhantomJS OUS implemented in Slitheen was connected to Slifox to drive overt content generation. We replaced the PhantomJS OUS with OUStral by configuring OUStral to use the Slifox binary to browse the Internet. This change is an improvement because PhantomJS is no longer supported or widely used and as we show in Section 5, PhantomJS is easily detectable by bot detection software.

[4]https://www.WebMproject.org/
[5]https://www.matroska.org/technical/specs/index.html
[6]Slifox is a portmanteau of Slitheen and Firefox.

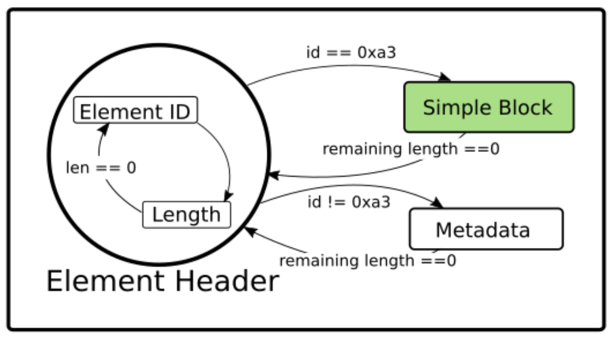

Figure 4: A state machine of the WebM protocol. The shaded green box indicates a state in which the replacement module can replace overt data with censorship resistance traffic.

Our replacement of video and audio frames in web streaming sessions allowed us to tap in to a very high-bandwidth channel that OUStral already supported given that it supports any protocol that it encounters while browsing. We provide the results of our overhead and throughput experiments in the next section.

5 EVALUATION

5.1 Experimentation platform

We conducted several experiments on OUStral to evaluate its ability to evade detection and to measure how much replaceable overt traffic it is capable of generating. We also evaluated our video and audio replacement improvements, described in Section 4, to measure the overhead and bandwidth available for censorship-resistant traffic. We ran these experiments using a test environment of two networked Docker containers built with Ubuntu 18.04 LTS images and run on a machine with 4 CPU cores and 16 GB of RAM.

Our test environment emulates a typical decoy routing deployment scenario in which client traffic passes through a decoy routing relay station deployed on a router in the middle of the network. The client container runs a SOCKS proxy connected to Slifox and OUStral and exposes port 1080 to the local machine. To send and receive covert traffic, processes on the local machine would then use 127.0.0.1:1080 as a SOCKS proxy. The client container is networked to the relay station container using veth pairs, so that all client traffic to and from the outside Internet is sent through the relay container. We illustrate our experimental setup in Figure 5.

5.2 Evaluation of OUStral

5.2.1 Detection and Evasion. There are two places that OUStral could be detected: by a censor observing traffic between the relay station and the client, and by the websites that OUStral is browsing. We wish to measure how detectable the traffic generated by OUStral is as being generated by a bot. To separate out the effects of the traffic replacement scheme attached to OUStral, for the purposes of this section, we disable the relay station container, and allow the bot detector (as would be used by the end website or a similar detector by the censor) to view the traffic directly output by the client container, as it would in a real deployment.

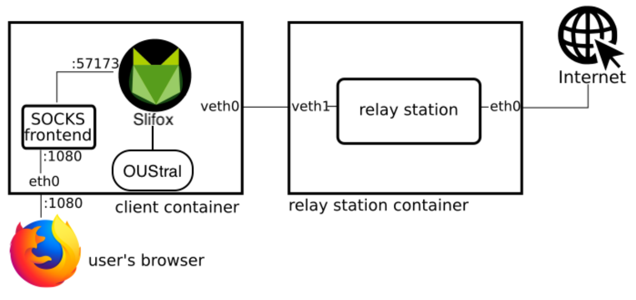

Figure 5: The experimental setup of our throughput experiments. We run the client and relay code separately in two different Docker containers. Traffic to and from the client is routed through veth0, such that is passes through the relay station on its way to and from the outside Internet. This mimics a decoy routing deployment. The user can then connect to the client software on port 1080 to tunnel censorship-resistant traffic, similar to a normal SOCKS proxy.

We compare our implementation of a human-like OUS to three existing overt content generation schemes (Slitheen, Waterfall, and Slitheen++) in addition to an actual human browsing the Internet. The original implementation of an OUS in Slitheen used a modified version of PhantomJS [35], a now discontinued scriptable headless browser, and naively reloaded the same site over and over with no crawling or human-like behaviour. Waterfall [50] uses vanilla PhantomJS as the framework for its OUS as well. The Waterfall OUS also only loaded a single site, but it waits one second between page refreshes. Slitheen++ extended the original Slitheen OUS by implementing "Think Time", which we refer to as dwell time in our implementation of OUStral.

