Building Performant, Privacy-Enhancing, and Blocking-Resistant Communication Systems

By

Piyush Kumar

Under the supervision of Sambuddho Chakravarty and Mukulika Maity

Computer Science and Engineering

Indraprastha Institute of Information Technology Delhi

New Delhi– 110020

August, 2021

Building Performant, Privacy-Enhancing, and Blocking-Resistant Communication Systems

By

Piyush Kumar

A Thesis

submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Computer Science and Engineering

Indraprastha Institute of Information Technology Delhi

New Delhi– 110020

August, 2021

Thesis Evaluation Committee

- Dr. Amir Houmansadr, Associate Professor, University of Massachusetts
- Dr. Michalis Polychronakis, Associate Professor, Stony Brook University
- Dr. Kent Seamons, Professor, Brigham Young University

Certificate

This is to certify that the thesis titled *Building Performant, Privacy-Enhancing, and Blocking-Resistant Communication Systems* being submitted by *Piyush Kumar* to the Indraprastha Institute of Information Technology Delhi, for the award of the degree of Doctor of Philosophy, is an original research work carried out by him under our supervision. In our opinion, the thesis has reached the standard fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree or diploma.

August, 2021

Dr. Sambuddho Chakravarty and Dr. Mukulika Maity

Indraprastha Institute of Information Technology Delhi

New Delhi 110020

Abstract

Free and open communication over the Internet is essential for the overall advancement of modern societies. However, there are numerous ways with which malevolent adversaries try to control the flow of information, by either selectively restricting access to content or in extreme scenarios completely shutting down the Internet. Hence, we aim to augment existing research as well as build novel systems to facilitate open communication over the Internet. Thus, in this thesis, we attempt to answer the question — Can we facilitate open access to Internet by building privacy-enhancing technologies that can also provide blocking resistance, good QoS and deployability? To that end, we propose and build solutions for three broad categories of applications. **(i)** applications for which there does not exist any functional and deployed solutions (*e.g.,* anonymous calling over Internet) **(ii)** applications (*e.g.,* accessing censored websites) for which there are functional solutions, but those have great scope for improvement (*e.g.,* blocking resistance, performance) **(iii)** accessing lightweight Internet applications during events of Internet shutdowns (*e.g.,* posting tweet and retrieving news snippets *etc.*).

Firstly, we explore the feasibility of anonymous voice communication over the Internet as it is of great interest to whistleblowers, privacy practitioners *etc.* Current literature sidesteps existing anonymity systems such as Tor for voice calling, citing the performance issues that would render real-time voice calling unusable over Tor. Thus, novel systems tailored for voice have been proposed (*e.g.,* Herd). However, the newly proposed systems are not functional, and thus there does not currently exist any usable anonymous voice calling system. Hence, we revisit Tor and perform a comprehensive analysis of performing voice calling over it. With the help of nearly half a million voice calls performed over the real Tor network over a year under various setups, we establish that it is indeed possible to perform anonymous voice calls using Tor.

Secondly, we look at accessing censored content (*e.g.,* websites, files *etc.*) over the Internet. There already exist solutions such as VPNs, Tor *etc.*, that help accessing restricted content by routing it through single (or multiple) proxy nodes. The censor blocks such proxies as soon as they are discovered, while the researchers attempt to build systems to evade such blocking, effectively leading to an arms race. Decoy Routing (DR) is an approach to potentially break this arms race, where a network router (known as the decoy router) doubles up as a proxy, instead of the end hosts. This makes it extremely difficult for the adversary to censor, as it can no longer block based on end hosts IP, but require blocking routers which carry a significant amount of innocuous traffic.

However, existing DR solutions use commodity servers to act as decoy routers, as traditional routers cannot be programmed to perform the complex operations required for proxying connections. This leads to (1) poor performance for end users due to handling of ISP scale flows by commodity servers, and (2) privacy violations of oblivious users (non-DR clients) due

to the analysis of ISP flows by third party DR maintainers. We thus propose *SiegeBreaker*, which overcomes the aforementioned problems (while also providing other salient features) by building a DR protocol with the help of software defined network (SDN) devices. We show that using SDN provides the essential modularity required in DR designs, eventually leading to good performance along with flexibility in trust relationships for providing better privacy guarantees. We implemented and deployed SiegeBreaker using hardware SDN switches and show with the help of extensive evaluation that SiegeBreaker provides performance comparable to direct TCP downloads. However, despite being a promising solution, DR poses deployment challenges as it relies on the ISPs support to help place the DR infrastructure. Thus, on one hand we have solutions that are readily available (VPNs, Proxies *etc.*) but are easy to block, and on the other hand we have solutions that are hard to block but face deployment challenges (DR).

Thus, next, we attempt to build a solution which is readily available to end users and at the same time provides effective blocking resistance without compromising the performance. To that end we propose a new system *Camoufler* that utilizes the Instant Messaging (IM) apps to transfer censored content. Using IM makes the system immediately usable as these apps are an integral part of the Internet ecosystem. Moreover, using IM apps as-is to transfer censored content makes it difficult for the adversary to distinguish it from normal IM traffic. We implement Camoufler on five popular IM apps (Signal, Telegram, Slack, Skype and Whatsapp) and demonstrate that a user can access censored websites in a few seconds (3.6s on an average for Alexa top-1K websites).

Lastly, we attempt to address a recent and an extreme form of censorship *i.e.,* Internet shutdowns. Such an extreme step of complete Internet disconnectivity leads to various problems for users in such regions (*e.g.,* reporting of power failures, immediate access to medical emergency *etc.*). To the best of our knowledge there do not exist systems that can provide Internet access during shutdowns. Thus, we build a novel system *Dolphin* that can provide access to basic Internet services such as Twitter, email, accessing news *etc.* Dolphin utilizes the cellular voice channel to encode the data bits into audio and transfer it via the cellular call. However, the cellular voice channel is unreliable, inherently lossy, insecure and highly bandwidth constrained. We overcome all the aforementioned challenges and build a complete end to end system. We experimentally demonstrate that Dolphin can be used to access emails, tweets and news all within a few minutes (an email of 500 characters was delivered securely and reliably within 3 minutes).

Acknowledgements

I would like to thank my advisor, Dr. Sambuddho Chakravarty who has immensely supported and guided me throughout my PhD. I would also like to thank my co-advisor Dr. Mukulika Maity for her insightful suggestions. Additionally, I would like to thank Dr. H.B. Acharya for his involvement and guidance in my PhD. I am also thankful to Dr. Vinayak Naik for providing his valuable feedback and insights.

I would also like to take this moment to thank my PhD colleague Dr. Devashish Gosain, who has been a great support to me throughout my PhD journey. Moreover, I would like to thank all the students, faculties and other technical and non-technical people with whom I have worked for the research projects. Special thanks to the IIIT Delhi administration, IT and facility management staff for making work at IIIT hassle free.

Lastly, I would like to thank my parents, who have been extremely supportive throughout my journey.

Research Papers From This Work

1. The Road Not Taken: Re-thinking The Feasibility of Voice Calling over Tor.
   Proceedings on Privacy Enhancing Technologies Symposium (PETS), 2020.
2. SiegeBreaker: An SDN Based Practical Decoy Routing System.
   Proceedings on Privacy Enhancing Technologies Symposium (PETS), 2020.
3. Camoufler: Accessing The Censored Web By Utilizing Instant Messaging Channels.
   Proceedings on Asia Conference on Computer and Communication Security (AsiaCCS) 2021.
4. Dolphin: A Cellular Voice Based Internet Shutdown Resistance System.
   Under submission to a reputed venue.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Internet has grown well beyond its original purpose of providing a resilient communications infrastructure; in fact, it has become so essential to the free circulation of speech, information, and ideas that Internet access has been considered as a fundamental right by many countries [1]. However, in recent times, adversaries have attempted to contain the Internet with activities like network surveillance, IP address blocking, DNS injection, etc. It is becoming common that Internet Service Providers, working under the orders of malevolent adversaries (e.g., repressive regimes themselves), block, filter, intercept, or even modify traffic between clients on their networks and "controversial" Internet-based services [2, 3]. This has further led to a great deal of tension between civil liberty activists and repressive regimes. Thus, to facilitate freedom over the Internet, researchers have proposed various circumvention schemes. However, existing schemes witness multiple challenges. We broadly divide them into two categories:

Category I: *Applications (such as anonymous voice calling) for which there are proposed schemes but none that are functional or immediately usable.* This is because, the recently proposed systems (*e.g.,* Herd [4]) are difficult to deploy in practice due to dependence on the recruitment of volunteer operators. Moreover, existing anonymization infrastructure (e.g., Tor [5]) is believed to not provide adequate QoS guarantees particularly for real-time applications such as VoIP [6, 4]. Thus, it becomes of paramount importance to take strides in providing such a solution.

Category II: *Applications (e.g., those that allow accessing blocked web content) for which the proposed schemes have functional and usable implementations, but suffer from other drawbacks*. Arguably, such systems do not simultaneously provide blocking resistance and good performance to the users, while also keeping up with the privacy guarantees.

But all existing schemes that solve the above challenges would only function if the censor is not willing to completely cut-off from the Internet. Sadly, in past few years, we have now witnessed an increase in such extreme censorship cases *i.e., Internet Shutdowns*. Thus, we aim to devise a novel scheme to combat such extreme measures.

Category III: *Providing basic Internet access even during total Internet shutdowns* (*e.g.,* accessing email and news, posting a tweet *etc.*). Currently, there does not exist any scheme that can be used to access the Internet during such shutdowns.

We thus aim to improve upon the existing research by proposing and building novel systems for the above three categories. Thus, in this thesis, we aim to answer the questions — "Can we build privacy-preserving communication systems that also provide real-time performance guarantees to the end-users, when connected to the Internet?" "In extreme scenarios of Internet shutdown, can we still provide minimal access to the Internet in a privacy-preserving manner?"

We commenced our research by focusing on the first category of problem i.e., achieving anonymous calling over the Internet. Anonymity has always been of great interest to whistle-blowers, free speech activists, journalists, political activists, etc. There exist solutions that provide anonymity over the Internet for accessing the web, but there does not exist any functional system that supports anonymous voice calling. Hence, we attempt to answer questions like — *"Is anonymous voice calling possible over the Internet?", "Does providing anonymity have any quantifiable impact on perceived call quality?", "Do we need a new anonymization network to support VoIP traffic, or can existing anonymous networks (like Tor) be directly used?"*

To assess the feasibility of anonymous VoIP calling we considered Tor — a pervasive overlay network with volunteer-operated nodes that are built to support anonymous communication. However, existing research deems popular anonymization systems like Tor unsuitable for providing the requisite performance guarantees that real-time applications like VoIP need. Their claims

are backed by studies[6, 7] that may no longer be valid due to constant advancements in Tor. Moreover, these studies lacked the requisite diversity and comprehensiveness. Thus, conclusions from these studies led them to propose novel and tailored solutions (for achieving anonymous VoIP calling), eventually rejecting Tor as a possible solution.

However, no such system is available for immediate use. Additionally, operating such new systems would incur significant costs for recruiting users and volunteered relays, to provide the necessary anonymity guarantees. It thus becomes imperative that the exact performance of VoIP over Tor be quantified and analyzed so that the potential performance bottlenecks can be amended. We thus conducted an extensive empirical study across various in-lab and real-world scenarios to shed light on VoIP performance over Tor. In over half a million calls spanning 12 months, across seven countries and covering about 6650 Tor relays, we observed that Tor supports good voice quality (Perceptual Evaluation of Speech Quality [8] (PESQ) >3 and one-way delay <400 ms) in more than 85% of cases. Further analysis indicates that in general for most Tor relays, the contentions due to cross-traffic were low enough to support VoIP calls, that are anyways transmitted at low rates (<120 Kbps). Our findings are supported by concordant measurements using iperf that show more than the adequate available bandwidth for most cases. Hence, unlike prior efforts, our research reveals that Tor is suitable for supporting anonymous VoIP calls. Thus, in parts of the globe, where Tor is not blocked, it proves to be a readily available solution for achieving anonymous VoIP calling, in addition to accessing the web.

However, determined adversaries like China put constant efforts to block circumvention systems such as Tor, proxies, VPNs, etc. Conversely, researchers keep developing systems that counter the measures taken by such adversaries. This has led to an unsaid arms race between the Internet free speech activists and the censor. However, to put an end to this arms race, researchers proposed a novel solution, viz., Decoy Routing (DR) [9]. It is a promising approach to censorship circumvention, that uses routers (rather than end hosts) as proxy servers. Users of censored networks, who wish to use DR, send specially crafted packets, nominally addressed to an uncensored website. Once safely out of the censored network, the packets encounter a special router (the Decoy Router) which identifies them using a secret handshake, decrypts their content, and proxies them to their true destination (a censored site).

However, DR has implementation problems: it is infeasible to reprogram routers for the complex operations required. Existing DR solutions fall back on using commodity servers as a Decoy Router. But as servers are not efficient at routing, most web applications show poor performance when accessed over DR. A further concern is that the DR has to inspect all flows in order to identify the ones that need DR. This may itself be a breach of privacy for other users (who neither require Decoy Routing nor want to be monitored).

Thus, in the second part of this thesis, we attempt to answer questions like — *"Can we build a practical privacy-preserving DR solution?", "What performance guarantees can be assured with such a system?" and "How hard would it be to block the proposed DR system?"*

To answer such questions, we present a novel DR system, *SiegeBreaker*, which solves the aforementioned problems using an SDN [10] based architecture. Previous proposals involve a single unit that performs all major operations (inspecting all flows, identifying the Decoy Routing requests, and proxying them). In contrast, SiegeBreaker distributes the tasks for Decoy Routing among three independent modules. (1) The SDN controller identifies Decoy Routing requests via a covert, privacy-preserving scheme, and does not need to inspect all flows. (2) The reconfigurable SDN switch intercepts packets and forwards them to a secret proxy efficiently. (3) The secret proxy server proxies the client's traffic to the censored site. Our modular, lightweight design achieves performance comparable to direct TCP downloads, for both in-lab setups, and Internet-based tests with commercial SDN switches. Thus, we experimentally demonstrate that privacy-preserving highly effective blocking resistant solutions can be built that could further end the arms race between free speech activists and authoritarian regimes.

It must be noted that on one hand, we have systems that are readily available but are relatively easy to block (*e.g.,* VPNs, proxies, *etc*.). While on the other hand, the systems that provide highly effective blocking resistance (*e.g.,* decoy routing) incur deployment costs (require collaboration with the ISPs to install decoy routers, *etc.*), rendering them currently unavailable for users and preventing their wide-scale adoption.

Thus, to further aid anti-censorship research, we answer questions like — *"Is it possible to build a system that is effective in resisting adversaries and at the same time is readily available*

to end-users?", "What would be the deployment costs and performance overheads (if any), in building such a system?"

To find a possible answer, we attempt to use the Instant Messaging (IM) channels as a medium to tunnel blocked content. Our motivation to use the IM channel is two-fold. Firstly, IM apps are readily available to end-users, as they are pervasive. Thus, a system built on top of it could be easily accessed by the end-users. Secondly, completely blocking IM apps would result in collateral damage to the censor; an adversary would need to block all IM platforms to completely render the system ineffective, as IM apps are an integral part of today's economic as well as social activities.

Thus, we propose a new anti-censorship system, called *Camoufler*, which overcomes challenges in existing systems, while attempting to maintain similar blocking resistance available in existing schemes. Camoufler leverages Instant Messaging (IM) platforms to tunnel the 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. As a consequence, the censor would observe regular IM traffic being exchanged between the IM peers. Thus, utilizing IM channels as-is for transporting traffic provides unobservability, and also provides good QoS, due to its inherent properties, such as low-latency message transports. Moreover, it does not pose new deployment challenges or incur additional costs.

We implemented Camoufler on five popular IM apps (Telegram, Signal, Whatsapp, Slack and Skype) and it provides adequate QoS for web browsing. E.g., the median time to render the homepages (regular webpages) of Alexa top-1k websites was recorded to be about 3.6 s when using Camoufler implemented over Signal IM application. This is comparable to what one may observe when browsing these pages directly (*e.g.,* via browser), in which case it takes around 2.7 s for the same.

Notably, traditional censorship revolves around blocking access to some websites (or services) over the Internet. However, recently there has been a rise in the events of an extreme form of censorship *viz.,* deliberate Internet shutdowns [11] that lead to complete Internet disconnectivity, severely impacting the people residing in such regions. Moreover, these shutdowns render all

existing circumvention schemes unusable. Thus, in the final part of this thesis we aim to answer questions such as — *Is it possible to build systems that can provide Internet access during such shutdowns? If so, what type of applications can we access using it? What are the security guarantees we can provide in such a system?*

To that end, we present *Dolphin*, a first of its kind system that can provide access to lightweight and delay tolerant Internet applications (email, tweets, news snippets, *etc.*) during Internet shutdowns. Dolphin uses the cellular voice channel to transmit data bits. A user in the shutdown region (who wishes to access these applications) requires a trusted peer (*e.g.,* a friend) in non-shutdown region to send and retrieve content on its behalf. The data bits between the peers are sent by first encoding them into audio and then transmitting them over a cellular voice call.

We overcome multiple challenges while designing and implementing Dolphin. *E.g.,* the cellular voice channel is inherently lossy and unreliable. But the Internet applications need reliable transfers. Thus, in Dolphin we develop a TCP style reliability layer to overcome the losses. This layer works atop any underlying encoding/modulation technique. Further, to evade eavesdroppers over the insecure voice channel, we use this layer to derive keys for encrypting data. Also, Dolphin can work even without human intervention, by using cellular voice automation services. We experimentally show that Dolphin works for Internet applications, by testing it for sending email, tweets and accessing news snippets. All these applications take a few minutes to be accessed (*e.g.,* a 500 character email was securely received in under 3 minutes).

To summarize, following are the major contributions of this thesis:

- To solve the problems of category I, we demonstrate for the first time that a functional and usable anonymous voice calling system is possible using an existing anonymity system Tor.
- Subsequently for category II, we present the design and implementation of a performant and privacy-preserving decoy routing system in the form of SiegeBreaker.
  Moreover, since DR systems pose deployment challenges, we designed another system, Camoufler, that is easy to deploy and at the same time provides good performance and

blocking resistance.

- In case of category III, *i.e.,* for extreme scenarios of Internet shutdowns, we develop a novel system Dolphin, that provides access to lightweight Internet applications such as twitter, news *etc*.

Chapter 2

Background

In this thesis we attempt to build privacy-enhancing systems to facilitate free flow of information over the Internet while also conducting measurement studies to assess their performance. First, we study the feasibility of performing VoIP calls over a popular anonymous network Tor (in Chapter 3). However, in parts of the globe where Tor is censored, we present a new hard to block and performant decoy routing system (SiegeBreaker) built with the help of software defined networks as an underlying technology (presented in Chapter 4). But, SiegeBreaker may face deployability challenges as it requires collaboration from ISPs. Thus, we show the design and implementation of a new and easily deployable circumvention system (Camoufler), that utilizes IM apps to tunnel restricted content. IM apps are ubiquitous and may cause collateral damage to the censor if recklessly blocked (explained in Chapter 5). Lastly, we consider the scenario of Internet shutdowns where all existing circumvention solutions cease to function. Therefore, in Chapter 6 we present a system (Dolphin) that can provide access to basic Internet apps during the events of Internet blackouts. To conclude the thesis, we present our overall takeaways and the potential future work in Chapter 7. The overall thesis road map is depicted in Fig. 2.1.

To better understand the contributions in the individual chapters, that include varied technologies like VoIP, Tor, SDN, and IM applications, we now describe the relevant background.

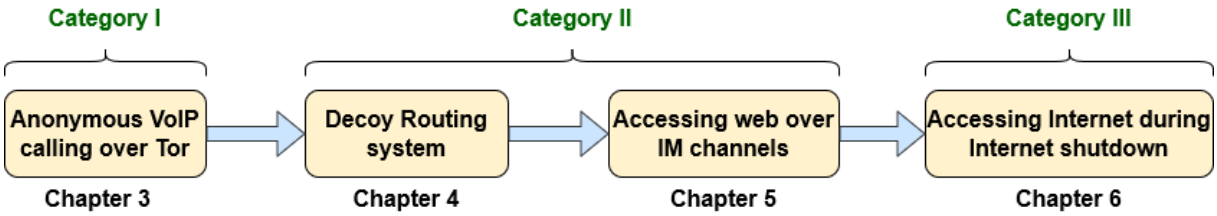

Figure 2.1: Overall thesis road map depicting different projects covered in each chapter.

2.1 On anonymous VoIP calls

In this section, we begin by describing the basic concepts behind VoIP calling and its evaluation metrics. Next, we briefly describe Tor, followed by a discussion on different types of anonymous calling scenarios.

2.1.1 Basics of VoIP

Voice over IP enables real-time voice communication over IP networks. Any VoIP based system comprises of two primary channels: one for control and signaling, and the other for transporting the actual encoded voice traffic. Session Initiation Protocol (SIP) [12], is an example of a popular VoIP signaling protocol. It includes functionality like authenticating users, establishing and terminating calls, *etc.* SIP, along with Session Description Protocol (SDP) [13], allows for negotiating the call related parameters like codecs. Once the call is set-up, Realtime Transport Protocol (RTP) [14], a UDP based protocol, solely manages voice traffic. It adds sequence numbers to packets for in-order delivery, and buffers them to minimize the impact of jitter.

2.1.2 QoS Metrics for Voice Calling

The following metrics are often used for measuring the quality of voice calls.

1. **Perceptual Evaluation of Speech Quality:** PESQ [8] is the ITU specified and standardized metric for voice call quality evaluation. It estimates the user perceived call quality. PESQ computation requires both the source and recorded audio for comparison. The PESQ metric generates an objective score using its own algorithm which is then mapped to a

subjective Mean Opinion Score (MOS). It is demonstrated that the score generated by PESQ highly correlates with the MOS score reported by actual users. The metric largely reflects distortions in recorded audio, which in-turn indicates the impact of network losses and jitters.

It returns values between 0 and 5, 5 being excellent, and 0 being poor and unusable. However, in practice, values between 1 and 4.5 are observed. Following ITU specifications, in our tests, we considered calls with PESQ greater than three as acceptable.

2. **Jitter and Packet Loss:** Jitter represents variations between subsequent packet arrivals. Such variations may arise due to packet reordering and losses. While VoIP users can endure minor losses, jitter may dramatically hamper the perceived call quality. Even non-permissible variations in either of these two metrics can sharply perturb PESQ, which incorporates the impact of both the metrics.

3. **One Way Delay (OWD):** OWD is the time duration between when voice packets are encoded at the sender and when they are successfully decoded at the receiver. According to the ITU specification [15], OWD should ideally be less than 150 ms. Further, OWD below 400 ms is considered permissible for international calls [16]. Hence, we chose the permissible limit of 400 ms to evaluate the performance of voice calls.

Interestingly PESQ score, other than losses, only captures the impacts of jitters, which represents variations in OWD. It would suffer no perturbations if all the voice packets were uniformly delayed, without jitter or losses. However, for humans, such delays lead to time-shifted audio, eventually leading to poor perceptual quality. Thus, in our evaluation study we considered both PESQ and OWD as metrics for estimating call quality.

2.1.3 The Onion Router (Tor)

Tor [5] is a widely-used low-latency anonymization network. It allows it's users to communicate without revealing their IP addresses. It consists of globally distributed volunteered hosts acting as relays. Clients communicate to servers by proxying their traffic via a cascade of three such

relays, *viz.*, the *entry*, the *middle*, and the *exit* nodes. The client encrypts the traffic using a three-layered encryption scheme, each corresponding to the three relays, using keys negotiated with each of them, respectively. These encrypted packets are then forwarded via the three chosen relays. Each of these relays de-crypts one layer of encryption and forwards it to the next one in the cascade. Thus, no one ever, other than the client itself, knows *the IP address of all the relays and the server.* Each relay only knows about the previous and next one in the cascade. The server only sees connections arriving from the exit node, but knows nothing about the client. By design Tor only supports TCP streams.

2.1.4 Types or Use Case of Anonymous Calling

As already mentioned, anonymous calls are of great use to whistleblowers, activists, undercover reporters, *etc.* However, it may be inquired as to what are the current alternatives (used by such groups) to conduct anonymous calls. To the best of our knowledge, there are no publicly available alternatives. This is the reason that in the past decade, prior efforts [4, 17] attempted to design and build such systems. However, as discussed in Subsec. 3.2, none of these systems are functional. Moreover, there are secure messaging and end-to-end encrypted communication apps such as Signal and Telegram which are highly popular among privacy practitioners. Though these provide security against eavesdroppers, the centralized architecture of all such apps allows the central server to know the details such as who is calling to whom. Thus, even though such apps provide secure calls, they may not be fit to be used for anonymous calls.

We now describe the anonymous voice calling scenarios tested in our study.

1. **Caller (one-way) anonymity:** Here, the caller wants to achieve anonymity (by hiding IP address) against an adversary that may monitor and (or) filter its traffic. Such scenarios represent journalists and whistle-blowers who communicate sensitive information to other individuals or groups (*e.g.*, news headquarters), while evading the adversary.

2. **Caller-Callee (two-way) anonymity:** Here, both parties want to achieve anonymity. Two individuals who both wish to communicate covertly, while remaining anonymous to their respective adversaries, may use such setups.

Further, these setups and their use cases are discussed in detail in Sec. 3.4.

2.2 SDN Based Decoy Routing

2.2.1 Decoy Routing

DR is an anti-censorship mechanism that that uses "friendly" routers in the Internet as proxy servers. Being routers, they are very difficult to block (ref. Fig. 2.2). The steps for DR are as follows: **(1)** A user hosted in a censor regime sets up a TLS connection with an unfiltered website (the OD), hosted outside the boundary of the regime. **(2)** The client-OD traffic carries a special cryptographic signature which can be identified by DRs en-route. On identifying this signature, the DR hijacks the connection, and engages a proxy (the DR proxy), that finally fetches the content from the CD. **(3)** The traffic from CD is returned to the client by the SP, setting TCP/IP header fields to appear as if they were part of the initial client–OD TLS connection.

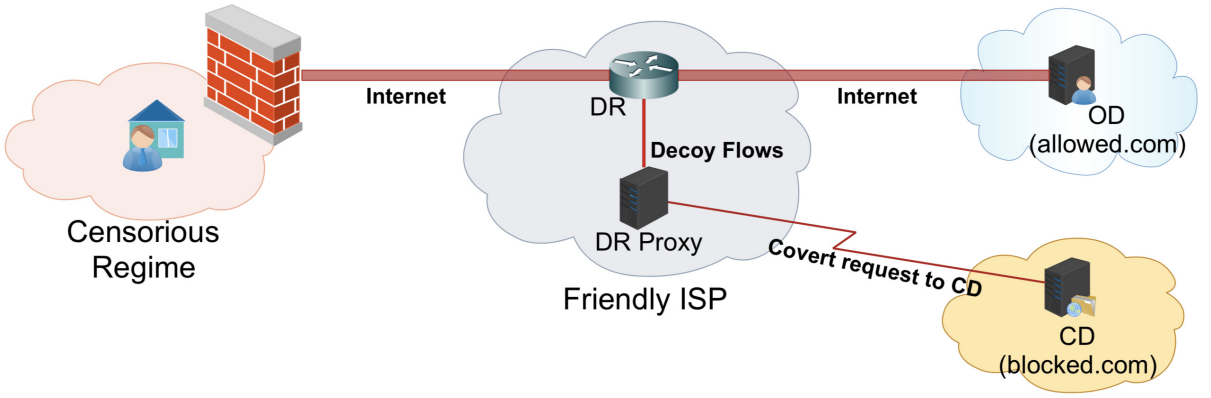

Figure 2.2: Decoy Routing: The user access `blocked.com`, through DR. To the censor it appears that the client is communicating to an unfiltered site, `allowed.com`.

2.2.2 Software Defined Networking (SDN)

Software Defined Networking [10] is an emergent network design paradigm that decouples the control and data plane of the network. An SDN architecture has a centralized controller, which controls the operation of the switches in the network (usually, by sending commands over a

standard protocol named OpenFlow [18]).

In SDNs, complex operations (such as computing network-wide control policies) are the responsibility of the *controller*. The policies, in the form of *flow tables*, are installed dynamically on the network switches which are simple dedicated devices, merely forwarding traffic based on these tables. The advantage of such an architecture is that control plane functionality is consolidated in a single, programmable entity – the controller.

Also, the SDN controller is capable of adding (or updating) any flow table rule or to redirect traffic to itself for analysis. By default OpenFlow compliant switches redirect any previously unseen packet, for which it has no matching rule, to the controller for inspection.

Since SiegeBreaker is also an SDN based anti-censorship system, it naturally inherits the salient features of SDN. In our design the controller and the SDN switches maintained by a friendly ISP are trusted; whereas the Secret Proxy managed by the DR operators/volunteers is not. Thus, any traffic analysed by the controller, will not compromise the ISP's regular customers' privacy, as it is maintained by the ISP itself.

2.3 Utilizing IM Platforms For Exchanging Blocked Content

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 [20] *etc.*) as well as professional communication (Slack [21], Flock [22] *etc.*). This is reflected in a recent report [23] which depicts that the number of monthly active users of IM platforms, were ≈ 2.4 billion in January 2019. These users are expected to go beyond 3 billion in 2022.

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 [24]. 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. 2.3). 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.* On the other hand applications such as Skype, Wechat *etc.* rely on E2M encryption.

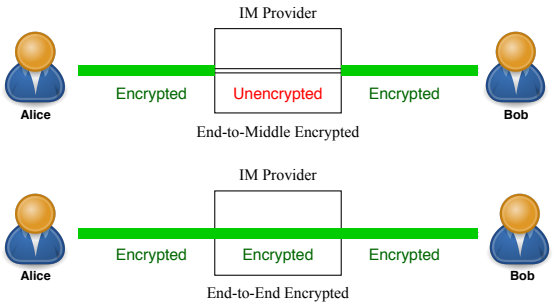

Figure 2.3: Difference between end-to-middle (E2M) and end-to-end (E2E) encryption in IM applications.

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* [25]). 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 [1]. As evident from Fig. 2.3, 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.

2.4 Internet Shutdowns and Outages

Internet shutdowns are the deliberate act of turning off the Internet connectivity by the competent authorities at the behest of the governments. Such shutdowns have been on the rise, with 213 documented cases reported in 2019 alone. There is a steady increase in the number of countries performing Internet shutdowns with the number increasing from 25 in 2018 to 33 in 2019. These shutdowns could last for less than a day to over a year in some cases (472 days in Chad) [11]

Various projects keep track of these shutdowns at country as well as global scale. *E.g.,* `accessnow` project [11] categorically reports incidents of shutdowns occurring across the globe, presenting detailed statistics of such events. Further, there are country specific projects such as [26] which maintain a record of all the shutdowns that happen in India (country with the highest number of shutdowns). Some projects even attempt to estimate the economic losses inflicted due to Internet shutdowns *e.g.,* `internetsociety` [27].

Other projects attempt to identify Internet outages in general. *E.g.,* IODA [28] keeps track of Internet outages by performing active measurements using various probes, as well as using passive measurements by identifying anomalies in publicly available BGP paths and characterizing them as possible cases of outages. There are some proprietary projects such as ThousandEyes (managed by Cisco) [29] which also keep track of Internet outages across the globe in real-time.

Overall, while there are various studies and platforms that report Internet shutdowns and outages, but none provide solutions to circumvent them. Thus, we present Dolphin, a novel system which provides basic Internet connectivity to the users in shutdown and outage regions by using cellular voice (utilizing just a mobile phone and a laptop/desktop).

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

Chapter 3

The Road Not Taken: Re-thinking the Feasibility of Voice Calling Over Tor

3.1 Introduction

Voice-over-IP (VoIP) applications that support traffic encryption are popular among users who are concerned about their communication privacy. However, these applications do not safeguard the anonymity for Internet users residing in regimes which may conduct surveillance (*e.g.*, the ability of NSA to intercept conversations is widely known [30, 31, 32]). Moreover, to the best of our knowledge, there does not exist any functional VoIP based system that ensures communication privacy and anonymity (along with real-time communication guarantees), required by privacy conscious Internet users and whistle-blowers alike.

Popular privacy and anonymity preserving systems like *Tor* [5] hide the actual IP address of the communication peers by routing their traffic via a cascade of proxies. Since such systems reroute traffic via circuitous paths, it is a widely held belief that they would incur intolerable delays for real-time applications like VoIP. More importantly, while traditionally VoIP relies on UDP traffic to ensure real-time guarantees, popular systems like Tor are designed to transport TCP traffic. Further, complex cryptographic handshakes, essential to the anonymity guarantees provided by such systems, may exacerbate the impact on performance, making them unsuitable

for real-time applications.

Prior efforts on anonymous calling [33, 4, 17, 34] unanimously agree with aforementioned shortcomings of Tor. Some conclude this based on potentially biased results [6], involving relays only in Europe or only conducted a few hundred calls [17], lacking diversity. Others, such as Le Blond *et al.* [4], did not conduct any experiments to measure VoIP performance over Tor. Overall, we believe that existing literature does not quantify the actual interplay of network performance attributes (*e.g.*, one-way delay (OWD), available bandwidth, *etc.*) and how it impacts VoIP call quality over Tor. However, the belief that Tor is not competent enough to transport VoIP traffic is still prevalent [17].

Thus several novel VoIP architectures were proposed [33, 4, 17] to tackle the shortcomings. Some of these, like *Phonion* [17] and *Herd* [4], require a new volunteer-run network with millions of active users (like Tor) for providing anonymity guarantees. This requirement can be a major stumbling block. Moreover, in the absence of active users and significant cross-traffic, comparable to that of Tor (that transports over 200 Gbit/s traffic per day [35]), one cannot adjudge future anonymity and performance assurances of these proposals. *Importantly, no such system is currently functional.*

Hence, in the absence of an existing anonymous voice calling system, we chose to determine the root cause(s) of poor performance over Tor. After ameliorating them, one may expect to achieve adequate voice call quality with anonymity guarantees equivalent to that provided by Tor. To that end, we began by conducting a pilot study where we made VoIP calls over the Tor network. We also captured the network performance attributes, in order to eventually identify how they impact voice call quality. Following ITU guidelines and prior proposals [17], we used one-way delay (OWD) and *Perceptual Evaluation of Speech Quality* (PESQ) [8] as metrics to judge VoIP call quality. The PESQ ascribes a value to judge the user-perceived audio quality in an automated manner.

We made 1000 consecutive calls through individual Tor circuits[1], with the callee and caller machines under our control. Contrary to the prevalent notion of poor quality of calls via Tor,

[1] Using the standard Tor client utility.

we observed good call quality, with average PESQ ≈ 3.8 and average OWD ≈ 280 ms. Overall, 85% of the calls were acceptable (PESQ >3 and OWD <400 ms) as per ITU [16, 15].

In the absence of substantial evidence of poor quality calls, we went ahead and conducted an extensive longitudinal study involving 0.5 million voice calls over the Tor network, spread across 12 months. These measurements not only involved varied in-lab and real-world scenarios with diverse Tor relays, VoIP applications, caller-callee locations but also a user study. To our surprise, even then more than 85% of voice calls had acceptable perceptual quality.

The PESQ metric varies inversely with distortions in the perceived audio. Packet drop and jitters, which cause such distortions, are an artifact of increased cross-traffic contentions. *A high PESQ score, accompanied with low overall OWD, in the majority of the cases thus indicates low cross-traffic contentions, for VoIP calls. Moreover, VoIP calls which are mostly transmitted at low bit-rates (<120 Kbps), incur less routing costs, and thus may not suffer much distortions.*

Other network performance metrics like available bandwidth also varies with network cross-traffic volume. Thus, concomitant available bandwidth during the call (over 1 Mbps in 90% cases[2]), along with data published by Tor Metrics [35], confirms the reason for obtaining good results.

Overall, this first ever long-term study involving extensive evaluations of calls over Tor, bore some interesting results and insights. We summarize them as follows:

1. In the vast majority of our experiments (>85%), involving voice calls over individual Tor circuits, we observed acceptable call quality with PESQ above 3 and OWD less than 400 ms. This holds for a diverse set of scenarios:
   (a) Caller and callee spread across 7 countries (in three continents).
   (b) Coverage of a total of 6650 Tor relays, 22 times more than previous studies [6].
   (c) Popular VoIP apps such as Telegram and Skype.
   (d) Both caller and (or) callee using Tor circuits to establish calls with one another.
   (e) Different codecs and call duration.

[2]The bandwidth was measured by `iperf`.

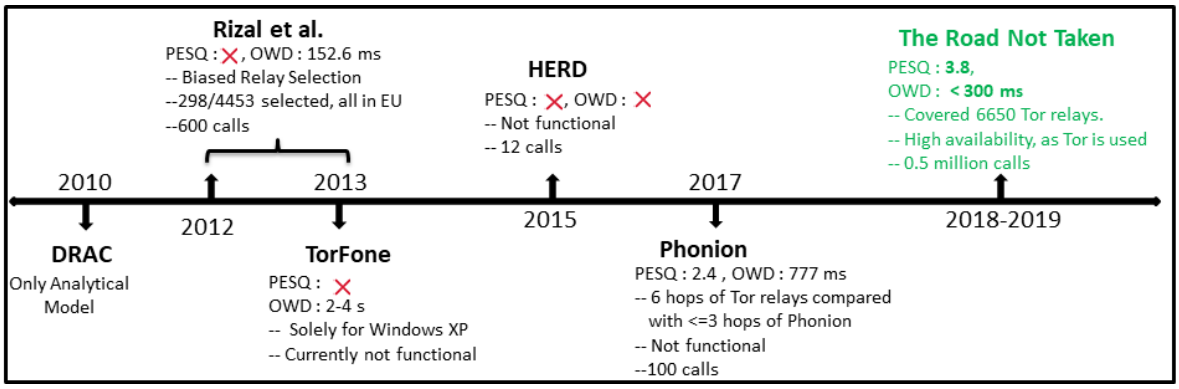

Figure 3.1: Existing literature on anonymous voice calling highlighting their inconsistencies and incompleteness. **X** implies metric not considered or evaluated in the study.

