Camoufler: Accessing The Censored Web By Utilizing Instant Messaging Channels

Piyush Kumar Sharma
IIIT Delhi, India
piyushs@iiitd.ac.in

Devashish Gosain
IIIT Delhi, India
devashishg@iiitd.ac.in

Sambuddho Chakravarty
IIIT Delhi, India
sambuddho@iiitd.ac.in

ABSTRACT

Free and open communication over the Internet is considered a fundamental human right, essential to prevent repressions from silencing voices of dissent. This has led to the development of various anti-censorship systems. Recent systems have relied on a common blocking resistance strategy *i.e.,* incurring collateral damage to the censoring regimes, if they attempt to restrict such systems. However, despite being promising, systems built on such strategies pose additional challenges, *viz.,* deployment limitations, poor QoS *etc.* These challenges prevent their wide scale adoption.

Thus, we propose a new anti-censorship system, *Camoufler*, that overcomes aforementioned challenges, while still maintaining similar blocking resistance. Camoufler leverages Instant Messaging (IM) platforms to tunnel client's censored content. This content (encapsulated inside IM traffic) is transported to the Camoufler server (hosted in a free country), which proxies it to the censored website. However, the eavesdropping censor would still observe regular IM traffic being exchanged between the IM peers. Thus, utilizing IM channels *as-is* for transporting traffic provides unobservability, while also ensuring good QoS, due to its inherent properties such as low-latency message transports. Moreover, it does not pose new deployment challenges. Performance evaluation of Camoufler, implemented on five popular IM apps indicate that it provides sufficient QoS for web browsing. *E.g.,* the median time to render the homepages of Alexa top-1k sites was recorded to be about 3.6s, when using Camoufler implemented over *Signal* IM application.

CCS CONCEPTS

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

KEYWORDS

Anti-censorship; Instant Messaging; IM

ACM Reference Format:
Piyush Kumar Sharma, Devashish Gosain, and Sambuddho Chakravarty. 2021. Camoufler: Accessing The Censored Web By Utilizing Instant Messaging Channels. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21), June 7–11, 2021, Virtual Event, Hong Kong*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3433210.3453080

ASIA CCS '21, June 7–11, 2021, Virtual Event, Hong Kong

ACM ISBN 978-1-4503-8287-8/21/06…$15.00
https://doi.org/10.1145/3433210.3453080

1 INTRODUCTION

In the last decade the Internet has become an integral part in various aspects of our lives, ranging from education, healthcare to social interactions, technological advancements in different spheres of science *etc.* Free flow of ideas and unrestricted access to information has become a necessity not only for personal development but also for the advancement of the society. However, repressive regimes continuously attempt to surveil and censor this flow of information by restricting the content, users can share and access.

In opposition, free speech activists and researchers developed systems [32, 46, 63] which aim at providing unhindered access to information for clients in repressive regimes. This strife of ideologies between the censors and the free speech advocates has led to the evolution and development of effective censorship as well as anti-censorship technologies [22, 46, 63]. This has led to an arms race between the censors and free speech activists. As censors advance their craft, researchers try to stay a step ahead by developing hard to block anti-censorship systems [66].

To that end, recent anti-censorship systems [33, 38, 45] are designed on a fundamental principle *i.e., to incur collateral damages to the censor, if the they attempts to disrupt the circumvention scheme.* This makes it difficult for the adversary to completely block these systems. Approaches like *decoy routing* [45], *domain fronting* [33] *etc.* are examples of such systems. Restricting Decoy Routing requires the censor to update nation-wide routing policies, and in the process sustain heavy collateral damages, *e.g.,* increased performance overheads [52]. Similarly, Domain Fronting requires the adversary to block cloud services (such as Google App engine), which might also be hosting essential services for oblivious users. Other systems such as *Conjure* [35] and *MassBrowser* [53] require the censor to block some IPs or IP prefixes, thereby also blocking other innocuous services running behind those. Lastly, *tunnelling systems* such as SWEET [43], Covercast [50], Freewave [41] *etc.*, transport censored traffic via services and protocols essential for smooth functioning of businesses, and thus a censor's economy. These systems exploit Email [43], VoIP [41], video streams [24, 50] *etc.*, as covert channels to transport content. Since these systems use the underlying protocol as-is, it becomes hard for an adversary to distinguish circumvented traffic from the underlying protocol's messages. Hence, a determined adversary may attempt to disrupt the use of the underlying protocol itself (*e.g.,* emails in case of SWEET). Although, blocking such channels may incur collateral damage to the censor.

However, despite these systems providing efficient blocking resistance from the adversary, they exhibit other challenges which

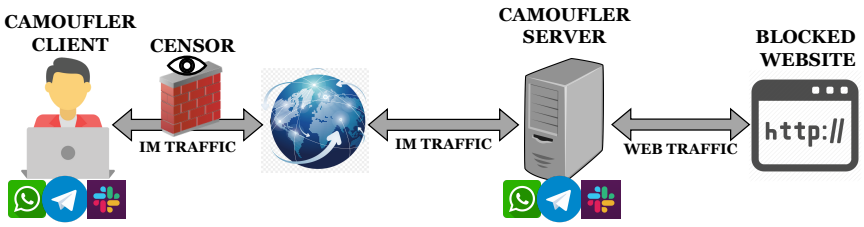

Figure 1: Camoufler basic architecture.

hinders them from being widely used. These challenges include deployment limitations, high cost of operation and low performance for the users. For instance, Decoy Routing and Conjure requires collaboration from ISPs to install and maintain additional network hardware for them to function. Similarly, Domain Fronting requires hosting a proxy on a fronting service (such as Google App Engine, Amazon Cloudfront *etc.*) which incur high periodic subscription [33] costs. Moreover, DeltaShaper [24] (a tunneling based system) provides 2.56 Kbps throughput, which is insufficient for providing web browsing.

Thus, we propose and build a new tunnelling based system, *Camoufler*, that aims at overcoming the shortcomings of the existing systems while maintaining similar blocking resistance. Camoufler utilizes Instant Messaging (IM) platform as a medium to tunnel the censored traffic. IM channel seems to be better suited to act as a tunnelling medium for developing such a system in comparison to other existing counterparts. This is because it has some salient features, that in general, anti-censorship schemes strive to achieve. They are:

(1) **Minimized latency:** IM platforms aim at minimizing the latency ($< 1s$) [1] when user exchange messages with each other (ref. Sec. 4). This provides good QoS, unlike other non-realtime channels such as emails.
(2) **Adequate throughput:** IM platforms have sufficient data transport capacity (in the form of attachments), in comparison to channels such as VoIP (which encode data at low bit rate). Thus IM proves to be more suitable for regular web browsing.
(3) **Reliability:** IM is also a reliable channel in comparison to others such as VoIP and video. While IM ensures reliable delivery of messages, real-time VoIP and video generally do not incorporate this feature, as the information lost in the latter channels becomes irrelevant and is thus not recovered.
(4) **Blocking resistance:** Similar to existing systems, restricting the underlying IM applications may incur collateral damage to the adversary as IM apps are an important part of personal as well as professional spaces [23, 55, 58, 59, 62][1]. At present, businesses utilize IMs as a medium to advertise, communicate and expand. *E.g.,* several airline reservation and movie ticketing services utilize IMs to directly send e-tickets to customers. Further, IM based collaboration platforms such as Slack, Flock *etc.*, are now widely used as an alternate to email for professional communication [3, 11, 14].
(5) **Deployment ease:** IM applications are ubiquitously used by netizens. As a consequence, to use Camoufler, a user merely needs to install programs at the client and server ends. Apart from that, there are no additional requirements as assumed by previous circumvention proposals (*e.g.,* collaborating with ISPs [34, 36, 45]).

Thus, we developed Camofler with IM channels as its underlying tunneling protocol. The basic architecture of Camoufler (depicted in Fig. 1) requires the censored user to have an IM ID on any popular IM platform. The user using the Camoufler client, tunnels requests to a Camoufler server hosted in a free country, which acts as a proxy and serves the censored content to the client. All the censored content is encrypted and exchanged via the underlying IM platform that the user has access to. This would give the pretense to the censor that regular IM clients are communicating, providing unobservability to Camoufler.

Camoufler has been implemented and tested to work on several popular IM applications including Whatsapp, Signal, Telegram, Slack and Skype, and can be extended to others as well. Camoufler clients take an average and median time of 4.1s and 3.6s respectively, to access webpages of Alexa top-1000 sites.

To summarize, following are our major contributions:

- The design of a new anti-censorship system Camoufler, which utilizes IM apps as covert media to tunnel censored content.
  - Usage of IM covert channel serves multiple advantages in comparison to existing tunneling channels by ensuring: (i) low latency (ii) reliability (iii) similar blocking resistance and (iv) high data transport capacity.
- A prototype implementation with a detailed performance evaluation of Camoufler on five popular apps including, Signal, Telegram, Slack, Whatsapp, and Skype, depicting the feasibility of our design.
- A detailed security analysis, depicting Camoufler's robustness against variety of attacks including traffic analysis.

2 BACKGROUND AND RELATED WORK

2.1 Instant Message Applications

Instant Messaging (IM) applications are one of the most widely used medium of communication over the Internet. They are used for both personal (Whatsapp [19], Telegram [15] *etc.*) as well as professional communication (Slack [12], Flock [2] *etc.*). This is reflected in a recent report [61] which depicts that the number of monthly active users of IM platforms, were $\approx$ 2.4 billion in January 2019. These users are expected to go beyond 3 billion in 2022.

[1]There were 2.5 billion active IM users till January 2019 and this number is expected to easily cross 3 billion till 2022 [61].