We determine if OUStral risks being blocked (and therefore blocking the flow of replaceable overt content) by the sites that it is browsing, or by the censor. We emphasize that OUStral is not a bot that attacks the websites it browses. OUStral does not perform credential stuffing, denial of service, or other attacks that would harm the website or its operators. However, OUStral could be blocked by a website as part of the website's protection mechanisms that use machine learning to determine wether or not it is a bot. Due to the general unavailibility of data sets of human browsing behaviour, we used Cloudflare's Bot Detection Software [16] to determine if OUStral could be blocked by the sites it browses, since Cloudflare does have access to such data, upon which it can train its system. Cloudflare's Bot Detection system uses machine learning, rulesets, and heuristics to determine if traffic is human or bot generated and provides information about the "human-ness" of each request to a website to the website operator. For each request, a score from 1 to 99 is assigned to that request, 1 being certainly bot-generated and 99 being certainly human-generated [15, 16]. CloudFlare recommends blocking any traffic with a score of at most 30 [14]. We chose to use Cloudflare's machine-learning based detection over deploying our own machine-learning evaluation due to the quality of data that Cloudflare is able to access.

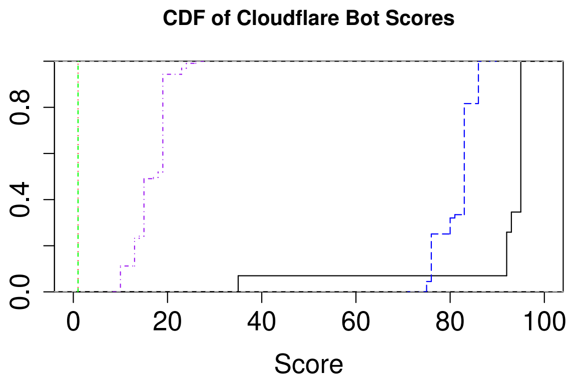

Figure 6: A CDF of the Cloudflare Bot Management scores assigned to Slitheen-generated (red, dot line) and Waterfall-generated (green, dashed line) which overlap since both had consistent bot scores of 1, Slitheen++-generated (purple, dot dashed line), human-generated (blue, long dash line), and OUStral-generated (black, solid line) requests to the test websites.

Unlike a typical adversarial censor that would only have access to the encrypted TLS traffic to analyze, Cloudflare's Bot Management system has access to the HTTP queries, responses, and headers inside the TLS connection, making it a more powerful bot detector than a censor could typically use to monitor OUStral traffic. We posit therefore that a censor attempting to detect OUStral would be *less* powerful in detecting non-human traffic than Cloudflare's Bot Management system. Since we are unable to test OUStral against a real censor, the intent of this evaluation using Cloudflare is to sanity check that OUStral cannot be detected by a commonly used tool that approximates a censor's capabilities.

We deployed Cloudflare's Bot Management system in front of our own mirrors of popular news sites and several subreddits. We listed the mirrored sites as available overt sites for OUStral to use and logged the bot scores in the request logs for each of the sites.

The CDF in Figure 6 shows the results of running our OUStral implementation, Waterfall, Slitheen, Slitheen++, and human-generated requests against Cloudflare's Bot Management system. Cloudflare rejects any connection that does not use TLS 1.3, so we also implemented a simple client-side proxy for the systems that used PhantomJS in order to force a TLS version upgrade. Without forcing the version upgrade, connections from PhantomJS were outright rejected by Cloudflare and were not assigned a score. We note that Waterfall, Slitheen, and Slitheen++ all have a much lower than "human" bot score. We expect that the PhantomJS headers are seldomly seen by Cloudflare's Bot Management system and thus requests made through PhantomJS are immediately flagged as likely bots. Importantly though, Slitheen++, which also used PhantomJS, performed far better than both Waterfall and Slitheen by using some simple implementations of human-like browsing, namely user agent randomization and a brief pause between requests; i.e., "think time".

As mentioned above, we ran OUStral unconnected to the Slitheen relay station container, using Selenium's driver for Firefox 70.0.1 (the most recent Firefox release at the time of the experiment) in the client Docker container (described in Section 5.1). We used the single-tab user mode since we did not have access to any cloned YouTube videos for the other user modes. During a multi-hour browsing session, OUStral generated 471 requests. In order to compare OUStral's scores to that of a real human, we generated 470 requests by navigating the cloned sites manually, browsing the websites as we would normally. Note that the human-generated scores were not generated as part of a large user study; therefore, they only serve as a reference point and not novel research about users' browsing behaviour.

While there are differences between the human-generated scores and the OUS-generated scores, as shown in Figure 6, all of OUStral's scores were consistently greater than the recommended cut-off point (30) [14] and are therefore far out of the range that would alert a website operator (who unlike the censor has visibility into the unencrypted HTTP session) of non-human traffic. As mentioned above, a censor would be even less likely to detect OUStral as non-human traffic to overt sites. Additionally, given that OUStral contains tunable elements such as the amount of dwell time, links clicked, user mode, and list of browsable sites, future work could look towards optimizing these scores for identification as human.

Given our results, and that Cloudflare's advanced Bot Management system with insight into the HTTPS stream is unable to flag our OUStral-generated traffic as generated by a bot, we argue that a censor, with a more limited view of the stream would have even more difficulty detecting our OUStral-generated traffic as being unnatural.