(f) User study involving humans rating voice calls.

2. Acceptable performance in the vast majority of cases was due to *relatively low contention for VoIP in most Tor relays.*

3.2 Related Work

There exists scant literature on performance evaluation of voice calls over Tor (ref Fig. 3.1). The only attempts made were by Rizal *et al.* [6] and Heuser *et al.* [17]. Their efforts involved instantiating a few hundred calls through Tor for evaluating their quality.

However, the aforementioned efforts were limited in scope. *E.g.*, rather than using PESQ (an established call quality metric), Rizal *et al.* relied only on network parameters like OWD, jitter, and packet loss as evaluation criteria. Also, this study involved Tor relays only in Europe (with only 298 out of the available 4453) for their evaluation, and may not be representative of VoIP performance, for the complete Tor network. Based on these preliminary studies, previous researchers ignored Tor and proposed novel architectures [4, 17, 33] for providing anonymous voice calls. Next, we describe all such efforts.

Inspired by Chaum's mixes [36], Pfitzman *et al.* [37] proposed ISDN mixes for anonymous voice communication. Authors proposed two simplex Mix channels, one for the sender and the other for the recipient, thereby enabling full-duplex anonymous communication over the telephony network.

Drac [33] by Danezis *et al.* involves an architecture for anonymous low latency voice communication using social networks as relays to route traffic. The system, while providing anonymity to a particular user, relies on using the identity of other users in its social circle as alibis. However, Drac presents an analytical model with no functional deployment.

Torphone [38], a system designed over Tor, was an extension to send VoIP traffic via Tor. It reported having achieved an OWD of 2–4 s, which is unsuitable for acceptable call quality. However, it is presently non-functional (and was last used on Windows XP).

In 2015, Le Blond *et al.* [4], proposed a novel architecture, *Herd*, to prevent global passive or active adversaries attempts to de-anonymize users, based on call metadata (correlating start and end times of a call). Herd relies on a set of dedicated mixes that relay VoIP traffic to other mixes and endpoints while hiding any distinct traffic patterns. The mixes are also known as *zones*, and a user can select available trustworthy zones to route its call. They believed prior results on evaluating the performance of VoIP over Tor (published back in 2008 [7]), and concluded the RTT to be high (2–4 s), deeming Tor unsuitable for VoIP. However, they refrained from conducting a fresh study to gauge the (then) recent performance of Tor. Additionally, they concluded that the calling entities on Tor could be easily correlated, given the start and end times of a call. They validated this claim by analyzing the call records of a service provider. However, obtaining such data for calls conducted via Tor might not be easy, as Tor is a globally distributed system, and it would require gathering data from geographically diverse ISPs. Moreover, they tested their prototype on only four cloud hosts, performing just 12 calls, in the absence of active users and significant cross-traffic when compared to Tor.

Phonion [17] is one of the recent anonymous VoIP systems. It is fundamentally similar to Tor, but specifically developed for voice communication. It uses *relays* (similar to Tor relay), *relay services* and *broker system* (similar to Tor directory authorities). Phonion attempts to anonymize call data records (CDRs) against different adversaries. In Phonion, the calls are relayed via various service providers. This prevents a single provider to gather all the call records for a particular call. The advantage of Phonion is that it works across different voice calling technologies — VoIP, cellular or PSTN (public switched telephone network). Additionally, it requires Internet access only for an initial bootstrap phase after which, anonymous calls could be

instantiated over carrier services. However, the authors of Phonion also side stepped using Tor. Through a pilot study of 100 calls, they concluded Tor to be unsuitable for transporting VoIP calls. The study involved comparing one, two and three-hop circuits of their prototype, with six-hop Tor circuits (two-way anonymity). However, we show in Sec. 3.4.2.1 and Sec. 3.4.2.2 that regular three-hop Tor circuits provide suitable performance for VoIP. Usage of this setup and performing 100 calls might have led them to report an unacceptable average OWD of 777 ms for Tor circuits along with an average PESQ of about 2.4. Moreover, the implementation of their system relied on `google voice`, a service only available in N. America, making the system unusable elsewhere. Lastly, it required the relay operator to pay an operational fee to telcos, which might serve as a stumbling block for recruiting relays.

Overall, a major hindrance in the wide-scale adoption (and availability) of such systems, is the recruitment of new volunteered relays and users (which Tor already has in abundance). Schatz *et al.* [34] also highlight this issue. Additionally, as previously mentioned, prior evaluations over Tor, provide very few insights as to how the interplay of network performance attributes affects the voice call quality.

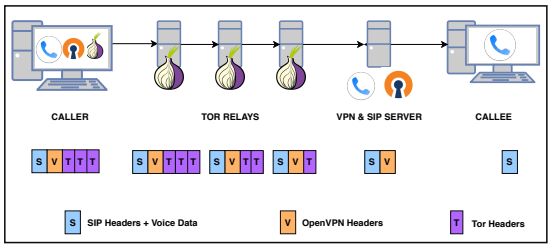

Figure 3.2: V-Tor: The caller establishes a VPN tunnel through Tor, so that all the UDP VoIP traffic traverses the Tor network. On reaching the VPN server, the traffic is sent to the SIP server, which initiates a voice call to the callee.

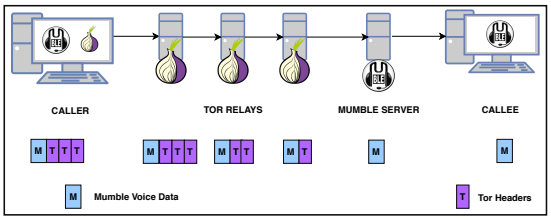

Figure 3.3: M-Tor: The caller configures the mumble client to work in TCP mode. Thereafter, mumble's traffic is sent over Tor eventually reaching the mumble server, which handles the call procedure between the caller and callee.

Comparison with prior VoIP measurements over Tor: As already mentioned, only two studies actually measured the performance of VoIP over Tor — Rizal's study [6] and Phonion [17] by Heuser *et al.* We now explain how our measurement study is methodologically different from them.

Rizal's study: This study transported VoIP traffic over Tor by tunnelling it via VPN tunnels. The experiments conducted in this study, involved Tor relays hosted only in Europe. Thus, Rizal

reported a reasonably low average OWD of ≈ 152 ms likely due to the geographic proximity of relays. On the other hand, our study included relays from all parts of the globe and hence provided better coverage of the entire Tor network[3]. Thus, our study measured an average OWD of ≈ 280ms likely due to the diversity of involved Tor relays. Moreover, unlike Rizal's study, we used PESQ as an evaluation metric, which is an industry standard for measuring user perceived voice quality.

Phonion: It utilized the Mumble VoIP software [39] to transport VoIP traffic via Tor. Authors of Phonion used PESQ as an evaluation metric. However, their study involved measuring performance only for two-way anonymity, and not the one-way anonymity. On the other hand, our study involved different scenarios corresponding to both the cases *viz.*, one-way and two-way anonymity. Hence, we involved comparatively more diverse and comprehensive set of measurement setups.

Additionally, both the previous studies involved performing only a few hundred calls — 100 in Phonion and 600 in Rizal. In contrast, we performed ≈ 0.5 million calls involving a diverse set of geographic locations, media codecs, *etc.,* along with controlled experiments involving private Tor setups.

Overall, these studies lack the requisite comprehensiveness and a detailed analysis of different performance attributes that could provide deeper insights for VoIP performance over Tor. Thus, our study aimed to fill this research gap by conducting an extensive study with an intent to better understand the behavior of VoIP calls over Tor in varied setups and network conditions.

Other Tor performance measurement studies : There exist abundant studies which measure the Tor network's performance in terms of observed bandwidth and latency [40, 41, 42, 43, 44, 45, 46, 47, 48, 49]. Few, like Shadow [42] and Chutney [50], developed simulators to analyze the performance of Tor. Others like Torflow [47], EigenSpeed [48], and Peerflow [49] *etc.,* focused on measuring relay bandwidth for calculating relay weights by either directly measuring relay bandwidth (Torflow) or by indirectly inferring it using statistics such as relays reporting the amount of data exchanged between them (Peerflow). Measuring relay bandwidth is a crucial task

[3] Achieved using the standard Tor utility that helped create different circuits based on default Tor circuit selection algorithm.

in Tor, as it is used to assign weights to different relays that govern the selection probability of a relay in a Tor circuit and thus the amount of traffic the relay might serve. Cangialosi *et al.* [51] specifically focused on measuring RTT between relay nodes. Tor metrics [52] is another popular (and actively maintained) project that periodically measures various performance attributes of the Tor network *e.g.,* circuit RTTs, bandwidth for downloading files of various sizes, *etc.* However, none of these existing projects, measure the perceptual VoIP quality and its associated network characteristics. In our study, we primarily measure the VoIP performance over Tor, along with network characteristics (like bandwidth and RTT) of millions of Tor circuits. The metrics, such as available bandwidth are essential in our study as they help to infer the potential reasons for good or bad performance while conducting VoIP calls. We often referred to the Tor metric results to further support our claims.

3.3 Measurement Approach

In this section, we describe our experimental setups and the approach taken for performing experiments to measure the quality of voice calls over Tor.

3.3.1 Experimental Setup

Previous efforts acknowledge that transporting VoIP traffic anonymously over Tor is non-trivial, as VoIP generally uses UDP and Tor only supports TCP. There are two ways in which one can succeed in sending VoIP traffic over Tor. One way is to tunnel UDP packets inside TCP flows. The other way is to directly encode and send VoIP traffic in TCP packets. Thus, some previous studies [6] utilized VPN tunnels to encapsulate and transfer VoIP packets, while others like Phonion [17], relied on Mumble [39] VoIP software to generate TCP packets and transfer them directly to Tor. Similar to these studies, we tested VoIP performance using both the above approaches. These setups are now described in detail below:

1. **SIP client via VPN through Tor *(V-Tor)*:** The V-Tor setup (ref. Fig. 3.2), involves a SIP client (caller) connecting through a Tor circuit to a VPN server for establishing a TCP

tunnel. A step by step walkthrough of this setup is described below:

Step1 The caller runs the VPN and Tor client utilities. VPN client is configured to establish a VPN tunnel (to the VPN server) over a Tor circuit, by forwarding VPN traffic to the Tor SOCKS interface. This ensured that all the traffic from the caller would reach the VPN server via the Tor network.

Step2 Client would initiate a VoIP call using a SIP utility. The VoIP traffic (generated from SIP) would reach the VPN server via Tor (explained in step 1). VPN server decapsulates SIP packets and forwards them to the SIP server.

Step3 SIP server then helps negotiate the call between the caller and the callee.

2. **Mumble with TCP mode over Tor (*M-Tor*):** This setup relies on using the Mumble client program (in TCP mode) for encoding voice traffic through TCP streams. These streams are transported via Tor to a Mumble server that mediates the voice call between the parties.

It must be noted that we performed experiments for both the V-Tor and M-Tor setups. However, since we obtained similar experimental outcomes from both the setups, we present description of V-Tor experiments in the subsequent sections and sub-sections, and the details of M-Tor experiments in Appendix A.1.

3.3.2 Overview of Experiments

We now describe the experiments performed to measure the VoIP call quality over Tor. Our experiments were primarily conducted to identify the root cause of the hitherto believed poor voice call quality of Tor.

We began with a pilot study that involved conducting 1000 consecutive calls over Tor, with the caller and callee under our control, but in different geo-locations. Our results, using V-Tor (and M-Tor) setup, showed high call quality in a large fraction of the cases. Considering these results to be potential outliers, compared to findings of previous authors, we went ahead and conducted a comprehensive measurement study spread across 12 months.

We conducted two different sets of experiments — (1) involving in-lab setups and (2) involving circuits through the public Tor relays.

In-lab experiments: In the in-lab setup, our goal was to measure the performance of VoIP over V-Tor (M-Tor) setups with competing cross-traffic entirely under our control. For this, we setup a private network in our lab, consisting of Tor nodes along with client and server (VPN, SIP, etc.) machines. This private network was deployed on real machines, with three of them serving as relays, while one of them was also serving as a directory authority. Other machines acted as clients and servers. In these experiments, we measured performance attributes and established baseline values (*e.g.,* bandwidth requirement) for a VoIP call. These experiments were performed over setups involving different combinations of VPN, Mumble, and Tor in the following manner:

1. *Direct VoIP calls:* VoIP calls between the caller and callee were conducted without involving Tor and VPN. The caller and callee communicated directly using SIP or Mumble protocol. This helped us in observing the minimum bandwidth and delay requirements when no overhead was introduced (due to Tor or VPN).

2. *VoIP calls over VPN:* Caller and callee communicated using SIP protocol. However, calls were encapsulated through VPN connections, in order to measure the impact (if any) due to the overhead of running a VPN.

3. *VoIP calls over Tor:* Encapsulating the calls through VPN connections (or Mumble) *and* then transporting them via Tor circuits, to evaluate the impact of the overheads due to Tor.

Moreover, we performed some additional experiments to observe the impact of variation in background cross-traffic on VoIP calls. We observed the variation in performance when VoIP call(s) were made in the presence of heterogeneous background traffic (*e.g.,* other VoIP calls and web traffic). Such in-lab experiments may potentially present clues regarding the number of VoIP clients that could be simultaneously supported by the real Tor network.

Internet based experiments over public Tor: After performing various in-lab tests, we carried out multiple experiments involving public Tor relays, where we had no control over the background cross-traffic and network conditions. These experiments involved measuring performance

across diverse scenarios (Tor relays, end-points, codecs, *etc.*) with the intention of studying the variation in performance under real-world conditions. More specifically, the experiments involved measuring the call quality by varying:

- **Tor circuits:** Involved measuring quality across a large number of circuits (6650 unique relays) created using the regular Tor client program.
- **Geo-location of communication peers:** Involved measuring quality by instantiating several calls, varying the location of the calling peers.
- **Type of anonymity achieved:** Involved measuring quality while achieving one-way and two-way anonymity, in accordance with the use cases already described in Sec. 2.1.4.
- **Circuit lengths:** Involved measuring quality over two-hop circuits. By default, Tor circuits are built using three relays.
- **Codecs:** Involved measuring quality by varying the call codecs used.
- **Call duration:** Involved measuring quality when call duration was varied.
- **Type of relays used:** Involved measuring quality when using bridges instead of public Tor relays.

We also measured the voice call quality for few popular voice calling apps such as `Telegram` [53] and `Skype` [54], when used over Tor. Users may choose to rely on using these already popular and familiar apps, instead of having to setup V-Tor and M-Tor. Additionally, we also conducted a user study involving 20 participants who rated calls via Tor. Most of experimental results are presented in the next section (Sec. 3.4). A small fraction, addressing important concerns related to VoIP performance, *e.g.*, impact of call codecs, Tor bridges, *etc.* are presented in Sec. 3.6.

3.3.3 Implementation Details

Host configurations: All machines of the in-lab experiments used Intel Core i5 8th gen CPUs with 8 GB of RAM. The hosts used in the experiments involving public Tor relays were hosted on

Digital Ocean's cloud based infrastructure, distributed across seven countries. These machines were equipped with Intel Xeon 2.2 GHz single core CPUs and had 2 GB RAM. Since the latter experiments required initiating only a single call at a time, we did not require machines with higher memory.

Tor configuration: In the basic configuration, all the caller and callee hosts had the latest Tor v0.3.9 installed. We used the Tor `stem` python library [55] to ensure that (1) only a single circuit was enabled at a time (2) the performance metrics (such as bandwidth measured using `iperf`, RTT using `ping`, *etc.*) were measured for the same circuit through which the call was performed.

Communication peers: For V-Tor experiments, the end hosts were installed with OpenVPN [56] v 2.4.7. The VoIP traffic generated was encapsulated through OpenVPN client and transported via Tor (by specifying the SOCKS port in the OpenVPN configuration). The clients used the python based SIP client, `pjsua` [57], a command-line softphone, to automate SIP calls. The module supported audio playout and recording.

VPN/SIP server: The V-Tor setup uses a machine configured to be a VPN as well as a SIP server. OpenVPN [56] v 2.4.7 was configured to work as the VPN server. It forwarded the encapsulated SIP calls to the SIP server. Freeswitch PBX [58] was used as the SIP server for handling SIP calls. We used different configuration files that come with Freeswitch to make call extensions, route calls, select codecs, *etc.*

Popular VoIP apps: Among all the popular apps, Telegram is the only one that provides user APIs. Still, we require to overcome several challenges to measure the performance of Telegram calls over Tor.

Firstly, the API provides only instant messaging automation facility. We thus mined the source code and discovered that it relies on `libtgvoip` [59] for making voice calls. We used this library for automated calls. Secondly, the library uses UDP for transporting VoIP. We thus modified it to enforce transporting voice calls via TCP streams. Thirdly, `libtgvoip` is configured, by default, to play audio clips endlessly in a loop. We applied appropriate modifications to control the playout duration, to suit our calls. Fourthly, we also required

modifying the library so as to synchronize the call setup and termination events with starting and stopping of voice recording. This synchronization is required for accurately measuring PESQ. Additionally, pyrogram [60] redirected Telegram traffic to the Tor SOCKS port.

Other popular voice calling apps like Skype poses two challenges—*viz.,* absence of resources like APIs or source codes to aid call automation, and ability to redirect traffic through Tor by configurable SOCKS interface. To overcome the first obstacle we initially synced the caller and callee machines. Thereafter once the call was initiated, the caller plays out the audio clip using `mplayer` [61], while the callee captured and recorded the call audio directly from the sound card using pactl [62] utility. To overcome the second challenge, we used OpenVPN to encapsulate and redirect Skype call traffic to the Tor SOCKS interface.

3.4 Measurement Results

In this section, we describe the experiments conducted to test the performance of anonymous calls over Tor, and their corresponding outcomes. We begin by enlisting some common steps we followed while conducting the experiments:

- In all our experiments, a caller host played out an audio clip containing 30 s of human speech. It was encoded and transported, via a unique Tor circuit, to the callee. The callee recorded the audio, which is later used for computing PESQ.
- For every call, we recorded the network traffic through `pcap` files, and also measured various network performance attributes of the Tor circuit (through which the call is performed) like available bandwidth and RTT using `iperf` and `ping`, respectively. The `ping` utility was run during a call, as it is not a bandwidth intensive test. On the contrary, `iperf` (bandwidth intensive) tests were conducted after the completion of the call, so that it does not have any impact on the call quality.

  For the in-lab experiments also, we measured the stream bandwidth using `iperf`.
- PESQ score for every call was calculated by comparing the original (one played out at

the caller) and recorded (at the callee) audio clips. Any score above three was considered good [15].

- One way delay was also calculated for the duration of the call. We used `ping` to calculate OWD. As per ITU guidelines for international calls [16], the upper limit of OWD for acceptable call quality is 400 ms.
- We ensured that for a call, all the performance metrics were measured for the same circuit through which the call was instantiated.

In every experiment, the above steps were repeated for each call. Thereafter, we analyzed the measurements and performance metrics across all these iterations.

3.4.1 In-Lab Experiments

We performed these experiments, with an intent of measuring call performance under different testing conditions, while fully controlling network link capabilities and background cross-traffic. To establish the baselines, we created three test scenarios for the setup. These involved: (1) Direct SIP calls (2) SIP calls over VPN tunnel (3) SIP calls through VPN over Tor (V-Tor). All these experiments followed the setup described in Fig. 3.2 For the scenarios where Tor was not used, the nodes between caller and callee (in Fig. 3.2) merely functioned as routers. This was done to minimize any biases in performing experiments, by ensuring that the packets traverse the same number of hops[4].

Experiments using V-Tor setup: We started by initiating VoIP calls over all the three setups and computed their respective PESQ scores and OWD values. The capacity of the link between the caller and callee was 100 Mbps. The measured PESQ score averaged across 100 individual samples was the same for all the three scenarios *i.e.*, 4.5. Whereas OWD was below 50 ms. This result established that for a single call, with no competing cross-traffic, the overheads introduced by the VPN and Tor had no significant impact on the call quality. We additionally observed that the available bandwidth requirement for a single call in all the three scenarios was no more than

[4] Additionally, removing these hops do not have any observable impact on our results. We kept them just to have uniformity among our experiments.

120 Kbps (ref. Tab. 3.1). As expected, direct calls transmitted at the lowest rates. Additional overheads due to the headers introduced by VPN and Tor progressively increased the bandwidth requirements.

Call category	Bandwidth (Kbps)
Direct SIP call	84
SIP call via VPN	≈ 108
V-Tor	≈ 120

Table 3.1: Baseline bandwidth (in Kbps) requirement of VoIP in different scenarios.

Next, to understand the impact of cross-traffic on VoIP call quality, we initiated VoIP calls in the presence of cross-traffic. The experiments were carried out for three link bandwidth configuration—2 Mbps, 5 Mbps and 10 Mbps. Studying the performance under cross-traffic, for different link bandwidth would help us understand if the observed behavior is consistent or not. These experiments were specifically conducted for the VPN via Tor (V-Tor) setup. Further, these lab experiments provided us insights on the number of calls that could potentially be made under varied network conditions on the real-Tor network.

Thus, we gradually introduced the cross-traffic by increasing the number of parallel file downloads (using `wget`) from another client that shared the link with the caller. We made sure that the cross-traffic was in the direction of call, to ensure adequate cross-traffic contention. We measured the degradation in the call quality, by computing the average PESQ score, for every new parallel connection introduced. The cross-traffic was gradually increased such that it utilized the total link capacity from 5%, to 10%, and then all the way up to the point where the call under consideration received less than 120 Kbps of the total available bandwidth. At this point, we observed a sharp decline in PESQ for the call (*i.e.,* 2.3). This corresponds to unacceptable call quality. Our findings are summarized in Tab. 3.2.

Competing Streams	Link Bandwidth	Available Bandwidth Per Stream	Call Requirement	PESQ Score
< 75	10 Mbits	$>$ 133 Kbps	120 Kbits	> 4.2
80	10 Mbits	125 Kbits	120 Kbits	$\approx 3.4 \downarrow$
> 85	10 Mbits	$<$ 117 Kbits	120 Kbits	$< 2.3 \downarrow\downarrow$

Table 3.2: Analysis of V-Tor under the presence of competing non-VoIP (web or file downloads) cross-traffic.

We then performed experiments, where the background cross-traffic constituted of other

VoIP calls. Similar to the previous experiment, we performed this test for three different link capacities (2, 5 and 10 Mbps) for the V-Tor setup. The results of the 5 Mbps link bandwidth test are summarized in Tab. 3.3. We obtained similar behavior for the other two bandwidth categories.

Competing VoIP Calls	**Link Bandwidth**	**Available Bandwidth Per Call**	**Call Requirement**	**PESQ Score**
< 35	5 Mbits	> 145 Kbps	120 Kbits	> 4.2
40-43	5 Mbits	128 Kbits	120 Kbits	$\approx 3.3 \downarrow$
> 43	5 Mbits	< 120 Kbits	120 Kbits	$< 2.3 \downarrow\downarrow$

Table 3.3: Analysis of V-Tor under the presence of competing VoIP cross-traffic.

The results indicate that when the contention on the shared link increases, the PESQ drops. The PESQ metric is very sensitive to the impact of even minor network drops or delays. Even a small increase in contention, *e.g.,* only five additional download streams reduce PESQ from 4.2 to 3.4 (ref. Tab. 3.2). Corresponding to increased contentions, the available bandwidth for every stream (including VoIP) drops. The constant bit-rate voice traffic of 120 Kbps suffers significant distortions that may, however, have little impact on the non-VoIP flows. This held when the cross-traffic was non-VoIP as well as when it was VoIP.

These in-lab measurements indicate that, a client should be able to conduct good quality calls, if the constructed circuit provides a bandwidth of above 120 Kbps. This should hold true on the real Tor network as well. *E.g.,* if a circuit has an available bandwidth of about 1.2 Mbps, then it is capable to simultaneously support a maximum of 10 VoIP clients (with acceptable call quality).

3.4.2 Experiments Involving Public Tor Relays

Having obtained good performance in in-lab tests, we went ahead to evaluate the performance of voice calls over public Tor network. We expected the results to vary significantly due to the dynamic nature of competing cross-traffic and network conditions over the Internet.

We began with our pilot study that involved a client host (caller), positioned in our university, establishing VoIP call to a cloud hosted peer (callee). The call traffic was transported via Tor. We

sequentially started 100 calls, each transported via a different Tor circuit, and measured the call quality by computing PESQ score. To our surprise, for both the setups we observed an average PESQ of 3.8 and an acceptable OWD of 280 ms. Even after 1000 calls, each transported via a freshly created Tor circuit, we observed very similar performance measures (PESQ $\approx$ 3.86 and OWD $\approx$ 273 ms).

However, one may argue that our positive results might have been a small anomalous fraction. These may have been different from the bulk of poor outcomes that may have led others to deem Tor as unfit for VoIP. In order to test that this was not a fluke, we conducted a longitudinal experimental study covering diverse scenarios. The experiments are described below.

3.4.2.1 Caller anonymity: co-located voice server and callee (Scenario I).

We begin with the fundamental scenario where a caller is positioned in a censored network and wishes to call someone who is beyond the censor's control. The caller makes calls to the callee, through the public Tor network, using setups similar to one shown in Fig. 3.2. It is assumed that callee runs a publicly accessible VPN server (for V-Tor) on its host.

In our experiments, we chose seven individual cloud machines as callers, and three other as callees. Each of these was selected from Europe, N. America, and Asia. For every caller–callee pair we made 1000 calls using the V-Tor setup. In each case, we measured the PESQ and OWD. A total of 42000 calls were made. The average PESQ and OWD across all measurement was 3.88 and 217 ms, respectively.

By default, Tor circuits are three hops long. To reduce the potential impact of hop length on performance, we repeated the said experiments by making calls over two-hop circuits. This did not led to any significant impact on the PESQ (3.90). However, as expected, the OWD reduced to 205 ms. The CDF of PESQ scores obtained is depicted in Fig. 3.4. Results clearly show PESQ above 3 in over 93% of the calls.

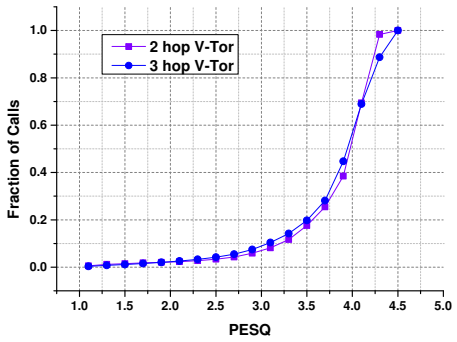

Figure 3.4: V-Tor: CDF of PESQ for Caller Anonymity when server is co-located with callee (Scenario I).

3.4.2.2 Caller anonymity: separate VPN/voice server (Scenario II).

There are, however, certain limitations of Scenario 1. Firstly, the setup requires a publicly accessible VPN server, which may be infeasible when the callee is behind a NAT. Secondly, there may be cases where multiple whistleblowers or covert reporters (*i.e.*, several callers), communicate to callees working for a common organization. In such cases, having a commonly shared VPN/SIP server, with high availability, supporting features like voicemail, removes the need for the callee to be always online. Thirdly, it reduces the hassle for every callee to port VoIP server to different platforms.

Therefore we considered an alternative setup where the VPN / SIP server and callee were not hosted on the same host. They were distributed among seven different cloud hosts, positioned across Europe, N. America and Asia. This separation may incur higher OWD between the communication peers, due to the intervening network between the VPN/SIP server and the callee, thus impacting call quality.

To test this, the caller made 1000 individual calls (through Tor) to every callee (seven locations), via VPN/SIP servers (three locations). Similar to the previous experiment, a total of 42000 calls were conducted. We observed acceptable quality with average PESQ 3.81 and average OWD 270 ms, slightly higher than the previous scenario.

Here again, we further tried to optimize the performance using shorter 2-hop circuits. We

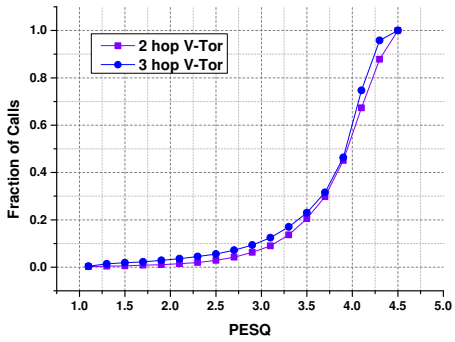

Figure 3.5: V-Tor: CDF of PESQ for Caller Anonymity when server is separately hosted (Scenario II).

saw the average PESQ increased to 3.91, and the average OWD reduced to 210 ms. The results are presented in Fig. 3.5.

The CDF of OWD for V-Tor is depicted in Fig. 3.6. Evident from the results, we observed PESQ above 3 and OWD less than 400 ms in over 92% of the calls.

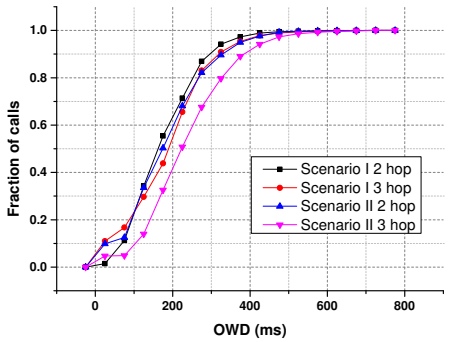

Figure 3.6: V-Tor: CDF of OWD variation for Caller Anonymity in both Scenario I and II.

3.4.2.3 Caller and Callee (two-way) anonymity (Scenario III).

There may be cases where both the caller and callee are positioned in censored networks. In such cases, they may connect via Tor to a VPN/SIP server placed outside their respective censors' jurisdictions. Their calls would be routed via their individual Tor circuits. We thus tried to observe the impact of such scenarios (traffic traversing two circuits) on the overall call quality as the additional network hops may increase OWD.

Similar to previous experiments, we varied the caller and callee locations across seven

countries, while the VPN/SIP server was distributed across three. Each caller initiated 1000 voice calls to a callee, resulting in a total of 42000 calls. For V-Tor we observed an average PESQ of about 3.2 with 81% calls above PESQ 3. However, the average OWD, as expected due to the increased network hops, was about 458 ms. This is slightly above the acceptable limit.

We thus tried to optimize performance by using shorter two-hop circuits. We hence repeated the above tests using two-hop circuits and observed a reduction of the average OWD to 396 ms. The results are shown in Fig. 3.7. In general, for such scenarios, regular three-hop circuits incur higher OWD, compared to two-hop ones. Hence, two-hop circuits seem a better choice for such cases. However, the results of our user study (ref Sec. 3.4.4) shows that users did not have any noticeable performance impact (due to the delay introduced) when both the users conversed via Tor for two-way anonymity.

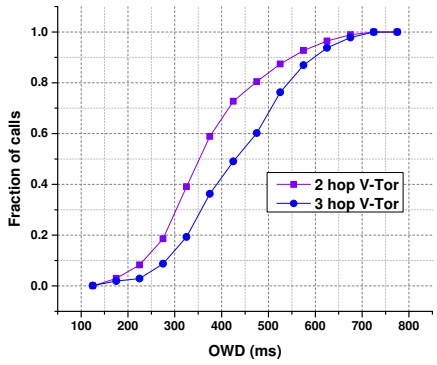

Figure 3.7: CDF of delay for V-Tor setup when two-way Anonymity was achieved (Scenario III).

*To summarize, in all the three scenarios over the public Tor network we observed good call quality (PESQ >*3 *and OWD <*400 *ms) in about* 85% *cases.*

We use 400 ms as a threshold for good call quality following the ITU recommendation for international calls [15]. However, these recommendations also reported some user dissatisfaction even when OWD was between 300 ms and 400 ms. On analyzing the results, we found that more than 80% of calls had OWD below 300 ms. This has been further analyzed in detail in the Appendix. A.2.

3.4.2.4 Experiments involving popular apps

Next, we evaluate the performance of two popularly used VoIP apps, Telegram, and Skype, when running over Tor. Evaluating these apps would be beneficial from the usability point of view as most users generally use these apps for their day to day tasks. Hence using them for anonymous calls would be relatively easy, as they would not require installing V-Tor or M-Tor setups. However, the users might not be completely secure or anonymous, when using these apps as the app maintainers would know the calling parties.

In this experiment, we instantiated 1000 consecutive calls using each of these apps (for both the setups) and computed their call quality. The average PESQ score was 3.8 and 3.54 for Telegram and Skype, respectively. Skype had more than 80% calls with PESQ score more than 3, whereas Telegram had approx. 85% above 3. Overall, there was not much difference in terms of call quality between the two applications.

In our results, we observe that popular voice calling apps perform acceptably well over Tor.

3.4.3 Direct Calls

In the previous subsections, we have established that users in the majority of the cases would obtain good call quality when calls are performed over Tor. However, it would be interesting to compare the performance when calls are instantiated with and without Tor to understand the relative change in call quality.

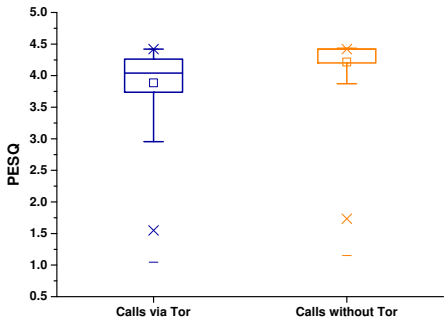

Figure 3.8: Comparison of call quality in terms of PESQ obtained with and without Tor.

Hence, we performed direct calls involving both the setups. In the V-Tor setup, the calls were carried via VPN, and for M-Tor they were carried out directly using mumble. We instantiated 1000 calls for each scenario described in earlier subsections. The results are shown in Fig. 5.6. As can be seen from the results, there is an expected relative drop in performance when moving from non-Tor to Tor setups. This is obvious as there are expected to be much fewer distortions on non-Tor setups. However, the performance of calls via Tor is good enough for the thresholds of performance metrics used in our study (OWD <400 ms and PESQ >3).

3.4.4 Users' Perspective

In our research, we conducted an extensive experimental study to adjudge the quality of voice calls via Tor. The study relied on two standard metrics *i.e.*, PESQ and OWD. These metrics are robust in judging the call quality. In fact, PESQ was established as a substitute for subjective tests (as already discussed in Sec. 2.1.2).

However, in all our experiments, we conducted calls with a maximum duration of 30 s. This is because, PESQ does not support the evaluation of calls whose duration is longer than 30 s [63]. Moreover, the PESQ score evaluation is a non-linear function, with respect to call duration. Hence, the mean of multiple samples of 30 s will not correspond to the PESQ of the actual combined duration call, as clearly stated by ITU [63]. Also, as per ITU recommendations [63] and PESQ [8] draft, a duration of 8 s - 12 s is sufficient to judge the call quality of the channel under test. However, one might argue that the real users' experience might be different for calls above 30 s duration, as these are not tested directly with PESQ. Therefore, we conducted a user study that involved human subjects evaluating the call quality with prior approval from ethical review committee (ref. Appendix A.2).

The user study involved 20 participants[5], which were randomly divided into five groups of four participants each. The groups were given individual sets of calls, each containing three different recorded calls. These calls were conducted over the Tor network, using setups described in Subsection 3.3.1

[5]The location of participants does not have much effect as Tor clients by default select relays in varied locations. However, our participants were spread across three countries.

The first call was 30 s long, and the remaining were two and four minutes long. We calculated PESQ for the 30 s call, and recorded OWD and jitter for longer duration calls (as PESQ cannot be calculated for calls longer than 30 s). The users listened to these recorded files and gave an Absolute Category Rating (ACR) [6] in the range of $1-5$. We then calculated the Mean Opinion Score (MOS) by averaging the score given by all four participants of each group (results of which are summarized in Tab. 3.4). The users reported an average MOS of 4.25, 4.5 and 4.0 for the $30s$, 2 mins, and 4 mins calls, respectively. These results show that, in general, users reported good call quality over Tor.

Group No.	**MOS**		
	30s	**2min**	**4min**
I	4.5	4.5	4
II	4.5	4.25	4.5
III	3.5	4.75	4.25
IV	4.25	4.5	3
V	4.75	4.5	4.25

Table 3.4: MOS by different user groups for varied call length.

Comparing MOS with network attributes: We confirmed the aforementioned MOS values (obtained from users) against recorded performance metrics. For the $30s$ call, we compared the MOS values with the corresponding PESQ scores. We observed an average MOS of 4.25, and a correlated average PESQ of 4.2, thus supporting our observations. Similarly, for the remaining two calls, we found that OWD and jitter were well within bounds for "good quality" (OWD $<$ 400 ms and jitter $<$ 30 ms). The average OWD was $\approx$ 278 ms, and jitter was about 24 ms.

All the above experiments involved the users listening to recorded calls. Thereafter, we went a step ahead, and asked ten users to converse daily via Tor for usual conversations. It must be noted that both the users connected via Tor, simulating Scenario III (two-way anonymity). Users reported a score for each of the calls. Call length varied from 1min to a max of half an hour. This experiment was conducted for about 15 days, and users in the majority of the cases reported good call quality comparable to that achieved via popular VoIP applications rating an average MOS of 4.1.

The results in this user study further strengthen the claim about obtaining adequate call

[6] This is in accordance with ITU guidelines [64] for rating calls in subjective tests.

quality for anonymous calls over Tor.

3.5 Insights from Measurements

We now present explanation of our experimental results, along with other interesting insights we observed from these results.

3.5.1 Overall Performance Analysis

Computing PESQ involves the comparison of the original audio clip, as played out by the caller, with what is recorded at the callee. Effects of network delays, jitters, and losses, reflected in the recorded audio, are captured by this metric. Variations in network conditions, like increase in contentious cross-traffic, leads to an increase in the drops and delays, and thus negatively impacts the perceived quality (and thus PESQ). Besides PESQ, such contention also impacts other network performance metrics like OWD, RTT and available bandwidth.