IM applications provide variety of features which are similar across all IM platforms. These include real time messages, image, video and file sharing (for individuals as well as for groups). Some IM apps also support integration of payment wallets [18]. Apart from these features, such apps consider providing security (and privacy) to their clients as one of their design principles. To that end, almost all IM apps provide encryption by either requiring a TLS connection from the client to the IM app providers *i.e., end-to-middle (E2M)* or by having an *end-to-end (E2E)* encrypted connection between the clients (ref. Fig. 2). This provides confidentiality to the IM clients from eavesdroppers who may attempt to snoop on the network traffic. However, it is desirable that the applications use E2E encryption as it ensures that neither the local eavesdropper, and nor the IM provider is able to see any content. Many applications support this feature, including Whatsapp, Telegram, Viber, Signal, Line, Flock *etc.*

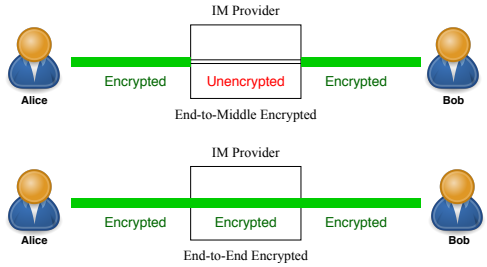

Figure 2: Difference between end-to-middle (E2M) and end-to-end (E2E) encryption in IM applications.

E2E and E2M encryption schemes We now give a brief overview of these two different encryption schemes and how they are implemented in IM applications. In an IM application supporting E2E encryption, the peers depend on a centralized server which is responsible for distributing public keys of the peers and relaying messages between them. The client utility uploads a *key bundle* (containing public keys) to the central server. When an IM user Alice initiates communication with another user Bob, her client utility retrieves the key bundle of Bob from the key server, and derives session key (*e.g.,* using *Triple Diffie Hellman algorithm* [49]). Then, Alice forwards her key bundle to Bob, which also derives the same session key. At this point, they establish an E2E encrypted connection with each other. With E2E encryption scheme, even the key server (*i.e.,* an IM provider) cannot compromise the confidentiality of the messages being exchanged.

On the other hand, applications supporting E2M encryption generally use the TLS protocol to establish an encrypted connection between the users and the provider. *E.g.,* if Alice and Bob wish to communicate with one another, they first establish individual TLS connections to the IM provider's server [2]. As evident from Fig. 2, E2M encryption scheme allows the provider to observe the content (being relayed) in plain-text. However, it protects the confidentiality of the users' messages from any third-party.

[2]A server managed by app maintainers which stores and relays messages.

2.2 Related Work

To promote free speech and unhindered information exchange over the Internet, researchers and free speech activists have proposed several anti-censorship solution [29, 31, 33, 45, 46, 51, 63, 67, 70] over the years. Given the plethora of such systems and proposals available, we try to categorize them and enlist their advantages and disadvantages.

(1) **Proxy based systems:** These systems include proxies, VPNs, Tor [32] *etc.* These involve clients relaying their traffic via intermediate proxies, that connect to the censored sites on behalf of the clients. Such systems are easily deployable and thus readily available to end users. However, they can be easily blocked by the adversaries, as generally such systems publicly advertise the proxies' IP addresss. This, makes it trivial for the adversary to block them as soon as discovered.

(2) **Decoy routing systems:** Decoy routing is a promising approach, that requires client in the censoring regime to connect to unfiltered websites. These requests contain covert information, that allows special intermediate routers (*decoy routers*) en-route to intercept them, and decipher the true censored destination that the client wishes to access. These decoy routers then proxy the requests and responses between the clients and censored sites, while keeping up with the pretense to the adversary that the client is connected to the unfiltered website. Examples of such systems include, Telex [68], Cirripede [40], Slitheen [27], Tapdance [67], Waterfall of Liberty [54],SiegeBreaker [57] *etc.* To censor decoy routers, the adversary may require undertaking daunting measures such as changing entire nation's routing policy [56] in order to bypass such systems. Such routing changes are prohibitively expensive to achieve in practice [36, 42]. Thus it becomes very difficult for the adversaries to block them. However, such systems require collaboration from the ISPs in order to function and thus pose a hurdle in deployment [36].

(3) **Mimicry based systems:** These systems attempt to disguise and transfer censored content as regular applications' protocol messages. *E.g.,* SkypeMorph [51], helps acccess censored websites by mimicking Skype's communication protocol. Others, like CensorSpoofer [65], obfuscate requests to camouflage censored sites as VoIP messages transported over SIP. However, such systems are relatively easier for the adversary to block as it is very difficult to mimic all the features of the underlying protocol [39]. Moreover, their performance depends on the traffic rate of the cover protocols. VoIP, used commonly, has very low transmission rates (*e.g.,* 5 - 40 Kbps in Skype), thus leading to low QoS for web browsing.

(4) **Tunneling based systems:** Tunneling based systems rely on encapsulating covert traffic in standard application protocol messages – *e.g.,* email, VoIP, video streams, online games *etc.* Examples include SWEET [43], Covertcast [50], Delta Shaper [24], Freewave [41], CloudTransport [28], Rook [64], Games without frontiers [37], Autoflowleaker [44] *etc.* These systems are an improvement over mimicry based systems as they do not mimic protocol messages, rather they directly use them as the covert channels. This ensures that the features of the underlying protocol (*e.g.,* packet size) remain unaltered, while the censored content is encapsulated inside the payload. In turn,

this makes it difficult for the censor to disambiguate them from regular (underlying) protocol messages. Thus such systems provide more efficient blocking resistance against adversaries, as the latter may require blocking the complete underlying applications (such as email, cloud services *etc.*) which may result in massive collateral damages. However, such systems provide low QoS for web downloads (*e.g.,* due to low offered bandwidth in these channels) and are limited in their deployment.

Recently, another tunneling based system Protozoa [25] (with design goals similar to Camoufler) was proposed. It also aims at ameliorating the challenges associated with existing tunnelling systems. Interestingly, it was developed roughly around the same time as that of Camoufler, and we believe it is a promising circumvention solution.

(5) **Miscellaneous systems:** There exist other anti-censorship systems, which do not fall in any of the above categories, like domain fronting [33], CacheBrowser [38], MassBrowser [53] *etc.* Domain fronting makes use of different popular cloud services such as `Google app engine`, to access censored content. The request to the proxy server (hidden behind a cloud server), is concealed in a HTTPS request, which is destined to the domain name of an innocuous front-end of the cloud server. This front-end decrypts the HTTPS request and forwards it to the proxy server. To block such services the adversary may require blocking the entire fronting service (*e.g.,* Google app engine), thereby blocking other third party applications that use the platform. However, leveraging fronting services is not cost effective for deployment [33].

Similarly, CacheBrowser works by utilizing content distribution networks (CDNs) frontends, located outside the censor's boundary. It requires the blocked content to be hosted on the CDN network, and thus also has an associated cost. Moreover, websites not hosted on these CDNs cannot be accessed.

A recent system MassBrowser by Nasr *et al.*, leverages some of the existing techniques (such as Domain fronting and CDN browsing) to build a *client-to-client proxy system.* It is similar to proxy based systems (described before), with a fundamental difference that these proxies are not hosted on a public IP but are rather behind public NATs. Thus, blocking the MassBrowser is not as easy as blocking the IP addresses as it would lead to blocking other clients behind those NATed IPs.

Lastly, similar to Decoy Routing a recent approach by Frolov *et al.* (Conjure [35]) tries to host proxies on an ISP's unused IP addresse space. However, Conjure also requires collaboration from ISPs to install network taps that would allow them to inspect the traffic transiting the ISP.

Camoufler is an example of tunneling based systems. It aims to address the shortcomings of tunneling systems such as low QoS, low-bandwidth channel *etc.* Additionally, similar to other systems, we attempt to achieve effective blocking resistance.

3 CAMOUFLER ARCHITECTURE

This section describes the threat model we consider for Camoufler, and present its design (ref. Fig. 3) as well as a step-by-step walkthrough of the protocol.

3.1 Threat Model

We assume the adversary to be a nation state, which can employ any existing censorship technique, *e.g.* IP/DNS filtering, URL and keyword filtering [22, 30, 69] *etc.* However, we assume that the adversary is not willing to be disconnected from important Internet services (or from the complete Internet for that matter) and inflict loss to itself, for achieving extensive censorship. Specifically, censor would allow atleast some IM based channel(s) to function within its jurisdiction. However, it may attempt to selectively block certain IM platforms. This is because IM platforms have proliferated significantly and are widely used in personal as well as professional space [23, 58, 59]. Hence, it would be difficult for the censor to completely block them without sustaining significant collateral damage.

Additionally, we also assume that the IM channels are encrypted, either end-to-end or end-to-middle, as is already the case with most of the existing popular IM applications [60]. Thus the adversary can monitor or actively analyze the encrypted traffic. He may also attempt to drop, modify or replay packets for deliberately inducing perturbations (for some suspicious IM connections) to detect the usage of Camoufler. However, he would refrain from launching these active attacks at a large scale so as to not disrupt the communication of regular IM clients.

3.2 System Design

We now describe the protocol design of Camoufler. It tunnels the web requests of IM users (*i.e.,* Camoufler clients) residing in censored regime to an IM peer in a free country (acting as Camoufler server), using IM applications. The users of Camoufler could themselves rent out VPS servers in such countries and run the Camoufler server, or may rely on trusted peers (friends in free countries) for running the server on residential or educational hosts. The Camoufler server proxies the received web requests to the censored destination. This enables the clients (in a censored regime) to access blocked content.

General working of our end-to-end system can be understood by referring to Fig. 3. We begin by describing the individual components of Camoufler and their functioning, followed by a step by step walkthrough of Camouflers' operation.

Camoufler client consists of the following components:

(1) Local Proxy (LP): It is a standard HTTP proxy that acts as an interface between the user's browser and the *client engine* (described ahead). More specifically, it accepts content requests from the browser and passes them to the client engine and vice versa.

(2) Client Engine (CE): Client engine acts as an interface between the LP and the underlying IM application. It receives a web request from the LP, processes it (encryption, compression *etc.*), and then forwards the processed web request to the underlying IM application.

IM application oblivious to the aforementioned process sends the received content to the other IM peer (Camoufler server) as standard IM packets. Similarly, CE receives content from the IM application, decompresses and decrypts it before sending it to the LP.