5.2.2 Throughput. To measure the overall throughput of censorship-resistant traffic generated by OUStral, we instrumented our implementation of Slitheen to report the total bytes of received, replaceable data for each of OUStral's user modes. We calculate the average throughput of our system in each mode to be the total number of replaceable bytes received divided by the total time that bytes were recorded for each run at the Slitheen relay.

First we used Slitheen's relay station instrumentation to measure the throughput of OUStral using the single-tab user mode. The results are shown in Figure 7. We ran ten 10-minute browsing sessions and had OUStral crawl the same cloned sites used in Section 5.2.1. Though these sites could contain WebM videos as well as other image data (JPG, PNG, etc.), we purposely avoided WebM streams for these measurements to distinguish throughput from the video-only mode. We found that OUStral's dwell times caused burstiness and that long dwell times had a significant effect on the amount of overt data OUStral was capable of generating. The mean throughput for OUStral in single-tab mode was 117.1 kbps with a maximum reported throughput of 164.1 kbps and a mininmum of 64.6 kbps.

We then took ten measurements for a duration of ten minutes each of OUStral streaming videos using its video only user mode. The video only user mode navigates to YouTube's home page, randomly selects a video to stream and then plays the video until the end before navigating to the next video. We provide a step function that show the total bytes received over time for all ten runs in Figure 8. The mean throughput for OUStral in video-only user mode was 581.7 kbps with a maximum reported throughput of 2023.3 kbps (2.0 Mbps) and a mininmum of 78.2 kbps. There is an extremely large amount of variance in our bandwidth measurements. This is unsurprising given that Figure 8 shows the highly variable amount of wait time while the video playback engine of Slifox waits to buffer more content.

There is a significant amount of variation between different overt video streams. We note that video streams with more frequent requests for more video data, and therefore shorter buffer times, are better for censorship-resistant traffic that requires multiple network round trips between the user and the covert site.

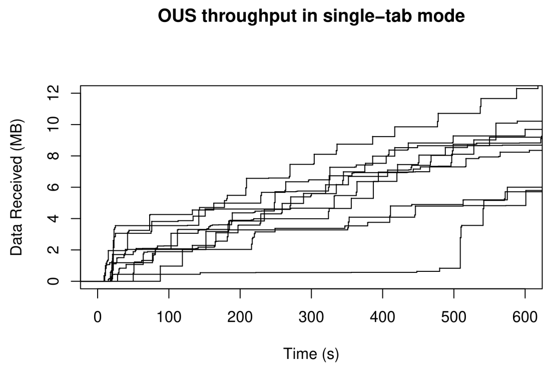

Figure 7: A step function of the throughput received by replacing leaf content while running the OUS in single-tab mode.

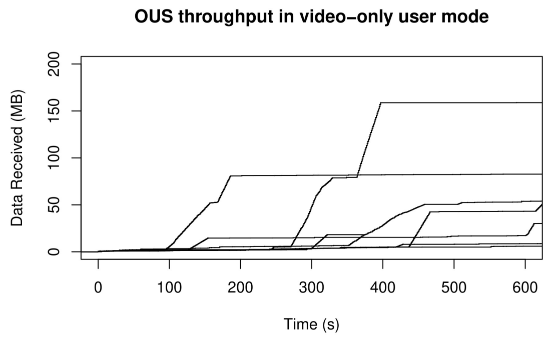

Figure 8: A step function of the overt channel throughput achieved using OUStral's video only user mode. This plot shows the replaceable bytes received (in MB) over time for ten runs of OUStral using the video only user mode. For each run, the video only mode used the Slifox browser to navigate to YouTube. It randomly selected a video from the YouTube homepage and navigated to a different random video once the video had finished playing. Each run initiated with a different video and followed a unique browsing path.

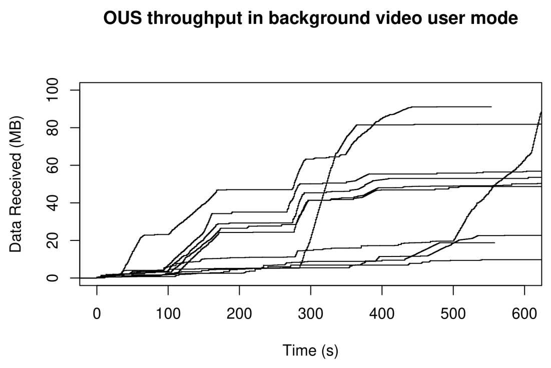

Figure 9: A step function of the overt channel throughput achieved using OUStral's background video user mode. This plot shows the replaceable bytes received (in MB) over time for ten runs of OUStral using the background video only user mode. For each run, the background video mode used the Slifox browser to navigate to a 10 minute YouTube video while performing browser actions in a separate tab(s).

Although we achieve reasonable overall throughput, the bytes received are bursty, and there are long periods during which the client receives no downstream data. This is due to both the time required to choose and start a new video as well as the buffering behaviour of pre-recorded video streams in which the client will load a large amount of content in a buffer, and then play back that video before loading more content.