Our overwhelmingly positive results, with PESQ above 3 and OWD under 400 ms in 85% cases are indicative of relatively low network contentions that can impact VoIP call quality. VoIP calls are encoded at low sending rates (<120 Kbps) and thus require low available bandwidth.

Further, in $\approx 90.6\%$ of our Tor circuits, we observed adequate available bandwidth ($>$ 1 Mbps), as reported by `iperf`. About 95% of these 90.6% circuits supported calls with acceptable performance. This indicates that circuits with sufficient available bandwidth improves the chances of call obtaining good perceptual quality. This can be further understood by analyzing the 90.6% circuits where we measured over 1 Mbps bandwidth. We tabulate the frequency of these circuits, along with their corresponding PESQ scores. As evident from Tab. 3.5, for calls where we obtained good perceived quality ($PESQ > 3$), the frequency of occurrence of circuits where available bandwidth was more than 1 Mbps was also very high. Similarly, we also observed that for bad quality calls ($PESQ < 3$), the instances of circuits obtaining a bandwidth above 1 Mbps were relatively low.

PESQ	1-2	2-3	3-4	4-5
Frequency	6K	22K	162K	259K

Table 3.5: Variation of frequency of Tor circuits (>1 Mbps bandwidth) with PESQ of calls via them.

In general, we observe that with an increase in network contention, both call quality and available bandwidth decrease. This is evident from Fig 3.9. As incidences of high PESQ coincide with cases when the recorded available bandwidth is high, and vice versa.

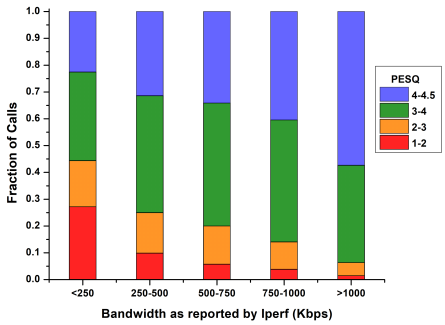

Figure 3.9: Fraction of PESQ scores at different available bandwidths.

Impact on performance over time: Additionally, we also analyzed whether the performance of VoIP calls changed over time. For this, we randomly sampled 100 calls from our measurements and analyzed the distribution of calls based on quality. We repeated this experiment several times, and in *each* iteration, we observed that more than 85% calls always observed good perceived quality. This indicates that the overall distribution is uniform, and there was no observable change over time.

To ascertain the above observation, we plotted a graph incorporating results from our complete dataset. We draw a box plot consisting of variation in PESQ scores for different months. The whiskers in the plot represent 10 and 90 percentile values. The 'x' represents the 99 and 1 percentile values, with '-' representing the min and max values respectively[7]. As evident from Fig. 3.10, we did not obtain any significant change in performance over time. Thus depicting that there was no observable change in performance over time.

[7]All subsequent box plots follow the same style as described.

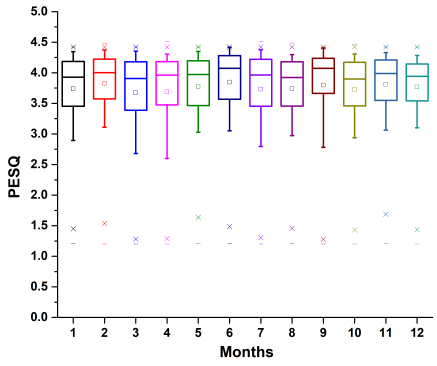

Figure 3.10: Change in performance over time.

3.5.2 Performance Dependence on Types of Relays

We next analyzed whether different type of relays (*viz.,* Guard, Middle or Exit) along with their frequencies of occurrence in circuits, had any observable impact on the VoIP call quality over Tor. To begin with, we distributed the relay frequency into three bins — low (< 10), moderate ($150 - 200$) and high (> 500). Then, from each group, we randomly selected a few entry, middle, and exit relays (around ten each) and manually inspected the PESQ scores of the calls involving them. For each type of relays (in all frequency bins), we observed PESQ scores ranging from $1 - 4.5$, with the majority of them being over 3. Overall, there was no obvious difference in the distribution of calls with different PESQ scores. This observation indicates that PESQ neither depends on any specific type of relay, and nor on their corresponding frequency of occurrence.

To further ascertain our claims and to obtain a comprehensive picture for all our measurements, we plotted the distribution of PESQ scores corresponding to all guard, middle, and exit nodes. The box plot for exit nodes is shown in Fig. 3.11. As evident, the PESQ values show no dependence on the frequency of occurrence of Tor relays. Corresponding to both less (< 50) and more frequently appearing relays (> 500) we observed PESQ above 3 for a large fraction of calls. The trend was similar for guard and middle relays.

Tor churn analysis : Tor relay churn is defined as the rate of relays joining or leaving the network from one consensus to the other (according to Tor Metrics [52] and Winter *et al.* [65]). To that end, we determined whether Tor relay churn had any impact on our results. We calculated

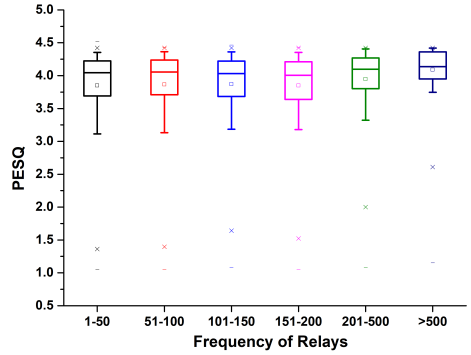

Figure 3.11: PESQ of individual calls vs frequency of exit relays.

the monthly relay churn[8] for the entire duration of our study (ref Fig. 3.12). It is evident from Fig. 3.12 that relay churn was low (an avg. of $\approx 0.2\%$). Further, we also observed that more than 85% of our measured VoIP calls had acceptable quality. Our overall results thus indicate that such a low value of churn has an insignificant impact on call quality.

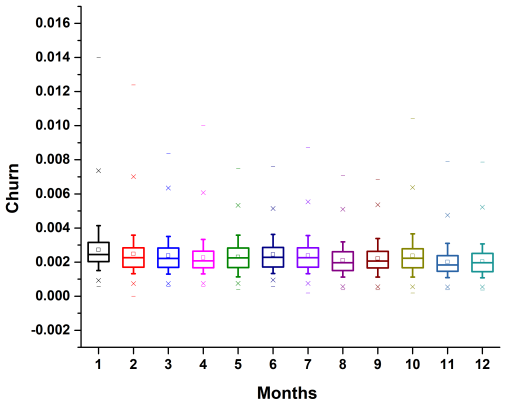

Figure 3.12: Tor relay churn for the entire duration of our study *i.e.,* 12 months.

However, one may argue that a large proportion of exit relays might not support the default ports used by VoIP applications (SIP or Mumble) involved in our study. This might lead to bias in the churn analysis, as we may be considering only a small fraction of all the available exit relays that support VoIP calls. Thus, in all our experiments, we configured our VoIP servers to listen on port 80 and 443 as these are generally allowed on the majority of the exit relays.

Further, we studied the prevalence of exit nodes supporting VoIP applications by default and

[8]The Tor consensus is updated every hour; in one month it is updated around 720 times. The individual box plot corresponds to the change in consensus of these 720 values. Thus, there are 12 box plots each corresponding to a different month.

compared them with those who support port 80 and 443. For this, we analyzed the exit relays in the Tor consensus files for the duration of our study. First, we identified the number of exit relays allowing the default VoIP applications ports (64738 for M-Tor and 1194 for V-Tor), and also the ports used for our study (80 and 443). Once we obtained the number of exit relays allowing these ports, we added their corresponding bandwidth weights and divided it with the cumulative bandwidth weight of all the exit relays. This gave us a normalized value of the bandwidth weight of these exit relays (that allow the aforementioned ports). The results are depicted in Fig. 3.13.

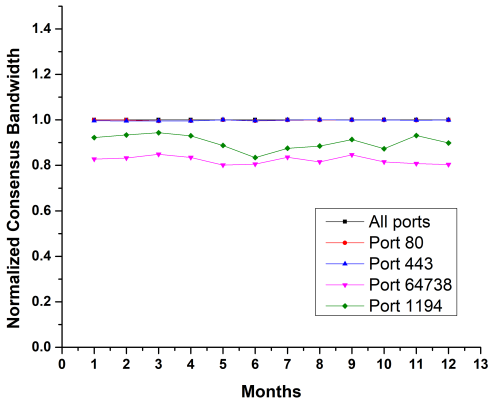

Figure 3.13: Analysis of the bandwidth coverage of exit relays when using default VoIP application ports (64738 for Mumble and 1194 for VPN), compared to when port 80 and 443 were used.

As evident from the results, almost all the exit relays supported port 80 (99.9 %) and 443 (99.6 %). On the other hand, 90.1 % and 83.4 % relays allowed the default VPN and Mumble ports, respectively. Thus, it indicates that our results did not rely on some specific set of exit relays. Moreover, a great fraction of exit relays ($> 80\%$) would support our VoIP applications by default. These results, along with the churn analysis conducted previously, depict that Tor relay churn did not have any significant impact on our results.

Overall, the analysis in this subsection clearly indicates that the performance was not dependent on any specific type of relay. The apparent independence is because, for a call to be of good quality, we require less cross traffic contention, and bandwidth of about 120 Kbps for the entire call duration. The prevalence of such an ecosystem naturally on Tor has already been argued in Sec. 3.5.

3.6 Discussion

In this section, we address concerns like how VoIP performs over Tor when using bridges, using different codecs, *etc.*

Call quality over Tor bridges: Tor bridges [66], are unadvertised entry (guard) relays, whose information is closely guarded. They are conservatively distributed either through *BridgeDB* [67], or covertly via out-of-band means (*e.g.*, emails). Censors may identify and filter Tor traffic using entry node IP addresses and (or) port numbers. User residing in such networks may use bridges to access other Tor relays and set up the circuits.

Hence, to study the impact (if any) when using a bridge, we performed 1000 measurements, using the V-Tor setup. In this experiment, we used a bridge to connect to a new Tor circuit each time. The average PESQ score was about 3.7. In about 85% calls, we observed good performance (PESQ >3.0 and OWD <400 ms). It must be noted that, we restricted our measurements to a single bridge node as they are scarce.

Impact of codecs: Codecs define the way audio is encoded and decoded to transmit them as packets. Hence we measured the impact of different codecs on call quality. The calls for this experiment were conducted over the real Tor network. We used some popular codecs for our

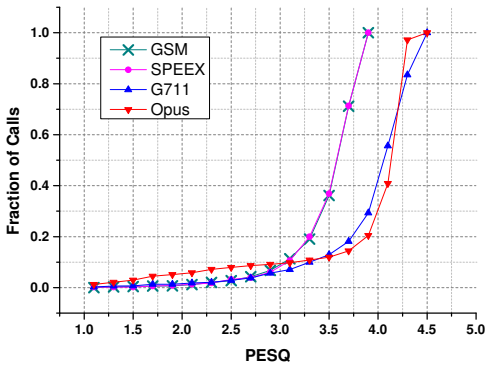

Figure 3.14: CDF of PESQ scores when different codecs were used.

evaluation, *viz.*, *Opus, G.711, Speex, and GSM*. The codecs were selected with the help of configuration changes in the Freeswitch SIP server. We conducted 1000 calls corresponding to each of these codecs and measured their PESQ scores (ref. Fig. 3.14). We observed that the

fraction of calls with PESQ above 3 were roughly equal ($\approx 85\%$) in all the cases. Thus any of the tested popular codecs could have been used for initiating good quality calls. However, GSM and Speex are lossy codecs, compared to lossless codecs like G.711, and thus may not provide very high quality calls (PESQ >4) [68]. Besides, such lossy codecs encode at lower bitrates compared to the lossless ones[9]. Thus, they may be used for Tor circuits with low available bandwidth to receive adequate call quality.

Thresholds for PESQ: The MOS scale proposed by ITU delineates perceived performance through sharp integral differences. *E.g.*, two corresponds to "annoying but usable" and three corresponds to "fair". However, it does not extend such annotations for values in-between. Further, upon listening to a few calls manually where PESQ was between 2.8 and 3.2, we observed no audible differences. Moreover, recent studies [69] indicate that humans may be unable to differentiate the quality of samples, when the difference between their MOS scores is less than 0.4.

In our study, we *conservatively* selected a PESQ over 3 as "acceptable", and anything below it as not (in accordance with ITU). We also observed a significant 7% cases which maybe categorized as "annoying but usable." Finally, we also observed about 8% with PESQ under 2. ITU classifies these as "totally unacceptable." Thus, we believe that actual user experience may be even better than what our study reports (as evident from the conducted user study in Subsec. 3.4.4).

Coverage of Tor relays: We recorded and analyzed Tor circuit information for all our experiments. We now present some interesting insights we observed from this analysis. A total of about 600 K Tor circuits were created during our study.[10] These circuits involved a total of 6650 unique Tor relays. Prior research (Rizal *et al.* [6]) reportedly used only about 298 relays that too restricted to Europe.

Selection of caller-callee endpoints: Throughout our experiments, we used endpoints either as cloud hosts or in-lab machines. All of these were sufficiently provisioned for a voice call

[9] GSM and Speex encode at bitrates <40 Kbps whereas G.711 encodes at 84 Kbps.

[10] The number of Tor circuits are slightly higher than the total number of experiments as the two way anonymity experiments involve creating two Tor circuits for a single call.

with more than adequate bandwidth at the caller-callee end. However, one might argue that our results might be biased as all our endpoints may have sufficient bandwidth for VoIP calls. But in general, the performance bottleneck was introduced at the Tor relays. As already described in Sec. 3.5, there were a significant number of Tor circuits which observed a low bandwidth (< 1 Mbps). Hence, even if the endpoints are well provisioned, it does not bias our results, as mostly the Tor relays were the bottlenecks.

3.7 Conclusion

Real-time anonymous VoIP calls are of interest to privacy and anonymity conscious citizens, whistle-blowers and covert reporters, *etc.* However, existing research on performance evaluation of VoIP calls over Tor is not comprehensive. They contraindicate transporting VoIP packets over Tor and thus favored novel architectures to support anonymous calling. Moreover, there does not exist a functional system to achieve the same. Additionally, the costs involved in recruiting volunteer operated relays (like Tor) along with users, and managing such a system, might outweigh the benefits.

Thus, it was essential to identify the causes of poor voice call quality over Tor by observing how the interplay of various network attributes (RTT, available bandwidth, *etc.*) impacts VoIP quality. To that end, we conducted a longitudinal study (spread across 12 months). It involved extensive testing of about half a million voice calls over Tor, including a user study, using various Tor circuits, peer locations, popular apps, *etc.* To our surprise, in over 85% cases, we observed good performance (PESQ>3 and OWD <400 ms), with only under 8% cases which were totally unacceptable (PESQ<2). The results of the user study also corroborate our findings. Our study is the first to demonstrate that anonymous VoIP calls are indeed possible using Tor.

Although, Tor can be used to perform anonymous VoIP calls along with its traditional usage of accessing web content, but various adversaries attempt to block Tor and in certain instances succeed in doing so. Thus, there is a need to build systems that are difficult to block in practice. Hence, in the next chapter we attempt to build a decoy routing system as it provides effective blocking resistance.

Chapter 4

SiegeBreaker: An SDN Based Practical Decoy Routing System

4.1 Introduction

Free speech on the Internet is a political battleground. On one hand, several governments claim that their sovereignty extends to cyberspace, and that they should control the content seen (or written) by their citizens. On the other hand, as censorship is frequently used to silence the opposition [70], access to free speech online is considered a human right by the United Nations [1]. This tension between censors and free speech advocates has led to an "arms race": users use proxy based services (*e.g.,* VPNs and Tor [71]) to access content on the Internet, and in response, censors design stronger counter measures [72]. Even the best anti-censorship solutions, such as relay addresses [73] and protocol obfuscation [74, 75], are temporary measures [76].

Decoy Routing [9, 77, 78] is an attempt to break out of this "arms race", through the use of "smart routers" that double as proxies. DR clients – residents of censor countries, who need to access a blocked site – start by sending packets addressed to allowed sites (the *overt destination (OD)*). These packets *appear* innocuous, but actually carry a covert cryptographic message, carefully chosen to make it hard for the censor to distinguish the packets from regular TLS messages. Once safely past the network boundaries of the censor, these packets are identified by

DRs that hijack the client's connection and redirect that flow to the *covert destination (CD)*, the actual censored site the client wishes to contact. Even if a DR is discovered, it is hard for the adversary to "Route Around Decoys" [79, 80] without losing connectivity from a major part of the Internet.

Interest in Decoy Routing has led to a rich body of prior proposals [78, 77, 81, 82, 83, 84], but *not* led to the development of practical and efficient DR systems. There are several challenges that must be addressed before DR becomes widely adopted in practice.

Firstly, Internet applications perform poorly when using existing DR systems. As reported in previous research, DR clients experienced overall low throughput [85] and high latency [84]. This is because existing solutions are implemented on commodity servers (rather than real routers); they cannot serve traffic at line rates while at the same time matching network flows using cryptographic operations. On the other hand, it is virtually impossible to simply port existing solutions to conventional routers (which are not designed to be reprogrammed).

Secondly, existing proposals assume (unfounded) mutual trust between third-party DR operators (analogous to Tor maintainers), and the "friendly" ISPs (who simply *host* the DR). These proposals involve DR operators inspecting *all* flows traversing the ISP. Thus, DR operators may compromise the privacy of non-DR flows passing through "friendly" ISPs, (*e.g.,* by recording their packet headers, metadata, or even content), breaching the trust extended to them.

A third issue is identifying the networks where DRs may be positioned (to intercept requests from several users), and providing incentives for these network operators to deploy DR. Finally, a fourth issue is defending against various attacks (traffic analysis *etc.*) from different adversaries.

The DR community has begun research efforts to address the third [79, 80] and the fourth issue [83]. However, the first two problems – assuring *performance* for regular DR flows, without compromising *privacy* of non-DR flows – remain open challenges to practical DR.

We thus propose the first practical system to answer such concerns: *SiegeBreaker (SB)*, a Software-Defined Network (SDN) [86] based, efficient, and privacy preserving DR system. It works as follows:

- The users of SB signal the Decoy Router through an email, addressed to an email ID associated with the SDN controller (that manages the SDN switches). The email bears encrypted information, that can only be decrypted by the SDN controller; at a later step, this information authenticates a legitimate DR client to the controller.
- After deciphering a DR request, the controller re-configures the switch (*i.e.* the Decoy Router), to redirect all subsequent TCP packets, *of only the SB client – OD connection*, to itself.
- These TCP packets carry covert information (indicated in the previous email message) that authenticate a legitimate DR request, *without* requiring the inspection of all other flows traversing the DR (as required in all previous proposals). This drastically reduces the inspection load on the controller.
- After authenticating the client, the controller re-configures the switch to divert subsequent SB client – OD packets (*i.e.* only the DR flows) to a third-party *secret proxy (SP)*.
- These subsequent packets help covertly establish a fresh session key with the SP, and expose to it the URL of the CD. The SP then communicates with the CD and sends the responses back to the client, ensuring reliable and efficient delivery of packets.

Overall, the protocol ensures that the third-party DR operator managing SP, observes only the DR requests, without having any knowledge of the non-DR flows, thereby preserving their privacy.

SB proves to be a practical and privacy-preserving DR system. It builds upon the idea that DR is not a monolithic task. Thus, it distributes the tasks of DR over three loosely-coupled, synergistic modules: (1) The SDN *controller*, that orchestrates the switch(es), independently identifies DR requests and re-configures SDN switches on-the-fly. (2) The *switch*, which primarily forwards packets between the client and the SP, *seldom* engaging the controller. (3) The *SP*, which focuses on communicating with CD, along with other tasks to ensure secrecy and reliability. Such a design not only leads to a massive improvement in performance, but also ensures that the DR system (and its operator) only sees those few flows which are associated with DR requests.

Our major contributions can be summarized as follows:

1. The prototype of SB, a practical and efficient DR solution, which achieves multiple goals:
    - Modularity. DR requests are handled by three loosely coupled entities – the SDN controller, the switch and the SP – reducing computational load on each.
    - Privacy preserving signalling. Using the above modular design, SB inspects a small fraction of all traffic and serves *only* the DR requests.
    - Dedicated hardware. SB uses SDN switches (instead of commodity servers proposed in previous designs) to divert flows efficiently at line rates.
    - Reliable communication. SP provides reliable and efficient client–CD communication, by preserving the characteristics of TCP like behaviour (when there is no actual TCP connection between them).
2. Comprehensive performance analysis of SB, on real testbeds (involving a commercial SDN switch), shows near-native (TCP) performance for *both* web surfing and bulk downloads, even in the presence of 50K parallel flows or 15Gbps network-traffic on the SDN switch. In ideal conditions, SB is capable of serving individual client requests at line rates of $\approx$ 1 Gbps.
3. Thorough security analysis of the protocol, showing how SB is resilient to a variety of advanced attacks (such as forced asymmetry [87]).

Finally, we note that existing efforts to deploy DRs [78, 79] could be very costly — even modest proposals [80] involve replacing about 11,000 routers across 30 ASes with DRs. However, in an SDN-based AS, *any* switch could double as a DR on-demand, without disrupting regular routing. This solves the problem of DR placement *within an AS*. We further demonstrate how the client could use inconspicuous protocol messages to covertly signal the controller, and help identify the best switch for installing flow rules, in Subsec. 4.4.4.

4.2 Related Work

Existing DR systems follow the general pattern described in the previous section, but vary in their choice of secret handshake, side channel, and whether they take additional measures to suppress unwanted messages from the OD to the client (which might make the censor suspicious). We

now provide a brief survey of existing DR systems.

Karlin *et al.* [9] proposed an initial DR architecture, called Curveball. However, the first-generation DR implementations include Cirripede [78] and Telex [77]. Cirripede uses the TCP Initial Sequence Number (ISN) field as a covert channel for registering the client to the *Registration Server*. This server provides a fixed set of sequence numbers that the client may use in subsequent connections. In Telex, the client embeds a cryptographic tag (created using the Telex station's public key) in the TLS nonce field of the Client Hello message to OD. Seeing a tag, the Telex station establishes a shared secret with the client (using Diffie-Hellman key exchange), which is subsequently used to covertly proxy traffic to the filtered sites. Telex requires *both* sides of the client–OD communication be intercepted; it is, thus, fragile where protocol messages (and acknowledgements) between the client and OD traverse asymmetric network paths. The recent systems by Bocovich [83, 88] and by Nasr [82] build upon the signaling architecture of Telex and Cirrepede respectively.

Tapdance [81], an improvement over Telex [77], uses chosen-ciphertext steganography to signal the DR in an incomplete HTTPS request destined to the OD. Further, this request eliminates the need for inline blocking of the OD [77] and the requirement to intercept both request and response, as the OD never responds to such requests. Recently, Frolov *et al.* [85] partnered with a small ISP (and a university) to deploy Tapdance for practical usage. Their experiments reveal that incomplete HTTPS requests, often terminate in roughly 30 seconds, requiring frequent re-negotiation. This issue, coupled with the average Tapdance client throughput of ≈ 5KBps, makes it unsuitable for accessing much of the web content. This reveals how existing DR solutions struggle with real-life deployment. Even on a very small ISP, with three uplink routers (compared to thousands, for a backbone ISP like Level-3 or Cogent [80]), there are serious performance problems. Deploying at scale would require a much larger infrastructure, capable of monitoring millions of flows, identifying the DR requests, and providing them proxy service.

Bocovich *et al.* [83] took a different approach where they address latency-based and website-fingerprinting attacks. They achieved resistance to such attacks by sending the covert content in the leaf elements (images *etc.*) of the webpage served by OD. The client maintained an active connection with OD, and only the leaf content in the response of the OD was replaced with covert

content. Further, as Slitheen still required analysing content in both the reverse and forward path, the same authors in their subsequent paper [88] proposed *Gossip*, a new protocol which allows all existing routing symmetry-dependent DR systems to work despite path asymmetry. Another approach was proposed by Nasr *et al.* [82], in which the *downstream* content, *i.e.* replies from OD, are used to detect DR, rather than the (upstream) requests. This helps in the mitigation of strong RAD [87] attacks. However, both Slitheen and Waterfall were developed and deployed on commodity servers, which are *not practical* to scale in an ISP.

In brief, existing DR solutions have not succeeded in utilizing commodity routers to match/decrypt messages and serve as proxies on-demand at an ISP scale.

4.3 SiegeBreaker vs Prior Research

Properties	Siege-Breaker	Curve-ball	Telex	Cirripede	Tap-Dance	Rebound	Slitheen	Waterfall of Liberty	Secure Asymmetry
Implementation on real routers	●	○	○	○	○	○	○	○	○
Real Internet Workload Performance	●	○	○	○	○	○	○	○	○
Privacy preservation of non-DR clients	●	○	○	●†	○	○	○	●†	○
Handling asymmetric routing	●	○	○	●	●	●	○	●	●
Resistant to replay attacks	●	●	●	●	○	●	●	●	●
Resistant to latency analysis	○	○	○	○	○	●	●	●	●
Resistant to website fingerprinting	○	○	○	○	○	○	●	●	●

Table 4.1: A comparison of different features of existing DR schemes. ● - feature supported; ○ - feature unsupported; ●† - feature supported with some assumptions.

We now describe how SB significantly differs from prior attempts at DR (summarized in Tab. 4.1).

- ***Implementation On Real Routers:*** Existing DR proposals use commodity servers as Decoy Routers – either by using the server as a router [78], or by attaching the server to one (through a wiretap) [81]. A server running a non-realtime OS, that has to inspect *all* the network connections to identify DR requests, and also proxy such requests to the intended CDs, is a clear bottleneck. SB uses hardware (SDN) switches instead of commodity servers, allowing it to process ISP-scale traffic at line rates. Although, Tapdance was deployed in an ISP in

2017 [85], it did all the processing on a server attached to a router. It is important to note that, unlike SB, their implementation was not on the router *itself*.

- ***Real Internet Workload Performance:*** Among existing DR prototypes, Frolov [85] demonstrates the practical use of Tapdance, inside a university and within an ISP. However, they reported overall low client throughput (5KB/s), and precludes testing with large files. Other approaches have restricted themselves to downloads of a few (less than ten) small (< 1 MB [78]) webpages, in a controlled laboratory environment [77, 81], and even so, report over half a minute to download the home pages of popular sites (< 1.5 MB in size) [84].

  SB's prototype is *extensively* tested for large files (up to 1GB) and high client link bandwidths (up to 1Gbps). Even under heavy cross-traffic conditions, and with CD and OD over the Internet, SB's client requests can be served at line rates of 1 Gbps. SB's performance was comparable to direct TCP downloads (near-native) in *every* scenario we tested.

- ***Privacy Preservation of non-DR clients:*** All prior approaches require the DR to inspect both the DR and non-DR flows, either by analysing TLS or TCP SYN packets[1]. This allows the DR volunteer hosts (*e.g.,* Telex/Tapdance/Slitheen station) to also see what sites other non-DR clients are visiting. This may be considered an unwelcome intrusion on their privacy. We introduce a novel signaling scheme that allows the controller to easily isolate DR requests (*i.e.* re-configure the SDN switch to selectively forward DR packets to the SP). Not only is this efficient (reduces load on the system) but also *privacy-preserving*, *i.e.* removes the need to inspect non-DR flows.

- ***Handling Asymmetric Routing:*** Routing of traffic in the Internet is not always symmetric, and most DR systems need to intercept client–OD traffic in both directions [9, 83, 77]. Some systems survive route asymmetry through special schemes (such as a gossip protocol) [88]. However, a few DR systems [81, 84] have no trouble with asymmetric routes; SB is one of these robust systems.

- ***Reliability and Efficiency:*** In all DR systems, the DR *hijacks* client–OD TCP connections.

[1] In Cirripede and Waterfall, if we assume that the registration server is maintained by the ISP's operator, then they could inherit this property. Additionally, how SDN implementation can be used to complement existing DR schemes (with this feature) is elaborated in B.2.

DR traffic between the client and the CD lacks the reliability and flow-control guarantees of regular TCP connections, and is thus easily impacted by background Internet cross-traffic. SB emulates TCP's message transmission mechanism, to ensure reliable, in-order and efficient delivery of packets to the client. This has significant positive impact on clients' performance.

- ***Limitations of SiegeBreaker:*** SB is not resilient to latency based [87] and website fingerprinting attacks [89]. Moreover, we have not tested SB in an ISP. (Such testing has only been demonstrated for Tapdance.) One may ask that in an SDN-based ISP, on which switch should the redirection rules be installed? As a solution, we have proposed a novel scheme (in Subsec. 4.4.4) using which a client can covertly signal the controller about the appropriate switch. Our proposed scheme, however, may be susceptible to fingerprint attacks by a determined adversary. In such scenarios, the controller would install rules on all the switches. But, due to the associated timeout, these rules would be eventually purged out from all the switches, except the one that would actually intercept the DR flows. This is explained in detail in Subsec. 4.4.4. However, we acknowledge, it still increases overhead on the system in terms of installing the unwanted rules whenever a user requests DR service.

4.4 System Design

4.4.1 Threat Model

We consider two types of adversaries for our system. Our primary adversary is a powerful nation-state that censors its citizens' Internet access. Not only can adversary censor regular Internet traffic using any known technique (DNS injection, IP address/URL filtering, DPI, *etc.*), it may also covertly monitor network packets. Further, in an attempt to detect/disrupt circumvention systems alone, it may also subtly manipulate network packets that cross its boundaries, without disturbing regular Internet users. Also, for the functioning of SB, we assume that users have access to *some* form of regular email service (either webmail or SMTPS).

We also consider a secondary (Byzantine) adversary — a motivated individual or group, who intends to disrupt normal network routing. This adversary sends forged messages, aiming to

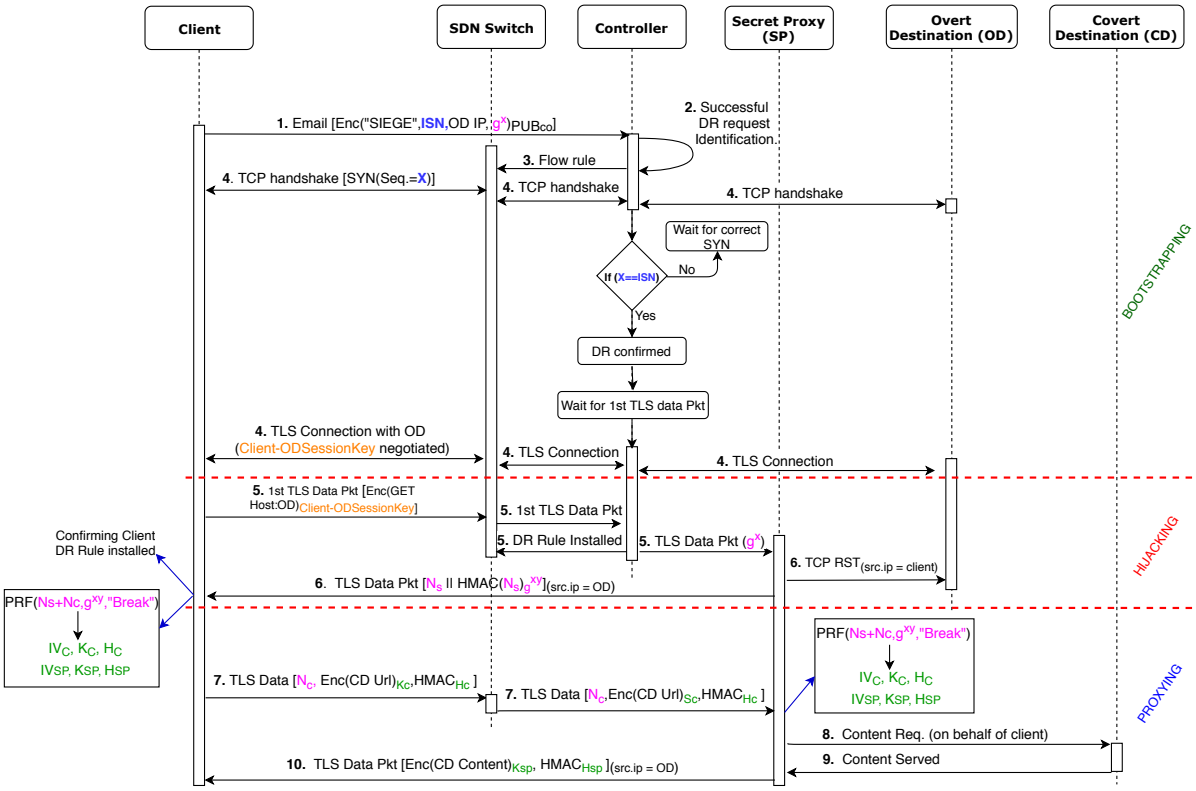

Figure 4.1: SB Protocol: The numbered arrows correspond to the various protocol messages, exchanged between the client, the SDN switch (acting as DR), the OD, the SP, and CD, as described in Subsec. 4.4.2.

install unwanted rules in SDN switches, or in any other way disrupt network routing through attacking the SB infrastructure.

4.4.2 SiegeBreaker Protocol

Pre-requisites

The SB architecture assumes a network of centrally controlled SDN switches, configured to function as regular routers (*i.e.* they forward packets based on their destination IP address). It is assumed that the client *a priori* has access to the following entities:

1. Public key (PUB_{CO})of the controller's 2048-bit RSA public-private key-pair.
2. The 256-bit DH public exponent g^y of SP.
3. The controller's email ID.

Protocol Description

SB's DR protocol comprises roughly of three phases—*viz. bootstrapping*, *hijacking* and *proxying*. We now present a step-by-step walk-through of a DR session, with reference to Fig. 4.1.

Bootstrapping (Phase I):

1. The client initiates DR by sending an email to the controller's email address. The payload of this email contains the DR request. It is encrypted with the public key of the controller, and thus not understandable to the client's email provider or the adversary.

2. The controller, on receiving an email, attempts to decrypt the content using its private key, and searches for four fields embedded in it: **(1)** The magic word "SIEGE", signifying DR request. **(2)** A TCP Initial Sequence Number (ISN) that the client will eventually use while handshaking with the OD. **(3)** IP addresses of client and OD **(4)** A DH exponent public value (g^x), the client derived using privately chosen number x.

3. On successfully identifying a DR request, the controller installs a rule on the SDN switch, so that *all* client–OD packets are redirected to it (the controller). This *inspection rule* will expire after a hard timeout.

4. Subsequently, the client initiates a fresh TCP handshake with the OD. The SYN packet of the handshake bears the *same* ISN, as the one sent in the initial email.

   This packet is redirected to the controller (as per the rule installed in step 3), who checks that its ISN matches the ISN specified in the initial email and forwards it to OD. Thereafter the client completes the TCP connection, and initiates a TLS handshake with the OD, negotiating the session key (Client-ODSessionKey).

   This initial ISN match is the vital step that authenticates the client and identifies the DR requests, without the need to inspect any other flow.

   In case the client's SYN packet does *not* have the expected ISN, the controller continues to poll the packets of the client–OD flow until the timeout period of inspection rule lapses, causing it to expire.

Hijacking (Phase II):

5. After completing the TLS handshake, the client sends a TLS data packet, carrying the `GET` request (with OD as the host field), encrypted with the session key that was negotiated with the OD.

   The inspection rule is still in place, so this packet goes to the controller which *replaces the payload* of the TLS data packet with the client's DH exponent g^x (which it received in step 2) and the OD IP. It then forwards this packet to the SP.

   Further, the controller also updates the redirection rule on the switch, to divert *all subsequent packets of the flow* to the SP. This rule matches the client-OD flow based on the client's source port, client IP and OD IP as seen in the TCP SYN packet. We denote this rule the *SP redirection rule*, and associate an idle timeout with it. Once a rule is installed for a specific client-OD pair, no other rule, for the same pair, would be installed until the said rule times out. Thus, clients behind a NAT firewall who try to use the DR service, would require selecting different ODs. This ensures that all such clients can simultaneously access the DR services. This is explained in detail in Subsec. 4.6.1.

6. The SP, on receiving the first TLS data packet, assumes that the client intends to use DR.

   SP terminates the existing client–OD connection by sending a RST packet (spoofed as client, with correct sequence and acknowledgement numbers) to the OD IP[2] (obtained in step 5).

   Further, the SP derives a pre-master key (PMK) (g^{xy}) using client's g^x (received via the first TLS data packet) and its own DH private number, y. The SP (spoofing as OD) then crafts a TLS data packet carrying a random nonce N_S, along with its HMAC computed using PMK.

   To the censor, this appears to be a regular TLS data packet, carrying random bits. The client, however, treats these bits as a nonce N_S and calculates its HMAC. It derives the PMK (g^{xy}) using the private DH number x, and the publicly known g^y. Successful verification of the HMAC allows the client to confirm that the DR request was successful.

Master-key derivation: Following a successful validation of the nonce N_S, the client selects its

[2]The RST causes the client–OD connection to be forcefully terminated, without any responses being sent to the client that may raise the censor's suspicion.

own nonce N_C. Using these nonces and the PMK, the client derives a 6-tuple key (IV_C,K_C, H_C, IV_{SP}, K_{SP}, H_{SP}). IV_C,K_C and H_C are the IV, cipher and HMAC keys for the client, while IV_{SP},K_{SP} and H_{SP} are those of the SP. These keys are generated using a Psuedo Random Function (PRF) involving SHA-256 hash function (Referring to TLS v1.2 [90]).

Proxying (Phase III):

7. Thereafter the client crafts a TLS data packet carrying N_C, the CD URL (encrypted with key K_C using 256-bit AES in CBC mode), and a HMAC of the URL and N_C (using HMAC key H_C).

   The client sends this packet, addressed to the OD. En-route the packet encounters the switch that redirects it to the SP.