The IM server consists of the following components:

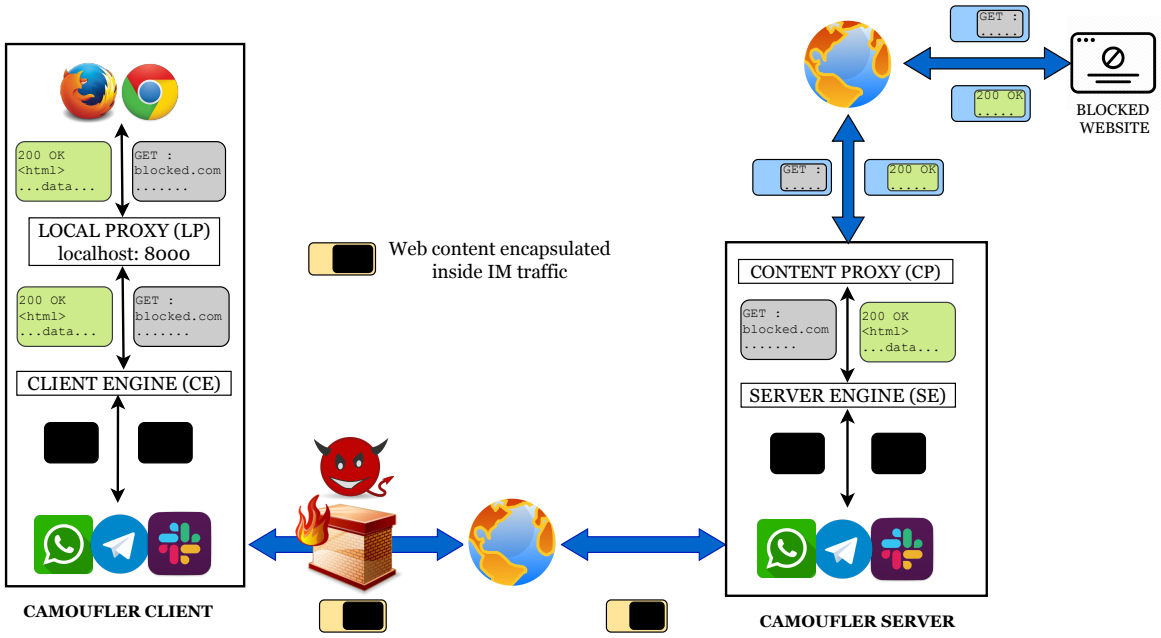

Figure 3: Camoufler detailed architecture.

(1) Server Engine (SE): The *server engine* works similar to CE. When SE receives a web request, it forwards it to the CP. Later, when SE receives the web content from the CP, it compresses and encrypts the web content and sends it back to the Camoufler client, using the underlying IM application.
(2) Content Proxy (CP): The *content proxy* retrieves the requested censored content from the blocked website and sends it back to the SE.

Furthermore, similar to other existing systems, we assume that the user has access to the Camoufler software. We now describe a complete walkthrough of our system. The steps involved in accessing Camoufler are as follows:

(1) The user configures his browser of choice (Firefox, Chrome, Opera *etc.*) to forward all the requests of the browser to the LP. Once configured, the client can use its browser to access the blocked content freely.
(2) Next, the user inputs the URL of a censored website in the browser. This is forwarded to the LP, which forwards this connection request to the CE and waits for the retrieved content.
(3) On receiving the content request (from LP), CE encrypts it (using a derived shared key as described ahead in §3.2.1), and uses the underlying IM application to transport this request to the Camoufler server (SE).
(4) SE, on receiving content request, decrypts it and then forwards the request to the CP. The CP, retrieves the censored content and sends it back to the SE. The SE then finally, encrypts and compresses this content and tunnels it back to Camoufler client using the IM channel.
(5) On receiving the website content from Camoufler server, the CE decompresses, decrypts and forwards it to the LP, which in turn provides it to the browser for appropriate rendering.

3.2.1 On encrypting exchanged content. Most of the IM applications support end-to-end encryption and thus ideally it is not required to additionally encrypt the messages exchanged via them (ref. §2.1). However, there might be scenarios where E2E applications are not allowed within the censor's jurisdiction (only E2M apps are allowed), or the client wants to be extra cautious by additionally encrypting the exchanged content. In such scenarios, the client derives a shared key with the server.

For this, the client program needs the RSA public key (KS_{pub}) of the server along with the public key of its DH exponent (g^y). Since the Camoufler server is managed by the user himself (or by his trusted peer), these keys are assumed to be with the client program.

Similarly, the client would generate its DH private key x and eventually derive the shared key g^{xy}. When the client utility is run by the user, as a background process, it informs the server that it intends to encrypt its communication. Then the utility sends the public key of its DH exponent g^x encrypted with the RSA public key of the server KS_{pub}.

The server extracts g^x by decrypting it using its RSA private key (KS_{priv}) and derives the shared key g^{xy} using its private part of DH exponent y. Once the key is derived, both the parties derive a hash of g^{xy} (using SHA-256), and use these resulting hashed bits to encrypt the subsequent messages exchanged between them using AES-256. The corresponding DH private keys x and y along with the derived key g^{xy} are then deleted to ensure perfect forward secrecy.

4 EVALUATION

We now evaluate the performance of Camaoufler using our prototype implementation on several IM apps *viz.*, Signal, Skype, Slack, Telegram and Whatsapp. It must be noted that Camoufler has a general architecture which can be implemented on any IM app.

Time to access Alexa top-1k websites: In the first experiment, we assessed the time, Camoufler takes to access different websites. Thus, we downloaded the default webpages of the Alexa top 1000 websites and recorded the time it took Camoufler to complete this operation. The Camoufler client and server were geographically apart by a distance of ≈ 8300 miles. The Camoufler client was running on a machine hosted in our university, whereas the server was running on a cloud hosting service. We performed this experiment for all the five IM apps. The results (in the form of a box plot for each IM app), are represented in Fig. 4. As evident from the figure,

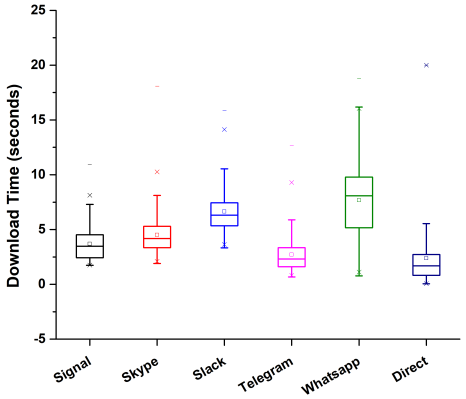

Figure 4: Download time of Alexa top-1k websites using different IM apps and its comparison with direct downloads.

we were able to access most of the websites in a few seconds that is comparable to direct download time. *E.g.,* for Signal, we recorded a median download time of 3.6s with the average being 4.1s. Similarly, using Telegram we were able to access these websites with an average time of 2.7s and median time of 2.3s. Time taken by Whatsapp was higher (average of 7.6s) compared to other IM apps, as it was automated using `Selenium` web automation framework. The details can be found in Appendix. A.4. However, it must be noted that the performance obtained by Camoufler using Whatsapp was still better than most of the existing systems such as Covercast [50], Deltashaper [24], *etc.* which incur an overhead of more than 10s for similar operations.

Time to first byte: This experiment was conducted to test the responsiveness efficiency of Camoufler server. We record the time it took for the first byte of the content to reach the Camoufler client (from the Camoufler server), after it sent the initial request.

For this experiment we accessed the Alexa popular 10 websites (100 times each) and plot the CDF of the results obtained for the Telegram app in Fig. 5. As evident from the graph, most of the websites (in over 90% of trials) were able to receive their first byte in less than 2s with half of the websites retrieving it under 1s. We obtained similar results for other apps. The details of the individual apps can be referred in Appendix. A.1.

Transmission time of messages from Camoufler client to server: Next, we evaluated the time spent for sending request from Camoufler client to Camoufler server via IM platforms (analogous to one way delay) using the aforementioned setup. This provides a

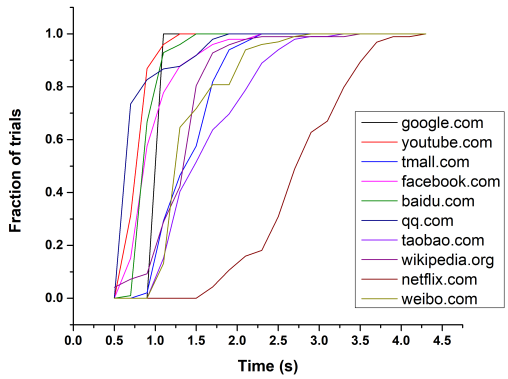

Figure 5: CDF of Time To First Byte (TTFB) for 10 popular Alexa websites (each downloaded 100 times).

measure of the end-to-end latency incurred due to the underlying IM platforms in transferring messages. Thus, in this experiment, we sent the same web requests via different IM platforms (100 times each), and recorded the time taken in receiving them at the other end. As evident from Fig. 6, in majority of the cases, the apps take less than a second to transfer the content in one direction. The inherent low overhead of the IM platforms is very helpful in providing good QoS for the clients, and is reflected in our results.

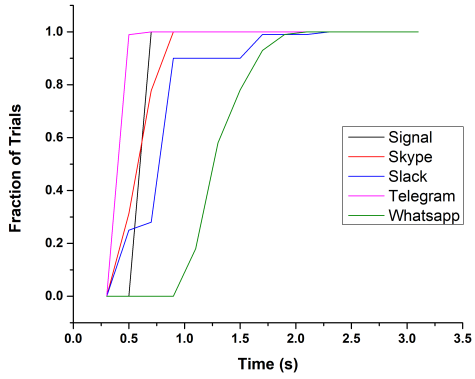

Figure 6: CDF of time taken by a message to travel from Camoufler client to Camoufler server for different IM apps.