We finally run OUStral with the background video mode, which has OUStral open a tab to a YouTube video and leave it to play while performing one of the browser actions in another (or several other) tabs. We took ten measurements for a duration of ten minutes each of OUStral using its background video user mode. We had OUStral navigate to and play the same long video in the background tab and then perform browser actions simultaneously in a separate tab(s). We would expect this to achieve a combined throughput of the video-only and single-tab measurements and indeed we see shorter periods of buffer time than in the video-only mode in Figure 9, which shows the total bytes received over time for all ten runs of the background video user mode. The mean throughput for OUStral in the background-video user mode was 721.6 kbps with a maximum reported throughput of 1528 kbps (1.5 Mbps) and a minimum of 144.4 kbps.

We hypothesize that using livestreams as the background video would significantly reduce the buffering time still present in our background video mode results, but as the majority of livestreamed content utilizes MPEG-4 or MPEG-TS format, rather than the WebM format, we save the analysis of livestreamed video tunnels for future work.

We also note that there is a significant loss in throughput specific to our Slitheen implementation due to the difficulties of parsing on packet boundaries at a decoy routing relay station that would not occur in an end-to-end deployment model such as that used by DeltaShaper [5] or Protozoa [7].

5.3 Improved Traffic Replacement Metrics

To create a baseline for our OUStral evaluation and to show what is possible given our WebM video and audio stream implementation for Slitheen, we additionally measured the overhead of our system. We measure the overhead as the amount of overt traffic that is needed to support Slitheen's covert streams. For reference, we compare our overall measurements of our Slitheen implementation using OUStral to the current state of the art traffic replacement systems and find that our results are comparable.

5.3.1 Overhead. In order to take full advantage of OUStral, we implemented WebM video replacement in Slitheen. To give an idea of the impact of adding a video channel to an existing censorship circumvention system, we measured the overhead of WebM video replacement in Slitheen. We calculate the overhead of our system to be the ratio of the total traffic received by the client to the amount of tunnelled covert traffic received. The overhead of our system is necessarily strictly larger than one, as preserving the illusion of the client's regular use of video streaming services requires that we replace only leaf data, and not the other data, headers, or metadata needed by the protocol. Our goal is to minimize this overhead as much as possible, both to improve the user experience of censorship-resistant browsing, and to minimize the number of decoy connections needed to provide a sufficient amount of tunnelled bandwidth.

We performed 100 ten-minute browsing sessions using OUStral. For each connection we measured the bytes of leaf data and compared that to the total number of bytes received by the client:

$$\texttt{overhead} = \frac{\texttt{total bytes received}}{\texttt{covert throughput}}$$

We found that the mean overhead was 4.7 (±1.6), meaning that for every byte of tunnelled traffic the user receives 4.7 bytes of total traffic. We note that we compute the overhead *pessimistically*: in our overhead calculations, the total bytes received includes TCP/IP headers, retransmits, etc., sent to the client while the covert traffic only includes traffic that Slitheen was able to replace at the application layer.

Adding WebM replacement to Slitheen had very low overhead compared with other similar tunnelling systems that provide defences against sophisticated traffic analysis attacks. Our comparison in Section 5.3.2 shows that overhead in our systems is lower than the original Slitheen system. At the same time, we maintain Slitheen's strong defences against detection. The overhead of our video streaming web browsing traffic is a vast improvement over the overhead presented of Slitheen [10]; by replacing just image resources, the mean overhead was about 20 (± 50) times the throughput with the large variance caused by the different ratios of image content to total content provided by each overt site. Thus, by moving from only replacing the images of overt sites to replacing the video and audio frames of streamed video content, we provide 4 times as much covert data as the original Slitheen system for the same amount of overt data.

We suspect that the overhead produced by our system can be reduced further through investigations into the leaf data of video and audio resources. Our replacement was very conservative and it is possible that much of the video streaming metadata can be

Table 2: A comparison of the overhead induced by existing systems that use protocol appropriation for censorship resistance. We measure overhead as the bytes of total traffic divided by the bytes of censorship resistance traffic. Some measurements were not reported in previous work and so receive an "Unknown" with a qualitative evaluation based on reported system descriptions.

System	Protocol	Overhead	Throughput	Traffic Shaping
DeltaShaper [5]	VoIP	2	7 kbps	●
Slitheen [10]	HTTPS	20	Unknown (low)	●
Waterfall [50]	HTTP/HTTPS	Unknown (low)	Unknown (high)	●
Slitheen + OUStral (This work)	WebM	4.7	up to 2.2 Mbps	●
Protozoa [7]	WebRTC	Unknown (low)	up to 1.4 Mbps	●
Conjure [29]	HTTPS	Unknown (low)	100 Mbps	○

safely replaced with covert content while maintaining the same overt traffic patterns. We leave a further reduction of the overhead to future work.

By adding a video replacement channel to Slitheen, we were able to take full advantage of OUStral's ability to support video channels while maintaining a low overhead, maintaining Slitheen's strong defences, and generating a large amount of replaceable overt content.