8. SP, on successful reception of the aforementioned TLS data packet, extracts the nonce N_C. Using N_C, N_S and the PMK, SP also computes the same 6-tuple as the client. Upon successful HMAC validation, the SP decrypts the URL (using K_C). Finally, SP connects to the CD and requests data. The SP focuses entirely on serving DR requests, *without the burden of identifying them from among all flows via costly cryptography operations*.

9. CD serves the requested content to SP.

10. The SP encrypts these responses with key K_{SP} and signs them with HMAC key H_{SP}. It then sends them back to the client, spoofing the source IP address of the OD (maintaining the state of Client–OD connection). This keeps up the pretense, to the client's censor ISP, that the client is communicating with the OD. The same session, can also be used for requesting content from various other CDs (before the idle timeout expires).

This walk-through raises several questions, such as why email was used as a covert channel? Among multiple switches, exactly on which switch(es) the controller would install the inspection/redirection rules? What rule timeout values should be selected in our system? Given that there is no true TCP session between client and CD, how do they compensate for dropped packets *etc.* We now explain the design decision of using email along with how SB would select switch(es) to install the rules and discuss the remaining concerns in Sec. 4.6.

4.4.3 Improved Covert Signalling

Existing DR implementations use TCP initial sequence numbers (ISNs) [78], the ClientRandom field inside TLS client hello [77], and the encrypted body of an HTTPS request [81] as covert signaling fields. These schemes require analyzing either *all* SYN packets [78, 82] or TLS flows [81, 77], as the covert patterns are embedded in the packets sent from the DR client to the OD. Inspecting large volume of traffic for identifying the embedded covert patterns, in innocuous looking packets, make it difficult for the censor to detect DR requests. However, these well thought signalling schemes pose new performance challenges for the DR users.

All existing DR systems rely on commodity servers to analyze the large volume of traffic [91]. This may pose as a performance bottleneck as NICs of these servers are not built to handle such high speed traffic. Further, these systems analyze all flows, including the non-DR ones, to detect DR requests. This may inadvertently compromise the privacy of non-DR users. In principle, SB can easily adopt any of the existing signalling schemes. However, SB aims to reduce the burden of analyzing all flows, while still retaining the same standard of unobservability (from the censor), compared to existing architectures. Additionally it also aims to preserve the privacy of non-DR users.

Thus, our covert signaling indicates the controller about the upcoming DR flows, such that it inspects *only the potential* DR traffic, reducing the amount of traffic to be analyzed. For this, a client can rely on any out-of-band channel (*e.g.,* IMs, SMS and email *etc.*). In our present implementation, SB clients use an email to covertly signal the controller. This email contains client's source IP, OD's IP and the TCP ISN that would be used by the client in the subsequent DR request. Thereafter, the controller installs inspection rule for only flows with indicated client IP and OD IP address, *i.e.* a potential DR flow, *disregarding the rest of flows*. For potential DR flows, the controller matches the ISN in the SYN packet, with the one indicated in the email. A successful match, confirms a DR flow, and the SP redirection rule is installed on the switch.

Thereafter, only the DR flows are diverted to the SP, which has no knowledge of non-DR flows. This final step helps preserves the privacy of non-DR users from the DR volunteers maintaining the SP.

It must be noted that, as an alternative to ISN, the TLS ClientRandom may be sent in the email. The controller would thus confirm DR flows by matching the content of TLS ClientRandom in ClientHello (of potential DR flows), to the one received in the email.

On the use of email: In the past, email has been successfully used as a covert channel for transporting censored content [92, 93, 94, 95]. Similar to such efforts, we also assume that users have access to some form of TLS supported email (either webmail or regular SMTPS). However, in our design email is not directly used as the data channel; it is a *control* channel, used for signalling the SDN controller.

A determined adversary may attempt to snoop and block all emails destined to controller's mail ID. Since the communication is TLS encrypted, our approach is not vulnerable to wire sniffing adversaries. They are only vulnerable to a very powerful adversary who can assume control over the clients' email services. However, even when the email provider is controlled by the adversary, one can issue unique email IDs to each individual client, as proposed by Houmansadr *et al.* [94]. This makes it impractical for the censor to learn and block *all* such email IDs.

Moreover, an adversary may cripple the system by randomly delaying suspicious emails, thereby delaying the installation of the inspection rule. Oblivious DR clients' packets would thus reach the switch before this rule is installed. Hence, they would go undetected by the controller, failing to avail DR service. Further, our email body contains four fields and thus may have a fixed length, thereby raising suspicion. However, we make it difficult for the adversary to identify such emails. *E.g.,* we pad the email body with random nonces.

However, a determined adversary may find some other identifiable patterns in the encrypted (registration) email. In such cases, we can resort to using other carriers, such as images, PDFs *etc.,* to steganographically encode the covert signal. as already used in existing anti-censorship systems [96, 97]. However, our current implementation does not incorporate this feature.

In the extremity, an adversary may attempt to delay every email (both DR and non-DR). In such cases, SB may be tailored such that the controller sends an acknowledgment email back to the client. Reception of this email, confirms the availability of DR services to the client. As a

drawback, this may incur a delay in the bootstrapping phase. However, in the presence of such disruptive adversaries, the email confirmation assures availability, albeit at cost of such delays.

4.4.4 Auxiliary Signalling Scheme

In SB, the controller needs to install `OpenFlow` rules to redirect the packets of a client–OD flow to itself or the SP. However, as a network will have multiple SDN switches, SB poses a new problem: how can the controller know *which* switch to install these rules on?

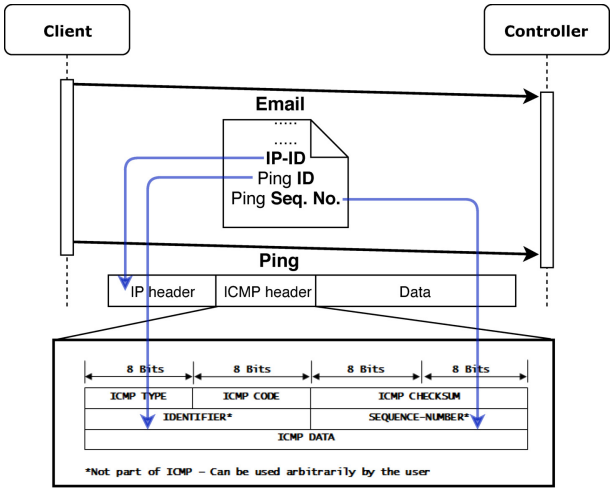

Figure 4.2: `Ping` packet assisting the controller in selecting the appropriate switch to install redirection rule.

The simplest solution would be to flood *all* switches in the SDN with the redirection rules. But intuitively, this approach maybe expensive considering rule installation time and the finite memory of the switch. (Note that a rule is added for *every* client–OD flow. They cannot be aggregated.) We therefore propose a secondary signal to achieve the same. The client sends a crafted `ping` packet as a covert signal, to help the controller identify the switch it should install the DR rule on (ref. Fig. 4.2). The overall approach is explained as following:

1. Switches are bootstrapped with a rule that forwards all `ping` packets to the controller.
2. While crafting the email to the controller, the client adds three more fields: (i) IP-Identifier (from IP header), (ii) `ping` sequence number and (iii) `ping` identifier (from ICMP header).
3. Next, the client sends a crafted yet innocuous looking `ping` packet to the OD (with three fields described in previous step). En route to OD, decoy AS's SDN controller receives the

`ping` packet and compares the aforementioned three fields to the one sent in the initial email. A successful match confirms that the `ping` packet is indeed from the authenticated client.

4. At this step, the controller selects the switch from which it received (and validated) the ping packet. Thus the redirection rules are installed on this switch, assuming the subsequent client-OD packets shall also arrive at the same switch.

Selection of three fields: The goal of the controller is to correctly associate the ping packet to the legitimate DR client that initially sent the email. An adversary may try to brute force this association by sending fake ping packets (spoofed as DR client), guessing the three header fields. In case the adversary succeeds, and the route taken by packets of the adversary differ from that of the client, the controller would end up installing a rule on a switch which might not appear on the client–OD route. This renders DR service unusable for intended DR clients. With our architecture, adversary needs to correctly identify 48 bits (IP-ID, ping seq. no. and ping ID, each being 16-bit field) making the brute force attack highly impractical (requiring ≈ 281 trillion `ping` packets per client–OD pair.)

Indistinguishable Pings: We only rely on the IP-ID, sequence number and identifier fields of the ping packet for covert signalling (which are by default random numbers). We keep rest of the fields (and payload) identical to *pings* generated by standard OSes. Thus our ping packet appear indistinguishable from regular ping packets generated by popular OSes.

However, we acknowledge that sending `ping`s for covert signalling, before accessing DR, may be classified as a pattern (*i.e.* a fingerprint attack). We try to make it difficult for the adversary to detect such patterns by making these `ping` packets innocuous. Thereafter, we introduce random delays between `ping` packets and the corresponding DR requests.

Our auxiliary signalling mechanism is prone to fingerprint attacks. However, SB can also function without this scheme. In the absence of this scheme, the rule would be installed on all the switches. But, these would be automatically purged, after a short duration (due to timeouts) from all the switches, except the one which later identifies the actual SB clients packets. However, the installation of rules on all the switches may incur an additional memory overhead. This may eventually impact the total number of clients that can be supported by the switches. We thus

analyzed the number of clients that can be supported simultaneously by the SDN switch. The `HP3500yl` SDN switch, that we used for evaluating our system (described ahead in Sec. 5.4), supports 74K table entries [98]. More advanced SDN switches [99] *e.g.,* `HP10500` series can support $1,152,000$ table entries (*i.e.* $1,152,000$ openflow clients). Such switches can transport a total of about 3.8 Tbps [100] traffic. Moreover, the SDN switches that we used, do not impede performance when the number of clients (or entries in the table) increases. These are built for commercial deployment within large ISPs, and can easily handle a large volume of traffic without any slowdown. Thus, we believe that our proposed system will be suitable for deployment.

4.5 Experimental Evaluation

In this section, we describe the experiments conducted to evaluate the performance of SB. First, to test the efficacy of the protocol, we tested SB on DETER [101] testbed. We describe this in detail in Appendix B.1. Next, to comprehensively evaluate the performance, we tested SB on a physical testbed (using a commercial SDN switch, *viz.* `HP3500YL`). Our performance tests are broadly divided into two categories:

1. *Controlled experiments*, which were designed to test the *robustness* of the complete system including the client, proxy, and the SDN switch in a lab setup.
2. *Internet experiments*, which were designed to show that our system works well for *realistic scenarios* — *i.e.* downloading content from Internet websites that were otherwise blocked.

We begin by describing the setup used in the experiments, followed by the tests performed, and their respective results.

Experimental Topology:

SiegeBeaker was tested using the setup in Fig. 4.3, for controlled experiments (where all the entities were hosted inside our lab), and the setup in Fig. 4.7 for Internet experiments (when OD and CD were websites hosted over the Internet).

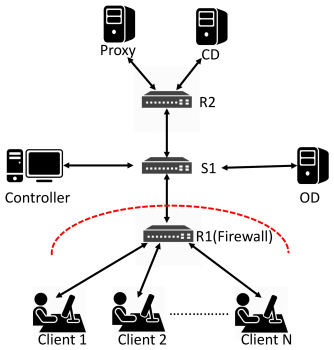

Figure 4.3: The topology used for evaluating SB; 1 hardware SDN switch (`S1`), 2 routers (`R1`,`R2`), an SDN controller and 6 host nodes. R1 blocks traffic to CD.

4.5.1 Controlled Experiments

Our first experiment involved the SB client downloading files of various sizes from the CD, and comparing the download times to that achieved when downloading via `wget`. The results of this test (ref. Fig. 4.4) depicts that SB performs considerably well in comparison to `wget`.

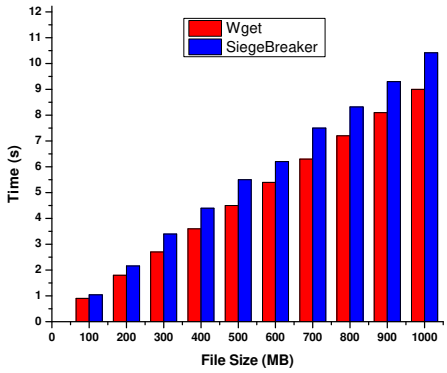

Figure 4.4: Comparison of SB and `wget` on a controlled setup, in terms of download time for file sizes up to 1GB.

Our next experiment was to evaluate the performance of SB's SP–client congestion and flow control (explained in Sec. 4.6) against native web (TCP) traffic. Both SB and `wget` clients simultaneously downloaded large files (varying between 100 MB and 1GB) from different CDs, over 1 Gbps shared network link (between R1 and S1, see Fig. 4.3). The throughput achieved over this shared link, for both SB and `wget` is shown in Fig. 4.5. As evident from the results, SB and the `wget` client achieve roughly the same throughput (sharing the link capacity almost uniformly). The result demonstrates that our system *effectively emulates TCP's congestion and*

flow control mechanisms, evenly sharing the link capacity with background TCP traffic.

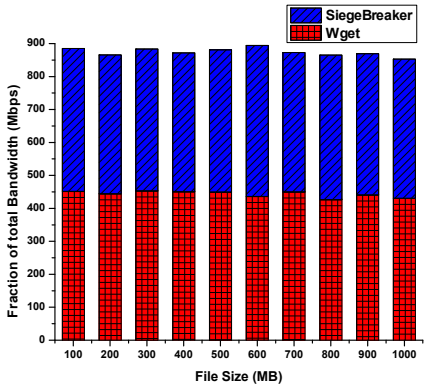

Figure 4.5: TCP performance: SB and `wget` simultaneously downloading a file on a common link. They share the available bandwidth almost equally.

To determine the impact of cross-traffic on SB's performance (by increasing the load on SDN switch), we connected several hosts to the switch and exchanged increasing volume of traffic between them, via P2P connections. At the same time, we used a SB client to download a 1GB file from the CD. The results of this test are shown in Fig. 4.6. As evident, increasing the cross-traffic load had no impact on the client's throughput. This is because, once configured, the switch merely forwards selective DR flows to the SP. This effectively isolates them from the rest of the traffic, avoiding contention.

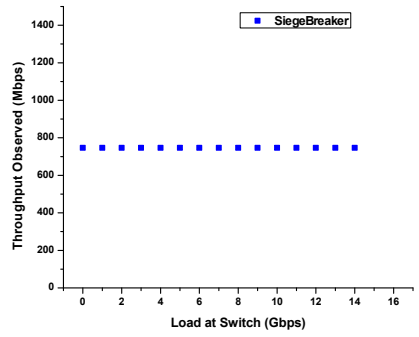

Figure 4.6: SB client's performance with increasing load on the SDN switch.

Further, we also included provisions at the proxy to handle multiple clients. For a sample run of 9 clients simultaneously downloading a 100 MB file, we observed a nearly even distribution of bandwidth *i.e.* $\approx 11\%$. Similar results were observed when this experiment was repeated for larger files (such as 200 MB).

4.5.2 Internet Experiments

To further evaluate our prototype, we conducted experiments where the OD and CD were chosen to be websites hosted on the Internet. We tested our system's performance against two kinds of network workloads — regular web browsing and large file downloads respectively.

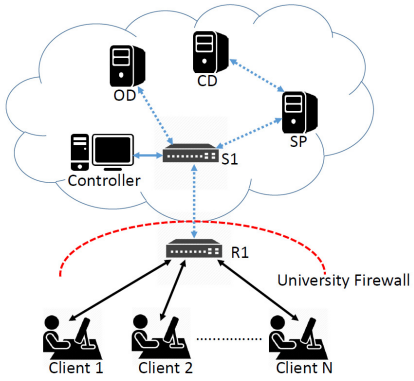

Figure 4.7: The topology used for evaluating SB over Internet with clients inside a university campus.

Assessing the performance for regular web browsing: This experiment involved recording the download times for the home pages of several blocked websites. To simulate web censorship, we configured our university firewall to filter access to Alexa top-500 sites, for a particular client under our control. The SDN switch and the SP were installed outside the firewall, and had unhindered access to the Internet.

In our test we spawned 500 concurrent SB client instances, using blocked Alexa top-500 sites as CDs, and several random unfiltered sites as ODs. In all cases, we were successful in downloading the home pages of the blocked sites, which varied in size from a few kB to 1.5 MB; we measured their respective download times. Next, we reconfigured the firewall to disable filtering, and again accessed the same web pages using `wget`. As Fig. 4.8 shows, the average download time for SB (1.8 s) was very close to that for `wget` (1.7 s).

Assessing the performance for bulk downloads: To test SB with bulk file downloads, we set up web servers on six geographically distributed machines, each serving files of various sizes. The firewall was set to block direct access to these machines from our clients, but we could download the files using SB, using a random unfiltered website as OD. Regardless of the background network conditions, the download times for SB do not significantly differ from our baseline, *i.e.*

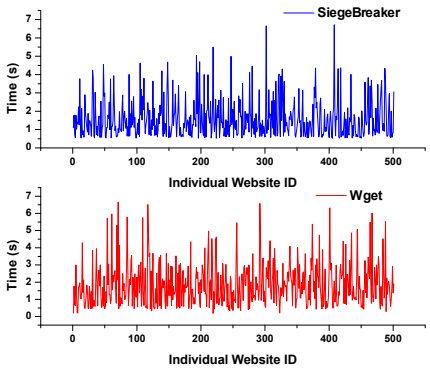

Figure 4.8: Download times for Alexa top-500 websites — accessed by 500 parallel clients of SB and `wget`.

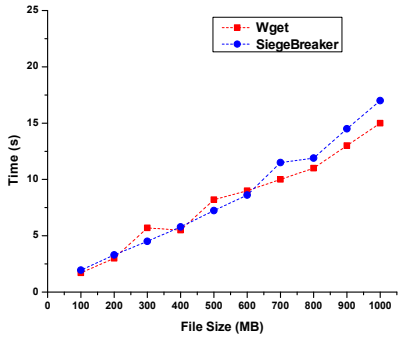

Figure 4.9: Comparing time taken by SB and `wget` for downloading files, with OD and CD hosted on Internet.

download times using `wget`. Fig. 4.9 represents one such case, corresponding to a cloud server, comparing the download times *w.r.t* various file sizes (varying from 100 MB to 1 GB).

To gauge the performance of SB under heavy cross-traffic loads, we initially planned to divert the entire university's traffic through the switch. However, we chose against doing so, for two reasons. Firstly, it would force users uninvolved in the study to send their traffic through our switch, raising ethical concerns. Secondly, the cumulative volume of our university's traffic rarely exceeded 500 Mbps, and would not saturate the switch.

We therefore continued with our original setup, but connected 15 hosts to the switch so as to generate varying background loads. These hosts generated cross-traffic by establishing P2P connections with one another and exchanging large volumes of data ($\approx$ 15 Gbps). At the same time, we downloaded large files (100 MB and 1 GB) from the cloud servers, using both SB and `wget`. As Fig. 4.10 shows, SB's download times are comparable to those of `wget`, regardless of variations in cross-traffic.

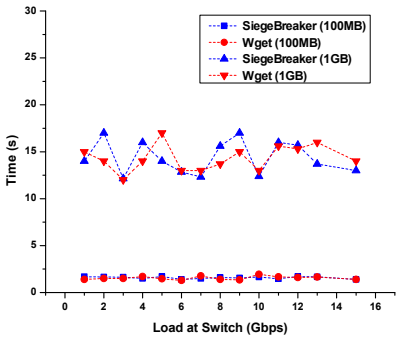

Figure 4.10: SB vs `wget` when increasing the cross-traffic on SDN switch, with OD and CD hosted on Internet.

Similar tests were repeated by varying the number of parallel flows traversing the switch. The hosts, instead of exchanging P2P data, ran `Apache Benchmark Version 2.4` and connected to a web server (also connected to the switch), spawning as many as 50k concurrent web connections[3]. Fig. 4.11 presents the outcomes of these tests. Increasing the number of flows, had *no impact* on the download times achieved using SB. This was comparable to what is achieved using `wget`. Thus, even in the presence of high cross-traffic, SB proves to be practical for downloading files of all sizes.

Additionally, SB has no measurable impact on non-DR traffic due to increasing DR traffic. Non-DR traffic is forwarded directly by the SDN switch without any intervention from the controller. Since SDN switch is a specialized hardware designed to transmit traffic at line rates, the non-DR traffic is thus not impacted.

Overall, the promising performance of SB can be attributed to our modular design (which makes use of hardware SDN switches), and avoids the unnecessary cryptographic inspection of non-DR flows.

Measuring the setup time: A SB client is required to covertly signal the controller to install a DR redirection rule, before it begins the actual download from the CD. The time elapsed between initial email and the installation of final DR redirection rule (*i.e.* between step 1 and step 5 of protocol) is called *setup time*.

[3]According to the NOC, this is *much* larger than the total number of flows at the university's edge router, that rarely exceeds 20k flows.

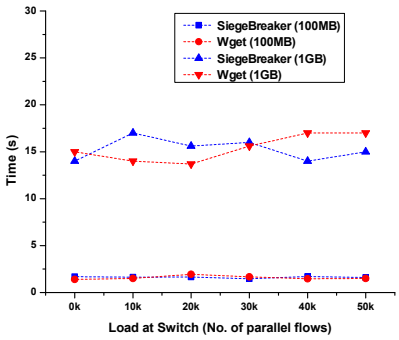

Figure 4.11: SB vs `wget` when increasing the number of flows on SDN switch with OD and CD hosted on Internet.

In 100 experimental trials, we observed an average setup time of 3-4 seconds. Similar to Sweet [94], we observed that most of this delay comes from email handling modules (selenium scripts composing mail, SMTP connection time at controller *etc.*) rather than network latency (which is of the order of milliseconds). [**Note:** The download time figures in previous subsections do *not* include setup time.]

Browsing experience: We used `wget` for benchmarking purposes. However, SB is also integrated in a browser. It can easily load static as well as dynamic websites without any significant performance degradation. Similar to Telex [77], we also used SB in our lab to access normal day to day websites for months without any problems.

4.5.3 Implementation Details

We now describe the implementation details of each individual component of SB. All in-lab machines, had Intel(R) Core(TM)i5-7400 CPUs @ 3.00GHz, provisioned with 8GB RAM, fitted with Gigabit Ethernet adapters.

Client: The client code is written in C ($\approx$ 1100 LoC). It uses the `OpenSSL` library v1.1.0f for handling TLS connections. We use raw sockets API for crafting packets and `libpcap` for receiving and processing packets.

The email component is implemented for both SMTP and webmail using Python. The webmail version uses `Selenium` 3.14 [102] and SMTP version uses smtplib [103] in Python.

The client program listens on a local port on the client machine for new websites requests and serves the content as and when requested.

Secret Proxy: The code for SP is also written in C ($\approx$ 1500 LoC). The client – SP TLS connections are handled using the OpenSSL library. We modify the header fields via raw sockets and use `libpcap` for packet processing.

SDN switch: Our implementation uses the `HP3500yl`, a 24-port Gigabit SDN switch. This switch works in hybrid mode, *i.e.* both Openflow and non-Openflow environments simultaneously. It has three different types of immutable tables: *Default* (0), *Hardware* (100) and *Software* (200). Our redirection and inspection rules all make use of the Hardware table, as this allows for the best performance.

Controller: We designed and implemented our controller application using `Ryu` v4.15, which is written in Python ($\approx$ 2068 LoC), and communicates with the switch using `Openflow` 1.3. Email reception and processing is implemented for both IMAP and webmail in Python. The webmail version uses Selenium and the IMAP version uses Python `imaplib` [104]. In our experiments we used IMAP for fetching Emails, as it was easier to integrate with `Ryu` controller.

OD and CD: We use Linux nodes running Apache v2.4.18 configured as HTTPS server, to serve as the OD and CD.
The source code of SB can be found at at [105].

4.6 Discussion

In this section, we discuss the various design aspects along with the security analysis of the SB protocol.

4.6.1 System Design

1) ***Selection of timeouts:*** In our design, the controller needs to install redirection rules at steps 3 and 5. As the switch has limited memory, these rules need to be purged eventually.

The inspection rule (step 3 of protocol) match packets based on client–OD IP pair and has a hard timeout associated with it. We cannot use an idle-timeout for it because if the DR and non-DR clients are behind a common NAT device (*i.e.* share same source IP), then the aforementioned rule would also capture packets for the non-DR clients accessing the same OD. The rule may remain active even when there are no DR flows that match the rule. Enough of these (NAT based) inspection rules may unnecessarily remain in the switch exhausting its limited memory. Since, in all our experiments, we always observed that the inspection rule (in step 3) was installed in less than 3s, so we set the value of hard time out to be 4s.

For SP redirection rule (step 5 of protocol), controller installs the rule based on client IP, source port and OD IP. Here again, non-DR flows (sharing the same source IP, accessing the same OD as of DR client), shall not match the rule, because different flows have different ports associated with them. Thus, we use an idle timeout with this rule which expires once the flow is inactive for the said duration. Following [106], we set the value of idle timeout to 5s.

2) Proxy–Client traffic reliability, efficiency, and congestion control: The connection between SP and CD is a standard TCP connection, but the "connection" between client and SP merely *appears* to be so. Client–SP packets carry spoofed headers, so they can appear as client–OD traffic to the censor. However, they *cannot* rely on the kernel's TCP stack (due to spoofing). Hence, in order to ensure packets between client and SP are reliably and efficiently delivered (despite the absence of a true connection), *we emulate TCP's congestion and flow control mechanisms at the application layer*. The SP–client traffic, not only bear sequence and acknowledgement numbers corresponding to the initial client–OD traffic, but also mimic TCP sending patterns (varying the sender window sizes, based on acknowledgements and timeouts, as in TCP).

3) Handling clients behind NAT: It must be noted that attempts by multiple clients behind a NAT to use the same OD for DR may lead to service unavailability. This can be explained with an example. Let us assume that for a client C_1, that is behind a NAT, an SP redirection rule (bearing NAT IP_{IP1}, OD IP_{IPO1}, and client's source port) is already installed. At the same time, another client C_2 attempts using the DR service with same OD (IP address: IP_{O1}). This would lead to installation of a new inspection rule for the client C_2, for the same $IP1$ and $IPO1$ pair. Due to this new rule, the DR packets of C_1 would also be redirected to the controller, rather than

to the SP. Thus, disrupting the C_1's DR session. As a preventive measure, we do not install any inspection rule for a client IP–OD IP pair, if there is already a SP redirection rule in place for the same. Thus, the clients behind a NAT device would require trying out different ODs to access DR service.

4) Location of Decoy Station (SP): Previous proposals (except Cirripede and Waterfall) require Decoy Stations to be located close to the Decoy Router. In our design, the Decoy Router is simply an SDN switch, which forwards packets based on flow rules; it can transport DR flows to the SP regardless of its network location. This helps defend against the *Forced Asymmetry* attack discussed ahead in Subsec. 4.6.2.

4.6.2 Possible Attacks and Countermeasures

We now consider attack scenarios that may be launched by attackers within our threat model, and describe how SB addresses these threats.

1) *Fingerprinting Attacks*

a) **TLS handshake fingerprinting:** To prevent an attacker from characterizing the TLS handshake signature (*e.g.,* the list of ciphers supported), we use the cipher suite of a popular browser Mozilla Firefox[4].

b) **TCP/IP protocol fingerprinting:** An adversary that records TCP/IP header values for traffic to ODs, may use it to distinguish the DR flows from non-DR flows. The DR flows may have different TTL values, window sizes *etc.* (due to packets originating from SP) compared to non-DR flows, making them easily distinguishable. In such a scenario the SP can generate identical TCP/IP characteristics as that of OD, making them relatively indistinguishable from the regular flows.

2) *SB Confirmation Attacks:* The aim of the adversary is to distinguish DR flows from the non-DR flows. To achieve this, an adversary may modify, drop or replay some packets of a TLS connection. However, tampering (or replaying) of regular TLS packets in a normal connection

[4]We can also use a recent library `uTLS` [107] to achieve the same.

results in the receiver replying with TLS error messages. Since SB hijacks an existing TLS connection, it does not natively respond with such error messages. The adversary might use this distinctive feature to confirm a DR connection. We now describe the kind of attacks such an adversary may launch, along with the preventive measures employed in SB.

a.) Record and Replay (without disrupting regular Internet clients): To confirm usage of DR, censor can record all traffic of a suspected DR user and can try to replay a part of (or complete) connection, at a later time. The first interaction of DR client with SP occurs at step 7. All prior steps corresponding to the original client–OD TLS connection. Thus replaying packets of all the prior steps would not result in confirmation of DR. Hence, the censor may initiate a new DR connection and try to replay *only* the second TLS data packet, carrying the CD URL (step 7 of Fig. 4.1). Any response from SP, other than an TLS alert, would confirm the use of DR. However, our design is resilient to such an attack as the said packet contains a HMAC derived from the session's HMAC key. Hence, replaying it will lead to failure in HMAC verification. Additionally, none of the subsequent packets can be replayed because of the usage of AES in CBC mode.

b.) Modification and/or Replay (disrupting regular Internet clients): An adversary can attempt to confirm if an arbitrary TLS flow is intended for using DR service. Such an adversary can actively *modify* and/or *drop packets* of an *ongoing* flow. We believe that modifying the packets of such arbitrary TLS flow may prove costly to the adversary. It will disrupt regular Internet communication, eventually degrading the user experience. Such attackers are outside the scope of our threat model (ref. Subsec. 4.4.1). However, even if the adversary carries out such attacks, we enumerate the possible solutions to mitigate them in Appendix B.3.

3) *Forced Asymmetry:* This attack [87] assumes that traffic from anywhere in the Internet, destined to IP prefixes of censored regime can be (1) classified into traffic originated (or transited), from *Decoy ASes* (ASes which host DR infrastructure) and *non-decoy ASes* and (2) will enter the censorious regime at different locations (*e.g.,* BGP routers) in the network using a load balancing technique called as hole punching [108]. *E.g.,* if a client accesses OD without DR, its responses will enter at one particular network location. Whereas, if it *pretends* to use OD while using DR, traffic will enter at different location, confirming the use of DR to the censor.

In the existing DR systems (*e.g.,* TapDance [81], Slitheen [83] *etc.*) the decoy router and the decoy proxy are placed together within a same decoy AS. The decoy router intercepts the DR requests of the client and forwards it to decoy proxy, which on behalf of client fetches content from the CD and sends it to the client (spoofing as OD). Thus, response packets will always come from decoy AS to the client. This makes the Forced Asymmetry attack successful against such systems.

However, due to SB's modularity, the DR infrastructure is not confined to an AS, because the SP (decoy proxy) can be hosted anywhere on the Internet, even in non-DR ASes. As a consequence, response packets originate from non-DR ASes rather than DR ASes. This would result in DR and non-DR traffic entering the censor's network through the same location, making the attack ineffective.

4) *Routing Capable Attacks:* A powerful adversary may disrupt *any* DR system (including SB), by *preventing* traffic originated from its networks to reach the decoy AS. Schuchard *et al.* [87], demonstrated few such attacks, which rely on the censor's ability to control the route taken by packets, either through a *tainted* path (which has DR between client and OD) or a *clean* path (devoid of DR):

a) Crazy Ivan attack: For ongoing client–OD connections (either through a tainted or a clean path), the censor can enforce changes in routing policies such that these connections (DR and non-DR) follow only the clean path. The non-DR flows would not be affected by this change whereas, DR flows would be disrupted.

b) Packet Injection: For a suspected ongoing connection, an adversary can send packets (spoofing as client) along a clean path in order to reveal the actual connection state between client and OD. *E.g.,* an adversary could send a crafted TCP packet, with sequence number corresponding to the initial client–OD connection, to the OD. Responses like TCP ACK or Duplicate ACK would indicate absence of DR session, whereas a TCP RST response from the OD would reveal the presence of an ongoing DR connection.

c) Routing around decoys: An adversary might change its routing policies permanently, in order to avoid the DR ASes altogether (by always selecting a clean path), thereby denying DR

service.

All the aforementioned attacks rely on the assumption that clean paths can be easily obtained by the adversary. However, recent DR placement strategies (local [79], as well as global [80]) suggest that, obtaining clean paths would be *difficult* for the adversary, considering the massive topological changes (with the associated costs) required to obtain the same [79]. On the contrary, Schuchard in [109] argued that, if the censor manages to obtain a small (yet significant) fraction of clean paths — rerouting its traffic via clean paths would incur heavy economic losses to decoy ASes (which are likely transit ASes that earn revenue by transiting the Internet traffic of their customer ASes). This might build pressure on decoy ASes to remove DRs by inflicting economic losses.

However, to successfully launch former [87] and latter RAD attacks [109], there are some practical challenges. Firstly, a censor needs to find all possible tainted paths from ISPs under its jurisdiction to different Internet destinations, by active probing [87] — a non-trivial exercise for the censor. Secondly, after obtaining tainted and clean paths, censor would have to introduce nationwide BGP policy changes, including changes to routing business relationships, and may lead to network downtimes *etc.*

5) *Delayed SYN:* The client is expected to initiate a TCP connection before the redirection rule expires on the SDN switch (step 3 of the protocol). If client's connection request is delayed and misses this window, intentionally by an adversary (or due to congestion), it would arrive at the OD (without being observed by the controller). The regular TCP and TLS handshakes would ensue. Packets sent by the client would reach OD, not the controller or SP. This does not pose a problem: after the handshakes (TCP and TLS), the subsequent GET request by the client fetches a regular response from the OD. The client immediately realizes that it missed the window, and initiates a new connection request. The censor sees nothing suspicious.

6) *DoS Attacks:* We discuss different ways in which an adversary may abuse the system to deny service to DR and/or non-DR clients. Like others, SB is also vulnerable to such attacks. We also discuss the strategies that are in place, or can be adopted to mitigate them.

a) *Forced Decoy Routing:* We consider the Byzantine attacker that tries to disrupt normal

routing for a non-DR user. To do so, the attacker simulates the client side of the SB protocol, while spoofing the source address of the victim (*i.e.* the non-DR user). Eventually, the controller installs redirection rules for the victim's IP address (assuming it to be a DR client). Thus when the victim tries to access OD, its traffic is redirected to the SP which fails to decrypt these packets, and drops them. Thus, by abusing the DR, non-DR traffic is prevented from reaching its intended destination.

However, in our design, the SP redirection rule matches packets based on source IP, destination IP and *source port number*. Thus, to successfully launch the attack, the adversary must *anticipate* the port number the victim will use while connecting to a particular OD; the probability of success of this anticipation is 0.00006 (assuming ephemeral port range = 16383).

b) Memory Exhaustion Attacks: As already mentioned in previous attacks (like Forced Decoy Routing), a powerful adversary can force installation of enough flow rules to exhaust the TCAM memory of SDN switch by pretending to be many legitimate DR clients. To minimize this threat, we incorporate timeouts for different flow rules (step 3 and 5 of protocol). Methods to minimize TCAM memory usage in SDN [110, 111, 112, 113, 114, 115] can be further used to mitigate this attack.

c) Spamming the Controller: A simple attack would be to flood the controller's email with spam. The adversary could send random emails, or more subtly, emails requesting DR service, from thousands of email addresses. This noise might prevent the controller from detecting legitimate requests. In such scenarios, we resort to issuing a unique email ID for the controller to individual (or groups of) clients. Thus, even if one email ID is spammed, clients can use other email IDs to access DR service. Additionally, similar to Mailet [95], we can also enforce usage limitations on clients or can use Captcha [116] and/or puzzles.

d) Fake Sessions Attack: As a more sophisticated attack, the adversary could send the controller "legitimate-looking" emails, which set up decoy routing sessions with random source IP addresses and ISNs. The controller, receiving an email, would install an inspection rule and begin to analyze the corresponding (irrelevant) flows. *To minimize the impact of this attack, we include a hard timeout with the inspection rule.* A fake email can make the controller inspect unwanted traffic

only for the timeout duration, *i.e.* 3s (ref. Subsec. 4.6.1).

e) Fake DR Request Registration: With an intent to deny some specific users of DR services, an adversary can send an email containing their source IP, some OD-IP and some random ISN. When the actual DR user sends an email, with the same source and OD IP, but a different ISN, the controller stores both the ISNs for matching. Hence, when the DR user sends a TCP SYN to the aforementioned OD IP, it's ISN would match the one indicated in the latter email, and the SP redirection rule would be installed.

An adversary could send multiple such emails with different ISNs. Now the controller would require maintaining a set of ISNs for every client and OD IP pair. Searching through such sets to match ISNs of the incoming SYN packets could marginally increase the overhead at the controller. However, DR services would not be hindered.

Moreover, the presence of the hard timeout of the inspection rule, would force the adversary to keep sending such emails regularly. This could be very expensive for the adversary if the attack has to be carried out for all possible clients. Thus, SB is more resilient to such attacks, unlike other registration based systems (such as Cirrepede and Waterfall of Liberty) where the adversary just needs to register once for a client.

4.6.3 SDNs and SiegeBreaker

It can be argued that SDNs are primarily deployed in small networks (like data centers), and are not suitable for use on the Internet. However, SDNs have been shown to scale to an entire AS [117, 118, 119], and practical ISP-scale SDN has been demonstrated by Rexford *et al.* [120]. Moreover, there is a recent research that suggests how ISPs can (and should) migrate to SDNs [121]. Further, SB can work even for a non-SDN AS. Friendly non-SDN ISPs could position openflow switches such that they intercept traffic to critical network entities (*e.g.,* routers which transport a large fraction of traffic [80]), *without replacing the existing infrastructure*.

Scalability of Controller: Would the SB *controller* become a bottleneck at ISP scale? Recent work [122, 123, 124, 125, 126] suggests that SDN controllers can scale to very large use cases –

for instance, `Cuttlefish` [124] proposes a hierarchy of local and global controllers, which can offload tasks to each other, providing higher control plane throughput and better scalability. We suggest that, if needed, such approaches can be used when SB is deployed at scale.