Location diversity: In this set of experiments we varied the location of both the Camoufler client and the server to analyze if there was any significant change in performance. We varied the client across six locations, and the server across three (each in America, Europe and Asia). Alexa top-1k websites were downloaded for each of the 18 (6x3) client-server pairs. The result for the server in Asia (Singapore) is depicted in Fig. 7. It is evident from the box plot that there was not much variation when the client location was varied. The trend remained similar for other server locations as well (ref. Appendix. A.2).

Bulk downloads: Camoufler also supports bulk content transfer by transmitting the large files as compressed attachments. The Camoufler server first downloads the requested content, compresses it and then transfers it back to the client. In our tests, Camoufler

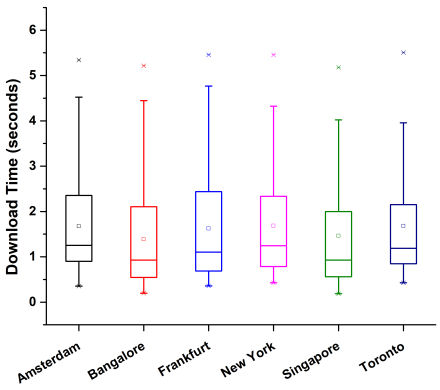

Figure 7: Box plot depicting download time of Alexa top-1k websites for varying client locations. The Camoufler server was hosted in Singapore.

client successfully downloaded files of various sizes (10, 20,...,100 MB) that were hosted on cloud servers. A sample of our results depicting download time variation is summarized in Tab. 1. Using

Downloaded	Time (in s)						
Using	10 MB	20 MB	30 MB	40 MB	50 MB	75 MB	100 MB
Direct (Wget)	7.9	15	23	29	35	51	68
Camoufler	13.6	23.5	34.6	45.3	52.1	77.2	93.3

Table 1: Large File Downloads: Comparison of download times of Camoufler and Wget.

Telegram as our underlying IM channel, we downloaded each file five times. Across different measurements we observed that download time with Camoufler is higher when compared to download time with `wget`. This is because, Camoufler server first downloads the complete file (at its end) and then sends it to the Camoufler client. However, large file downloads are generally delay tolerant and thus we believe this additional delay could be acceptable by the Camoufler clients. Overall, we observed a similar performance with other IM channels as well.

4.1 Implementation Details

We now describe the details of the proof of concept implementation of Camoufler. As an example we describe the implementation details on the `Signal` messenger platform as it is very popular among security and privacy practitioners. However, we similarly implemented Camoufler on other platforms as well, including Whatsapp, Telegram, Skype and Slack *etc.* The details of their implementation can be referred in Appendix A.4.

Camoufler Client: The client implementation was performed on a Linux host running Ubuntu 18.04 LTS, consisting of a 4 GHz processor, and provisioned with 8GB of RAM. The client's browser was configured to forward its requests to local port 8000 where LP listens for Camoufler requests. LP is written in python and uses the `sockets` API [7] to manage connections to and from the browser.

CE comprises of scripts written using python and shell-scripting. It interacts with the underlying `Signal` messenger to send and receive messages, using the `signal-cli` [8] interface. `signal-cli` helps automate exchanging messages (using CLI commands), over the signal messenger. Thus, CE uses it to craft and send blocked websites request to the SE, using the `send -m` command of `signal-cli`. Secondly, CE use the daemon mode and the `dbus` feature, shipped with the `signal-cli` interface to listen for incoming messages from SE. The dbus feature allows applications to create a listener which can easily receive and process different events that are generated when the signal app receives messages. `Pydbus` [5] was used to interact with the dbus interface. On receiving the response, CE decompresses and decrypts it using the `gzip` [6] and `Crypto` [4] libraries in python, before forwarding it to the LP. The compression and decompression process are lossless.

Camoufler server: Camoufler server was also implemented on a Linux machine running Ubuntu 18.04 LTS OS with 4 GHz processor and 8 GB RAM. The source code was also written in python and shell-scripting, and similar to CE it utilizes `pydbus` [5], `gzip`, and `Crypto` library for performing various tasks.

SE also utilizes the `signal-cli` interface and accesses the dbus feature for processing incoming messages. This processing enables SE to extract the censored website. The CP connects to the blocked website using python `socktes` API to retrieve responses. The SE compresses and encrypts the responses using `gzip` and `Crypto` library before sending it back to CE (using the `signal-cli` interface).

5 SECURITY ANALYSIS

We now describe various attacks the censor might attempt to block access to Camoufler.

5.1 Traffic Analysis

Censors may attempt to analyze Camoufler client's traffic to identify distinguishing features and block them from using Camoufler. As already described in our threat model, the censor can inspect the encrypted Camoufler client content within its network boundaries.

It must be noted that the functionality of all IM apps (with respect to their traffic characteristics) are very similar [60]. Thus the proposed attacks (and defenses) discussed subsequently are applicable across all IM platforms. We now enlist the possible attacks.

On traffic patterns: Camoufler involves downloading and accessing blocked content by exchanging IM messages. As analyzed previously (*e.g.,* by authors of SWEET [43] and Mailet [47] *etc.*), an adversary could attempt to distinguish regular IM traffic from IM flows that transport Camoufler traffic. Such attacks work on the premise that the behaviour of the tunneling/encapsulating protocol could be different when used with and without anti-censorship schemes. For instance, if there are differences in the packet exchange rates between a regular IM client and a Camoufler client, then it could be used to disambiguate the two. On one hand, it is already known that, IM clients (other than chatting) exchange significant amount of multimedia content [20]. In a recent study [26], authors reported that above 50% of the messages exchanged over IM applications constituted multimedia content. However, on the other hand, Camoufler clients would mostly fetch blocked content (*e.g.,* websites). Thus a determined adversary may attempt to differentiate web content downloaded (using Camoufler) from multimedia content downloaded (using standard IM apps). But, since

the underlying traffic (both of regular IM and Camoufler) is encrypted, it is plausible that adversary may opt of for traffic analysis based on differences in packet exchange rate and packet sizes [43].

Thus to observe such differences, we performed tests involving regular IM clients accessing multimedia content and Camoufler clients accessing websites. In our experimental setup, we used one machine (located in our lab) for running both regular IM and Camoufler clients. This machine communicated with another one (located in a different country) that ran an IM client (the peer) and the Camoufler server. We began by measuring the packet exchanged rate when (1) multimedia content was downloaded by a regular IM client and (2) web content was downloaded by using Camoufler. Fig. 8 depicts the scenario when we downloaded a PDF file, a GIF animation, an image and a video clip using regular IM app. Further, we downloaded the webpages from `cnn.com`, `youtube.com` and `github.com`, and a MS-Word document (.DOC) file using Camoufler. It is evident from the figure that, there is a sudden rise (spike) in packet exchange rate when multimedia content is shared between IM clients. This is because, multimedia attachments (*e.g.,* large video) involves a lot of content being transferred. Thus, underlying IM applications send data (packets) at a faster rate resulting in a spike in packet exchange rate.

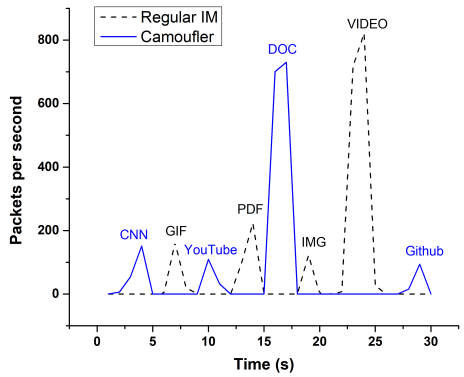

Figure 8: Packets exchange rate of regular IM client accessing multimedia content (images, GIF animation, video *etc.*) vs a Camoufler client accessing websites (`cnn`, `github` *etc.*) and a doc file. As evident, traffic characteristics of both regular IM and Camoufler are very similar.

It must be noted that, depending upon the size of the multimedia object being shared over the IM channel, one could expect spikes in packet exchange rate. For a large object (*e.g.,* a video of size 1 MB) the spike would be very high when compared to a smaller object (*e.g.,* an image of size 100 KB). This trend hold good for Camoufler traffic as well, as it uses the same IM app for transferring the content. For instance, when we downloaded a 1.5 MB video using regular IM client, we observed packet exchange rate peaked at 800 packets per second (ref. Fig. 8). Further, on downloading a 1.3 MB document using Camoufler, we observed a similar packet exchange rate *i.e.,* more than 700 packets per second. This trend holds good for smaller size files, websites and multimedia objects as well. Our experiment thus indicates that it is hard to differentiate Camoufler traffic from the regular IM traffic.

Further, we also plotted the packet size distribution for the above set of experiments for multimedia content (exchanged by regular IM clients) and websites (accessed by Camoufler clients). It is evident from the histogram presented in Fig. 9, that the maximum number of packets are clustered in two bins, the first bin with less than 100 byte packets, and the second bin with more than 1200 byte packets. The former corresponds to mostly the acknowledgement packets generated from the regular IM (or Camoufler) client and the latter corresponds to the data packets sent by the other IM client (or the Camoufler server). Overall, it is evident that depending on the size of the content, the number of data packets vary; large content download would result in large number of larger sized data packets (over 1200 bytes) while smaller content download would result in fewer bigger size data packets. For instance, the video downloaded by the regular IM client resulted in about 1200 data packets (over 1200 bytes each). In contrast, the image download resulted in only about 100 data packets of comparable sizes. The trend was very similar for Camoufler client as well, a document download resulted in 1000 data packets, whereas accessing website such as `cnn.com` resulted in only around 150 data packets.

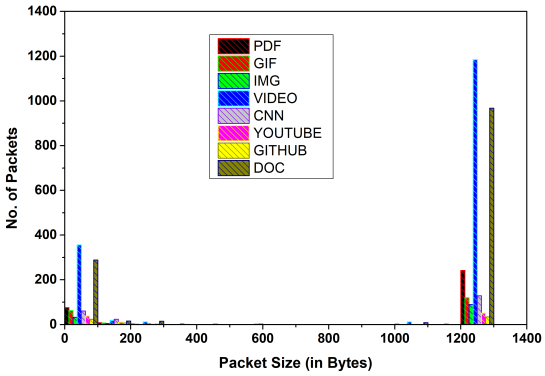

Figure 9: Histogram of packet sizes when regular IM client accessing multimedia content and when they use Camoufler to access websites.