5.3.2 Comparison to existing systems. Table 2 provides a comparison of the work we did to extend Slitheen's traffic replacement for our OUStral evaluation (i.e., implementing WebM traffic replacement) to other recently proposed censorship circumvention systems that use tunnelling methods. We note that while our overhead and bandwidth measurements could be improved with optimizations in what constitutes leaf data, our implementation does perform far better compared to Slitheen [10] and DeltaShaper [5]. Our method also defends against an adversary capable of performing traffic analysis without making assumptions about the computational restrictions of censors which means that our system will continue to be secure in the future as machine learning techniques for network classification improve.

At an average throughput of 581 kbps and achieving up to 2.2 Mbps we provide high throughput comparative to previous tunnelling systems that perform some kind of traffic shaping to hide traffic patterns (i.e., DeltaShaper [5], Slitheen [10]).

In addition to throughput, we provide inter-connection shaping both due to the use of a web browser that loads all resources associated with an overt site and our work on overt user simulation. Waterfall [50] is another system in the decoy routing and tunnelling space that uses the same basic replacement technique as Slitheen but claims higher throughput and lower overhead due to the replacement of non-leaf, but cached resources. Although they do not provide measurements on their exact throughput, we suspect that our improvements would still outperform Waterfall as video and audio resources cannot be cached and if they are replaced as a whole, the video metadata will be overwritten and the playback engine will fail to request more buffer. Our method provides not only more throughput but also prevents identifiable traffic patterns in the dynamic resource request patterns of video streaming sites.

The results of our experiments show that our system is well suited for censorship-resistant traffic that requires a large amount of downstream bandwidth but minimal back-and-forth communication between the user and the covert site compared with the current state of the art. We perform very well compared to existing systems in loading large covert sites over time.

Protozoa [7] reports a maximum throughput of 1.4 Mbps as well as much improved latency. OUStral has only been implemented to support web browsing traffic and so is incompatible with systems like Protozoa that require video and audio frames to mimic a video call. An interesting direction for future research would look to making OUStral more versatile so that it could seamlessly integrate with Protozoa-like systems.

6 CONCLUSION

In this paper we presented OUStralopithicus (OUStral for short), an Overt User Simulator that generates human-like overt traffic for traffic-replacing censorship resistant tunnelling systems. We further demonstrate that such a tool can be used in tandem with high-throughput protocols such as WebM in order to provide strong resistance to traffic analysis attacks.

We find that combining traffic replacement with popular high-throughput protocol implementations and overt user simulation is an effective technique for hiding both the censored traffic and a user's participation in the system itself. We provide not only shaping for each outbound flow from the client, but also inter-connection shaping to defend against censors that evaluate a client's traffic over time for suspicious behaviour. By carefully selecting which protocols to use for traffic replacement, we leverage a censor's unwillingness to block an entire protocol and provide a way to tunnel covert traffic to users even as censorship activity becomes bolder.

Our results show that our method of user simulation to generate overt traffic tunnel traffic successfully evades sophisticated bot detection infrastructure and therefore can appear as a human user operating within the censor's bounds. In combination, we believe that traffic replacement and overt user simulation can give censorship resistors the upper hand in the fight for a free and open Internet.

ACKNOWLEDGMENTS

This research was undertaken, in part, thanks to funding from the Canada Research Chairs program. This work was also partially funded by an Ontario Graduate Scholarship and a Canada Graduate Scholarship. This material is based upon work supported by DARPA and AFRL under Contracts FA8750-19-C-0079 and FA8750-19-C-0500. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA. We would like to thank Nick Sullivan, Sergi Isasi, and Andrew Li from Cloudflare for their support and access to their Bot Management system. This work was conducted on land stolen from Indigenous peoples who have historically lived and who currently live in this territory. These groups include the Haudenosaunee, Neutral, and Anishinaabeg peoples.

REFERENCES

[1] Qurat-Ul-Ann Danyal Akbar, Marcel Flores, and Aleksandar Kuzmanovic. 2016. DNS-sly: Avoiding Censorship through Network Complexity. In *6th USENIX Workshop on Free and Open Communications on the Internet (FOCI 16)*. USENIX Association. https://www.usenix.org/conference/foci16/workshop-program/presentation/akbar

[2] Collin Anderson, Philipp Winter, and Roya. 2014. Global network interference detec- tion over the RIPE Atlas network. In *4th USENIX Workshop on Free and Open Communications on the Internet (FOCI 14)*. https://www.usenix.org/system/files/conference/foci14/foci14-anderson.pdf

[3] Yawning Angel and Philipp Winter. 2014. obfs4 (The obfourscator). https://gitweb.torproject.org/pluggable-transports/obfs4.git/tree/doc/obfs4-spec.txt. [Online; accessed May 30, 2021].

[4] Simurgh Aryan, Homa Aryan, and J. Alex Halderman. 2013. Internet censorship in Iran: A first look. In *3rd USENIX Work- shop on Free and Open Communications on the Internet (FOCI)*. https://www.usenix.org/system/files/conference/foci13/foci13-aryan.pdf

[5] Diogo Barradas, Nuno Santos, and Luis Rodrigues. 2017. DeltaShaper: Enabling UnobservableCensorship-resistant TCP Tunneling overVideoconferencing Streams. In *Proceedings on Privacy Enhancing Technologies*. 1–18. https://petsymposium.org/2017/papers/issue4/paper15-2017-4-source.pdf.