4.7 Conclusion

We present SiegeBreaker, a practical and efficient Decoy Routing system. Using an SDN architecture, SiegeBeaker divides the tasks of Decoy Routing among three loosely-coupled modules. The SDN controller focuses on detecting packets of interest, using an efficient *privacy-preserving* signaling scheme, by re-configuring SDN switches on-the-fly. The switch then redirects all packets of the Decoy Routing flow to a secret proxy server. Finally, the proxy server communicates with the covert website on behalf of the client, and transmits back the responses *reliably and efficiently*.

In extensive tests involving commercial SDN switches, our prototype shows promising performance — nearly equal to that of direct TCP connections. Additionally, SiegeBreaker's flows uniformly share the available link bandwidth with other non-DR connections. Along with privacy preserving signaling, such performance results show promise for future implementation and deployment by SDN-based networks.

It must be noted that DR systems provide great blocking resistance. However, such systems pose deployment challenges as they require collaboration from ISPs to deploy DR infrastructure. Thus, DR provides a promising but futuristic solution for circumventing censors. However, in the meantime we require systems that can be immediately used without posing deployment challenges. Thus, in the next chapter we aim to build such a system by utilizing instant messaging applications to transfer restricted content.

Chapter 5

Camoufler

5.1 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 [71, 127, 128] 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 [129, 128, 127]. 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 [130].

To that end, recent anti-censorship systems [9, 131, 93] are designed on a fundamental principle *i.e., to incur collateral damage to the censor, if it attempts to disrupt the circumvention scheme*. This makes it difficult for the adversary to completely block these systems. Approaches

like *decoy routing* [9], *domain fronting* [131] *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 [132]. 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* [133] and *MassBrowser* [134] 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 [94], Covercast [96], Freewave [135] *etc.*, transport censored traffic via services and protocols essential for smooth functioning of businesses, and thus a censor's economy. These systems exploit Email [94], VoIP [135], video streams [96, 136] *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.

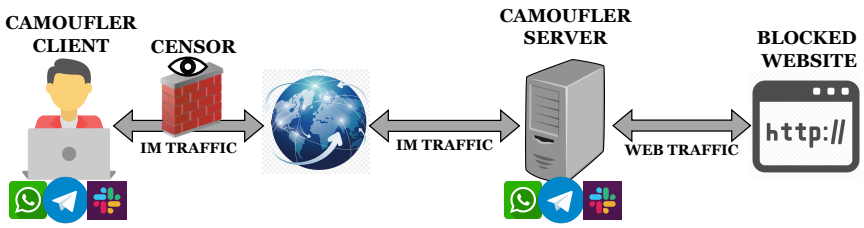

Figure 5.1: Camoufler basic architecture

However, despite these systems providing efficient blocking resistance from the adversary, they exhibit other challenges which 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 [131] costs. Moreover, DeltaShaper [136] (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 over-

coming 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 ($< 1s$) [137] when user exchange messages with each other (ref. Sec. 5.4). 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 [138, 139, 140, 141, 142][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 [143, 144, 145].

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 [9, 80, 85]).

[1]There were 2.5 billion active IM users till January 2019 and this number is expected to easily cross 3 billion till 2020 [23].

Thus, we developed Camofler with IM channels as its underlying tunneling protocol. The basic architecture of Camoufler (depicted in Fig. 5.1) 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 the major contributions of this work:

- 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.

5.2 Related Work

To promote free speech and unhindered information exchange over the Internet, researchers and free speech activists have proposed several anti-censorship solution [146, 147, 148, 74, 128, 127,

9, 131, 149] 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 [71] *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 address. 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 [150], Cirripede [78], Slitheen [151], Tapdance [149], Waterfall of Liberty [82],SiegeBreaker [152] *etc.*

   To censor decoy routers, the adversary may require undertaking daunting measures such as changing entire nation's routing policy [87] in order to bypass such systems. Such routing changes are prohibitively expensive to achieve in practice [79, 80]. 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 [80].

3. **Mimicry based systems:** These systems attempt to disguise and transfer censored content as regular applications' protocol messages. *E.g.,* SkypeMorph [74], helps access censored websites by mimicking Skype's communication protocol. Others, like CensorSpoofer [153], 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 [76]. 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 calls, streaming video *etc.* Examples include SWEET [94], Covertcast [96], Delta Shaper [136], Freewave [135], Cloud-Transport [154] *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 poor QoS for web downloads (*e.g.,* due to low offered bandwidth in these channels) and are limited in their deployment.

5. **Miscellaneous systems:** There exist other anti-censorship systems, which do not fall in any of the above categories, like domain fronting [131], CacheBrowser [93], MassBrowser [134] *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 [131].

   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 [133]) 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.

5.3 Camoufler Architecture

This section describes the threat model we consider for Camoufler, and present its design (ref. Fig. 5.2) as well as a step-by-step walkthrough of the protocol.

5.3.1 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 [129, 155, 156] *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 at least 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 [138, 139, 141]. Hence, it would be difficult for the censor to completely block them without sustaining significant collateral damage.

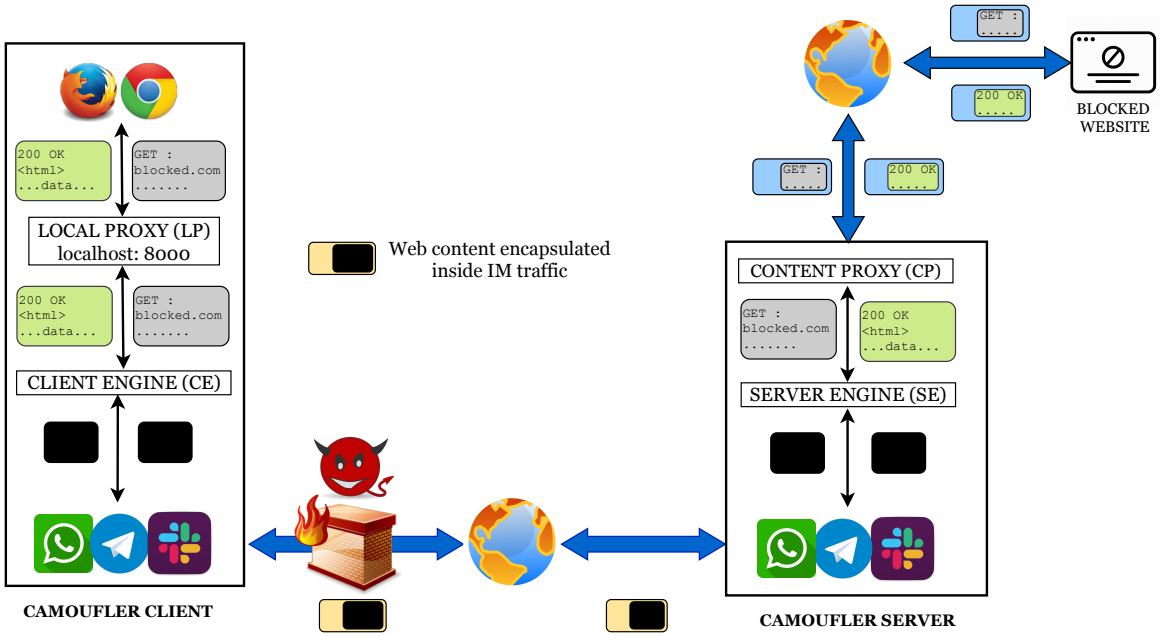

Figure 5.2: Camoufler Detailed Architecture.

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 [157]. 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.

5.3.2 System Design

We now describe the protocol design of our proposed system. Our system 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. 5.2. 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:

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 §5.3.2.1), 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.

5.3.2.1 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. §2.3). 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.

5.4 Evaluation

We now evaluate the performance of Camoufler 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 $\approx$ 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. 5.3. As evident from

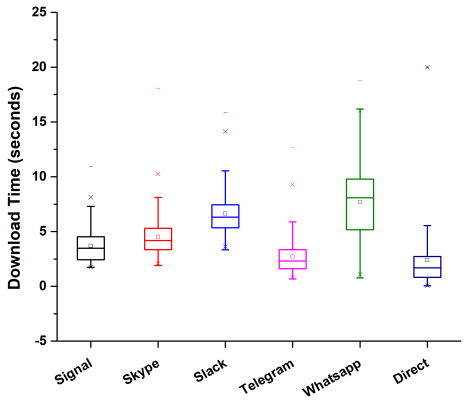

Figure 5.3: Download time of Alexa top 1000 websites using different IM apps and its comparison with direct downloads.

the figure, 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. C.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 [96], Deltashaper [136], *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. C.1. 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. C.1.

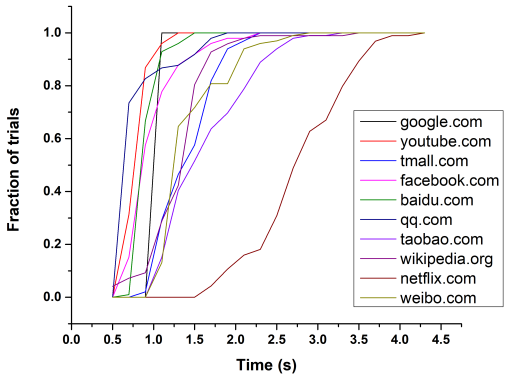

Figure 5.4: CDF of Time To First Byte (TTFB) for 10 popular Alexa websites (each downloaded 100 times).

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 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. 5.5, 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.

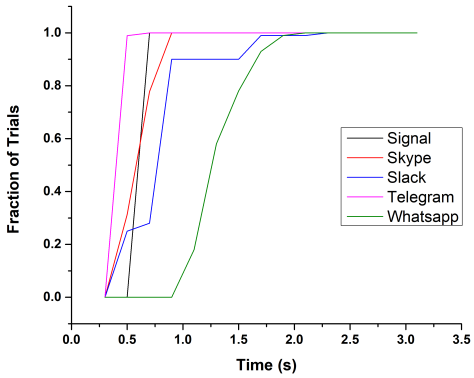

Figure 5.5: CDF of time taken by a message to travel from Camoufler Client to Camoufler server for different IM apps.

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. 5.6. 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. C.2).

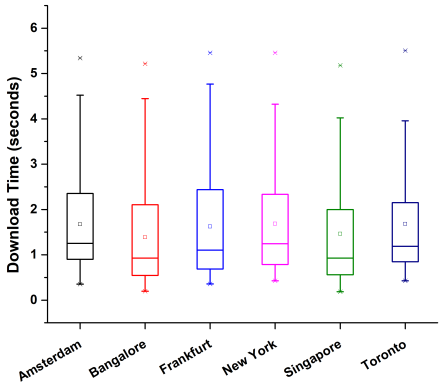

Figure 5.6: Box plot depicting download time (from a server in Singapore) of top Alexa-1000 websites for varying client location.

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 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. 5.1. Using Telegram as

Downloaded Using	**Time (in s)**						
	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 5.1: Large File Downloads: Comparison of download times of Camoufler and Wget.

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.

5.4.1 Implementation Details

We now describe the details of our 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 C.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 `socket` API [158] to manage connections to and from the browser.

CE comprises of scripts written using python and shell-scripting language. It interacts with the underlying `Signal` messenger to send and receive messages, using the `signal-cli` [159]

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` [160] was used to interact with the dbus interface. On receiving the response, CE decompresses and decrypts it using the gzip [161] and Crypto [162] 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 language, and similar to CE it utilizes `pydbus` [160], `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).

5.5 Security Analysis

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

5.5.1 Traffic Analysis

Censors may attempt to analyze Camoufler client's traffic in order 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 [157]. 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 [94] and Mailet [95] *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 [163]. In a recent study [164], 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 [94].

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. 5.7 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 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.

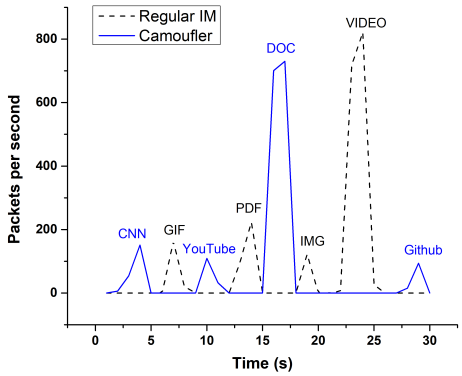

Figure 5.7: Packets exchange rate of regular IM client accessing multimedia content (images, GIF animation, video *etc.*) vs a Camoufler client accessing websites (`cnn`, `github` *etc.*) and a doc file. As evident, traffic characteristics of both regular IM and Camoufler are very similar.

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. 5.7). 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. 5.8, 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.

This is further highlighted in Fig. 5.9. 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[2].

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 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. C.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 [164] 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

[2]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.

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.

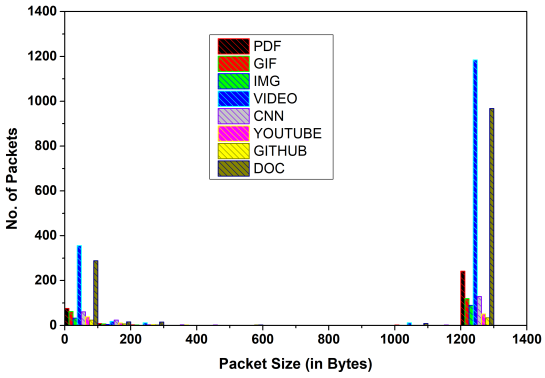

Figure 5.8: Histogram of packet sizes when regular IM client accessing multimedia content and when they use Camoufler to access websites.

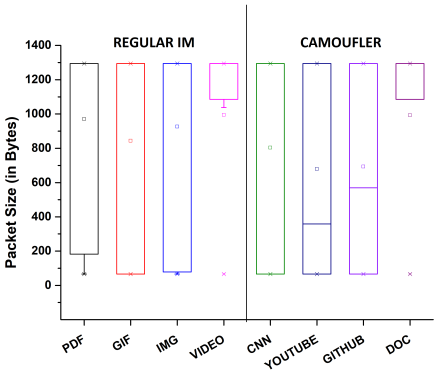

Figure 5.9: Box plot of packet sizes observed when regular IM client accessing multimedia content and when they use Camoufler to access websites.

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 the Camoufler or not.

5.5.2 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. §2.3). 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. §5.3.2), 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. 5.3.2.1), 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 [153] to hide our content from the adversary in plain sight, as also assumed in other anti-censorship systems [94, 96]. 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.

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

Table 5.2: 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 5.3: A comparison of different features of other existing anti-censorship schemes. ● - feature supported, ○- feature unsupported, ◑ - feature partially supported.

5.6 Comparison With Prior Systems

We now compare Camoufler with existing systems (described in Sec. 5.2) 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. 5.2 and Tab. 5.3 respectively.

- **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, Cache Browser relies on content publishers to host their content on some CDNs, thereby posing deployment challenges[3]. 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

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

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. Camoufler does not incur any cost when the client has a peer to run Camoufler server. Otherwise, if the client decides to run Camoufler 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 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**

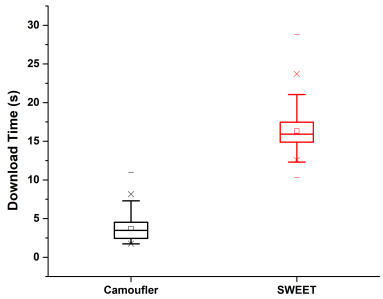

Figure 5.10: Camoufler vs Sweet: Download time of Alexa top-1k websites.

Tunnelling Systems: Most tunnelling systems fail to 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 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.* [96] report that loading BBC news homepage along with three articles takes almost 120s. Further, DeltaShaper only provides an effective bandwidth of 2.56 Kbps [136]. 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[4] (ref. Fig. 5.10). CloudTransport traffic also transits multiple hops (cloud provider, cloud bridge *etc.*), incurring delays and reducing the overall QoS.

Other Systems: Except for Decoy Routing and Domain Fronting, all other systems provide satisfactory QoS. The recent deployment of Decoy Routing within an ISP do not provide adequate QoS for the end user [85, 133]. On the other hand, Domain Fronting incurs a significant amount of delay due to functional overheads. *E.g.,* Domain Fronting introduces a delay of 16s more compared to Psiphon, when downloading a file [131]. 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.

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

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

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.

5.7 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 [136], Facet [165], Freewave [135] 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. §5.5.2), 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 §5.5.2.

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. 5.11).

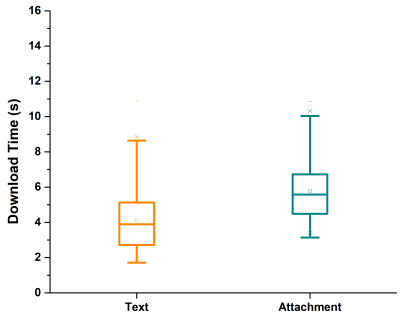

Figure 5.11: Download time of Alexa top-1k websites using Camoufler, when downloaded as text and as an attachment.

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 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.

On the use of Camoufler

In general, tunnelling systems involve sending censored content using some underlying tunnelling channel (email, video streams, IM apps, etc.). It could be argued that when such circumvention

systems would become popular, the channel providers might be coerced by the censor to stop their services. Thus building circumvention solutions that use such channels might prove to be detrimental to the providers' business in future.

However, similar to Camoufler, there already exist multiple anti-censorship solutions that hide the censored content in plain sight—*e.g.,* CovertCast [96], DeltaShaper [136], Sweet [94], Protozoa [166], Skypemorph [74], Decoy Routing/Refraction Networking [150, 78, 9]. Notably, anti-censorship research inherently involves a trade-off between unobservability (of censored content) and the danger of inadvertently blocking the underlying channel. Thus to reduce the collateral damage, proper care must be taken while using such systems; to mitigate the inadvertent harm they can be used as a last resort when the communication over the Internet is severely disrupted.

5.8 Conclusion

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 may be hard to be detected by an adversary (*e.g.,* an ISP working at the behest of an authoritative regime).

Notably, all the systems that have been developed in the previous chapters assume that the adversary does not aim to completely disconnect the Internet for its citizens. However, in the recent past there is a rise in the cases of such deliberate Internet blackouts, performed by the adversaries. Thus, in the next chapter we try to develop a system that can provide access to Internet in such blackout regions.

Chapter 6

Dolphin

6.1 Introduction

The original idea of Internet was to provide a platform to facilitate free flow of information across the globe. This unhindered access to information has promulgated the rampant growth in all walks of life (including technology). On one hand the Internet is so vital to the modern world that free speech over it is considered a fundamental human right by the UN [1]. But on the other hand many censoring nation states attempt to disrupt the free flow of information (as per their convenience), opposing the original idea of Internet. As a result, in the past decade, there has been an exponential rise in the events of Internet censorship globally [3, 2, 167]. This has lead to an ongoing arms race between adversaries and free speech activists across the globe; adversaries continue to evolve various censorship techniques [130, 155, 168, 169, 170], whereas civil liberty activists counter them with wide range of novel circumvention systems [171, 133, 128, 172].

Traditional censorship involves restricting access to a particular resource (such as a website) on the Internet. However, in the recent past, an extreme form of Internet censorship *viz., Internet shutdowns*, has been on the rise. With such extreme measures, the adversary has gone a step ahead in the arms race by completely disabling Internet connectivity in a particular region. These shutdowns can range from a day to over a year in some cases (like in Myanmar and Chad [11]). The complete cut-off from Internet renders all available circumvention tools non-functional.

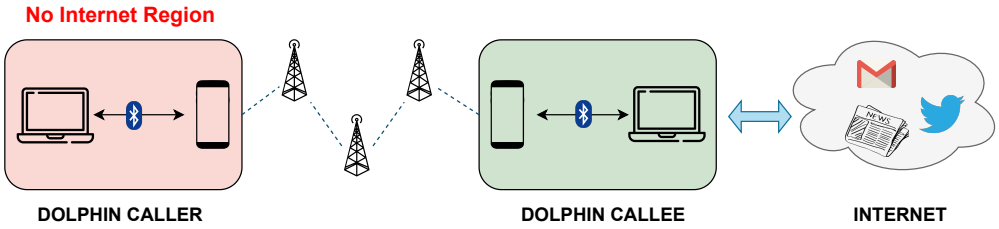

Figure 6.1: Overview of Dolphin's architecture.

Moreover, such a step has a severe impact on the people residing in the shutdown region. They are even devoid of accessing essential services over Internet *e.g.,* access to news, reporting power failures and outages, sending and receiving important emails *etc.* The recent COVID-19 pandemic further exacerbates the impact — a large population of the globe has moved to working online, both for professional and personal tasks. Thus, the regions with such shutdowns have been adversely impacted in these trying times. *E.g.,* due to Internet shutdown in Myanmar some villages in the country were not even aware of the pandemic for many months [173]. It is even alarming that more and more countries are opting for Internet shutdowns. *E.g.,* the number of countries who performed Internet shutdown increased from 25 in 2018 to 33 in 2019, with overall documented shutdown events increasing from 75 in 2016 to 213 in 2019 [11]. Considering that such trends are becoming common, it is plausible that more nation states opt for such measures [174]. Thus, it becomes an imperative to explore solutions using which people living in Internet shutdown regions could access basic Internet services like email, accessing news articles, tweets *etc.*

To that end, we introduce *Dolphin*, a novel system that can provide access to the aforementioned "lightweight" delay-tolerant Internet applications even during shutdowns. Dolphin uses only the cellular voice channel to transmit data. This idea rests on the observation that in Internet shutdown regions cellular voice connectivity is maintained (possibly for performing important executive and administrative tasks, by the governments). There are multiple documented evidences to support this observation [175, 176, 177, 178].

However, cellular voice channel is by design not built for running Internet applications. It is an unreliable, lossy, insecure and highly bandwidth constrained channel. We describe ahead how we overcome these challenges. But before that we begin by describing the overall working of

Dolphin.

Dolphin user requires running the Dolphin client program on its host, while also requiring a trusted peer (*e.g.*, a friend) in a non-shutdown region to run a server program at its end, that *must be* beyond the censor's control. Both the peers also require mobile phones connected to their respective hosts, through which the cellular call will be placed. When the setup is ready, the Dolphin client's utility will initiate a cellular call to the peer, that the Dolphin server program will automatically receive. Once the user has some data to send (email, tweet *etc.*), it will provide it to the Dolphin client which will encode (and encrypt) the data bits to audio with the help of an underlying modulation and framing technique. This audio is then played into the ongoing call, which is transmitted over the cellular network and received by the Dolphin server. The server program would then demodulate (and decrypt) the received audio and recover the data bits. Thereafter, the data is forwarded to the respective application (such as Twitter client) that performs the necessary operation (such as posting the tweet). The overall high-level functioning of Dolphin can be understood from Fig. 6.1.

Since Dolphin transmits data using the cellular voice channel, its functioning might seem similar to the legacy dial-up systems [179]. However, both are fundamentally different. Dial-up requires additional infrastructural support from the cellular service provider [179], whereas Dolphin does not. Importantly, cellular provider disambiguates and differently processes the standard calls and the dial-up traffic. Since, in Dolphin, data is relayed as a standard call, it requires no additional dial-up infrastructural support. Rather, it is an end-to-end system completely managed at the client ends.

This poses various challenges in sending data using the cellular voice channel. First, the encoded data (that is to be transmitted over voice channel) should be similar to human vocal frequency. This is because, cellular networks use variety of optimizations such as voice activity detection (VAD), automatic gain control (AGC) *etc.*, that attempt to suppress any audio signal that does not belong to human vocal frequency. Thus, Dolphin encodes data to human speech frequency before sending it over the cellular voice channel.

Second, real-time voice channel is unreliable by design *i.e.,* the lost audio data will not

be recovered. Intermittent connectivity issues with the base station can further deteriorate the condition. However, most of the Internet applications are built with reliability in consideration. Thus, in order to run these applications, in Dolphin we present a new TCP style (framing, sequence numbering, acknowledgements *etc.*) reliability layer atop the voice channel, which ensures end-to-end reliability and in order delivery of data.

Third, the voice channel lacks end-to-end confidentiality. Thus, Dolphin also provides end-to-end data encryption with additional security features that resists various other attacks (*e.g.,* channel perturbation) explained in detail in Sec.6.6.

We thus designed Dolphin and successfully demonstrate that using it, the users can tweet, send an email, and access news excerpts. Even on a severely bandwidth restricted cellular voice channel, Dolphin takes close to a minute to tweet (280 characters). Additionally, depending on the size, email can also be delivered in a few minutes, *e.g.,* 500 character email takes less than 3 minutes, including the time to establish a secure channel. It must be noted, that Dolphin has a modular design, as it provides a data link and a transport layer (on top of cellular calls) which ensures reliable *end-to-end* transfer of data. Thus, it can be easily extended to support other "lightweight" applications as well.

Moreover, by design, Dolphin is easy to adopt and use – it requires access to a computer and Bluetooth enabled smartphone, and relies on commonly available open source libraries. Additionally, we also provide a way for users to access Internet, even with a fully-automated peer, that *requires no human support*. This is achieved using cellular voice automation services (such as Twilio [180]) that enables hosting the Dolphin server program on a cloud, while providing a local number that users could call. This automation service forwards the audio (from the call) to the cloud hosted Dolphin server, that serves the encoded requests. During a shutdown, the Dolphin caller would only require knowing the above phone number to access Internet (ref. Sec. 6.4.3 for more details).

To summarize, following are our major contributions:

- The design of Dolphin, a system that provides a way to combat the extreme form of censorship due to Internet shutdowns by using the cellular voice channel. The design

ensures security and reliability on top of the insecure, unreliable and bandwidth constrained cellular voice channel.

- An extensive evaluation exploring the feasibility of transmitting data bits in the cellular voice channel by varying data encoding rates, cellular operators, location of the peers *etc.*
- A working implementation of Dolphin that can be used for emails, posting tweets, accessing news *etc.*, all within a few minutes. Due to its modular design it can be easily extended to support other "lightweight" applications as well. Moreover, Dolphin not only works with a human peer, but can also operate without one (using cellular voice automation services).

6.2 Related Work

Several prior research efforts have explored the feasibility of sending data bits over the cellular voice channel. However, most are simulation studies that attempt to provide theoretical bounds for sending data over voice, propose a new modulation schemes (for encoding data to voice), or focus on a specific codec *etc*.

For instance, LaDue *et al.* [181] proposed an initial approach to send data over GSM voice channel. But, their approach was designed for a fixed rate codec (EFR) and may not work with other GSM codecs (*e.g.,* AMR). Further, Peric *et al.* [182] conducted a feasibility study for sending data bits over GSM channel globally, and attempted to improve upon the previous work by reducing noise in the audio signals. Similarly, Ali *et al.* [183] proposed a new data modulation scheme to reduce errors when data is transmitted over voice. Along the similar lines, Ozkan *et al.* [184] propose a new approach to encode data directly into "speech like" symbols. Through simulations they demonstrated that a data rate of 1.6 Kbps may be achieved. Further, Kazemi *et al.* [185] even attempted to use the cellular voice channel for vehicular communication and provided an information theoretic bound on the achievable data rates. However, Dhananjay *et al.* [186] tried to build a generalized encoding and modulation scheme for the GSM channel and presented results using real cellular operators. Ahmad *et al.* [187] took a different direction — they propose a architecture where the base stations could be used to provide cellular based data services for rural areas. A recent attempt by Dogar *et al.* [188] provides a communication

channel using missed calls to transmit data (bits) between smartphones. Authors proposed MissIt, a free and low-cost system considering developing countries with low income users. MissIt achieves a very low bit rate of about 0.39 bps to support some very specific and extremely low bit rate applications such as conducting surveys to spread awareness in less informed regions.

However, all such schemes completely ignored practicality and usability of such schemes and systems. None of the existing approaches attempted using actual Internet applications, nor depicted the challenges in doing the same. The cellular voice channel is unreliable and can lead to data distortion and losses (*e.g.,* due to poor connectivity of mobile devices with the base stations). This may hamper the functioning of existing schemes as they do not consider compensating for data losses (inherent in cellular voice). Thus, with Dolphin we depict the practicality and usability of sending data over voice by running actual applications over the Internet using the voice channel, along with a security and reliability layer, that can work with any underlying modulation scheme.

6.3 Dolphin System Design

We now describe the overall design of Dolphin. We begin by describing the individual components of Dolphin (depicted in Fig. 6.1) and their functioning, followed by a step-by-step walk-through of Dolphin's operation. Dolphin has two major components: caller and callee. Dolphin caller infrastructure consists of the following components:

- Dolphin caller machine: This machine runs the Dolphin client utility. It accepts input from the user (*e.g.,* text) that is to be sent over the Internet (as email or tweet). The client utility inputs the text to an audio encoder (explained in detail ahead in Sec. 6.4.2), which encodes the text to audio format (*e.g.,* mp3). This audio is streamed into the audio input of the mobile handset (connected to this machine using Bluetooth).

- Dolphin caller mobile phone: This phone is paired to the client machine via Bluetooth in a manner that it accepts audio input from the said machine (details in Sec. 6.4.1). The audio received from the host is relayed to the Dolphin callee mobile using a standard voice call.

Dolphin callee infrastructure consists of the following components:

- Dolphin callee mobile phone: This phone receives the call from the caller mobile phone and sends the received audio to the server machine, paired using Bluetooth.
- Dolphin callee machine: Upon receiving the audio, from the callee mobile, it is forwarded to the Dolphin server program which decodes the audio to the corresponding data bits (text). These bits are processed by the server program which performs subsequent actions (sending the text as email or tweet on the Internet *etc.*).

6.3.1 Dolphin communication protocol

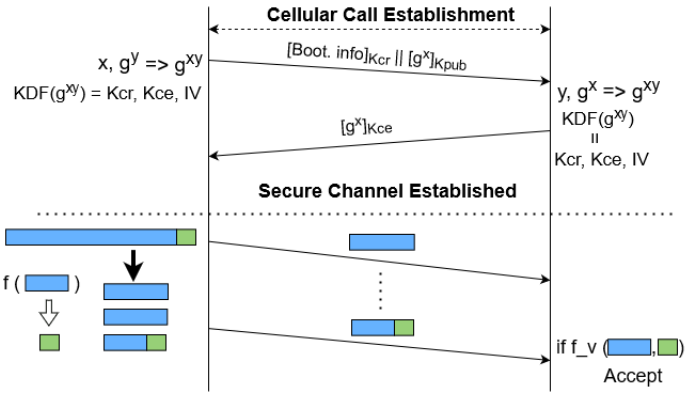

Figure 6.2: Dolphin's secure channel and data transmission phases. Function f() computes HMAC tag and f_v() verifies it.

We now describe the communication protocol of Dolphin. We assume that Dolphin's caller and callee infrastructure is installed and respective programs are running. Additionally, we assume that the caller knows the trusted callee's phone number, the Diffie Hellman (DH) public exponent (g^y) and the public key (K_{pub}) of the callee out of band.

Once a cellular call is established, Dolphin then operates in two phases. First phase deals with establishing a secure encrypted channel between the caller and the callee, required to evade an eavesdropping adversary. Once the secure channel is established, the second phase then deals with actual transmission of data (refer to overall design in Fig. 6.2). The details of these two phases are as follows:

Secure channel establishment phase:

1. In this phase, the caller and callee establish a shared secret to encrypt the data bits, for which they rely on a Diffie-Hellman (DH) key exchange.

2. The caller's client utility first selects a DH private part x, and derives the shared secret g^{xy}, using the already known g^y of the callee.

   Then the encryption/decryption key (K_{cr}, K_{ce}), and the initializing vector (IV) are derived from the shared secret using a key derivation function (KDF) by the caller. We use AES-128 in GCM mode (an AEAD cipher [**?**]) for encryption/decryption. The derived IV is considered as an input nonce to AES-GCM.

3. Once the keys are derived, the caller prepares the bootstrapping information (the application requested to access, current timestamp and plain-text secret string) and encrypts it with its encryption key (K_{cr}). Additionally, the caller encrypts its DH public part g^x with the already known public key (K_{pub}) of the callee (for callee authentication) and appends it with the encrypted bootstrapping information.

   The caller's client utility then sends this data to the callee.

4. The server utility, on successful reception of data, computes g^{xy}, by extracting g^x with the help of its private key. It then derives the respective keys (K_{cr}, K_{ce}), decrypts the received bootstrapping information (using K_{cr}) and sends back an acknowledgement (containing g^x) encrypted with its encryption key K_{ce}. Notably, successful retrieval of the plain-text string provides a quick way to check the integrity and authenticity of the received data.

5. The secure channel establishment phase completes on successful reception and decryption of the acknowledgement by the caller.

Data transmission phase:

1. Once the key is derived, the caller or the callee initiates data transmission based on the bootstrapping information. Since we use AES-GCM, the encrypted data to be sent is appended with a one time HMAC tag that provides integrity as well as authentication of the data. To efficiently utilize the capacity, the plaintext data bits are first compressed, before encrypting and subsequently encoding them as audio.

2. The resulting data is divided into data frames (or chunks) and are transmitted one by one to the receiver.

3. These data frames are received and stored by the receiving end until all frames for the current transmission are successfully received.

4. The above steps are repeated for subsequent data transfers as and when required in either direction.

Notably, the peers derive a new key every time some fresh data is to be transferred. However, for performance efficiency, they can derive a key that stays active for multiple data transfer sessions (*e.g.,* a day or a week).

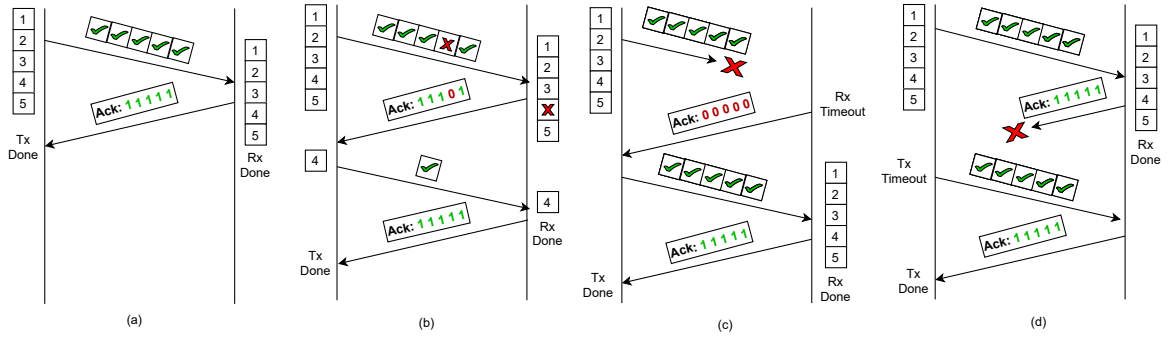

Figure 6.3: Some representative scenarios that are handled by Dolphin's reliability protocol: (a) represents the best case where no data is corrupted/lost, (b) depicts the case where one (or more) chunks are corrupted/lost, (c) is the case where a complete batch of chunks is corrupted/lost, and in (d) the acknowledgement(s) are corrupted/lost. All other scenarios that exist are the variation of these base cases and are thus handled by our reliability protocol.

6.3.2 Dolphin Reliability Protocol

The above walkthrough raises several important questions *i.e.,* how is the data flow controlled, how are the timeouts computed and how is the data integrity verified *etc.* are all important questions that we answer in this section. Moreover, it is known that the voice channel is lossy, and does not provide reliability. Thus, a natural question is how we ensure reliability over the lossy cellular voice channel.

Thus, to achieve reliability and in-order delivery of data, we designed a new reliability protocol. Our protocol is (in part) similar to TCP, but tailored specifically for Dolphin, considering

the underlying lossy and low capacity cellular voice channel. Our reliability protocol specifically incorporates the re-transmission, sequencing and timeout mechanisms, for the in-order and reliable transmission of data. Also, as described ahead (ref. Sec. 6.5.1), we select a low fixed bit rate for transmitting data and thus do not require congestion control mechanisms.

Our protocol involves dividing the data into fixed sized chunks and transmitting each of them with their respective checksums. Thus, the corruption of each chunk can be individually detected and the callee can solicit the caller to re-transmit only the corrupted chunk, rather than the entire data. This scheme helps in reducing the number of possible re-transmissions while transferring data. *E.g.,* one way to transmit 100 bytes is through a single chunk, or dividing it into five 20 byte chunks. In case of former, the corruption of a single bit would require the re-transmission of the entire data (100 bytes), while in the latter the callee may only solicit for the lost chunk (20 bytes). This could potentially lead to a five fold decrease in the amount of data to be re-transmitted in the said example.

Moreover, we transmit the data at a bit rate of 64 bps (relatively higher), and the acknowledgements (or other control messages) at a relatively lower bit rate of 16 bps. The control information is sent at a low rate to minimize the chances of its corruption. Moreover, since the control information is only a few bytes, transmitting them at low bit rates would not impact the overall performance drastically.

We now describe the end-to-end functioning of **Dolphin's reliability protocol**.

1. In order to transfer data in either direction (caller to callee or vice-versa), first the data is divided into smaller chunks of fixed size. Each chunk consists of data bits and the corresponding integrity checksum. These chunks are also prepended with a sequence numbers for managing their order (ref. Fig. 6.5).

2. Thereafter, the sender transmits a batch chunks of sequentially. The exact number of chunks in a batch are fixed and known to both the parties apriori. Once the chunk batch is completely transmitted, the sender waits for an acknowledgement.

3. The receiver listens for, and stores, the incoming data. Since total data to be transferred,

and the transmission rate are fixed, the receiver calculates and sets an appropriate timeout. *E.g.*, if a batch of five chunks (20 bytes each) are to be transferred at a rate of 64 bps (8 bytes/sec), then the total timeout should be 12.5 s (100 ÷ 8 s). Thus, the receiver sets a timeout of 13 s (additional δ say 0.5 s) to compensate for any stochastic delays.

4. After receiving a batch, the receiver pre-processes the chunks by validating their integrity. All the correctly received chunks are queued as per the sequence numbers. The incorrectly received chunks are marked. Subsequently, the receiver sends an acknowledgement, indicating the correctly received and corrupted chunks (thereby soliciting re-transmission).