This is further highlighted in Fig. 10. Large content download would result in large number of bigger size packets (over 1200 bytes) irrespective of the type of client (regular IM/Camoufler) and the content type (video/document *etc.*). *E.g.,* the box plot for a video download (by a regular IM client) and a document download (by a Camoufler client) are very similar — packet size distribution mostly consisting of more than 1200 bytes packets. Similarly, the box plot for smaller objects (like image/GIF) download by a regular IM client looks very similar to the website downloaded using Camoufler. However, the box plots in this case have more spread; packet size distribution consist of a fewer bigger size data packets[3].

Additionally, we performed multiple such experiments to further strengthen our claims. We downloaded different multimedia objects using regular IM clients as well as web content using Camoufler, and measured the packet exchange rates and packet sizes. Across all

[3]Additionally, if the average size of multimedia objects being exchanged over IM platforms differ from the average webpage sizes, the censor may attempt to distinguish them. However, in such a scenario, Camoufler can easily add cover traffic (by padding extra bytes) to even out the differences.

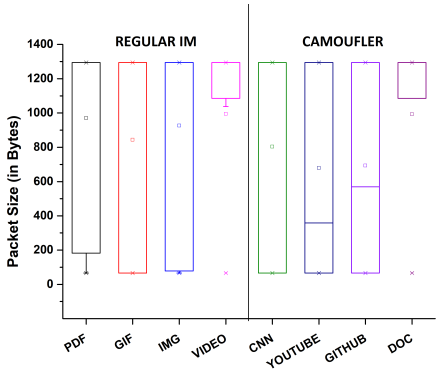

Figure 10: Box plot of observed packet sizes when regular IM client accessing multimedia content and when they use Camoufler to access websites.

our tests, the observations remain consistent, *i.e.,* the traffic characteristics of a web download using Camoufler is akin to multimedia download using regular IM apps (ref. App. A.5 for details).

However, it could still be argued that our observations would hold good only if regular IM clients often download multimedia content. Otherwise, the spikes in packet exchange rate, or a packet size distribution consisting of large number of bigger size packets, could be a uniquely identifiable characteristics of Camoufler. To that end, recent studies [19] suggest that IM clients very often exchange multimedia content. For instance, in [26] authors report that more than 50% of the messages exchanged over IM platforms constitute multimedia objects. These would also result in spikes in packet exchange rate and also the transmission of bigger sized packets, by regular IM clients. Hence, overall we believe that any attempts by a wire-sniffing adversary to distinguish Camoufler clients from regular IM clients could lead to high false positives.

Inducing traffic perturbations: Next, an adversary may attempt to identify Camoufler flows by actively dropping, delaying, modifying packets in some connections so as to see if Camoufler and regular IM clients behaved differently to compensate for such perturbations. However, it must be noted that, Camoufler is not mimicry based, rather a tunnelling system. It does not "pretend" to use the IM apps, rather it uses the underlying IM channel without modifying its default behavior. Therefore, such analysis would not provide any observable changes in the packet level features of the IM apps being used by Camoufler. The IM channel would continue to respond in exactly the same way to perturbations such as drops and modifications, regardless of whether they are used with Camoufler or not.

5.2 Other Attacks

Collusion attacks: The adversary can attempt to coerce, and thus collude, with the IM service provider which would enable it to access much more information than it could normally obtain by analyzing merely the encrypted traffic. This information could further be used for identifying and blocking Camoufler clients. Thus, there could be two possibilities:

(1) *Collusion with an end-to-end encrypted IM provider:* In this scenario, the IM provider, and thus the adversary, would not be able to inspect the content of IM messages as they are end-to-end encrypted (ref. §2.1). The lack of plain-text messages would hinder the adversary from obtaining any identifying information, such as the kind of content being transported.
Further, the censor could observe the metadata of messages from the IM peers, *e.g.,* their IM IDs. However, since the Camoufler server's ID is not publicly known (ref. §3.2), the censor would not be able to differentiate it from regular IM IDs.

(2) *Collusion with end-to-middle IM providers:* In end-to-middle IM applications, an encrypted channel is established between the client and the IM providers. Thus, if the adversary colludes with the IM provider, it would be able to inspect IM clients' content in plain text. However, Camoufler client derives a shared secret with the Camoufler server (using the scheme described in Subsec. 3.2.1), and uses that to encrypt the messages. This would not allow the censor or the IM provider to inspect the plain-text traffic and thus they could not attempt to identify clients by filtering requests seeking censored URLs.
However, in extreme cases the adversary could attempt to identify and drop all encrypted IM messages, to disrupt Camoufler. In such a scenario, we could use stenographic techniques [65] to hide our content from the adversary in plain sight, as also assumed in other anti-censorship systems [43, 50]. This may reduce the overall QoS and thus could be seen as a trade-off between unobservability and QoS.

Identifying Camoufler servers: An adversary may attempt to identify Camoufler servers' IM IDs, after which he/she may attempt to censor it. If the adversary owns the IM platform it could simply filter the IDs by itself, otherwise it may coerce the IM provider to block the said IDs. However, as already mentioned, Camoufler servers' IM IDs are not publicly known — either a Camoufler client would run its own Camoufler server in some hosting service or would request someone trusted to run the Camoufler server utility. Thus, it is extremely hard for an adversary to determine the Camoufler servers' IM IDs. Additionally, a determined adversary may further attempt to actively probe different IM IDs (on all IM platforms) by pretending to be a Camoufler client. Responses to such probes could lead to the detection of the Camoufler server. As a mitigation, the Camoufler server responds only to the trusted IM IDs. Thus, active reconnaissance by censors would be futile.

Long term user profiling: An adversary could attempt to longitudinally profile individual IM clients. Any deviations from the profiled behavior (such as sending messages at odd times of the day *etc.*) may evoke suspicion of use of a tunneling based system (including Camoufler). The success of such attacks would largely depend on accurately profiling clients *e.g.,* using some advanced machine learning techniques. Studying such attacks is an important part of our future work.

6 COMPARISON WITH PRIOR SYSTEMS

We now compare Camoufler with existing systems (described in Sec. 2.2) based on different features that circumvention schemes strive to provide. We compared Camoufler with existing tunnelling systems (*e.g.* SWEET, CloudTransport *etc.*), and also with other anti-censorship systems (*e.g.* Proxy based system, Decoy Routing *etc.*). A summarization of this comparison is done in Tab. 2 and Tab. 3 respectively.

Properties	Camoufler	SWEET	CloudTransport	Facet	CovertCast	Mailet	DeltaShaper	Rook	Protozoa	Freewave
Blocking Resistance	●	●	●	●	●	●	●	●	●	●
Deployment ease	●	●	●	●	●	●	●	●	●	●
No Cost of operation	◑	●	○	●	●	●	●	◑	◑	●
Requisite QoS	●	○	○	○	○	○	○	○	◑	○
Collusion Defense	●	●	●	○	○	○	○	○	●	○

Table 2: A comparison of different features of existing tunnelling based anti-censorship schemes. ● - feature supported, ○ - feature unsupported, ◑ - feature partially supported.

Properties	Camoufler	Proxy Systems	Domain Fronting	Decoy Routing	MassBrowser	Cache Browser	Conjure
Blocking Resistance	●	○	●	●	●	●	●
Deployment ease	●	●	●	○	●	○	○
No Cost of operation	◑	◑	○	○	○	○	○
Requisite QoS	●	◑	◑	◑	●	●	●
Collusion Defense	●	○	○	○	○	●	○

Table 3: A comparison of different features of other existing anti-censorship schemes. ● - feature supported, ○- feature unsupported, ◑ - feature partially supported.

- **Blocking Resistance**
  Tunnelling Systems: All such systems rely on using some underlying channel to covertly transport censored contents. Attempts to block the channel often incurs collateral damages to the adversary itself. Thus, all these systems, including Camoufler provide blocking resistance, against adversaries that attempt to censor the entire communication channel (*e.g.,* blocking all email services, IM services *etc.*).
  Other Systems: Most of the current promising systems also provide adequate blocking resistance by using the same principle of incurring collateral damage to the adversary. *E.g.,* decoy routing requires the adversary to change nation-wide routing policies in order to prevent users from accessing the system. However, proxy based systems are relatively easier to block as the adversary merely needs to filter traffic destined to their IP addresses, incurring no collateral damages.
- **Deployment Ease**
  Tunnelling Systems: Camoufler, like most tunnelling based systems, require installing programs at the client and server ends. Apart from that, there are no additional requirements.
  Other Systems: Decoy routing systems require collaboration from the ISPs, and thus poses a hurdle for deployment. Similarly, CacheBrowser relies on content publishers to host their content on some CDNs, thereby posing deployment challenges[4]. Similar to Decoy Routing, Conjure requires ISPs assistance to install multiple servers with taps having access to all content transiting the ISP and thus pose challenges for deployment. Apart from these, other systems do not pose much hurdle for deployment.
- **No Cost of Operation**
  Tunnelling Systems: Except for CloudTransport, tunnelling based systems do not incur any upfront cost to the users, or to the content providers. CloudTransport requires hosting servers (*e.g.,* Amazon s3), incurring monetary operational costs. Moreover, Rook uses games to tunnel content, and thus may incur subscription charges if paid games are used. Camoufler and Protozoa do not incur any cost when the client has a peer to run the server end. Otherwise, if the client decides to run the server on a cloud host, it would incur hosting charges.
  Other Systems: Domain fronting, Decoy routing and MassBrowser incur a monetary cost for their functioning. While Massbrowser and Domain Fronting requires subscription for services like `Google App Engine`, `Amazon CloudFront` *etc.,* most Decoy Routing proposals (including Conjure) ideally require ISPs to change or add in their existing network routing infrastructure. CacheBrowser requires that the censored content is hosted on CDNs, thereby incurring periodic subscription charges. Some proxy based systems such as VPNs *etc.* also require subscription costs.
- **Requisite QoS**
  Tunnelling Systems: Most tunnelling systems do not provide requisite QoS due to the limitation of the underlying channel they use for exchanging content. *E.g.,* freewave uses VoIP, which encodes and transports data at low bit rates (19 Kbps), insufficient for providing requisite QoS for applications like web browsing. Similarly, Facet, CovertCast, Protozoa and DeltaShaper use video streams to encode censored contents. However, factors like lossy video encoding of the underlying platforms, result in unsuitable QoS for web browsing. *E.g.* Mcpherson *et. al.* [50] report that loading BBC news homepage along with three articles takes almost 120s. Further, DeltaShaper only provides an effective bandwidth of 2.56 Kbps [24]. Protozoa attempts to improve upon the

[4]Non-CDN (blocked) websites can not be accessed by the CacheBrowser.