[6] 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, Baltimore, MD, 169–185. https://www.usenix.org/conference/usenixsecurity18/presentation/barradas

[7] 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 ACM SIGSAC Conference on Computer and Communications Security*. Virtual Event, USA.

[8] Benedikt Birtel and Christian Rossow. 2020. Slitheen++: Stealth TLS-based Decoy Routing. In *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*. USENIX Association. https://www.usenix.org/conference/foci20/presentation/birtel

[9] Cecylia Bocovich. 2018. *Recipes for Resistance: A Censorship circumvention cookbook*. Ph.D. Dissertation. University of Waterloo, Waterloo.

[10] Cecylia Bocovich and Ian Goldberg. 2016. Slitheen: Perfectly Imitated Decoy Routing Through Traffic Replacement. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, 1702–1714.

[11] Cecylia Bocovich and Ian Goldberg. 2018. Secure asymmetry and deployability for decoy routing systems. *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2018, 3 (July 2018).

[12] Owen Bowcott. 2017. Turks detained for using encrypted app 'had human rights breached'. https://www.theguardian.com/world/2017/sep/11/turks-detained-encrypted-bylock-messaging-app-human-rights-breached. [Online; accessed May 30, 2021].

[13] Jacopo Cesareo, Karlin Karlin, Michael Schapira, and Jennifer Rexford. 2012. *Optimizing the Placement of Implicit Proxies*. Technical Report. Princeton, NJ, USA. https://cs.princeton.edu/~jrex/papers/decoy-routing.pdf

[14] Cloudflare. [n.d.]. Bot Management Dashboard. https://developers.cloudflare.com/logs/tutorials/bot-management-dashboard/. [Online; accessed May 30, 2021].

[15] Cloudflare. [n.d.]. Understanding Cloudflare Bot Management (beta). https://support.cloudflare.com/hc/en-us/articles/360027519452-Understanding-Cloudflare-Bot-Management-beta-. [Online; accessed May 30, 2021].

[16] Cloudflare. [n.d.]. What Is Bot Management? How Bot Managers Work. https://www.cloudflare.com/learning/bots/what-is-bot-management/. [Online; accessed May 30, 2021].

[17] Nicholas de Pencier. 2017. Black Code. Mongrel Media.

[18] Roger Dingledine. 2011. Iran blocks Tor; Tor releases same-day fix. https://blog.torproject.org/iran-blocks-tor-tor-releases-same-day-fix. [Online; accessed May 30, 2021].

[19] Roger Dingledine. 2012. Obfsproxy: The Next Step in the Censorship Arms Race. https://blog.torproject.org/blog/obfsproxy-next-step-censorship-arms-race. [Online; accessed May 30, 2021].

[20] Patrick Dubroy and Ravin Balakrishnan. 2010. A Study of Tabbed Browsing Among Mozilla Firefox Users. (2010). https://www.dgp.toronto.edu/~ravin/papers/chi2010_tabbedbrowsing.pdf.

[21] Kevin P. Dyer, Scott E. Coull, and Thomas Shrimpton. 2015. Marionette: A Programmable Network-traffic Obfuscation System. In *24th USENIX Security Symposium* (Washington, D.C.). 367–382.

[22] Daniel Ellard, Christine Jones, Victoria Manfredi, Timothy W. Strayer, Bishal Thapa, Megan Van Welie, and Alden Jackson. 2015. Rebound: Decoy routing on asymmetric routes via error messages. In *Local Computer Networks (LCN), 2015 IEEE 40th Conference on*. 91–99.

[23] Roya Ensafi, David Fifield, Philipp Winter, Nick Feamster, Nicholas Weaver, and Vern Paxson. 2015. Examining How the Great Firewall Discovers Hidden Circumvention Servers. In *Internet Measurement Conference*. ACM.

[24] Roya Ensafi, Jeffrey Knockel, Geoffrey Alexander, and Jedidiah R. Crandall. 2014. De-tecting intentional packet drops on the Internet via TCP/IP side channels. In *Passive and Active Measurement*.

[25] David Fifield. 2017. *Threat modeling and circumvention of Internet censorship*. Ph.D. Dissertation. University of California, Berkeley.

[26] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. 2015. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies* 2015, 2 (2015), 46–64.

[27] David Fifield and Lynn Tsai. 2016. Censors' Delay in Blocking Circumvention Proxies. In *6th USENIX Workshop on Free and Open Communications on the Internet (FOCI 16)*. USENIX Association, Austin, TX. https://www.usenix.org/conference/foci16/workshop-program/presentation/fifield

[28] Sergey Frolov, Fred Douglas, Will Scott, Allison McDonald, Benjamin VanderSloot, Rod Hynes, Adam Kruger, Michalis Kallitsis, David G. Robinson, Steve Schultze, Nikita Borisov, Alex Halderman, and Eric Wustrow. 2017. An ISP-Scale Deployment of TapDance. In *7th USENIX Workshop on Free and Open Communications on the Internet (FOCI 17)*. USENIX Association, Vancouver, BC.