5. The sender receives the acknowledgement, verifies its integrity, identifies the corrupted chunks, and subsequently re-transmits them. In case the acknowledgement gets corrupted, the sender re-transmits the entire batch sent in the previous iteration.

6. The received re-transmitted chunks are processed similar to step 4. Upon successfully receiving the re-transmitted chunks, the receiver accordingly acknowledges the sender.

7. Thereafter, both caller and callee repeat steps 1 to 6 for any subsequent data transmission.

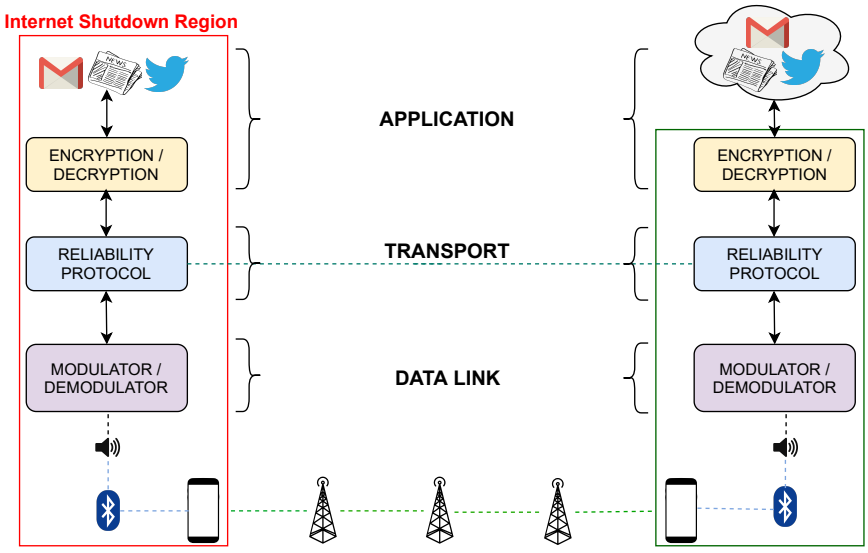

Figure 6.4: Dolphin's block diagram depicting its different functionalities (end-to-end).

Thus, using the above protocol, we are able to ensure reliable delivery of data over the cellular voice channel. A concise version depicting different scenarios and how the protocol handles them is shown in Fig. 6.3 and the overall working of Dolphin is depicted in Fig. 6.4.

However, there might be a few questions about what exactly is sent in the acknowledgements, how are sequence numbers assigned *etc.* We now describe the answers to such questions.

Delineating chunks: It is important to delineate chunk boundaries. Design of the reliability protocol categorically addresses this issue. A naïve approach is to delineate the chunks based on their sizes. *E.g.,* if five 20 byte chunks are transferred (total of 100 bytes), then the initial 20 bytes would belong to first chunk, the next 20 to the second and so on. However, if a single byte is lost in a chunk, then the boundary for all subsequent chunks would be miscalculated. More specifically, if a byte is lost in the first chunk, then even if all the subsequent four chunks are received correctly, they would be discarded due to inaccurate delineation. Though, this strategy is easy to implement, it can lead to unnecessary re-transmission even when the data is correctly received.

The other strategy would be to use a delimiter to delineate each chunk. There can be multiple approaches to add a delimiter. However, we use a technique known as byte stuffing [189]. This technique allows us to use a character (say *e.g.,* the null character), to be used as a delimiter to mark the end of a chunk. All other instances of the character (selected as the delimiter) in the original data are masked (by using extra bytes) in a manner such that the original characters can be easily recovered at the receiver.

However, the traditional byte stuffing algorithms can lead to large overheads, with worst case scenario leading to doubling of the original data. Thus, in order to minimize the overhead, we use the Constant Offset Byte Stuffing (COBS) [190] algorithm. This algorithm ensures, that there will be a constant overhead of only 1 byte per delimiter. Thus, effectively, a 100 byte data (5 chunks) would be converted to 105 byte data, with each chunk ending with the delimiter.

Sequence numbering: First byte of each chunk is reserved for assigning a sequence number. Thus, the maximum sequence numbers that can be assigned is 256, implying that a batch can have at most 256 chunks. Dolphin can be configured to transmit more chunks per batch, by reserving multiple (sequence number) bytes per chunk. Selecting a single byte for sequence number minimizes the overhead.

Each chunk within every batch is assigned a relative sequence number, *i.e.,* the first chunk of

every batch has sequence number 1, the second has 2 and so on.

Acknowledgements: Acknowledgements identify the correctly and incorrectly received (or lost) chunks. Each chunk corresponding to its seq. no. is assigned either a bit 0 (correctly received) or 1 (incorrectly received), within a bit sequence. Thus, the acknowledgement is this bit sequence of 0s and 1s. *E.g.,* if eight chunks are transmitted in a batch, and the fifth and sixth are corrupted or lost, then the acknowledgement will be the bit sequence with the corresponding bits set to 1, *i.e.,* "00001100". The acknowledgement will also contain 1 byte for integrity verification.

Timeout calculation: The duration for which the peers need to wait for receiving the data/acknowledgement can be easily calculated from the length of data and the transmission rate (already known apriori to both parties). The approx. timeout could then be calculated using the formula: timeout = (total data (in bits) $\div$ bit rate) + δ. We fix the value of δ to a small one (*e.g.,* 0.5 s) to account for any unexpected delay.

Data compression: We compress the data before transmitting it. We perform the compression of data before encrypting it, as text data can be compressed with high compression ratio, as compared to cipher-text [191]. In our experiments, compression reduced the data size by 20%-60%, leading to effectively less data transmission over the voice channel.

Integrity check: The integrity for each chunk is calculated using the CRC algorithm [192] (CRC-8). Thus, each chunk consists of an additionally appended one byte to verify the integrity of the received data. Other mechanisms such as CRC-32 (4 bytes) can also be used, but we use CRC-8 (1 byte) to minimize the overhead of verifying integrity. Also, CRC-8 is sufficient for our requirements as it can be used to verify integrity of data up to 64 bytes [193], as we generally select a much smaller chunk size *i.e.,* 20 bytes.

Effective data transport capacity: Overall, a 20 byte chunk would include one byte for sequence number and another one for checksum. Thus, effectively 18 useful bytes are transmitted per chunk (ref. Fig. 6.5). Thus if 100 bytes are sent via 20 byte chunks, then effectively 90 bytes of data and 10 bytes of checksums and sequences number are transmitted. The overheads can be minimized by selecting a larger sized chunks, say 50 bytes each. Thus effectively transmitting 96 bytes of data, along with only four additional bytes. However, in such cases, re-transmission

of larger chunks (upon errors or losses) would incur higher overall delays. Our experience shows us that using 20 byte chunks, minimizes the latency, without reducing the data transport capacity (90 bytes of data for every 100 bytes sent) drastically. Therefore, for all our measurements we use 20 byte chunks. Additionally, data compression also increases the data transmission efficiency.

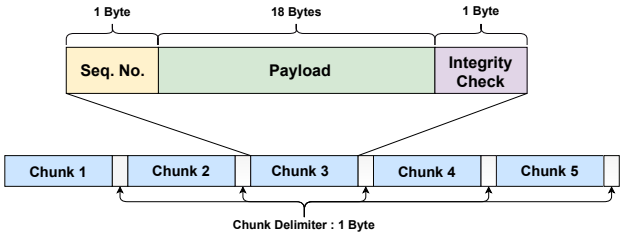

Figure 6.5: Details about individual chunks and how they are stacked before sending.

6.3.3 Modes of operation

There are two operating modes of Dolphin. We now list them.

Human callee mode: This mode requires the user in Internet shutdown region to find a trusted peer (or friend) in a region with uninterrupted Internet connectivity. The Dolphin user (caller) would then request this peer to setup the Dolphin callee infrastructure, for accessing Internet applications (such as Twitter, email *etc.*).

This is similar to users running circumvention systems in non-censoring countries, to support those living under repressive regimes. However, this model is not always conducive — what if one has no trusted peers (*e.g.,* friends) in non-censoring countries. To answer this question, we introduce the second mode of operation for Dolphin.

Automated callee mode: This mode provides a way for users to access Internet, even without a friend. We achieve this using cellular voice automation services (*e.g.,* Twilio [180]). Such services enable hosting the Dolphin server on a cloud, while providing a local number that users could call. Their automation engine forwards the audio (from the call) to the cloud hosted Dolphin server, that serves the encoded requests. During a shutdown, the Dolphin caller would only require knowing the phone number provided by Twilio (or other similar services) to access

Internet (implementation details in Sec. 6.4.3). However, unlike the trusted peer mode, such services would incur periodic subscription fee.

6.4 Implementation Details

In this subsection we describe how we implemented the above protocol to easily use it in real-world scenarios. It is assumed that both the caller and callee have access to a mobile phone with Bluetooth connectivity and a computer host (laptop/desktop). The caller runs the Dolphin client program and the callee runs the server.

6.4.1 Setup

The major components of the setup include caller and callee mobile phone and a host machine to which they are paired via Bluetooth (ref. Fig. 6.1). Pairing phones with the hosts ensures that during a cellular call, the audio input and output is captured from the host's sound card, rather than the mobile phones' inbuilt microphones and speakers, respectively. Data encoded audio is played out via the hosts' sound card. The output is treated as microphone input by the mobile phone, due to Bluetooth pairing. At the receiver, a similar pairing joins phone's speaker output to the host's sound card's input, allowing for decoding of received audio.

We used Android 10 version mobile phones for our setup. The host machines were provisioned with 4GB RAM, Intel i5 8th gen processor and ran Ubuntu 20.04. We assumed the caller to be in an Internet shutdown region. We ensured this by disconnecting the caller's phone and it's host to any sort of Internet access (WiFi, LAN or cellular data). On the other hand, callee is assumed to be in a region with Internet access *i.e.,* in our setup, the host on the callee's side had access to uninterrupted Internet via LAN/WiFi.

6.4.2 General Implementation Details

Connection establishment and call automation: The phones need to be paired to the host via Bluetooth manually, for the first time. Once paired, the subsequent pairing is automatic. We use `ofono` framework [194] for call automation as it helps manage various calling features – dialing and disconnecting, tracking call related events (call established/missed etc.), on the phone via Bluetooth. `Ofono` is accessed using a `dbus` interface (using `pydbus` [160] library).

Sending and receiving data: Since, we cannot directly send the data over the cellular voice channel, we first encode it into an audio signal. Additionally, sending the encoded data over the cellular voice channel, while ensuring minimal losses is not trivial. Various background processing and optimizations in the cellular infrastructure, *e.g.,* Voice Activity Detectors (VAD), Automatic Gain Control (AGC), can deteriorate the encoded bits significantly. VAD filters out all frequency components outside the human speech range, *i.e.*, it significantly attenuates frequencies close to 0 Hz or above 4 Khz. Thus, our modulation scheme must ensure that the data encoded audio lies between such a frequency range. Similarly, AGC dynamically adjusts the transmitted signal's amplitude. Hence, the modulation technique must also not rely on the amplitude of the voice signal to encode data. Hence, we selected Frequency Shift Keying (FSK) [195] to modulate the data bits. Since, it uses frequency to modulate data, AGC will not have much impact. Similarly, we ensure that the generated audio does not go beyond 4 Khz frequency range, and thus remains unaffected by VAD.

Thus, Dolphin relies on `minimodem` [196], a software modem which encodes (or decodes) data bits into (or from) audio tones using FSK. The rate at which data can be encoded/decoded can be varied. We thus present experimental results in Sec. 6.5.1 to establish the suitable data rates for transmitting data over the cellular voice channel.

Establishing secure channel We aim to establish a shared secret between the caller and the callee using DH. Traditional DH uses 128 byte public DH exponents. For regular network speeds, transferring such keys takes under half a second. However, in Dolphin, low data rates ($\approx$ 64 bps) can incur significant delays to exchange such keys. Thus, in Dolphin, we minimize this delay by using Elliptic Curve Diffie-Hellman (ECDH), instead of DH to establish a shared

secret. In ECDH the keys are 32 bytes and can be transferred relatively quickly (4 times sooner, as compared to DH). Moreover, the smaller key size does not compromise the security of derived keys [197]. The established shared secret is used by the peers as input to the PBKDF (password based key derivation function) to derive the key and IV. We used `pycrypto` [162] and `coincurve` [198] to perform the crypto operations.

Running Internet applications In Dolphin the callee accesses the Interent services on behalf of the caller, using the server utility. The current implementation, integrates Dolphin with three applications *viz.,* email, Twitter and news. The email has been automated using `smtplib` [103], and Twitter using `twython` [199] library. The news application is automated using `newsapi` [200], which returns concise news snippets based on a keyword query (in the form of text only).

6.4.3 Automated Callee Mode

As discussed previously, Dolphin can also work in a mode where the callee is completely automated and implemented on a cloud host using cellular voice automation services. Thus, the caller would not need to rely on a trusted peer. To achieve this, we need a way to manage cellular voice calls (automatically answering, playing audio, recording audio *etc.*) from a cloud host. In Dolphin, we achieve this with the help of the `Twilio` platform[1]. `Twilio` provides a diverse API to automate a variety of tasks related to voice calls (cellular, PSTN as well as VoIP). For Dolphin, we use the `Twilio` API to specify the operations to perform when a cellular call arrives on a particular phone number[2].

We configure `Twilio` API to manage any incoming call using a `webhook`. Further, a *call management module*, hosted on the cloud, interacts with `Twilio` (via a `webhook`) to manage calls. This module relies on `fastAPI` [201] to process `webhook` messages.

Once a call is established, we use the `Twilio` stream API to manage audio playback and recording. This API sends the incoming audio data to a `websocket`. An *audio management*

[1]Dolphin is not coupled to Twilio, it can be integrated with any other similar platform that provides cellular call management functionality.

[2]This number can be leased from either `Twilio` or elsewhere.

module listens on this `websocket` and plays it on the cloud host's sound card (using `pyaudio` library), which is then finally decoded using `minimodem` (running in receiver mode). After correctly decoding the requests, they are processed by the Dolphin server program. The corresponding response data is encoded and played back on the host's sound card, which gets relayed back to the `websocket` using the audio management module. The `websocket` sends the audio back to the caller, via the `Twilio` stream API.

Overall, managing call audio with `Twilio` enables Dolphin clients without a peer to still access Internet applications. All they require is dialing a phone number managed by `Twilio`.

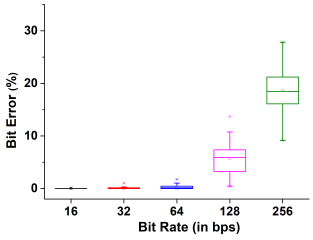

Figure 6.6: Bit error rate variation for different bit rates for 100B transfer.

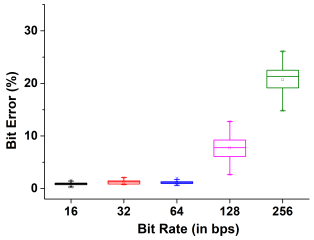

Figure 6.7: Bit error rate variation for different bit rates for 1000B transfer.

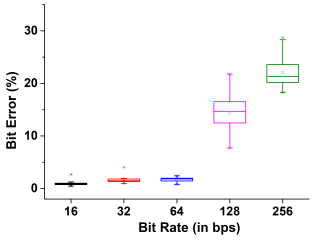

Figure 6.8: Bit error rate variation for different bit rates for 5000B transfer.

6.5 Data Collection and Results

We now present the details of various tests performed for evaluating Dolphin along with their corresponding results. Broadly we divided our experiments into two categories. First set of experiments are devised to test the viability of sending data at different bit rates over the cellular voice channel. Second set of experiments are conducted to gauge the performance of actual Internet applications when accessed via Dolphin. Additionally, we also conducted experiments to assess the performance of Dolphin for the automated callee mode configuration.

6.5.1 Performance of Dolphin at Various Encoding Rates

As already described, we encode and decode data bits into and from voice respectively. However, the underlying cellular voice channel used in Dolphin is lossy. Thus, our aim is to identify the

achievable bit rates with which the caller can transmit the data to callee over cellular telephony network.

Size (Bytes)	Bit Rate (bps) 16	32	64	128	256
100	0.01	0.15	0.29	4.86	18.76
500	0.61	0.9	0.92	5.71	19.32
1000	0.92	1.23	1.16	7.71	21.32
5000	0.97	1.8	1.69	16.07	22.28

Table 6.1: Error percentage for varying bit rates and file sizes.

Thus, we performed various experiments that involved encoding and sending of data bytes at different bit rates. In our experiments we used the setup as already described in Sec. 6.4.1. For these, we first established a cellular call from the caller to the callee and then sent the data of varying lengths (100, 500, 1000, 5000 bytes) at different rates (16, 32, 64, 128 and 256 bps). The goal of these experiments was to measure the bit error rate (BER) when the encoded data is transmitted over the cellular voice channel at different rates.

But, with Dolphin, the calculation of BER was not straightforward. The BER is defined as the percentage of corrupted bits in a transmission. However, standard BER calculation does not consider lost bits, but only bit flips. As Dolphin relies on lossy cellular telephony network, the resulting errors not only include bit flips but often also results in bit losses. Thus, general BER techniques cannot be directly used; rather we used edit distance [202] as a metric to measure the bit errors. The edit distance algorithm outputs the minimum number of bit operations required to convert the received data to its original form. For our scenarios these bit operations represent all possible errors – bit flips and losses. Since in our experiments we controlled both the caller and callee, we could compare the bits sent from those received. This enabled us to compute the edit distance.

Further, the edit distance represents the total bit errors. Dividing it by total bits transmitted would yield BER for data transmitted. In all our experiments, we computed the corresponding BER using the edit distance metric. We repeated each experiment for a particular bit rate and data length 30 times.

Corresponding to different bit rates (16, 32, 64, 128 and 256 bps) we tabulate the average

BER in Tab. 6.1 and present the complete error distributions in Fig. 6.6, Fig. 6.7 and Fig. 6.8 for 100, 1000 and 5000 bytes respectively. It is evident from the table and the graphs that upto 64 bps, the BER is relatively low *i.e.,* less than 2 %. At 16 bps the BER was even lower *i.e.,* less than 1%. However, the BER increases drastically with relatively higher data rates *i.e.,* 128 and 256 bps. *E.g.,* with 256 bps the BER is around 20%.

Ideally one would want to transmit the data at higher bit rates using Dolphin (*e.g.,* >256 bps). This would reduce the overall latency. However, as demonstrated through our extensive experiments, *higher bit rate results in higher error rates, eventually rendering the cellular voice channel unsuitable for data transmission*. On the other hand, if we send the data at extremely low rates (*e.g.,* < 16 bps), the data would be delivered with least errors, albeit increasing the overall end-to-end delay. Thus, 64 bps seems like a good trade-off point between latency and errors, and thus we selected it for performing subsequent experiments.

However, since control information (*e.g.,* acknowledgements *etc.*) is generally smaller in size, compared to data chunks, we sent them at low rates (*i.e.,* 16 bps), to further minimize their chance of corruption. This step does not impact the overall latency much.

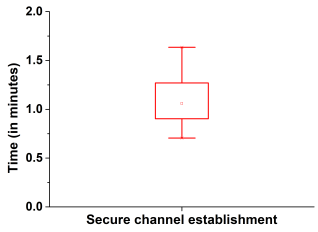

Figure 6.9: Dolphin's secure channel establishment time.

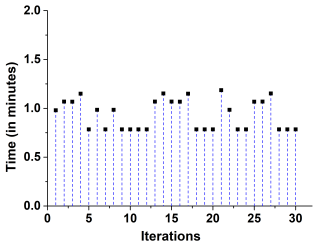

Figure 6.10: Time taken to tweet 280 characters (max. limit) using Dolphin.

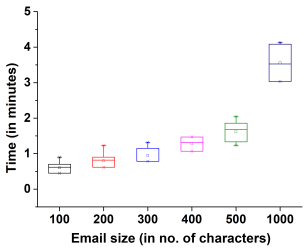

Figure 6.11: Time taken to send an email of varying sizes using Dolphin.

Varying cellular connectivity: Notably, the above set of experiments assumes the caller to be in a shutdown region, and the callee to be outside. The caller phone was thus manually switched to use only 2G voice[3] and the callee's phone was enabled with 4G voice connectivity. However, it may so happen that the peers may not always have such connectivity due to various reasons (such as intermittent signal or proximity to base station *etc.*).

[3]2G is a bare minimum cellular voice connectivity.

Thus, to test the feasibility of all cases for caller and callee, we repeated the above set of experiments varying bit rates and data sizes for different combinations of 2G, 3G and 4G voice connectivity[4]. The BER received in these scenarios (*e.g.,* caller (2G) callee (4G), caller (3G) callee (4G) *etc.*) did not vary much (ref. Appendix D.1 for details), depicting similar performance for different connectivity scenarios. Thus we continued using 64 bps as our default data sending rate.

Varying cellular providers: We also performed the aforementioned experiments for different cellular service providers. We observed similar BER (*e.g.,* < 3% error for bit rates < 64 bps) when we tested Dolphin for four popular providers. Thus, one can safely assume that Dolphin would successfully work across cellular providers.

Geographical variation: In the aforementioned experiments, both the caller and the callee were in a close proximity (in the same building) and may thus be connected to the same cellular tower. Thus, one may argue that the results may vary if the geographical distance between the caller and callee is increased as it would involve data to travel over multiple cellular towers. Thus, we repeated the experiments with the caller and callee in different cities ($\approx$ 1100 miles apart), as well as different countries ($\approx$ 3600 miles apart). We observed similar BER for downloading 100 to 500 byte files at different bit rates (16, 32,..., 256 bps), with less than 3% error for 64 bps. Moreover, we additionally show in the subsequent section (automated callee mode) that even when the callee infrastructure was hosted on a cloud service, the results did not vary much. This establishes that Dolphin's efficacy is not generally impacted by geographical variations.

6.5.2 Performance of Internet Applications

In the previous section we presented the BER observed when transmitting content at different bit rates under different scenarios. In this section we now quantify the performance when Internet applications (*e.g.,* email, twitter and news apps) are used over Dolphin.

As already described (Sec. 6.3.1), Dolphin works in two phases *i.e.,* the secure channel establishment phase and data transmission phase. Thus, first we quantify the time incurred to

[4]Here again, we revoked any Internet connection on the caller end, to ensure a shutdown scenario.

establish a secure channel between the caller and callee. As depicted in Fig. 6.9, we observe on an average a minute to establish an encrypted channel, with the worst case being around 1.7 minutes (experiment repeated 30 times). Ideally, to obtain forward secrecy, the caller should establish the secure channel every time it sends or accesses some content. However, in case the user wishes to reduce the overall latency, the Dolphin caller utility can be configured to establish a key once and use it for all data transfers for a specified time duration *e.g.,* a day.

Next, we measured the time taken by Dolphin to access (or send) content using different Internet applications. We tested three applications *viz.,* email, twitter and news. For each application, we measured the time taken by the caller to send (or receive) the complete data *reliably*. First we tested the time taken to tweet a 280 character message (maximum size for a single tweet). We repeated this experiment 30 times and observed that on an average it took under a minute to tweet this message (ref. Fig. 6.10). Similarly, we sent emails of varying size (100–1000 characters) and again recorded the time elapsed in reliably sending them. This experiment was also repeated 30 times for different email sizes. Overall results are depicted in Fig. 6.11. It is evident that it takes $\approx$ 1.7 minutes (102 seconds) to send an email of 500 characters. Lastly, we recorded the time taken (at the caller end) to retrieve 10 concise news snippets (around 60 characters each). We observed a total time of 2 minutes to receive these 10 snippets on an average.

Thus, the overall end-to-end time for accessing applications using Dolphin would be the sum of secure channel establishment time and the data transmission time. *E.g.,* sending an email of 500 characters would in average case take 2.7 minutes (about 160 s). Thus, by and large, our results depict that most of the implemented applications would take only a few minutes to deliver the content end-to-end reliably.

6.5.3 Automated Callee Mode Performance

Size	**Bit Rate (bps)**				
(Bytes)	**16**	**32**	**64**	**128**	**256**
100	0.23	0.82	1.27	8.9	22.51
1000	0.284	1.18	1.41	10.8	22.2

Table 6.2: Error percentage for varying bit rates and file sizes (100B and 1000B) for automated callee mode.

Similar to the previous experiments, we performed tests to gauge the efficacy of callee side automation. These experiments were essentially performed to measure if there is any potential impact on performance, when the callee infrastructure operates from the cloud. In the first experiment, we transmitted files of 100 and 1000 bytes at varying bit rates (16,32,...,256 bps) and recorded the BER. As depicted in Tab. 6.2, BER of 0.8% was observed when data was transmitted at 32 bps (for 100 byte content), and 1.3% when sent at 64 bps. Thus, it is evident that even with callee completely on the cloud, the overall performance (in terms of BER) did not vary much, indicating minimal processing overheads. Additionally, we also sent tweets and email in the automated callee mode and observed similar performance with an email of 100 characters delivered reliably in under a minute (54.8 seconds on an average).

Overall, the results establish that it is feasible to use delay tolerant and lightweight Internet applications using Dolphin in Internet shutdown regions with transmission times in the range of a few minutes.

Anecdotes: While conducting the experiments, we observed an Internet shutdown in the region of one of the authors. This provided us an opportunity to test Dolphin during an actual shutdown. Thus, we conducted experiments by transferring data from the shutdown region to a callee placed in another location with Internet connectivity (managed by another author). As expected, we observed similar performance in this scenario (300 bytes transferred reliably in about a minute), further establishing Dolphin's efficacy.

6.6 Security Aspects of Dolphin

We being by describing our adversary model and the different types of possible attacks.

Threat model: In general, it is difficult to assume a precise threat model for Dolphin, as to the best of our knowledge, telecom providers as adversaries have not been studied at great length in the existing literature. Thus their capabilities are difficult to characterize. Moreover, unlike regular network eavesdroppers, cellular voice channel cannot be analyzed by capturing packets;

cellular voice networks do not work on the regular Internet's store-and-forward model[5]. Thus, we believe that it will be difficult to perform real-time traffic analysis on ongoing calls for telecom operators, but they may easily intercept and record audio calls. We confirmed this by communicating with a major telecom provider operating in a developing country with frequent shutdowns.

Thus taking the above points into consideration, we assume in our threat model that the adversary will not be able to perform real-time analysis on cellular voice channel to actively detect Dolphin. However, it may perform offline analysis on recorded cellular calls to identify if a particular call was part of Dolphin. Further, as an extreme step, adversary might attempt to add noise or perturbations in voice data with an aim to completely disrupt Dolphin. But, he would refrain from degrading the quality of voice (from the added noise) to an extent that it becomes practically unusable for ordinary calls. Further, we also assume that the adversary has the capability to restrict cellular communication for calls destined to specific mobile numbers. Overall, we assume the adversary would not disable the cellular voice channel during the Internet shutdown, as it may negatively impact several critical services of the state. This is already observed in multiple recent Internet shutdowns [176, 175, 177, 178].

1. Man-in-the middle attack (MITM): Dolphin uses Diffie-Hellman (DH) scheme to derive a shared secret between the peers. It could be argued that an adversary may attempt to launch MITM attack during DH key exchange [203]. This attack requires the adversary to derive two shared keys, one with the caller and one with the callee by intercepting the voice call and playing it's own public DH keys as encoded voice data. However, since we use public key of callee to safeguard the public exponent of DH exchange, the adversary won't be able to carry out such attacks.

2. Voice perturbation attacks: To disrupt Dolphin, an adversary may attempt to induce intentional perturbations or noise in voice calls. The rationale behind this attack is that, these perturbations could corrupt the encoded data of Dolphin users' calls. However, innocuous cellular users may perceive it as some disturbance while conversing. This attack may turn out to

[5]Unlike routers that store, process and forwards Internet packets, the real-time nature of cellular channel would not allow for such operations.

be very powerful because the adversary can aim to completely block Dolphin without having to even detect if Dolphin is under use[6].

There exist largely two ways with which an adversary can try to induce these perturbations One way is to just drop or disrupt voice samples for some short duration say 0.1s, 0.2s *etc* after some fixed or random interval. The other way is to have some constant disturbance (such as a low frequency hum sound) throughout the duration of the call.

Case I: We start by exploring how the adversary can use the first way to disrupt Dolphin. A simple attack would be to drop voice samples repeatedly after some randomly selected intervals. However, the reliability layer in Dolphin helps recover from random data losses and thus this way the attack may not be very effective. But, a determined adversary may induce perturbations intelligently such that all the transmitted chunks are corrupted. This could lead to endless re-transmission of data between the Dolphin peers[7]. To do so, the adversary would need to induce perturbations at very small intervals. *E.g.,* the adversary may need to introduce perturbations every 2.5 s to corrupt each 20 byte chunk transmitted at 64 bps. But, in practice, this attack could render cellular voice unusable for regular callers due to the unpleasant periodic disturbance (after every 2.5 s) throughout the call.

To quantitatively verify this, we conducted experiments to determine how does periodic disturbance affect perceivable voice quality. Thus, to measure voice quality we used PESQ (Perceptual Evaluation of Speech Quality) [**?**], a metric standardized by International telecommunication Union (ITU) to measure the perceptual audio quality. PESQ scores show very high correlation with Mean Opinion Scores (MOS) given by actual humans.

The PESQ metric takes the original audio and the audio that undergoes degradation as input and outputs a score between 1 to 5, with 1 being the worst and 5 being the best audio quality. A PESQ score of above 3 is considered as good whereas a score less than 3 is not considered ideal. Moreover, a score of below 2 is considered as poor and unusable. Thus to perform our experiment, we took a sample audio containing human speech and introduced perturbations in

[6]It must be noted that this attack is possible because the data carrying channel is the voice channel and thus normal users may be able to perceive human audio despite the disturbances introduced in the voice channel.

[7]It is important to corrupt all the chunks because if few chunks are corrupted and few are not then the reliability protocol will be able to retransmit and recover the corrupted chunks over time.

it by removing samples of 0.1s from it after every 2.5s. Then we calculated the PESQ score between the original audio and the audio with the periodically disturbed samples. We observed a PESQ score of 1.6, clearly establishing that the audio in the cellular channel would become highly imperceivable if such a disruption is introduced.

However, as a workaround, the adversary can also try to disrupt the channel by attempting to corrupt only all the acknowledgements instead of the chunks. This way adversary would require to drop samples after every ≈12.5s as in the default configuration, we transmit five chunks before transmitting an acknowledgement. Moreover, we calculated the PESQ score for this scenario (*i.e.,* disruptions after every 12.5s) and got a score 3.6, depicting that such a disruption will lead to complete disruption of Dolphin without severely impacting the perceptual quality for normal calls. However, as a countermeasure to this attack, we can slightly alter Dolphin's default configuration by sending an acknowledgement after every chunk instead of after a batch of five chunks. This would force the adversary to again cause disruption after every 2.5s, which, as previously seen, cannot be done by the adversary as it leads to the voice channel becoming unusable for regular users. However, Dolphin will still be able to function. Hence, we believe, the adversary would refrain from performing this attack.

Case II: Next, we move to the scenario where the adversary can try to introduce continuous noise throughout the duration of the call, hoping that it will disrupt Dolphin's functioning, without making it unusable for regular users. To that end, the adversary can introduce a constant low frequency sound in all cellular calls. To normal users this should sound like a constant background sound (such as a hum or a continuous beep). We started with a continuous 50 Hz beep and it did not have much impact on quality of call (PESQ = 3.8) or on Dolphin (error rate = 1.3%). We kept increasing the frequency of the noise and found out that at about 440 Hz, the introduced noise lowers the PESQ score to about 1.7 thereby making it unsuitable for regular calls. However, the error percentage of Dolphin is still not affected much and is 2.1%[8]. Thus, it would prove to be a futile exercise for the adversary to disrupt Dolphin with continuous noise as well.

[8]This happens because Dolphin's functioning is only dependent on correctly decoding the frequency samples so that they can be converted to data bits, and it is observed that the low frequency tones do not affect this, but impacts human perception.

3. Active probing attacks: The aim of this attack is to enumerate possible Dolphin callee numbers and eventually dropping all calls made to them. To do so, the adversary can itself pretend as a Dolphin caller and may brute force some suspicious mobile numbers. The adversary may confirm the Dolphin callees by checking if it can avail Dolphin service through these suspicious numbers.

However, to avail Dolphin's service the adversary requires the DH public exponent of the callee, which is shared out of band with the caller and is a secret. If the caller fails to provide requisite data encrypted with this key, or the provided data is incorrect, the callee program just plays an audio containing the traditional "hello" sound a few random times and then disconnects the call. This behaviour is similar to how a normal user would react if it received a call with some gibberish sounds. However, effective active probing resistance is an open research problem and the current standard is to adapt according to the measures taken by the adversaries [**?**]. Thus, as Dolphin becomes popular, adversaries may be able to find unique ways with which they could actively probe and detect Dolphin peers. As and when such attacks evolve, we would accordingly design countermeasures to avoid such detection.

4. Replay Attacks: An adversary can attempt to replay a part of audio or complete audio in order to confirm if Dolphin service can be availed on a particular mobile number. For this the adversary can attempt to replay the starting few seconds of suspicious calls to the potential callee mobile number in those calls. If the adversary obtains an adequate response, she confirms that the callee is running a Dolphin server, and can block it. However, firstly, the suspicious audio may be noisy and probably contain arbitrary data due to lossy channel and multiple retransmissions, significantly decreasing the chance of successfully detecting the callee. But even in the unlikely scenario where the adversary obtains an audio with no losses and is able to successfully transfer it to the potential callee, the latter will still not respond as the initial boostrapping information (ref. Sec. 6.3) must include fresh timestamps (otherwise they are silently dropped by the callee and responded with as described in the active probing attacks).

5. Offline Analysis: As assumed in the threat model, the adversary can record all cellular calls of the region and analyze them offline to confirm if they were part of Dolphin. One easy approach that can be applied by the adversary is to try and decode the recorded audio using the public

information of Dolphin implementation.

After decoding the audio, the adversary can check if the resulting data contains valid CRC checksums, periodically after every few bytes. The signature of periodic checksums would be unique to Dolphin and would lead to its detection. We remove this detectable feature by simultaneously generating some extra bytes from the KDF (during secure channel establishment), and XOR-ing the CRC values with these extra bytes. Since the KDF is given the same input by both peers, they will be able to derive the same bytes to invert the XOR.

However, the adversary may apply some advanced techniques such as signal processing to differentiate normal audio from Dolphin audio. In such a scenario, the adversary may be able to detect Dolphin calls, identify callee numbers and block them. However, in this case, the caller can try to obtain other callee numbers which are not currently blocked. Thus, this scenario is similar to the one faced by traditional proxy based systems, where the adversary keeps finding and blocking them while the users keep hunting for the active ones. It must be noted that Dolphin will not be completely hampered in this scenario. The users can use it so long as they can find an unblocked callee number. Moreover, we can leverage platforms such as `Twilio` such that availability of active callee numbers can be maintained. Though, this is not a permanent solution, as it is even an open problem in traditional proxy systems' literature. However, in future, we shall strive to find a more permanent solution.

6.7 Discussion and Future Work

1. Using Dolphin as a covert channel: The primary focus of designing Dolphin has been to provide Internet access in regions experiencing shutdown.

However, Dolphin can also be used for various applications in non-internet shutdown regions such as a low bit rate covert channel to perform tasks such as exchanging secret information. Moreover, Dolphin cal also be used as an out-of band communication channel to bootstrap various anti-censorship systems [9, 82] *etc.*

2. On privacy implications of initiating connection by the peer on behalf of caller: In

dolphin, we assume that the peer in non-shutdown region is trusted and thus sharing password of protected accounts (email, twitter) should not be an issue. However, there are alternatives which one can use to protect the privacy of their accounts. First, caller can enable two-factor authentication on its password protected accounts so that every new access requires providing the OTP (received via SMS). This prevents the callee from accessing the caller's account without the latter's knowledge. Second, caller can use OAuth token based authentication schemes. OAuth tokens allow for stricter control and can be configured to perform specific tasks with confined scope. *E.g.,* in case of Twitter the user can generate tokens that allow only for tweeting and can share these via Dolphin whenever it wants to tweet from its account. The current Dolphin implementation has been tested to work with the above methods (for Twitter and Gmail). However, the user requires configuring its account for OTP access and generate tokens before any shutdown event. If the user is not able to perform this task beforehand, then an alternate approach as described in Mailet [95] could be used that relies on multiple parties to derive the password, with no one party having complete information.

3. Maximum achievable transmission rates for Dolphin: We experimentally demonstrate that Dolphin traffic experiences very low error rates, when transmitted at 64 bps. Further, as already depicted, this rate seems acceptable for various "lightweight" applications like email and Twitter. Higher data transmission rates incur significant error and eventually re-transmissions. Thus, increasing the data rate further is an important direction for future work.

However, it must be noted that Dolphin's reliability layer runs atop any underlying data framing and modulation mechanism. Thus, any high bit rate modulation schemes proposed in future, could be easily used with Dolphin.

4. Alternatives to cellular voice: One may argue that voice may not be the only medium using which people can communicate in an Internet disrupted region. An alternative may be automating cellular SMS to transfer data. SMS has an advantage of providing a reliable data channel. However, it suffers from a few drawbacks.

- Many countries restrict the number of SMS that can be sent/received per day [204, 205]. But no such restriction is generally imposed on voice calls.

- Sending and receiving bulk SMS messages in a short duration can make it very easy for the adversaries to suspect and identify Dolphin users. In contrast, a long duration (say an hour long) voice call is relatively less suspicious, as long duration calls are not generally unusual.
- Similar to the voice channel, SMS messages are not encrypted end-to-end. However, one cannot securely derive shared keys using DH key exchange in SMS. This is because, SMS are akin to "store and process" architecture. This makes it relatively easy for the adversary to perform MITM on the DH key exchange. The adversary could very easily intercept the key exchange and replace the exchanged keys with its own.