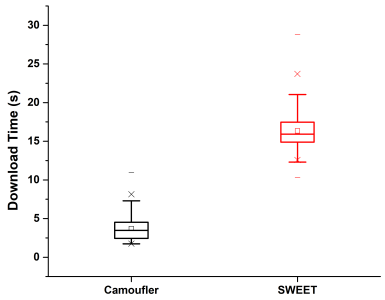

Figure 11: Camoufler vs Sweet: Download time of Alexa top-1k websites.

performance, but is limited by the unreliable nature of video streams and the video quality offered. Using a 480p video stream, it achieves 250 Kbps throughput and is further shown to achieve a maximum download rate of ≈ 1.5 Mbps. Mailet and SWEET rely on emails which can carry significant content but inherently involve substantial delays (when emails transit email servers). However, Camoufler due to its usage of IM applications (which involve minimum transit delay and substantial content carrying capacity) provides good QoS with significant improvement in comparison to other systems such as SWEET[5] (ref. Fig. 11). CloudTransport traffic also transits multiple hops (cloud provider, cloud bridge *etc.*), incurring delays and reducing the overall QoS. *Other Systems:* In general, all existing systems attempt to provide acceptable QoS. However, Decoy Routing systems do not provide adequate QoS guarantees [34, 35] (barring a few like [57], which provide throughput comparable to direct TCP downloads). Similarly, Domain Fronting incurs additional latency due to its functional overheads [33]. Lastly, popular Proxy systems such as Tor may sometimes incur substantial delays due to the selection of low bandwidth relays.

- **Collusion Defense**
  Tunnelling Systems: It is generally difficult to safeguard against a covert channel application provider who could collude with the censor to help identify clients. The CloudTransport architecture ensure that the cloud provider has no information about the destinations the client visits, even when the cloud provider colludes with the censor. SWEET attempts to resists collusion by distributing unique email IDs to individual users. In Camoufler, the Camoufler server IM IDs are known only to the specific clients. This makes it difficult for the adversary (that colludes with the IM provider) to block traffic by observing destination IDs in clients' requests. Protozoa, also distributes private ID and password among the peers using an out-of-band channel, mitigating the impact of collusion.
  Other Systems: Only CacheBrowser has a way to defend against collusion, as it uses frontends located outside the censor's boundary to access censored content. Thus, even if the CDN provider agrees to filter content in the censor's country, it may not do so in foreign countries, due to its own business motivations and regulatory compliance. In proxy systems, the collusion of the VPN provider or proxy maintainer with the adversary, makes it relatively easy for the latter to identify clients. Similarly for Domain Fronting, the adversary by colluding with fronting service provider, could identify and block the Domain Fronting servers hosted on these services. Also, in Decoy Routing and Conjure, if the ISP colludes, then the respective systems would cease to function.

[5]We could not obtain working codes of CloudTrasport for comparison. However, SWEET's code was publicly available.

7 DISCUSSION AND FUTURE WORK

On blocking of IM apps
It can argued that an adversary may attempt to block IM apps, and in extreme case may block all existing IM apps to disrupt Camoufler. In such a scenario, like any other tunneling system, Camoufler may cease to function. *E.g.,* if apps like Skype are completely blocked by the censor, then circumvention solutions like DeltaShaper [24], Facet [48], Freewave [41] etc. would be disrupted.

But we believe that such a move by the censor could be prohibitively expensive leading to collateral damage as IM apps are extremely popular and have penetrated deeply into businesses and commercial spaces as well. However, it must be noted that, even if the censor allows only a single IM app to function, controlled by itself (ref. §5.2), Camoufler would continue to function.

A more rational approach that could be opted by the censor is to selectively block only the suspicious IM IDs. Thus, a detailed analysis of such threats is already presented in §5.2.

On scalability of Camoufler
As and when the popularity of Camoufler grows, the system has the potential to scale up for larger deployments. Camoufler uses IM apps as an overlay network to tunnel traffic. The ubiquity of IM applications is potentially beneficial for scaling Camoufler into a distributed system with a large user base. Similar to Tor, Camoufler volunteer could act as the Camoufler servers. Thus, an increase in the number of Camoufler clients could be handled by these distributed volunteer Camoufler servers. We forsee that in future the popularity of Camoufler may drive the recruitment of server hosters and maintainer, much like Tor.

Text vs attachment for transporting content
Camoufler has a choice of selecting how it transports content using the underlying IM channel *viz.* as text, or as an attachment. The Camoufler server initially used text messages to transport content. However, we discovered that in a few cases the contents of the websites accessed were truncated. Upon investigation, we found that IM applications generally restrict the volume of data that can be sent via a single text message. To overcome such restrictions, the Camoufler server compressed the content before sending it to the other end. Thereafter, almost all the websites could be accessed, without data being truncated.

Next, we also considered transporting our content as an attachment. This is because, with attachments we can transport more data as compared to text. However, we noticed a slight increase in the download time of Alexa top-1k websites, compared to when data was transported via texts (ref. Fig. 12).

Thus, our design combines the best of both approaches. By default, we used text to transport content due to its obvious performance advantage. However, for the few cases where text could not be used, we relied on using attachments. Alternatively, the content can be segmented into multiple chunks, such that the length of each

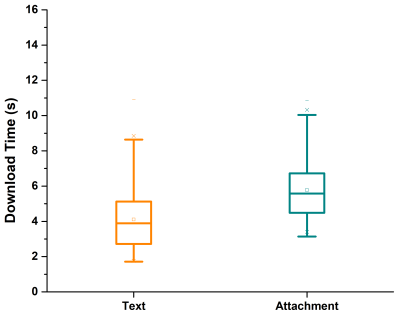

Figure 12: Download time of Alexa top-1k websites using Camoufler, when downloaded as text and as an attachment.

chunk is within the limits enforced on the length of text messages. However, unlike ours, this approach may incur extra round trips. and increase the overall delay.

8 CONCLUSION

In this paper we presented Camoufler, a new anti-censorship system to provide unhindered access to information over the Internet. Camoufler utilizes standard IM platforms (such as Whatsapp, Signal *etc.*), to tunnel and transport censored content, and thus attempts to make it difficult for the adversary to detect it. *Camoufler provides satisfactory performance, reliability, blocking resistance and deployment ease.* Using the prototype implementation of Camoufler on popular IM apps, we experimentally demonstrate that it provides acceptable performance for regular web browsing and bulk downloads. A detailed security analysis of Camoufler highlights that it is hard to be detected by an adversary (*e.g.,* an ISP working at the behest of an authoritative regime).

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their feedback. Additionally, we would like to thank Karan Tandon for helping out with some of the experiments.