[29] Sergey Frolov, Jack Wampler, Sze Chuen Tan, J. Alex Halderman, Nikita Borisov, and Eric Wustrow. 2019. Conjure: Summoning Proxies from Unused Address Space. In *Proceedings of the 26th ACM Conference on Computer and Communications Security (CCS '19)*. ACM.

[30] Sergey Frolov and Eric Wustrow. 2019. The use of TLS in Censorship Circumvention. In *Network and Distributed System Security Symposium (NDSS '19)*. https://tlsfingerprint.io/static/frolov2019.pdf

[31] John Geddes, Max Schuchard, and Nicholas Hopper. 2013. Cover Your ACKs: Pitfalls of Covert Channel Censorship Circumvention. In *Proceedings of the 2013 ACM Conference on Computer and Communications Security (CCS '13)*. ACM, 361–372.

[32] Yu Guo, Junzheng Shi, Zigang Cao, Cuicui Kang, Gang Xiong, and Zhen Li. 2019. Machine Learning Based CloudBot Detection Using Multi-Layer Traffic Statistics. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. 2428–2435. https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00339

[33] Javad Hamidzadeh, Mahdieh Zabihimayvan, and Reza Sadeghi. 2018. Detection of Web site visitors based on fuzzy rough sets. *Soft Comput* 22, 7 (April 2018), 2175–2188. https://doi.org/10.1007/s00500-016-2476-4

[34] Serene Han. 2011. Snowflake Technical Overview. https://keroserene.net/snowflake/technical. [Online; accessed May 30, 2021].

[35] Ariya Hidayat. [n.d.]. *PhantomJS Documentation*. https://phantomjs.org/.

[36] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. 2013. The Parrot Is Dead: Observing Unobservable Network Communications. In *2013 IEEE Symposium on Security and Privacy*. 65–79.

[37] Amir Houmansadr, Giang T.K. Nguyen, Matthew Caesar, and Nikita Borisov. 2011. Cirripede: Circumvention Infrastructure Using Router Redirection with Plausible Deniability. In *18th ACM Conference on Computer and Communications Security (CCS '11)*. 187–200.

[38] Amir Houmansadr, Thomas Riedl, Nikita Borisov, and Andrew Singer. 2013. I want my voice to be heard: IP over Voice-over-IP for unobservable censorship circumvention. In *Network and Distributed System Security*. The Internet Society. https://people.cs.umass.edu/~amir/papers/FreeWave.pdf

[39] Amir Houmansadr, Edmund L Wong, and Vitaly Shmatikov. 2014. No direction home: The true cost of routing around decoys. In *2014 Network and Distributed System Security (NDSS) Symposium*.

[40] Amir Houmansadr, Wenxuan Zhou, Matthew Caesar, and Nikita Borisov. 2017. SWEET: Serving the Web by Exploiting Email Tunnels. *IEEE/ACM Transactions on Networking* 25, 3 (June 2017), 1517–1527.

[41] Christos Iliou, Theodoros Kostoulas, Theodora Tsikrika, Vasilis Katos, Stefanos Vrochidis, and Yiannis Kompatsiaris. 2019. Towards a Framework for Detecting Advanced Web Bots. In *Proceedings of the 14th International Conference on Availability, Reliability and Security* (Canterbury, CA, United Kingdom) *(ARES '19)*. Association for Computing Machinery, New York, NY, USA, Article 18, 10 pages. https://doi.org/10.1145/3339252.3339267

[42] Imperva. 2020. Bad Bot Report: Bad Bots Strike Back. https://www.imperva.com/resources/reports/Imperva_BadBot_Report_V2.0.pdf. https://www.imperva.com/resources/reports/Imperva_BadBot_Report_V2.0.pdf/ [Online; accessed June 8, 2021].

[43] Tariq Jamil. 1999. Steganography: the art of hiding information in plain sight. *IEEE Potentials* 18, 1 (1999), 10–12.

[44] Josh Karlin, Daniel Ellard, Alden W Jackson, Christine E Jones, Greg Lauer, David P Mankins, and W Timothy Strayer. 2011. Decoy routing: Toward unblockable internet communication. In *USENIX Workshop on Free and Open Communications on the Internet.*

[45] Chao Liu, Ryen W. White, and Susan Dumais. 2010. Understanding Web Browsing Behaviors through Weibull Analysis of Dwell Time. (2010). https://www.microsoft.com/en-us/research/wp-content/uploads/2010/10/SIGIR2010-DwellTimeModel.pdf.

[46] Richard McPherson, Amir Houmansadr, and Vitaly Shmatikov. 2016. CovertCast: Using Live Streaming to Evade Internet Censorship. *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2016 (July 2016).

[47] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. 2012. SkypeMorph: Protocol Obfuscation for Tor Bridges. In *19th ACM Conference on Computer and Communications Security (CCS '12)*. 97–108.

[48] Zubair Nabi. 2013. The anatomy of web censorship in Pakistan. In *3rd USENIX Workshop on Free and Open Communications on the Internet (FOCI).* https://www.usenix.org/conference/foci13/workshop-program/presentation/nabi

[49] Milad Nasr and Amir Houmansadr. 2016. GAME OF DECOYS: Optimal Decoy Routing Through Game Theory. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, 1727–1738.