These limitations make SMS less suitable for sending data in comparison to voice channel. Additionally, some social media platforms (Twitter, Facebook *etc.*) provide a service that allows users to post and retrieve content by sending SMS to specific phone numbers provided by these platforms. Such services could also be used to bypass shutdowns. However, such phone numbers are uniquely assigned to each country and thus it is easy for the adversary to block them.

5. Alternative to Bluetooth for connecting laptop and phone: There are other alternatives for connecting the mobile phone and the host machines to transfer audio. One example is using an audio jack to connect a phone to a host machine essentially making the host as the source and sink of audio for the cellular call. However, it must be noted that audio jacks typically do not allow the host computer to control the cellular call (initiating or terminating call, checking call state *etc.*). Bluetooth by default allows the host computer to control and automate such call related functions on the phone, and thus is also used by Dolphin. However, in worst case, if bluetooth connectivity is unavailable, even audio cable (or other alternatives such as WiFi) could be used to transfer data between host and the phone, with the requirement of manual intervention to manage calls.

6. On availability of cellular voice services in shutdown regions: One may question the availability of cellular services in Internet shutdown regions. However, there are abundant documented instances [175, 176, 177, 178] where the regions undergoing Internet shutdown have fully-functioning cellular services. Moreover, "Internet shutdown" by name as well as definition implies, disruption of *only* the Internet services. It by default does not include disabling any other service. Thus, overall it can be assumed with sufficient confidence that the region of

Internet shutdown have cellular services fully functional.

However, in worst case scenarios one may fear that the adversary may eventually disable cellular services altogether, thereby disrupting any chance of external communication. We believe that doing so would be self-sabotaging for the adversary, as cellular connectivity is essential for maintaining and disseminating important governance information across a region. Disabling cellular service would not only hamper normal user living in that region but would also affect the adversaries' own communication, which may eventually lead to complete administrative as well as executive failure. However, a determined adversary may decide to allow only the landline connections and may disable all the mobile cellular connections. To counter this, Dolphin may be extended to work even with landline phones (by possibly connecting an external device that may interface landline phones to the host machine). This can be an interesting direction for future work.

7. On usability of Dolphin: Dolphin is designed keeping in mind its usability. Any person with a host computer and a Bluetooth enabled phone should be able to use it. In future one could build a smartphone only version of Dolphin. The Dolphin programs does not require superuser privileges and thus would not require a rooted/jail-broken phone.

8. On using error detection and correction techniques: One can argue that an alternative approach to provide reliability could be to employ error detection and correction techniques. However, there are multiple problems with using them in case of Dolphin. Firstly, such techniques need a bound on the maximum number of bit errors that can occur during transmission. Predicting the exact bounds in case of unreliable and unpredictably lossy voice channel is difficult. Secondly, even if we were able to somehow bound the bit errors, the error correction techniques are not built to tolerate bit losses (that happen in case of Dolphin). The standard techniques only work in cases of bit flips. Thirdly, such techniques incur a significant data overhead even when there are no errors in the received data. In contrast, Dolphin's reliability protocol ensures that data will be re-transmitted only when some data is lost or corrupted, thereby minimizing the overheads.

9. Potential harmful impact on Dolphin users: For applications such as twitter, one can argue that the repressive regimes could track individuals (using their official user accounts) from their

tweets, and penalize them, so as to discourage using Dolphin in future. This could be potentially harmful for such users. Thus, one needs to be careful while using such applications over Dolphin. As a possible mitigation for public platforms, the users could create secondary accounts and tweet using these accounts rather than their original ones. However, other non-public applications such as email, news *etc.,* are immune to such potential harms.

6.8 Concluding Remarks

Recently, we have globally observed a sudden rise in an extreme form of censorship — *i.e.,* Internet shutdowns. To circumvent such steps we present Dolphin, a system that can provide access to lightweight and delay tolerant Internet applications, by utilizing the cellular voice channel. Dolphin serves the request of a client in shutdown region by relying on trusted peers outside such regions which access Internet applications, on behalf of the client.

Dolphin works by encoding the data to be transferred, into voice, and sending it over a cellular call. The content is encoded such that it evades all forms of background processing performed in the cellular network. On top of that it provides reliability over the known lossy voice channel, as well as secure data transmission, resistant to eavesdropping. We demonstrate the feasibility of Dolphin by implementing and testing it for real Internet applications such as email, tweet and news. Across all our experiments for these applications we observed that it takes only a few minutes to access all of them. Moreover, Dolphin's modular design can be easily extended to other applications.

Overall, we believe that there is a need to build systems to stay ahead in the censorship arms race against the adversary's extreme measures. Dolphin is a step ahead in this direction and we hope this will further propel development of more such systems for these scenarios.

Chapter 7

Conclusion and Future Work

Over time we have seen consistent attempts by the adversaries to control and restrict the free flow of information over the Internet. This control impacts the netizens as they only have access to partial information about everything, eventually forcing them to have a biased view. Thus, privacy practitioners, free speech activists and researchers have attempted building various solutions that aim to evade such restrictions. However, there still exist various research gaps. These include no readily available solutions for certain applications (*e.g.,* anonymous voice calling), applications (e.g., accessing censored website or content) for which existing solutions could be improved (*e.g.,* performance, blocking resistance, and deployability *etc.*) and lastly there are certain extreme restrictions for which there are no existing solutions *i.e.,* Internet blackouts.

Thus, in this thesis, we attempted to fill the aforementioned gaps by analyzing, proposing and building various novel solutions. We first addressed the problem of anonymous voice calling for which there are a few proposed solutions, but all of them are non-functional. Thus, we studied Tor, a popular and functional anonymity system (with >2M daily users), for testing the feasibility of performing anonymous voice calls. We conducted a large scale measurement study that involved conducting 0.5 million voice calls over a span of a year, and contrary to existing literature, we demonstrated that it is possible to conduct anonymous voice calls using Tor. Our research shows that by and large most Tor relays provide adequate capacity to support good quality voice calls. This research thus established that an already existing functional system

(Tor) could be used to conduct anonymous voice calls.

Next, we focused at the problem of accessing censored content. There are existing solutions such as proxies, VPNs *etc.,* that can be used. But, the censors tend to block them easily. To make such systems extremely difficult to block a new approach *viz.* Decoy Routing (DR) was recently proposed. Rather than making end hosts as proxies, DR assumes network routers as proxies. This is because, it is trivial to block end-hosts using IP filtering. However, simply blocking the IP of the router would not result in censorship, as the web traffic is not destined to the routers, but some end-host (generally a server). There exist a few DR solutions, and they use commodity servers as DRs to analyze ISP scale traffic (for detecting the DR requests). Since, commodity servers are not built to handle ISP scale traffic, these solutions do not provide good performance. Moreover, such schemes require inspecting all flows of the ISP (DR and non-DR), posing privacy concerns for the non-DR users.

Thus, we proposed SiegeBreaker, a novel DR system that overcomes the above drawbacks. SiegeBreaker is built using SDN devices. As SDN devices are programmable and are built to handle ISP scale flows, we are able to remove performance bottlenecks. Moreover, the protocol is designed such that the DR maintainers only analyze and serve the DR flows ignoring the non-DR flows, thereby preserving privacy of non-DR users. We designed and implemented SiegeBreaker and tested it extensively using a hardware SDN switch. We showed that SiegeBreaker ensures performance comparable to direct TCP connections. Thus, we successfully demonstrated that it is possible to build performant and privacy preserving DR solutions. Although DR solutions provide excellent blocking resistance, but they require support from ISPs for deployment. Thus, overall, existing circumvention solutions fall under two broad categories — easily available but also easy to block, or difficult to block but pose deployment challenges.

Thus, as a next step we attempted to build a system that does not pose additional deployment challenges and at the same time is not easy to block. To that end, we built a new system Camoufler, that utilizes the Instant Messaging (IM) platforms to tunnel censored content. Firstly, IM's ubiquitous usage helps reduce the deployment challenges. Moreover, it is not trivial for the censor to block all IM apps in order to disable Camoufler. This is because, IM is now an integral part of netizens' daily lives, and its reckless blocking may cause heavy collateral damage to the

censor. Considering these salient features of IM, we built and implemented Camoufler for five popular IM apps and demonstrated that it can be effectively used to access censored websites with near-native performance.

Lastly, we examined an extreme scenario of Internet shutdowns, where all existing circumvention solutions are unusable. For such scenarios, we propose a first-of-its-kind solution, Dolphin, that can enable users in shutdown region to access basic Internet services with the help of a user in non-shutdown region. Dolphin uses the cellular voice channel to transfer data bits by disguising these bits as audio and transmitting them during a regular cellular call. We counter all the challenges posed by the cellular channel and implemented Dolphin. We demonstrated that users can easily access emails, news snippets and twitter with the help of Dolphin in a few minutes.

7.1 Future Work

There are various possible future directions that can be taken to build upon or improve the work done in this thesis.

For performing anonymous voice calls, we considered the standard codecs that by and large utilized less than 120 Kbps bandwidth. However, there are cases where the codec encodes at higher rates (> 200 Kbps). Thus it is an interesting future direction to investigate the performance in such cases that can even be extended to study the feasibility of performing video calls or video streaming over Tor. Additionally, approaches to facilitate recruitment of volunteer operators for the currently non-functional but promising systems (such as Herd [4]) is another important future direction.

Similarly, we built a DR system by using traditional SDN architecture which involves a controller that manages the SDN switches to perform ISP scale tasks. In the future, we can look at advanced SDN architectures such as using P4 switches to embed the complete DR protocol within such switches, without requiring the controller. Moreover, another future work is to remove the reliance of SiegeBreaker on some out of band channel for bootstrapping the system,

without compromising on the performance or the privacy guarantees.

All the IM applications we considered for Camoufler had centralized architectures *i.e.,* messages between peers are relayed via a central IM server. This is because all the popular applications with significant users follow such centralized architectures. However, other apps that have de-centralized or peer-to-peer architecture (Briar [206], Jami [207], firechat [208] *etc.*) can also be used with Camoufler. The messages in these apps may transit multiple IM peers before they are delivered, introducing delays. However, due to the lack of a single point to inspect all the client's traffic, such apps have the inherent advantage of being collusion resistant. Thus, we may explore strategies to mitigate the limitations introduced by these apps and to integrate them with Camoufler in future.

Lastly, Dolphin currently does not support session management *i.e.,* if a call disconnects in between transmission of data, the complete data transfer will start again. Thus, in future we aim improve Dolphin by incorporating such session management features for better usability. Additionally, we demonstrate running lightweight Internet applications using Dolphin. However, it will be desirable if we can provide some basic web browsing feature with Dolphin. Thus, it will be an important future work to try and increase the underlying bit rate for transferring data over the voice channel, such that we can support web browsing.

Insights

There are various takeaways and insights that we gathered while building performant, privacy-enhancing and blocking resistant communication systems.

Largely, we believe that approaching a problem in the research area targeted by this thesis, depends on the kind of use case and the application for which we are building it. In this thesis, we divided the problems in three categories and devised specific solutions for each of them. For instance, in the use case of anonymous voice calling, there are no functional systems that are available for usage. Thus, in such a scenario, we believe it is of paramount importance to have a functional system first, with the available traddeoffs. In contrast, for use cases where we already have functional implementations (*e.g.,* accessing blocked web content), one should try to minimize the tradeoffs inherent in these systems and should aim to make such systems more

robust with respect to blocking resistance, performance, privacy guarantees, deployability *etc.*. *E.g.,* Decoy Routing systems provide excellent blocking resistance but suffer while providing adequate performance, privacy guarantees and deployability. With SiegeBreaker, we attempted to provide good performance and preserve the privacy of users. In future, one can aim to improve the deployability of such systems.

However, in future, there might be cases where the given problems do not fall in any of our categories. In such a scenario appropriate categorization and solution approach would be required.

Overall, we believe that the researchers need to be a step ahead of the adversary in order to maintain free flow of information over the Internet. In the near future, as the adversaries evolve, the community needs to counter such evolution by building diverse, robust and resilient systems.

References

[1] U. N. GeneralAssembly, "Thirty-second session, human rights council, agenda item 3. the promotion, protection, and enjoyment of human rights on the internet," https://www.article19.org/data/files/Internet_Statement_Adopted.pdf.

[2] A. A. Niaki, S. Cho, Z. Weinberg, N. P. Hoang, A. Razaghpanah, N. Christin, and P. Gill, "Iclab: a global, longitudinal internet censorship measurement platform," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 135–151.

[3] R. Sundara Raman, P. Shenoy, K. Kohls, and R. Ensafi, "Censored planet: An internet-wide, longitudinal censorship observatory," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 49–66.

[4] S. Le Blond, D. Choffnes, W. Caldwell, P. Druschel, and N. Merritt, "Herd: A scalable, traffic analysis resistant anonymity network for voip systems," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 639–652.

[5] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," Naval Research Lab Washington DC, Tech. Rep., 2004.

[6] "A study of voip performance in anonymous network-the onion routing (tor)," Ph.D. dissertation, 2013.

[7] A. Panchenko, L. Pimenidis, and J. Renner, "Performance analysis of anonymous communication channels provided by tor," in *2008 Third International Conference on Availability, Reliability and Security*. IEEE, 2008, pp. 221–228.

[8] A. W. Rix, J. G. Beerends, M. P. Hollier, and A. P. Hekstra, "Perceptual evaluation of speech quality (pesq)-a new method for speech quality assessment of telephone networks and codecs," in *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, vol. 2. IEEE, 2001, pp. 749–752.

[9] J. Karlin, D. Ellard, A. W. Jackson, C. E. Jones, G. Lauer, D. P. Mankins, and W. T. Strayer, "Decoy routing: Toward unblockable internet communication," in *USENIX workshop on free and open communications on the Internet*, 2011.

[10] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[11] "Accessnow report on internet shutdowns, accessnow, 2019," https://tinyurl.com/y4c7w8gy.

[12] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "Sip: session initiation protocol," Tech. Rep., 2002.

[13] M. Handley, V. Jacobson, and C. Perkins, "Sdp: session description protocol," Tech. Rep., 2006.

[14] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "Rtp: A transport protocol for real-time applications," Tech. Rep., 2003.

[15] I. ITU-T, "Recommendation g. 114," *One-Way Transmission Time, Standard G*, vol. 114, 2003.

[16] R. ITU-T and I. Recommend, "G. 114," *One-way transmission time*, vol. 18, 2000.

[17] S. Heuser, B. Reaves, P. K. Pendyala, H. Carter, A. Dmitrienko, W. Enck, N. Kiyavash, A.-R. Sadeghi, and P. Traynor, "Phonion: Practical protection of metadata in telephony networks," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 1, pp. 170–187, 2017.

[18] "Openflow Switch Specification," https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[19] "Whatsapp," https://whatsapp.com/.

[20] "Telegram," https://telegram.com.

[21] "Slack messenger service," https://slack.com/.

[22] Flock messenger service. [Online]. Available: https://flock.com/

[23] Statista, "Number of mobile phone messaging app users worldwide," https://www.statista.com/statistics/483255/number-of-mobile-messaging-users-worldwide/.

[24] "Wechat," https://wechat.com/.

[25] M. Marlinspike and T. Perrin, "The x3dh key agreement protocol," *Open Whisper Systems*, 2016.

[26] "Internet shutdown tracker in india," https://internetshutdowns.in.

[27] "Policy brief: Internet shutdowns, Internet Society, december 2019," https://www.internetsociety.org/policybriefs/internet-shutdowns.

[28] "Internet outage detection and analysis," https://ioda.caida.org/.

[29] "Internet outages map by cisco," https://www.thousandeyes.com/outages/.

[30] *NSA collecting phone records of millions of Verizon customers daily*, The Guardian, june 2013, https://www.theguardian.com/world/2013/jun/06/nsa-phone-records-verizon-court-order.

[31] *Users guide for PRISM Skype collection*, August 2012, https://www.spiegel.de/media/media-35530.pdf.

[32] *NSA uses powerful toolbox in effort to spy on global networks*, December 2013, https://www.spiegel.de/international/world/\the-nsa-uses-powerful-toolbox-in-effort-to-spy-on-global-networks-\a-940969.html.

[33] G. Danezis, C. Diaz, C. Troncoso, and B. Laurie, "Drac : An architecture for anonymous low-volume communications," in *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2010, pp. 202–219.

[34] D. Schatz, M. Rossberg, and G. Schaefer, "Reducing call blocking rates for anonymous voice over ip communications," in *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2017 9th International Congress on*. IEEE, 2017, pp. 382–390.

[35] "Tor metrics," https://metrics.torproject.org/.

[36] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[37] A. Pfitzmann, B. Pfitzmann, and M. Waidner, "Isdn-mixes: Untraceable communication with very small bandwidth overhead," in *Kommunikation in verteilten Systemen*. Springer, 1991, pp. 451–463.

[38] *Torfone*, Tor Project, April 2013, http://torfone.org.

[39] *Mumble*, Lightspeed gaming LLC, March 2009, https://www.mumble.com/.

[40] R. Jansen, *Onionperf : A utility to track Tor and onion service performance.*, The Tor Project, May 2015, https://onionperf.torproject.org/onionperf.html.

[41] *Simple Bandwidth Scanner.*, The Tor Project, March 2018, https://github.com/torproject/sbws.

[42] R. Jansen and N. Hopper, "Shadow: Running tor in a box for accurate and efficient experimentation," *in Proceedings of Network and Distributed Systems Security (NDSS) 2012*.

[43] R. Dingledine and S. J. Murdoch, "Performance improvements on tor or, why tor is slow and what we're going to do about it," *Online: http://www. torproject. org/press/presskit/2009-03-11-performance*, 2009.

[44] R. Snader and N. Borisov, "Improving security and performance in the tor network through tunable path selection," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 5, pp. 728–741, 2010.

[45] A. Mani, T. Wilson-Brown, R. Jansen, A. Johnson, and M. Sherr, "Understanding tor usage with privacy-preserving measurement," in *Proceedings of the Internet Measurement Conference 2018*, 2018, pp. 175–187.

[46] R. Jansen, T. Vaidya, and M. Sherr, "Point break: a study of bandwidth denial-of-service attacks against tor," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1823–1840.

[47] M. Perry, "Torflow: Tor network analysis," *Proc. 2nd HotPETs*, pp. 1–14, 2009.

[48] R. Snader and N. Borisov, "Eigenspeed: secure peer-to-peer bandwidth evaluation," in *Proceedings of the 8th international conference on Peer-to-peer systems*. USENIX Association, 2009.

[49] A. Johnson, R. Jansen, N. Hopper, A. Segal, and P. Syverson, "Peerflow: Secure load balancing in tor," *in Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 2, pp. 74–94, 2017.

[50] *Chutney.*, The Tor Project, February 2011, https://github.com/torproject/chutney.

[51] F. Cangialosi, D. Levin, and N. Spring, "Ting: Measuring and exploiting latencies between all tor nodes," in *Proceedings of the 2015 Internet Measurement Conference*. ACM, 2015, pp. 289–302.

[52] *The Tor Metrics Project*, The Tor Project, January 2009, https://metrics.torproject.org/.

[53] *Telegram*, Telegram Messenger LLP, August 2013, https://telegram.com/.

[54] *Skype*, Microsoft, August 2003, https://skype.com/.

[55] *Stem*, Tor Project, March 2013, https://stem.torproject.org/.

[56] *OpenVPN*, OpenVPN INC., November 2006, https://www.openvpn.net/.

[57] *pjsua*, PJSIP, https://www.pjsip.org/pjsua.htm.

[58] *Freeswitch*, SignalWire, January 2006, https://freeswitch.com/.

[59] *libtgvoip*, Telegram, February 2017, https://github.com/grishka/libtgvoip.

[60] *pyrogram*, Telegram, January 2018, https://github.com/pyrogram/pyrogram.

[61] *Mplayer*, The Mplayer Project, January 2000, http://www.mplayerhq.hu/design7/news.html.

[62] *pactl*, Pulseaudio, June 2011, https://linux.die.net/man/1/pactl.

[63] I. Rec, "P. 862.3: Application guide for objective quality measurement based on recommendations p. 862, p. 862.1 and p. 862.2," *International Telecommunication Union, Geneva*, 2005.

[64] ITUT, "P. 830: Subjective performance assessment of digital telephone-band and wideband digital codecs," *International Telecommunication Union, Geneva (Switzerland)*, 1996.

[65] P. Winter, R. Ensafi, K. Loesing, and N. Feamster, "Identifying and characterizing sybils in the tor network," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 1169–1185.

[66] *Tor Bridges*, The Tor Project, https://2019.www.torproject.org/docs/bridges.html.en.

[67] "Tor bridges - bridgedb," https://bridges.torproject.org/.

[68] I. S. H. C. Ilias and M. S. Ibrahim, "Performance analysis of audio video codecs over wi-fi/wimax network," in *Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication*, 2014, pp. 1–5.

[69] S. Katsigiannis, J. Scovell, N. Ramzan, L. Janowski, P. Corriveau, M. A. Saad, and G. Van Wallendael, "Interpreting mos scores, when can users see a difference? understanding user experience differences for photo quality," *Quality and User Experience*, vol. 3, no. 1, p. 6, 2018.

[70] K. Apostol, "Internet censorship in the arab spring," 2012.

[71] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[72] R. Ensafi, P. Winter, A. Mueen, and J. R. Crandall, "Analyzing the great firewall of china over space and time," *PoPETs*, vol. 2015, pp. 61–76, 2015.

[73] "Tor: Bridges," https://www.torproject.org/docs/bridges.html.en.

[74] H. Mohajeri Moghaddam, B. Li, M. Derakhshani, and I. Goldberg, "Skypemorph: Protocol obfuscation for tor bridges," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 97–108.

[75] Z. Weinberg, J. Wang, V. Yegneswaran, L. Briesemeister, S. Cheung, F. Wang, and D. Boneh, "Stegotorus: a camouflage proxy for the tor anonymity system," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 109–120.

[76] A. Houmansadr, C. Brubaker, and V. Shmatikov, "The parrot is dead: Observing unobservable network communications," in *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 2013, pp. 65–79.

[77] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman, "Telex: Anticensorship in the network infrastructure," in *Proceedings of the 20th USENIX Security Symposium*, August 2011.

[78] A. Houmansadr, G. T. K. Nguyen, M. Caesar, and N. Borisov, "Cirripede: Circumvention infrastructure using router redirection with plausible deniability," in *Proceedings of the 18th ACM conference on Computer and Communications Security (CCS 2011)*, October 2011.

[79] A. Houmansadr, E. L. Wong, and V. Shmatikov, "No direction home: The true cost of routing around decoys." in *NDSS*, 2014.

[80] D. Gosain, A. Agarwal, S. Chakravarty, and H. Acharya, "The devil's in the details: Placing decoy routers in the internet," in *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM, 2017, pp. 577–589.

[81] E. Wustrow, C. M. Swanson, and J. A. Halderman, "Tapdance: End-to-middle anticensorship without flow blocking," in *Proceedings of 23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, August 2014.

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

[83] C. Bocovich and I. Goldberg, "Slitheen: Perfectly Imitated Decoy Routing Through Traffic Replacement," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16, 2016, pp. 1702–1714.

[84] D. Ellard, C. Jones, V. Manfredi, W. T. Strayer, B. Thapa, M. Van Welie, and A. Jackson, "Rebound: Decoy routing on asymmetric routes via error messages," in *Local Computer Networks (LCN), 2015 IEEE 40th Conference on*. IEEE, 2015, pp. 91–99.

[85] S. Frolov, F. Douglas, W. Scott, A. McDonald, B. VanderSloot, R. Hynes, A. Kruger, M. Kallitsis, D. G. Robinson, S. Schultze *et al.*, "An isp-scale deployment of tapdance," in *7th USENIX Workshop on Free and Open Communications on the Internet (FOCI)*. USENIX Association, 2017.

[86] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[87] M. Schuchard, J. Geddes, C. Thompson, and N. Hopper, "Routing around decoys," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 85–96.

[88] C. Bocovich and I. Goldberg, "Secure asymmetry and deployability for decoy routing systems," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 43–62, 2018.

[89] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, 2009, pp. 31–42.

[90] "The transport layer security (tls) protocol version 1.2," https://tools.ietf.org/html/rfc5246.

[91] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and S. C. Diot, "Packet-level traffic measurements from the sprint ip backbone," *IEEE network*, vol. 17, no. 6, pp. 6–16, 2003.

[92] A. Houmansadr, W. Zhou, M. Caesar, and N. Borisov, "Sweet: Serving the web by exploiting email tunnels," *arXiv preprint arXiv:1211.3191*, 2012.

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

[94] A. Houmansadr, W. Zhou, M. Caesar, and N. Borisov, "Sweet: Serving the web by exploiting email tunnels," *IEEE/ACM Transactions on Networking (TON)*, vol. 25, no. 3, pp. 1517–1527, 2017.

[95] S. Li and N. Hopper, "Mailet: Instant social networking under censorship," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 2, pp. 175–192, 2016.

[96] R. McPherson, A. Houmansadr, and V. Shmatikov, "Covertcast: Using live streaming to evade internet censorship," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 3, pp. 212–225, 2016.

[97] S. Burnett, N. Feamster, and S. Vempala, "Chipping away at censorship firewalls with user-generated content." in *USENIX Security Symposium*. Washington, DC, 2010, pp. 463–468.

[98] "Hp3500yl openflow enabled switch data sheet," http://www.curvesales.com/datasheets/switches/Campus-Access/HP-3500-3500-YL-Switch-Series-Datasheet.pdf.

[99] "List of hp sdn switches," https://techlibrary.hpe.com/ie/en/networking/solutions/technology/sdn/portfolio.aspx#.XjhyRtlS_CI.

[100] "Hp10500 series openflow enabled switches data sheet," http://www.hp.com/hpinfo/newsroom/press_kits/2011/InteropNY2011/HP_10500_Data-Sheet.pdf.

[101] "Deterlab: Cyber-Defense Technology Experimental Research laboratory," https://www.isi.deterlab.net/index.php.

[102] "Selenium webdriver and ide," https://www.seleniumhq.org/.

[103] "Smtp library for python," https://docs.python.org/2/library/smtplib.html.

[104] "Imap library for python," https://docs.python.org/2/library/imaplib.html.

[105] "Sigebreaker's source code," https://github.com/Piyush825/SiegeBreaker.

[106] A. Zarek, Y. Ganjali, and D. Lie, "Openflow timeouts demystified," *Univ. of Toronto, Toronto, Ontario, Canada*, 2012.

[107] S. Frolov and E. Wustrow, "The use of tls in censorship circumvention." in *NDSS*, 2019.

[108] M. J. Freedman, M. Vutukuru, N. Feamster, and H. Balakrishnan, "Geographic locality of ip prefixes," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. USENIX Association, 2005, pp. 13–13.

[109] M. Schuchard, "Adversarial degradation of the availability of routing infrastructures and other internet-scale distributed systems," http://hdl.handle.net/11299/182196.

[110] P. M. Mohan, T. Truong-Huu, and M. Gurusamy, "Tcam-aware local rerouting for fast and efficient failure recovery in software defined networks," in *Global Communications Conference (GLOBECOM), 2015 IEEE*. IEEE, 2015, pp. 1–6.

[111] M. Rifai, N. Huin, C. Caillouet, F. Giroire, J. Moulierac, D. L. Pacheco, and G. Urvoy-Keller, "Minnie: An sdn world with few compressed forwarding rules," *Computer Networks*, vol. 121, pp. 185–207, 2017.

[112] K. Kannan and S. Banerjee, "Compact tcam: Flow entry compaction in tcam for power aware sdn," in *International conference on distributed computing and networking*. Springer, 2013, pp. 439–444.

[113] C.-C. Chuang, Y.-J. Yu, A.-C. Pang, and G.-Y. Chen, "Minimization of tcam usage for sdn scalability in wireless data centers," in *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE, 2016, pp. 1–7.

[114] M. Obadia, M. Bouet, J.-L. Rougier, and L. Iannone, "A greedy approach for minimizing sdn control overhead," in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*. IEEE, 2015, pp. 1–5.

[115] X. Wen, B. Yang, Y. Chen, L. E. Li, K. Bu, P. Zheng, Y. Yang, and C. Hu, "Ruletris: Minimizing rule update latency for tcam-based sdn switches," in *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*. IEEE, 2016, pp. 179–188.

[116] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford, "Captcha: Using hard ai problems for security," in *Advances in Cryptology — EUROCRYPT 2003*, E. Biham, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 294–311.

[117] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and implementation of a routing control platform," in *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05, 2005, pp. 15–28.

[118] "Intra-domain routing convergence with centralized control," *Computer Networks*, vol. 53, no. 18, pp. 2985–2996, 2009.

[119] V. Kotronis, X. Dimitropoulos, and B. Ager, "Outsourcing the routing control logic: Better internet routing based on sdn principles," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XI. New York, NY, USA: ACM, 2012, pp. 55–60. [Online]. Available: http://doi.acm.org/10.1145/2390231.2390241

[120] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, and L. Vanbever, "An industrial-scale software defined internet exchange point." in *NSDI*, vol. 16, 2016, pp. 1–14.

[121] K. Poularakis, G. Iosifidis, G. Smaragdakis, and L. Tassiulas, "One step at a time: Optimizing sdn upgrades in isp networks," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[122] P. Song, Y. Liu, T. Liu, and D. Qian, "Controller-proxy: Scaling network management for large-scale sdn networks," *Computer Communications*, vol. 108, pp. 52–63, 2017.

[123] C. Wang and S. Yan, "Scaling sdn network with self-adjusting architecture," in *2016 IEEE International Conference on Electronic Information and Communication Technology (ICEICT)*. IEEE, 2016, pp. 116–120.

[124] R. Shah, M. Vutukuru, and P. Kulkarni, "Cuttlefish: Hierarchical sdn controllers with adaptive offload," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE, 2018, pp. 198–208.

[125] S. Woo, J. Sherry, S. Han, S. Moon, S. Ratnasamy, and S. Shenker, "Elastic scaling of stateful network functions," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018, pp. 299–312.

[126] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," *ACM SIGCOMM computer communication review*, vol. 43, no. 4, pp. 7–12, 2013.

[127] S. Khattak, T. Elahi, L. Simon, C. M. Swanson, S. J. Murdoch, and I. Goldberg, "Sok: Making sense of censorship resistance systems," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 4, pp. 37–61, 2016.

[128] M. C. Tschantz, S. Afroz, V. Paxson *et al.*, "Sok: Towards grounding censorship circumvention in empiricism," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 914–933.

[129] G. Aceto and A. Pescapé, "Internet censorship detection: A survey," *Computer Networks*, vol. 83, pp. 381–421, 2015.

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

[131] D. Fifield, C. Lan, R. Hynes, P. Wegmann, and V. Paxson, "Blocking-resistant communication through domain fronting," *Proceedings on Privacy Enhancing Technologies*, vol. 2015, no. 2, pp. 46–64, 2015.

[132] M. Nasr and A. Houmansadr, "Game of decoys: Optimal decoy routing through game theory," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16, 2016, pp. 1727–1738.

[133] S. Frolov, J. Wampler, S. C. Tan, J. A. Halderman, N. Borisov, and E. Wustrow, "Conjure: Summoning proxies from unused address space," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2215–2229.

[134] M. Nasr, H. Zolfaghar, A. Houmansadr, and A. Ghafari, "Massbrowser: Unblocking the censored web for the masses, by the masses."

[135] A. Houmansadr, T. J. Riedl, N. Borisov, and A. C. Singer, "I want my voice to be heard: Ip over voice-over-ip for unobservable censorship circumvention." in *NDSS*, 2013.

[136] D. Barradas, N. Santos, and L. Rodrigues, "Deltashaper: Enabling unobservable censorship-resistant tcp tunneling over videoconferencing streams," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 4, pp. 5–22, 2017.

[137] "Detailed latency analysis of whatsapp," https://developers.facebook.com/docs/whatsapp/tips-and-tricks/send-message-performance/#performance.

[138] V. C. Sheer and R. E. Rice, "Mobile instant messaging use and social capital: Direct and indirect associations with employee outcomes," *Information & Management*, vol. 54, no. 1, pp. 90–102, 2017.

[139] A. Andujar, "Benefits of mobile instant messaging to develop esl writing," *System*, vol. 62, pp. 63–76, 2016.

[140] L. Piwek and A. Joinson, ""what do they snapchat about?" patterns of use in time-limited instant messaging service," *Computers in Human Behavior*, vol. 54, pp. 358–367, 2016.

[141] S. So, "Mobile instant messaging support for teaching and learning in higher education," *The Internet and Higher Education*, vol. 31, pp. 32–42, 2016.

[142] Y. Tang and K. F. Hew, "Is mobile instant messaging (mim) useful in education? examining its technological, pedagogical, and social affordances," *Educational Research Review*, vol. 21, pp. 85–104, 2017.

[143] Slack vs email. [Online]. Available: https://slack.com/intl/en-in/why/slack-vs-email

[144] How slack is replacing email in the workplace. [Online]. Available: https://tech.co/news/slack-replacing-email-workplace-2018-08

[145] Slack is killing email. [Online]. Available: https://www.theverge.com/2014/8/12/5991005/slack-is-killing-email-yes-really

[146] R. Dingledine and N. Mathewson, "Design of a blocking-resistant anonymity system," 2006.

[147] S. Burnett, N. Feamster, and S. Vempala, "Chipping away at censorship firewalls with user-generated content." in *USENIX Security Symposium*. Washington, DC, 2010, pp. 463–468.

[148] H. Zolfaghari and A. Houmansadr, "Practical censorship evasion leveraging content delivery networks," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1715–1726.

[149] E. Wustrow, C. M. Swanson, and J. A. Halderman, "Tapdance: End-to-middle anticensorship without flow blocking," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 159–174.

[150] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman, "Telex: Anticensorship in the network infrastructure." in *20th {USENIX} Security Symposium ({USENIX} Security 11)*, 2011.

[151] C. Bocovich and I. Goldberg, "Slitheen: Perfectly Imitated Decoy Routing Through Traffic Replacement," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16, 2016, pp. 1702–1714.

[152] P. K. Sharma, D. Gosain, H. Sagar, C. Kumar, A. Dogra, V. Naik, H. Acharya, and S. Chakravarty, "Siegebreaker: An sdn based practical decoy routing system," *Proceedings on Privacy Enhancing Technologies*, vol. 3, pp. 243–263, 2020.

[153] Q. Wang, X. Gong, G. T. Nguyen, A. Houmansadr, and N. Borisov, "Censorspoofer: asymmetric communication using ip spoofing for censorship-resistant web browsing," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 121–132.

[154] C. Brubaker, A. Houmansadr, and V. Shmatikov, "Cloudtransport: Using cloud storage for censorship-resistant networking," in *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2014, pp. 1–20.

[155] T. K. Yadav, A. Sinha, D. Gosain, P. K. Sharma, and S. Chakravarty, "Where the light gets in: Analyzing web censorship mechanisms in india," in *Proceedings of the Internet Measurement Conference 2018*, 2018.

[156] A. Chaabane, T. Chen, M. Cunche, E. De Cristofaro, A. Friedman, and M. A. Kaafar, "Censorship in the wild: Analyzing internet filtering in syria," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, 2014, pp. 285–298.

[157] R. Soltani, A. Houmansadr, D. Goeckel, and D. Towsley, "Practical traffic analysis attacks on secure messaging applications," in *Proceedings of Network and Distributed Systems Security (NDSS) 2020*. Alireza Bahramali, 2020.

[158] "Python sockets – low-level networking interface," https://docs.python.org/3/library/socket.html.

[159] "Signal-cli," https://github.com/AsamK/signal-cli.

[160] "Python dbus library," https://pydbus.readthedocs.io/en/latest/legacydocs.

[161] "Python gzip library," https://docs.python.org/3/library/gzip.html.

[162] "Python crypto library," https://pycryptodome.readthedocs.io/en/latest/.

[163] "Whatsapp usage statistics," https://www.businesstoday.in/technology/news/whatsapp-users-share-texts--photos-videos-daily/story/257230.html.

[164] J. Baumgartner, S. Zannettou, M. Squire, and J. Blackburn, "The pushshift telegram dataset," in *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 14, 2020, pp. 840–847.

[165] S. Li, M. Schliep, and N. Hopper, "Facet: Streaming over videoconferencing for censorship circumvention," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, 2014, pp. 163–172.

[166] D. Barradas, N. Santos, L. Rodrigues, and V. 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*, 2020, pp. 35–48.

[167] A. Filasto and J. Appelbaum, "Ooni: Open observatory of network interference." in *FOCI*, 2012.

[168] R. Ramesh, R. S. Raman, M. Bernhard, V. Ongkowijaya, L. Evdokimov, A. Edmundson, S. Sprecher, M. Ikram, and R. Ensafi, "Decentralized control: A case study of russia," in *Network and Distributed Systems Security (NDSS) Symposium 2020*, 2020.

[169] J. Beznazwy and A. Houmansadr, "How china detects and blocks shadowsocks," in *Proceedings of the ACM Internet Measurement Conference*, 2020, pp. 111–124.

[170] A. A. Niaki, N. P. Hoang, P. Gill, A. Houmansadr *et al.*, "Triplet censors: Demystifying great firewall's {DNS} censorship behavior," in *10th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 20)*, 2020.

[171] K. Bock, G. Hughey, L.-H. Merino, T. Arya, D. Liscinsky, R. Pogosian, and D. Levin, "Come as you are: Helping unmodified clients bypass censorship with server-side evasion," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 586–598.

[172] S. Frolov and E. Wustrow, "{HTTPT}: A probe-resistant proxy," in *10th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 20)*, 2020.

[173] "Myanmar citizens not aware of covid-19 and human rights impact," https://tinyurl.com/yxuzab5q.

[174] "100 hours in the dark: How an election internet blackout hit poor ugandans, reuters, january 2021," https://tinyurl.com/y49d7shm.