REFERENCES

[1] Detailed latency analysis of Whatsapp. https://developers.facebook.com/docs/whatsapp/tips-and-tricks/send-message-performance/#performance.
[2] *Flock Messenger service.* https://flock.com/
[3] *How Slack Is Replacing Email in the Workplace.* https://tech.co/news/slack-replacing-email-workplace-2018-08
[4] Python Crypto Library. https://pycryptodome.readthedocs.io/en/latest/.
[5] Python Dbus Library. https://pydbus.readthedocs.io/en/latest/legacydocs.
[6] Python Gzip Library. https://docs.python.org/3/library/gzip.html.
[7] Python Sockets – Low-level networking interface. https://docs.python.org/3/library/socket.html.
[8] Signal-cli. https://github.com/AsamK/signal-cli.
[9] Skype Messaging API. https://skpy.t.allofti.me/.
[10] Slack Client API. https://python-slackclient.readthedocs.io/en/latest/.
[11] *Slack is killing Email.* https://www.theverge.com/2014/8/12/5991005/slack-is-killing-email-yes-really
[12] Slack Messenger service. https://slack.com/.
[13] Slack Real-Time Messaging API. https://python-slackclient.readthedocs.io/en/latest/real_time_messaging.html.
[14] *Slack vs Email.* https://slack.com/intl/en-in/why/slack-vs-email
[15] Telegram. https://telegram.com.
[16] Telethon Telegram API. https://docs.telethon.dev/en/latest/.
[17] Web Whatsapp automation. https://github.com/open-wa/wa-automate-python.
[18] WeChat. https://wechat.com/.
[19] Whatsapp. https://whatsapp.com/.
[20] Whatsapp Usage Statistics. https://www.businesstoday.in/technology/news/whatsapp-users-share-texts--photos-videos-daily/story/257230.html.
[21] xdotool: Linux GUI automation utility. http://manpages.ubuntu.com/manpages/trusty/man1/xdotool.1.html.
[22] Giuseppe Aceto and Antonio Pescapé. Internet censorship detection: A survey. *Computer Networks* 83 (2015), 381–421.
[23] Alberto Andujar. Benefits of mobile instant messaging to develop ESL writing. *System* 62 (2016), 63–76.
[24] Diogo Barradas, Nuno Santos, and Luís Rodrigues. DeltaShaper: Enabling unobservable censorship-resistant TCP tunneling over videoconferencing streams. *Proceedings on Privacy Enhancing Technologies* 2017, 4 (2017), 5–22.
[25] 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*. 35–48.
[26] Jason Baumgartner, Savvas Zannettou, Megan Squire, and Jeremy Blackburn. The Pushshift Telegram Dataset. In *Proceedings of the International AAAI Conference on Web and Social Media*, Vol. 14. 840–847.
[27] Cecylia Bocovich and Ian Goldberg. Slitheen: Perfectly Imitated Decoy Routing Through Traffic Replacement. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*.
[28] Chad Brubaker, Amir Houmansadr, and Vitaly Shmatikov. Cloudtransport: Using cloud storage for censorship-resistant networking. In *International Symposium on Privacy Enhancing Technologies Symposium*. Springer.
[29] Sam Burnett, Nick Feamster, and Santosh Vempala. Chipping Away at Censorship Firewalls with User-Generated Content. In *In Proceedings of USENIX Security Symposium*. 463–468.
[30] Abdelberi Chaabane, Terence Chen, Mathieu Cunche, Emiliano De Cristofaro, Arik Friedman, and Mohamed Ali Kaafar. Censorship in the wild: Analyzing Internet filtering in Syria. In *Proceedings of the 2014 Conference on Internet Measurement Conference*. 285–298.
[31] Roger Dingledine and Nick Mathewson. Design of a blocking-resistant anonymity system.
[32] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*.
[33] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies* 2015, 2 (2015), 46–64.
[34] Sergey Frolov, Fred Douglas, Will Scott, Allison McDonald, Benjamin VanderSloot, Rod Hynes, Adam Kruger, Michalis Kallitsis, David G Robinson, Steve Schultze, et al. An ISP-scale deployment of TapDance. In *In Proceedings of USENIX Workshop on Free and Open Communications on the Internet (FOCI 17)*.
[35] Sergey Frolov, Jack Wampler, Sze Chuen Tan, J Alex Halderman, Nikita Borisov, and Eric Wustrow. Conjure: Summoning Proxies from Unused Address Space. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2215–2229.
[36] Devashish Gosain, Anshika Agarwal, Sambuddho Chakravarty, and HB Acharya. The Devil's in The Details: Placing Decoy Routers in the Internet. In *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM, 577–589.
[37] 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)*. IEEE, 63–77.
[38] John Holowczak and Amir Houmansadr. Cachebrowser: Bypassing chinese censorship without proxies using cached content. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 70–83.
[39] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. The parrot is dead: Observing unobservable network communications. In *In proceedings of IEEE Symposium on Security and Privacy*. IEEE, 65–79.
[40] Amir Houmansadr, Giang T. K. Nguyen, Matthew Caesar, and Nikita Borisov. Cirripede: Circumvention Infrastructure using Router Redirection with Plausible Deniability. In *Proceedings of the 18th ACM conference on Computer and Communications Security (CCS 2011)*.
[41] Amir Houmansadr, Thomas J Riedl, Nikita Borisov, and Andrew C Singer. I want my voice to be heard: IP over Voice-over-IP for unobservable censorship circumvention.. In *In proceedings of Network and Distributed Systems Security (NDSS)*.
[42] Amir Houmansadr, Edmund L Wong, and Vitaly Shmatikov. No Direction Home: The True Cost of Routing Around Decoys.. In *In proceedings of Network and Distributed Systems Security (NDSS)*.
[43] Amir Houmansadr, Wenxuan Zhou, Matthew Caesar, and Nikita Borisov. Sweet: Serving the web by exploiting email tunnels. *IEEE/ACM Transactions on Networking* 25, 3 (2017), 1517–1527.
[44] Shengtuo Hu, Xiaobo Ma, Muhui Jiang, Xiapu Luo, and Man Ho Au. Autoflowleaker: Circumventing web censorship through automation services. In *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 214–223.
[45] Josh Karlin, Daniel Ellard, Alden W Jackson, Christine E Jones, Greg Lauer, David Mankins, and W Timothy Strayer. Decoy Routing: Toward Unblockable Internet Communication. In *Proceedings of USENIX Workshop on Free and Open*

Communication on the Internet (FOCI).
[46] Sheharbano Khattak, Tariq Elahi, Laurent Simon, Colleen M Swanson, Steven J Murdoch, and Ian Goldberg. Sok: Making sense of censorship resistance systems. *Proceedings on Privacy Enhancing Technologies* 2016, 4 (2016), 37–61.
[47] Shuai Li and Nicholas Hopper. Mailet: Instant social networking under censorship. *Proceedings on Privacy Enhancing Technologies* 2016, 2 (2016), 175–192.
[48] Shuai Li, Mike Schliep, and Nick Hopper. Facet: Streaming over videoconferencing for censorship circumvention. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. 163–172.
[49] Moxie Marlinspike and Trevor Perrin. The x3dh key agreement protocol. *Open Whisper Systems* (2016).
[50] Richard McPherson, Amir Houmansadr, and Vitaly Shmatikov. Covertcast: Using live streaming to evade internet censorship. *Proceedings on Privacy Enhancing Technologies* 2016, 3 (2016), 212–225.
[51] 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*. 97–108.
[52] Milad Nasr and Amir Houmansadr. GAME OF DECOYS: Optimal Decoy Routing Through Game Theory. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*.
[53] Milad Nasr, Hadi Zolfaghar, Amir Houmansadr, and Amirhossein Ghafari. MassBrowser: Unblocking the Censored Web for the Masses, by the Masses. In *Proceedings of Network and Distributed Systems Security (NDSS)*.
[54] Milad Nasr, Hadi Zolfaghari, and Amir Houmansadr. The Waterfall of Liberty: Decoy Routing Circumvention that Resists Routing Attacks. In *Proceedings of Network and Distributed Systems Security (NDSS)*.
[55] Lukasz Piwek and Adam Joinson. "What do they snapchat about?" Patterns of use in time-limited instant messaging service. *Computers in Human Behavior* 54 (2016), 358–367.
[56] Max Schuchard, John Geddes, Christopher Thompson, and Nicholas Hopper. Routing around decoys. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 85–96.
[57] Piyush Kumar Sharma, Devashish Gosain, Himanshu Sagar, Chaitanya Kumar, Aneesh Dogra, Vinayak Naik, HB Acharya, and Sambuddho Chakravarty. SiegeBreaker: An SDN Based Practical Decoy Routing System. *Proceedings on Privacy Enhancing Technologies* 3 (2020), 243–263.
[58] Vivian C Sheer and Ronald E Rice. Mobile instant messaging use and social capital: Direct and indirect associations with employee outcomes. *Information & Management* 54, 1 (2017), 90–102.
[59] Simon So. Mobile instant messaging support for teaching and learning in higher education. *The Internet and Higher Education* 31 (2016), 32–42.
[60] Ramin Soltani, Amir Houmansadr, Dennis Goeckel, and Don Towsley. Practical Traffic Analysis Attacks on Secure Messaging Applications. In *Proceedings of Network and Distributed Systems Security (NDSS) 2020*.
[61] Statista. Number of mobile phone messaging app users worldwide. https://www.statista.com/statistics/483255/number-of-mobile-messaging-users-worldwide/.
[62] Ying Tang and Khe Foon Hew. Is mobile instant messaging (MIM) useful in education? Examining its technological, pedagogical, and social affordances. *Educational Research Review* 21 (2017), 85–104.
[63] Michael Carl Tschantz, Sadia Afroz, Vern Paxson, et al. Sok: Towards grounding censorship circumvention in empiricism. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 914–933.
[64] Paul Vines and Tadayoshi Kohno. Rook: Using video games as a low-bandwidth censorship resistant communication platform. In *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society*. 75–84.
[65] 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*. 121–132.
[66] Zhongjie Wang, Yue Cao, Zhiyun Qian, Chengyu Song, and Srikanth V Krishnamurthy. Your state is not mine: a closer look at evading stateful internet censorship. In *Proceedings of the 2017 Internet Measurement Conference*. 114–127.
[67] Eric Wustrow, Colleen M Swanson, and J Alex Halderman. Tapdance: End-to-middle anticensorship without flow blocking. In *23rd USENIX Security Symposium (USENIX Security 14)*. 159–174.
[68] Eric Wustrow, Scott Wolchok, Ian Goldberg, and J Alex Halderman. Telex: Anticensorship in the Network Infrastructure.. In *20th USENIX Security Symposium (USENIX Security 11)*.
[69] Tarun Kumar Yadav, Akshat Sinha, Devashish Gosain, Piyush Kumar Sharma, and Sambuddho Chakravarty. Where the light gets in: Analyzing web censorship mechanisms in india. In *Proceedings of the Internet Measurement Conference 2018*.
[70] Hadi Zolfaghari and Amir Houmansadr. Practical censorship evasion leveraging content delivery networks. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1715–1726.

A APPENDIX

A.1 Time To First Byte when using Camoufler with different IM apps

In this subsection we present the TTFB of Signal, Skype, Slack and Whatsapp IM apps in Fig. 13, Fig. 14, Fig. 15 and Fig. 16 respectively. As evident from the graphs, majority of the websites' first byte was received withing 3s, across all IM apps. Except for Whatsapp, which as described in the previous subsection, was automated using selenium framework and this incurred extra latency, with most websites receiving content within 5s. Overall, the latency by Camoufler implementations on different platforms was satisfactory enough to support web browsing.

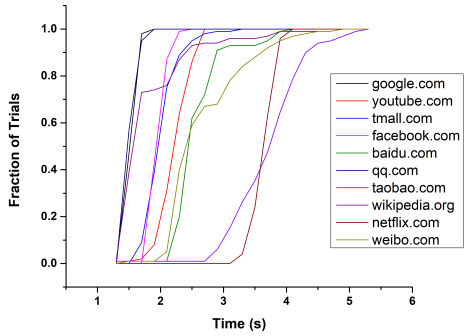

Figure 13: CDF of Time To First Byte (TTFB) for 10 popular Alexa websites (each downloaded 100 times) for Signal app.

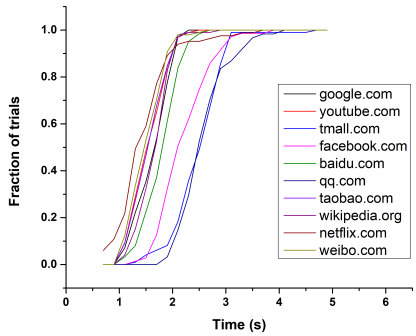

Figure 14: CDF of Time To First Byte (TTFB) for 10 popular Alexa websites (each downloaded 100 times) for Skype app.