[50] Milad Nasr, Hadi Zolfaghari, and Amir Houmansadr. 2017. The Waterfall of Liberty: Decoy Routing Circumvention that Resists Routing Attacks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. 2037–2052.

[51] Paul Pearce, Ben Jones, Frank Li, Roya Ensafi, Nick Feamster, Nick Weaver, and Vern Paxson. 2017. Global measurement of DNS manipulation. In *26th USENIX Se- curity Symposium (USENIX Security 17)*. https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-pearce.pdf

[52] Psiphon. 2020. Psiphon | Uncensored Internet access for Windows and Mobile. https://psiphon.ca/. [Online; accessed May 30, 2021].

[53] Stefano Rovetta, Alberto Cabri, Francesco Masulli, and Grażyna Suchacka. 2019. Bot or Not? A Case Study on Bot Recognition from Web Session Logs. In *Quantifying and Processing Biomedical and Behavioral Signals*, Anna Esposito, Marcos Faundez-Zanuy, Francesco Carlo Morabito, and Eros Pasero (Eds.). Springer International Publishing, Cham, 197–206. https://doi.org/10.1007/978-3-319-95095-2_19

[54] Shadi Sadeghpour, Natalija Vlajic, Pooria Madani, and Dusan Stevanovic. 2021. Unsupervised ML Based Detection of Malicious Web Sessions with Automated Feature Selection: Design and Real-World Validation. In *2021 IEEE 18th Annual Consumer Communications Networking Conference (CCNC)*. 1–9. https://doi.org/10.1109/CCNC49032.2021.9369591 ISSN: 2331-9860.

[55] Selenium. 2019. Selenium Documentation. https://www.seleniumhq.org/docs/. [Online; accessed May 30, 2021].

[56] Adrian Shahbaz and Allie Funk. 2020. *The Pandemic's Digital Shadow.* Technical Report. Freedom House. https://freedomhouse.org/report/freedom-net/2020/pandemics-digital-shadow, [Online; accessed May 30, 2021].

[57] Piyush Kumar Sharma, Devashish Gosain, Himanshu Sagar, Chaitanya Kumar, Aneesh Dogra, Vinayak Naik, H.B. Acharya, and Sambuddho Chakravarty. 2020. SiegeBreaker: An SDN Based Practical Decoy Routing System. *Proceedings on Privacy Enhancing Technologies* 2020 (July 2020).

[58] GraÅijyna Suchacka and Jacek IwaÅĎski. 2020. Identifying legitimate Web users and bots with different traffic profiles âĂŤ an Information Bottleneck approach. *Knowledge-Based Systems* 197 (June 2020), 105875. https://doi.org/10.1016/j.knosys.2020.105875

[59] Michael Carl Tschantz, Sadia Afroz, Anonymous, and Vern Paxson. 2016. SoK: Towards Grounding Censorship Circumvention in Empiricism. In *2016 IEEE Symposium on Security and Privacy.* 914–933.

[60] Benjamin VanderSloot, Sergey Frolov, Jack Wampler, Sze Chuen Tan, Irv Simpson, Michalis Kallitsis, J. Alex Halderman, Nikita Borisov, and Eric Wustrow. 2020. Running Refraction Networking for Real. *Proceedings on Privacy Enhancing Technologies* 2020, 4 (Oct. 2020), 321–335.

[61] Christian von der Weth and Manfred Hauswirth. 2013. DOBBS: Towards a Comprehensive Dataset to Study the Browsing Behavior of Online Users. *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)* (2013).

[62] Liang Wang, Kevin P. Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. 2015. Seeing Through Network-Protocol Obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. 57–69.

[63] Tim Wilde. 2012. Great firewall Tor probing circa 09 DEC 2011. https://gist.github.com/twilde/da3c7a9af01d74cd7de7. [Online; accessed May 30, 2021].

[64] Philipp Winter, Tobias Pulls, and Juergen Fuss. 2013. ScrambleSuit: A Polymorphic Network Protocol to Circumvent Censorship. In *12th ACM Workshop on Workshop on Privacy in the Electronic Society (WPES '13)*. 213–224. https://doi.org/10.1145/2517840.2517856

[65] Eric Wustrow, Colleen M. Swanson, and J. Alex Halderman. 2014. TapDance: End-to-middle Anticensorship Without Flow Blocking. In *23rd USENIX Security Symposium.* 159–174.

[66] Eric Wustrow, Scott Wolchok, Ian Goldberg, and J. Alex Halderman. 2011. Telex: Anticensorship in the Network Infrastructure. In *20th USENIX Security Symposium.*

[67] Yang Yang, Natalija Vlajic, and Uyen Trang Nguyen. 2015. Next Generation of Impersonator Bots: Mimicking Human Browsing on Previously Unvisited Sites. In *IEEE 2nd International Conference on Cyber Security and Cloud Computing.* http://www.cse.yorku.ca/~utn/papers/next-generation-of-impersonator-bots.pdf.