[175] "Myanmar shuts down internet but allows cellular services to function, telenor, may 2020," https://tinyurl.com/up2ytfo.

[176] "Russian authorities 'secretly' shut down moscow's mobile internet: Report, forbes, august 2019," https://tinyurl.com/47pnarm2.

[177] "Internet services restricted in 13 districts of haryana, indai, ndtv, november 2017," https://tinyurl.com/y5646kz9.

[178] "Internet shutdown in response to mega public gathering in india, business world, october 2020," https://tinyurl.com/yxa9e32u.

[179] G. Trewitt, "Triggered remote dial-up for internet access," Nov. 21 2000, uS Patent 6,151,629.

[180] *Twilio*, Twilio Inc., May 2008, https://twilio.com.

[181] C. K. LaDue, V. V. Sapozhnykov, and K. S. Fienberg, "A data modem for gsm voice channel," *IEEE Transactions on Vehicular Technology*, vol. 57, no. 4, pp. 2205–2218, 2008.

[182] M. Perić, P. Milićević, Z. Banjac, and B. M. Todorović, "An experiment with real-time data transmission over global scale mobile voice channel," in *2015 12th International Conference on Telecommunication in Modern Satellite, Cable and Broadcasting Services (TELSIKS)*. IEEE, 2015, pp. 239–242.

[183] B. T. Ali, G. Baudoin, and O. Venard, "Data transmission over mobile voice channel based on m-fsk modulation," in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2013, pp. 4416–4421.

[184] M. A. Özkan and S. B. Örs, "Data transmission via gsm voice channel for end to end security," in *2015 IEEE 5th International Conference on Consumer Electronics-Berlin (ICCE-Berlin)*. IEEE, 2015, pp. 378–382.

[185] R. Kazemi, M. Boloursaz, S. M. Etemadi, and F. Behnia, "Capacity bounds and detection schemes for data over voice," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 11, pp. 8964–8977, 2016.

[186] A. Dhananjay, A. Sharma, M. Paik, J. Chen, T. K. Kuppusamy, J. Li, and L. Subramanian, "Hermes: data transmission over unknown voice channels," in *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, 2010, pp. 113–124.

[187] T. Ahmad, E. Reed-Sanchez, F. Zarinni, A. Afutu, K. Adjaho, Y. Nyarko, and L. Subramanian, "Greenapps: A platform for cellular edge applications," in *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*, 2018, pp. 1–5.

[188] F. R. Dogar, I. A. Qazi, A. R. Tariq, G. Murtaza, A. Ahmad, and N. Stocking, "Missit: Using missed calls for free, extremely low bit-rate communication in developing regions," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–12.

[189] J. Romkey, "Rfc1055: Nonstandard for transmission of ip datagrams over serial lines: Slip," 1988.

[190] S. Cheshire and M. Baker, "Consistent overhead byte stuffing," *IEEE/ACM Transactions on networking*, vol. 7, no. 2, pp. 159–172, 1999.

[191] D. Klinc, C. Hazay, A. Jagmohan, H. Krawczyk, and T. Rabin, "On compression of data encrypted with block ciphers," *IEEE transactions on information theory*, vol. 58, no. 11, pp. 6989–7001, 2012.

[192] J. S. Sobolewski, “Cyclic redundancy check,” in *Encyclopedia of Computer Science*, 2003, pp. 476–479.

[193] “Cyclic redundancy check,” https://en.wikipedia.org/wiki/Cyclic_redundancy_check.

[194] “Ofono telephony management framework,” https://01.org/ofono.

[195] M. Masahisa, “Frequency-shift-keying phase-modulation code transmission system,” May 21 1968, uS Patent 3,384,822.

[196] K. Mostafa, “minimodem - general-purpose software audio fsk modem,” http://www.whence.com/minimodem/.

[197] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, *Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 119–132. [Online]. Available: https://doi.org/10.1007/978-3-540-28632-5_9

[198] O. Lev, “Cross-platform python cffi bindings for libsecp256k1,” https://pypi.org/project/coincurve/.

[199] “Python wrapper for the twitter api,” https://pypi.org/project/twython/.

[200] “A simple http rest api for searching and retrieving live articles from all over the web,” https://newsapi.org/.

[201] “Fast api web framework,” https://fastapi.tiangolo.com/.

[202] E. S. Ristad and P. N. Yianilos, “Learning string-edit distance,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 5, pp. 522–532, 1998.

[203] A. S. Khader and D. Lai, “Preventing man-in-the-middle attack in diffie-hellman key exchange protocol,” in *2015 22nd international conference on telecommunications (ICT)*. IEEE, 2015, pp. 204–208.

[204] “Bulk sms country wise restrictions,” https://www.sendmode.com/bulk-sms-country-restrictions-infographic.

[205] "Trai extends the 100 sms per day per sim limit to 200 sms per day per sim." https://www.trai.gov.in/sites/default/files/press_release_for_8th_amendmenet.pdf.

[206] "Briar secure messaging app," https://briarproject.org/.

[207] "Jami secure messaging and voip app," https://jami.net/.

[208] "Open-source real-time chat, built on firebase," https://firechat.firebaseapp.com/.

[209] *Torocks*, Tor dev team, December 2000, https://linux.die.net/man/8/torsocks.

[210] *socat*, Gerhard Rieger, April 2009, http://www.dest-unreach.org/socat/.

[211] "Ryu- Component Based Software Defined Networking Framework," https://osrg.github.io/ryu/.

[212] "Telethon telegram api," https://docs.telethon.dev/en/latest/.

[213] "Slack client api," https://python-slackclient.readthedocs.io/en/latest/.

[214] "Slack real-time messaging api," https://python-slackclient.readthedocs.io/en/latest/real_time_messaging.html.

[215] "Skype messaging api," https://skpy.t.allofti.me/.

[216] "Web whatsapp automation api," https://github.com/open-wa/wa-automate-python.

[217] "xdotool: Linux gui automation utility," http://manpages.ubuntu.com/manpages/trusty/man1/xdotool.1.html.

Appendices

Appendix A

The Road Not Taken

A.1 M-Tor Results

In this section, we describe the implementation details along with the experiments performed using the M-Tor setup. The client machine configuration along with Tor's setup was similar to the V-Tor setup (described in detail in Subsec. 5.4.1). The detail of the other new entities introduced in the M-Tor setup is described below:

Communication Peers: For M-Tor experiments, the end-point hosts were installed with Mumble client program, configured to work in TCP mode. This enables directly sending voice traffic over TCP streams. Further, like OpenVPN, the Mumble client also allows transporting the voice traffic through Tor, by specifying the SOCKS port in its configuration. `Torsocks` [209] utility was used to transport `iperf`'s traffic over Tor. Since `torsocks` does not allow programs with root privileges, we relied on `socat` [210] tunnels for transporting `pings`.

Mumble Server: The M-Tor setup had the Mumble server (Murmur) v1.2.19 (analogous to VPN/SIP server in V-Tor setup) installed for handling voice calls. A call channel was opened for every new call. The caller and callee utilities were configured to join this call channel so that whenever a caller initiated a call, it would reach the channel and the callee could record it for quality evaluation.

Now, we describe the experiments involving the controlled setups as well the ones performed over the public Tor network. *All these experiments followed the setups similar to those of V-Tor.*

A.1.1 Controlled Experiments

Similar to the V-Tor experiments, we performed two different sets of tests using M-Tor in the lab environment. These involved: (1) Direct Mumble calls without Tor (2) Mumble calls through Tor. We recorded an average PESQ (measured across 100 individual calls) of 4.5 for both direct calls and calls through Tor. Here also we observed a slightly higher bandwidth requirement ($70 - 80$ Kbps) for calls that were transported via Tor, compared to those that were not ($50 - 60$ Kbps). In general, the bandwidth requirement for M-Tor was much lower in comparison to V-Tor as the underlying codec used by Mumble encodes at a lower rate.

Further, similar experiments were performed where we increased the number of parallel connections gradually, to see its impact on call quality. Here also we observed a trend identical to V-Tor experiments.

A.1.2 Experiments over Public Tor

We considered the same three scenarios to perform experiments using M-Tor *i.e.,* (1) Caller anonymity: Co-located voice server and callee *viz,* Scenario I (ref. Subsection. 3.4.2.1) (2) Caller anonymity: separate VPN/voice server *viz,* Scenario II (ref. Subsection. 3.4.2.2) (3) Caller and Callee (two-way) anonymity *viz,* Scenario III (ref. Subsection. 3.4.2.3). The CDF of PESQ scores obtained in Scenario I and Scenario II are depicted in Fig. A.1, and Fig. A.2 respectively. It is evident from the figure that majority of the calls obtained PESQ values greater than 3. Similarly, the average PESQ score obtained in Scenario III, was 3.8 with 85% calls above a PESQ of 3.

The OWD results for both Scenario I and II (for three hop as well as two hop circuits) are shown in Fig. A.3, with the results of Scenario III in Fig. A.4. As evident from the results, moving to two-hop circuits in Scenario III (two-way anonymity) helped us improve the OWD

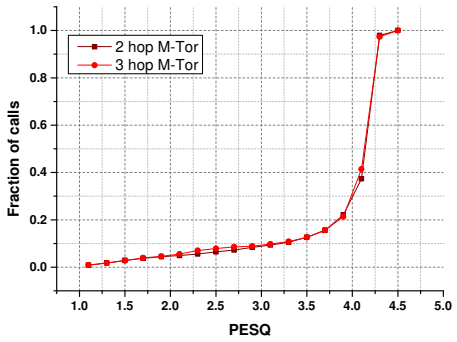

Figure A.1: M-Tor: CDF of PESQ for Caller Anonymity when server is co-located with callee (Scenario I).

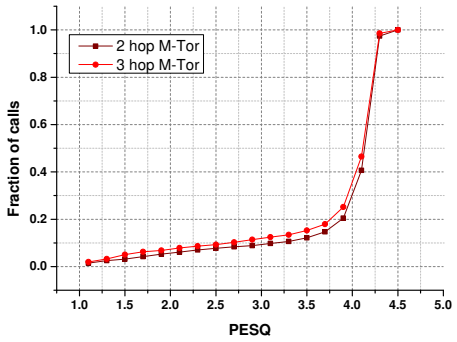

Figure A.2: M-Tor: CDF of PESQ for Caller Anonymity when server is separately hosted (Scenario II).

significantly with more than 90% calls below 400ms in comparison to 70%.

Overall, similar to V-Tor, M-Tor proved to be capable of performing good quality calls for the majority of the cases.

A.2 Miscellaneous Issues

VoIP applications with high bandwidth requirement: Our measurements involved testing VoIP applications that encoded at low bit-rates ($< 120Kbps$). However, these applications can be configured to encode at higher rates (≈ 800 Kbps). We evaluated the performance at these higher encoding rates (200 Kbps, 400 Kbps and 800 Kbps). We ran 1000 calls when the clients were configured to encode at these rates.

As expected, an increase in the rates progressively deteriorated the performance, and thus the

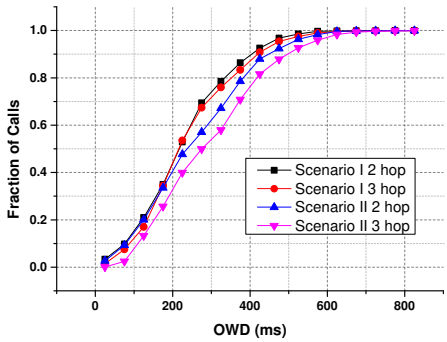

Figure A.3: M-Tor: CDF of OWD variation for Caller anonymity in both Scenario I and II.

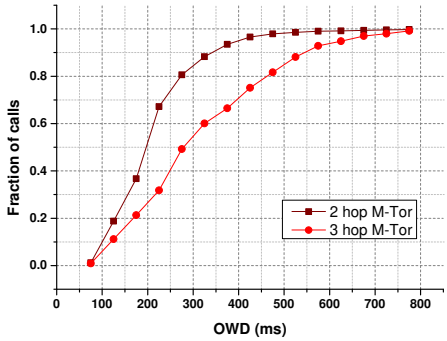

Figure A.4: CDF of delay for M-Tor setup when two-way anonymity was achieved (Scenario III).

measured PESQ. Average PESQ scores were 3.6, 3.2 and 3.0 for 200, 400, and 800 Kbps rates, respectively. However, even at 800 Kbps, we measured PESQ of above 3 for 65% of the cases. Thus even at higher encoding rates, one can expect reasonable call quality.

Coverage of Tor relays: We recorded and analyzed Tor circuit information for all our experiments. We now present some interesting insights we observed from this analysis. A total of about 600,000 Tor circuits were created during our study.[1] These circuits involved a total of 6650 unique Tor relays. Prior research (Rizal *et al.* [6]) reportedly used only about 298 relays that too restricted to Europe.

Threshold of OWD: We analyzed the distribution of OWD values for our study to obtain deeper insights about the overall performance. Fig. A.5 shows the percentage of calls which have OWD less than 50 ms, 100 ms so on till 400 ms. We notice that only about 10% of calls had a one-way

[1]The number of Tor circuits are slightly higher than the total number of experiments as the two way anonymity experiments involve creating two Tor circuits for a single call.

delay above 300 ms and below 400 ms, indicating only a small fraction of calls in that category. On the contrary, for about 81% of calls, the OWD was below 300 ms. This suggests that in the majority of the cases, the user would obtain satisfactory call quality with OWD less than 300 ms. Moreover, in about 28% of calls OWD was below 150 ms, which is regarded as an ideal quality call according to ITU. Thus, though we considered the ITU recommendations of OWD less than 400 ms to judge call quality, for the majority of our calls, we observed OWD below 300 ms.

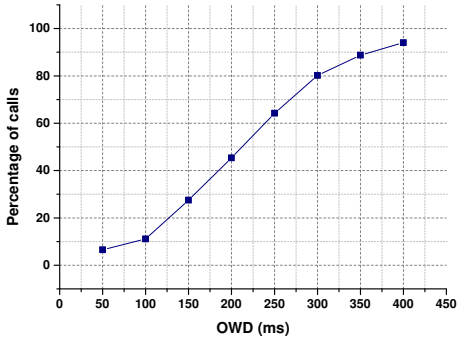

Figure A.5: Percentage of calls recorded at different OWD values.

Ethical Considerations: As described in the paper, we generated all our network traffic (*i.e.*, the voice calls) using machines in our control. We did not capture or use any third party's data/network traffic. Moreover, as our measurements were spread across a span of 12 months, and involved generating low bit-rate voice traffic ($\approx$ 120 Kbps), along with a short duration (<10 s) single `iperf` probes, we expect it to have had negligible to no impact on any un-involved Internet users' network performance.

Our user study involved human subjects in different geographic locations who heard audio samples and spoke to one another, via our setups. To the best of our knowledge the audio sample bore no information that may cause emotional or psychological trauma to the subjects involved. The quality and the contents of the clips were duly attested by the institutional research review committee that involved subjects who were not party to the research in any capacity. Further, we did not record and (or) decode, either manually or electronically, the speech between subjects, thereby preserving their communication privacy.

Appendix B

SiegeBreaker

B.1 Tests Over Emulation Environment

Experimental setup: Our experimental topology on DETER consists of ten Linux machines – one of which acted as a SDN switch (running `OpenvSwitch` v2.5.0), two nodes acted as normal routers (forwarding based on routing table lookups) and six host nodes (running Ubuntu 14.04 LTS) functioning as clients and servers. One node functions as the controller, running the `Ryu` [211] SDN controller (v4.15), a popular open-source SDN controller, fully compliant with `OpenvSwitch`. Fig. B.1 schematically represents the topology.

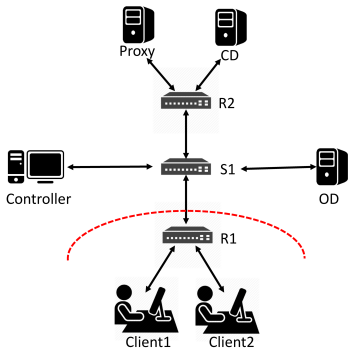

Figure B.1: The topology used for evaluating SiegeBreaker — one SDN switch (`S1`), two routers (`R1`,`R2`), an SDN controller and 6 host nodes all configured on real machines.

Experiments and results: In order to gauge the correctness of our prototype implementation, we tested SiegeBreaker under a variety of working conditions.

We tested SiegeBreaker by downloading relatively small-sized files (< 10MB). This emulates a user's everyday browsing activity. Next, we also tested it against large file sizes, of the order of 1GB, emulating bulk data transfer. Fig. B.2 shows the consolidated result of downloading files (varying from 1MB–1GB) using SiegeBreaker and `wget`. Evident from the figure, the download times of both SiegeBreaker and `wget` are comparable. These results indicate that SiegeBreaker works well for both web browsing and bulk downloads.

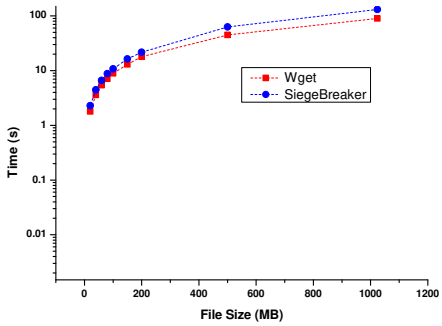

Figure B.2: Comparison of SiegeBreaker and `wget` in terms of download time for file sizes up to 1GB (in log scale).

To compare SiegeBreaker's and `wget`'s behavior in the presence of varying cross traffic, we used one client (Client1 in the topology) to download a file of size 100 MB using SiegeBreaker. Simultaneously, on a shared 100 Mbps link (between R1 and S1), another client (Client2) downloaded a file directly from CD. The second client provided a variable cross-traffic load. This is achieved by varying the download request rate of this client (Using the `--rate-limit` option of `wget`) from roughly 8 Mbps to 80 Mbps.

We then repeated the experiment, replacing the SiegeBreaker client with a `wget` client. Fig. B.3 shows that the performance of SiegeBreaker is consistently comparable to `wget`, and depicts similar degradation as we increase the background cross-traffic in our tests (which go up to 80 Mbps, *i.e.* 80% of link capacity).

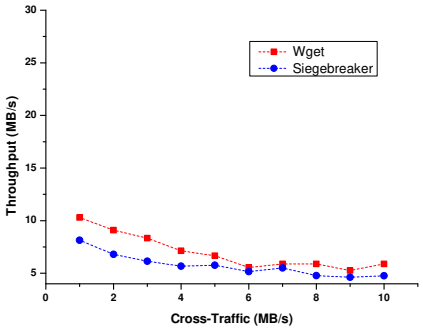

Figure B.3: Comparison of SiegeBreaker and `wget` in presence of varying cross-traffic (on a shared link).

B.2 Incorporating Salient Features of SiegeBreaker in Existing DR Systems

SiegeBreaker is a DR system which utilizes the programmability and reconfigurability of SDNs to redirect and inspect traffic on the fly. This helps in achieving two major goals :

(1) Reducing the overall load on the system by minimizing the amount of traffic to be analysed in order to detect DR flows.
(2) Protecting the privacy of non-DR flows from DR proxy (SP) maintainers.

We now enumerate the feasibility of incorporating these features in existing DR systems. Augmenting existing signalling schemes with the above goals would require SDN controlled switches acting as DRs. Additionally, they would also require some OOB channel (*e.g.,* email) to request DR service. This step ensures that only the *potential* DR flows are analysed. *E.g.,* Cirrepede and Waterfall would require to employ SDN controlled switches as DRs, the controller would require to act as the registration server, and would also have to manage the OOB signalling from the client. Thereafter, Cirripede may inspect only the SYN packets of potential DR flows. Moreover, in Waterfall, the controller may need to configure SDN switches to inspect the downstream path instead of the upstream ones.

Telex, Slitheen and Tapdance can also adopt these features. Here, the controller may inspect the respective ClientHello packets, or the incomplete HTTPS requests, to identify DR requests. Once identified, the controller can selectively divert DR flows to their Decoy Stations, which

may continue on with its normal functioning by deriving the session key and decrypting the TLS Finished messages *etc.* Slitheen would require the Decoy Station to be co-located with the controller in order to achieve its goal of minimizing the threats of latency based analysis. In Tapdance, after detection, the controller would need to forward all the client-OD traffic to the Decoy Station, and that would need to forward the packets to the OD, to preserve the connection between client and OD, as required in Tapdance.

B.3 SiegeBreaker Confirmation Attacks

In SiegeBreaker, step 5, 6 and 7 can be manipulated to confirm usage of DR. Packets exchanged before step 5 and after step 7 follow standard replay protection mechanism.

Step 5: The adversary can manipulate some bits of the payload sent in this step[1]. In this scenario, the recipient would generate a TLS `bad_record_mac` alert (in accordance with RFC 5246 [90]). However, the controller on receiving this packet has no way to verify if this packet was modified. It will anyways install DR rule and forward this packet to the SP. Unfortunately, the SP also would not be able to verify the message and would never respond with the expected TLS error. This would confirm the adversary of an attempted DR connection. To prevent this attack, we can adopt a scheme as proposed by Tapdance [81]. Here the DR client will covertly leak the client-OD session key to the controller[2]. This will allow the controller to verify if there is any change in the packet and generate appropriate responses.

Step 6: Similar to previous step, some bits of the payload of the TLS data packet (in step 6) can be modified by the adversary. As mentioned, the regular recipient would generate a TLS alert, while a SigeBreaker client would not. Here again the adversary may use this behavior to identify a DR client. To avoid such a situation, the SP includes a HMAC in this packet, generated using the DH key g^{xy}, which can only be derived by the client[3]. The client can thus verify if any modification took place and generate responses accordingly.

[1] Adversary when tampers the content of the TLS packet, also creates an appropriate TCP checksum, such that the modified packet is still accepted by the receiver's kernel.

[2] Client sends an incomplete HTTP GET request in this step, embedding the Client-OD session key.

[3] SP has obtained DR client's g^x in step 5.

Step 7: The adversary can intercept this second TLS data packet (step 7 in Fig. 4.1), and replace it with its own crafted packet. Reception of legitimate content from CD (via SP), confirms a DR connection. However, since the session keys used to encrypt this packet was derived using the DH parameter (g^x) shared via email. Thus, the adversary cannot derive the same session key, and hence will not be able to craft a legitimate packet.

B.4 SDN Setup

We now describe in detail the setup used in our experiments. The topological diagram of the setups used is shown in Fig. 4.3 and Fig. 4.7.

Controlled setup: It involved standard Linux machines as clients (shown as Client 1, 2,...,N). These machines were connected to another Linux machine (with multiple network interfaces) configured to work as a router (shown as R1). This machine was in turn connected to a `HP3500yl` hardware SDN switch. The controller (running Ryu controller application) and OD (two separate Linux machines) were attached directly to the switch. However, the VLANs of the controller and the OD were different as the controller needs to be connected to the switch via a dedicated and isolated channel. SP and CD were also Linux machines connected to `HP3500yl` via another Linux machine acting as a router (shown as R2). The websites were blocked for the client by adding an IPTABLES rule to drop packets destined to CD (on R1).

Internet setup: In this setup, the clients were hosted inside the university campus. The SDN switch was placed outside the purview of the university firewall. The controller and SP machines were directly attached to it. OD and CD were websites hosted on Internet. To capture the performance of large file downloads, we hosted servers on cloud machines, blocked them (via the firewall), and then downloaded the content using SeigeBreaker client.

Appendix C

Camoufler

C.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. C.1, Fig. C.2, Fig. C.3 and Fig. C.4 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.

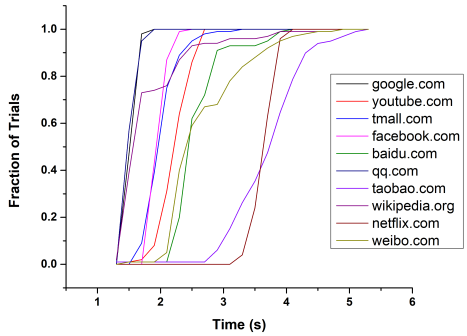

Figure C.1: CDF of Time To First Byte (TTFB) for 10 popular Alexa websites (each downloaded 100 times) for Signal app.

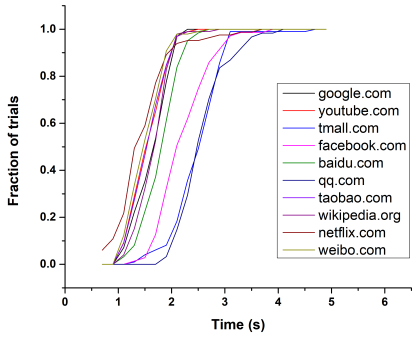

Figure C.2: CDF of Time To First Byte (TTFB) for 10 popular Alexa websites (each downloaded 100 times) for Skype app.

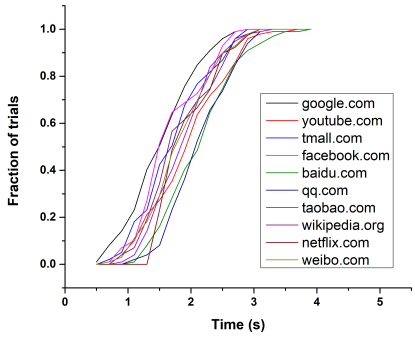

Figure C.3: CDF of Time To First Byte (TTFB) for 10 popular Alexa websites (each downloaded 100 times) for Slack.

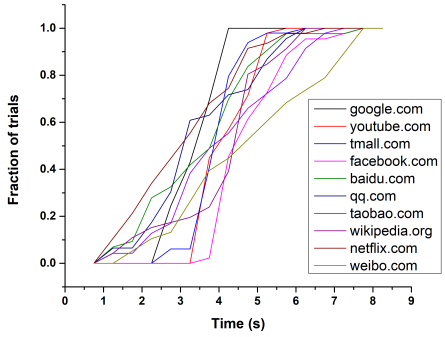

Figure C.4: CDF of Time To First Byte (TTFB) for 10 popular Alexa websites (each downloaded 100 times) for Whatsapp.

C.2 Location Diversity Additional Results

We demonstrate the results for impact of location diversity in Fig. C.5 and Fig. C.6. 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.

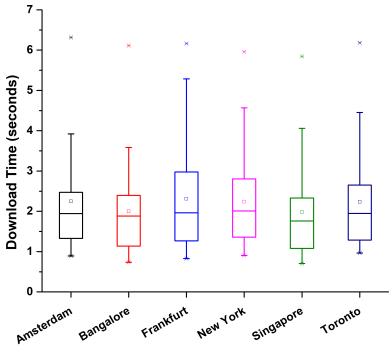

Figure C.5: Box plot depicting download time (server in Amsterdam) of top Alexa-1K websites for varying client location.

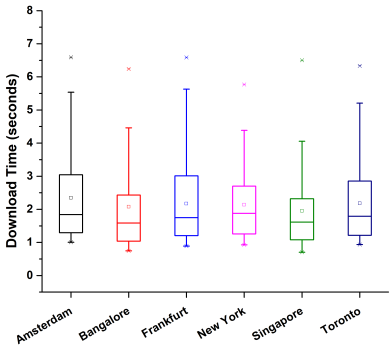

Figure C.6: Box plot depicting download time (server in San Francisco) of top Alexa-1K websites for varying client location.

C.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 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.

C.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 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. 5.4.1 we automated Telegram using its python API [212]. Additionally, we automated slack using its `slackclient` utility [213] to send messages and its Real Time Messaging (RTM) API [214] to automate processing of received messages carrying blocked content. Similarly we used the *skpy* API [215] 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 [216]. 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* [217]. `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.

C.5 On Traffic Analysis

As already described in §5.5.1, 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 §5.5.1), 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. C.7 and C.8, 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.

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 same set of multimedia

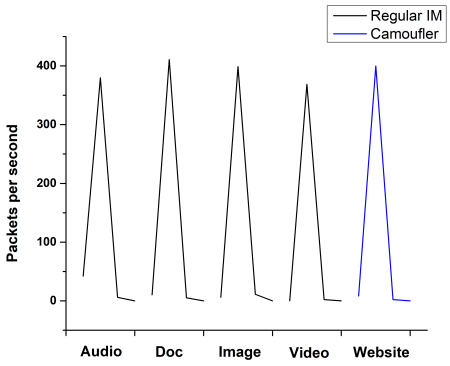

Figure C.7: Same size (300 KB) object download: Packet exchange rate for a webpage download (using Camoufler) is very similar to multimedia download using regular IM client (irrespective of the type of multimedia content).

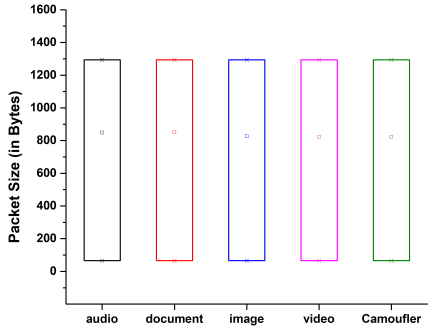

Figure C.8: Same size (300 KB) object download: Packet size distribution for a webpage download (using Camoufler) is very similar to multimedia download using regular IM.

objects using regular IM clients as well as Camoufler. It is evident from Fig. C.9, 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.

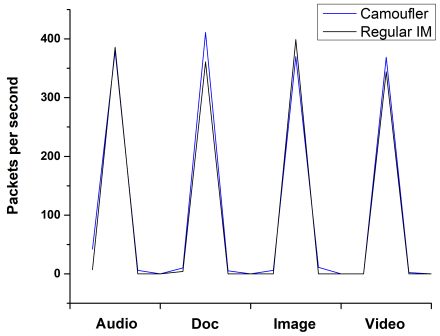

Figure C.9: Same object download using Camoufler and regular IM: Packet exchange rate for objects when downloaded using Camoufler and regular IM is almost identical.

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. C.10, large size image result in large number of packets exchanged per second. Download of image of size 1 MB resulted in more than 1000 packets per second, whereas 200 KB image download resulted in $\approx$ 300 packets per second.

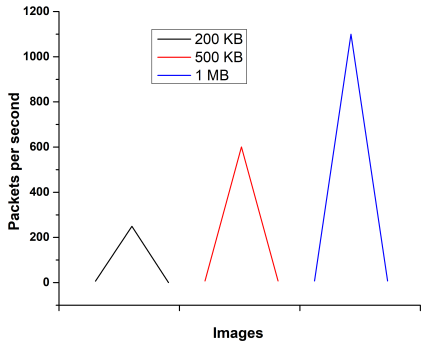

Figure C.10: Variable size images download: Packet exchange rate increases with increase in the image size.

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.

Appendix D

Dolphin

D.1 Results for Varied Cellular Connectivity

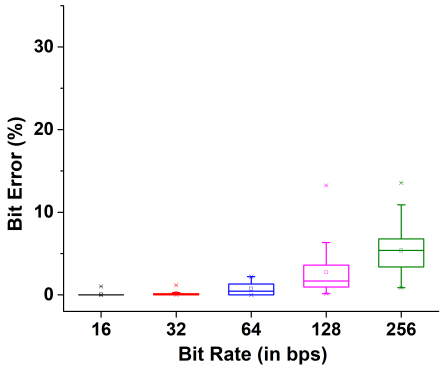

Figure D.1: 4G-4G: Bit error rate variation for different bit rate for 100B content transfer.

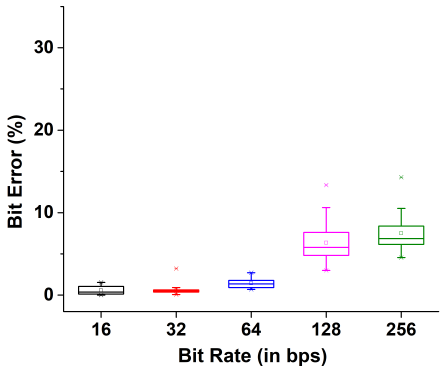

Figure D.2: 4G-4G: Bit error rate variation for different bit rate for 1KB transfer.

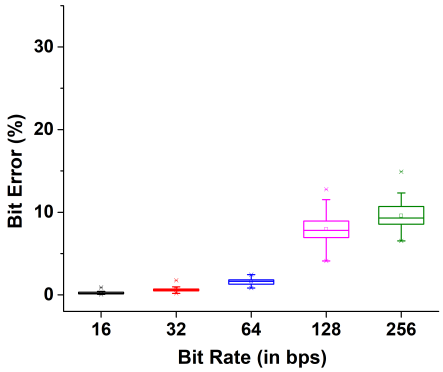

Figure D.3: 4G-4G: Bit error rate variation for different bit rate for 5KB transfer.

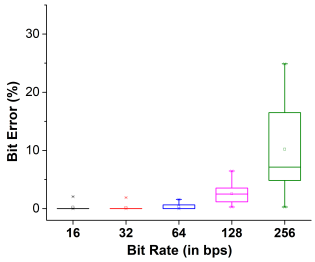

Figure D.4: 3G-4G: Bit error rate variation for different bit rate for 100B transfer.

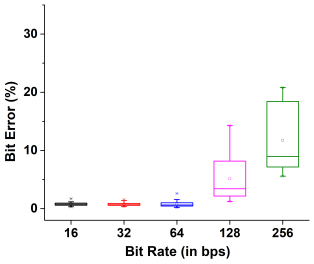

Figure D.5: 3G-4G: Bit error rate variation for different bit rate for 1KB transfer.

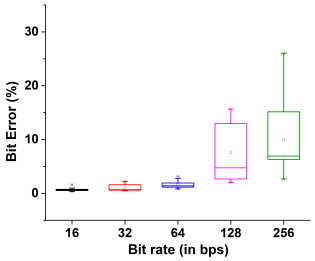

Figure D.6: 3G-4G: Bit error rate variation for different bit rate for 5KB transfer.

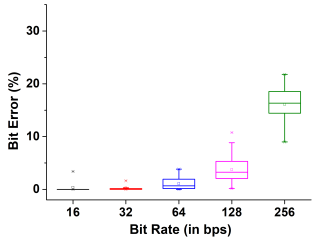

Figure D.7: 3G-3G: Bit error rate variation for different bit rate for 100B transfer.

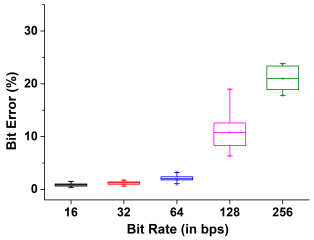

Figure D.8: 3G-3G: Bit error rate variation for different bit rate for 1KB transfer.

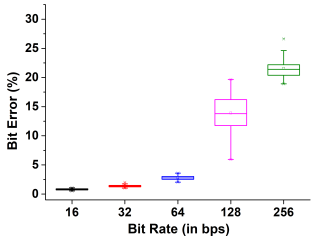

Figure D.9: 3G-3G: Bit error rate variation for different bit rate for 5KB transfer.

In order to gauge the performance of Dolphin with varying cellular connectivity, we conducted experiments with different connectivity scenarios. Overall there are six possible combinations of caller and callee for 2G, 3G, and 4G voice connectivity *i.e.,* 4G-4G, 3G-4G, 2G-4G, 3G-3G, 2G-3G and 2G-2G. For each combination, we transferred data of different sizes (100, 1000 and 5000 bytes) at different bit rates (16,32,,,256), and record the BER. We have already depicted the results of 2G-4G setting. Thus, here we present the results of remaining scenarios in Fig. D.1 to

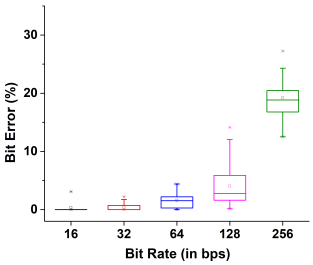

Figure D.10: 2G-3G: Bit error rate variation for different bit rate for 100B transfer.

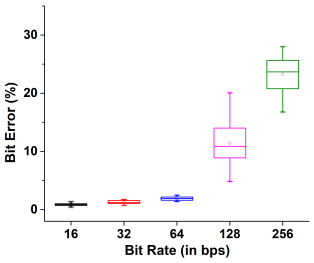

Figure D.11: 2G-3G: Bit error rate variation for different bit rate for 1KB transfer.

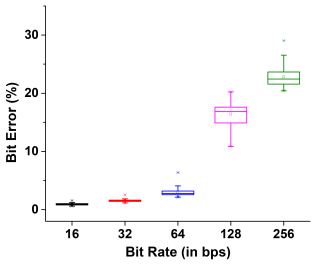

Figure D.12: 2G-3G: Bit error rate variation for different bit rate for 5KB transfer.

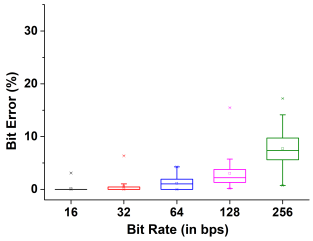

Figure D.13: 2G-2G: Bit error rate variation for different bit rate for 100B transfer.

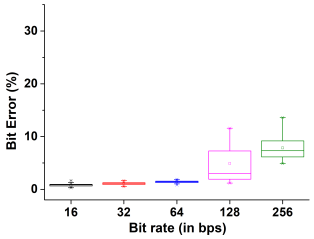

Figure D.14: 2G-2G: Bit error rate variation for different bit rate for 1KB transfer.

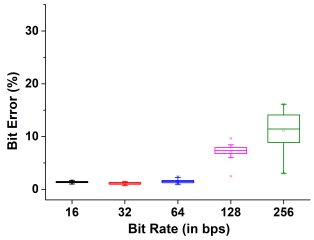

Figure D.15: 2G-2G: Bit error rate variation for different bit rate for 5KB transfer.

Fig. D.15.

It is evident that by and large, for all scenarios we obtain low bit error (<3%) for data rates below 64 bps. However, there is a significant increase in the BER at higher rates (20-30% for 256 bps). Thus, we selected 64 bps as our data sending rate as the error percentage was consistently low for all possible scenarios. Overall, these results establishes that Dolphin would work across all cellular connectivity scenarios.