A.2 Location Diversity Additional Results

We demonstrate the results for impact of location diversity in Fig. 17 and Fig. 18. It is evident from the figures that when server was placed at Amsterdam and San Francisco, clients at different location across the globe did not observe much variation in performance when accessing the Alexa top websites.

A.3 On SOCKS Implementation

The default implementation of Camoufler supports accessing web content. However, we have also implemented Camoufler with SOCKS support. SOCKS based implementation helps the client access any

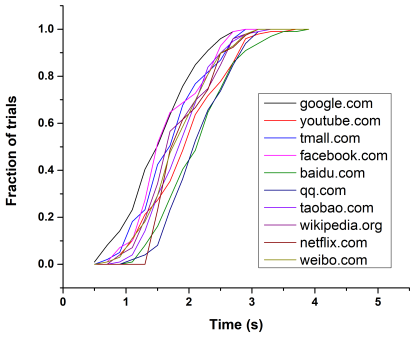

Figure 15: CDF of Time To First Byte (TTFB) for 10 popular Alexa websites (each downloaded 100 times) for Slack.

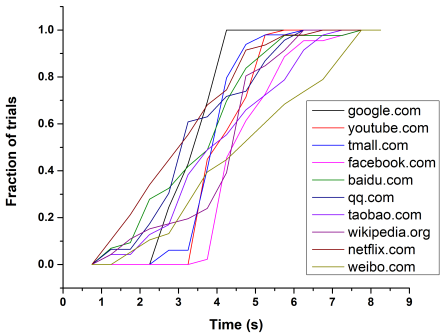

Figure 16: CDF of Time To First Byte (TTFB) for 10 popular Alexa websites (each downloaded 100 times) for Whatsapp.

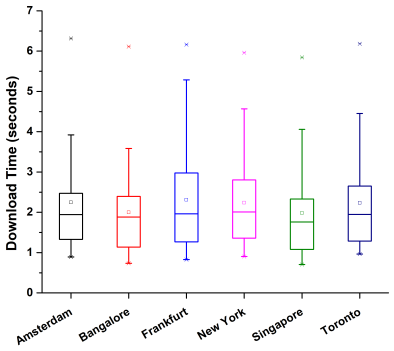

Figure 17: Box plot depicting download time (server in Amsterdam) of top Alexa-1K websites for varying client location.

TCP or UDP based protocol using Camoufler. Although, using SOCKS leads to a increase in download time by more than 1.5 times as compared to without SOCKS. Thus, if the goal is to access websites, then the default Camoufler implementation should be used. If other protocols needs to be accessed, then the SOCKS implementation can be used with a trade-off in performance.

A.4 Implementation details of Camoufler with different IM apps

We now describe how different IM applications could be used to transport traffic for Camoufler. The IM apps can be divided into

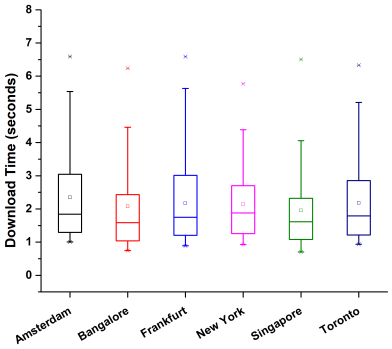

Figure 18: Box plot depicting download time (server in San Francisco) of top Alexa-1K websites for varying client location.

two categories with respect to implementation feasibility. First category applications are the ones which have a dedicated API readily available for public use (*e.g.,* Signal, Telegram, Slack, Skype and FB messenger). Second category contains apps which do not provide API for public use or the API is hard to procure (*e.g.,* Whatsapp, Wechat).

First category applications were automated utilizing their respective APIs, as they provide an interface to automate sending and receiving messages. Applications such as Signal and Telegram fall in this category. Similar to the signal implementation described in Subsec. 4.1 we automated Telegram using its python API [16]. Additionally, we automated slack using its `slackclient` utility [10] to send messages and its Real Time Messaging (RTM) API [13] to automate processing of received messages carrying blocked content. Similarly we used the *skpy* API [9] to automate Skype.

Second category apps could not be directly automated because of lack of APIs. We believe that, with the kind of penetration IM apps are having in businesses, building customized add-on applications over these IM apps would become more popular leading to public releases of the APIs for them. However, in the meantime, we devised approaches that could be used to automate such apps. The basic idea is to automate the GUI of such apps to achieve sending and receiving of messages. One way is to use selenium web automation framework to accomplish this task. Thus, we automated Whatsapp using its actively maintained selenium based API [17]. Similar approach can be used to automate other apps which provide a web based interface to send/receive messages.

However, selenium based automation requires regular maintenance as the HTML objects and their IDs (HTML class ID, table ID *etc.*), required for identifying individual elements (such as user chat, message send box *etc.*) are regularly updated by the IM app maintainers. Moreover, the approach would not work for apps who do not provide web based interface but rather a program binary to be run on desktop systems (*e.g.,* Wechat). Thus, to reduce the regular maintenance requirement, we used an alternate approach *i.e.,* we automated the GUI using desktop GUI automation utilities such as *xdotool* [21]. `xdotool` can be easily used to automate typing, clicking, copy paste, move mouse to a specific pixel on the app window *etc.*

We developed a framework, where at the client engine, we send the blocked content request message by clicking and typing the

request using `xdotool` in the app GUI. Similarly, The Camoufler server keeps polling for new message at a regular time interval in the app GUI. On receiving the content request, SE uses steps as described in Camoufler design to retrieve censored content. On receiving this content, we follow the same procedure to copy and paste this content in the response text box in the chat window . Finally, we click at the send button. On the Camoufler client's end, CE keeps polling for a response in the Camoufler servers' chat window, and on receiving one, processes it and sends to LP which forwards it to the browser for rendering. We could easily replicate similar process for all the apps whose APIs are not available or who do not provide web interface.

Thus, using all the approaches described above, we could automate roughly all the popular IM apps that are currently available.

A.5 On Traffic Analysis

As already described in §5.1, it is extremely difficult to disambiguate Camoufler traffic from regular IM traffic. We performed some additional experiments to strengthen our claims. As our first set of experiments (using our experimental setup described in §5.1), we downloaded different multimedia objects using regular IM client and a website using Camoufer. The size of multimedia objects and the webpage was roughly the same *i.e.,* around 300 KB. As evident from Fig. 19 and 20, packet exchange rate as well as the packet size distribution of all are very similar. Thus it is difficult to differentiate Camoufler traffic from regular the IM. This also indicates that, irrespective of the type of file being downloaded, the packet exchange rate and packet size distribution primarily depends on the file size.

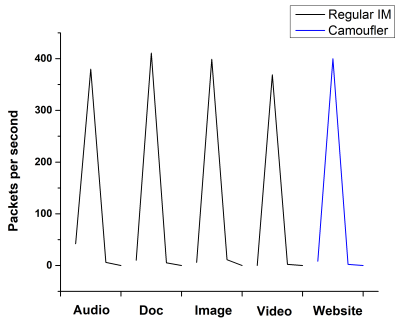

Figure 19: Same size (300 KB) object download: Packet exchange rate for a webpage download (using Camoufler) is very similar to multimedia download using regular IM client (irrespective of the type of multimedia content).

To further establish that Camoufler and regular IM apps result in very similar traffic characteristics, we conducted the second set of experiments. We downloaded the the same set of multimedia objects using regular IM clients as well as Camoufler. It is evident from Fig. 21, that rate of packets exchanged of Camoufler is akin to regular IM client. It establishes that Camoufler does not alter any underlying behavior of the IM channel.

Finally to establish that large size content download would result in generation of high packet exchange rate, we downloaded three images (with increasing sizes) using regular IM client (and Camoufler). It can easily inferred from Fig. 22, large size image result in large number of packets exchanged per second. Download of image

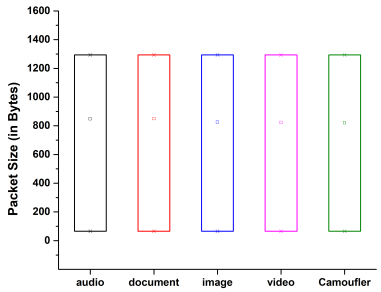

Figure 20: Same size (300 KB) object download: Packet size distribution for a webpage download (using Camoufler) is very similar to multimedia download using regular IM.

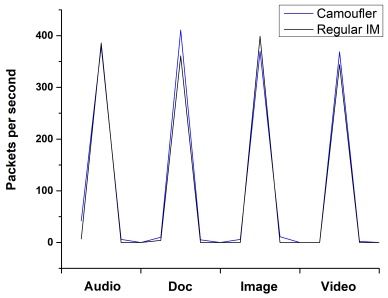

Figure 21: Same object download using Camoufler and regular IM: Packet exchange rate for objects when downloaded using Camoufler and regular IM is almost identical.

of size 1 MB resulted in more than 1000 packets per second, whereas 200 KB image download resulted in ≈ 300 packets per second.

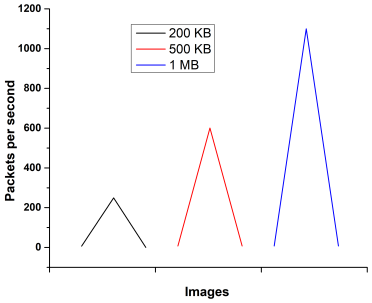

Figure 22: Variable size images download: Packet exchange rate increases with increase in the image size.

We repeated the experiments multiple times, varying the multimedia objects and the websites. Across all our experiments, our findings were consistent. (1) Since Camoufler use the underlying IM as-is, traffic footprints of its traffic are very similar to regular IM app. (2) Irrespective of the medium (Camoufler/Regular IM app) and the object being download (video, audio, document, image, or a website *etc.*), the packet exchange rate and packet size distribution only depends on the size of the object being downloaded.