DOLPHIN: A CELLULAR VOICE BASED INTERNET SHUTDOWN RESISTANCE SYSTEM

BY

RISHI SHARMA

Under the supervision of

Dr. Sambuddho Chakravarty

Dr. Mukulika Maity

INDRAPRASTHA INSTITUTE OF INFORMATION TECHNOLOGY DELHI

July, 2022

DOLPHIN: A CELLULAR VOICE BASED INTERNET SHUTDOWN RESISTANCE SYSTEM

BY

RISHI SHARMA

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology

TO

INDRAPRASTHA INSTITUTE OF INFORMATION TECHNOLOGY DELHI

July, 2022

Certificate

This is to certify that the thesis titled *Dolphin: A Cellular Voice Based Internet Shutdown Resistance System* being submitted by *Rishi Sharma* to the Indraprastha Institute of Information Technology Delhi, for the award of the degree of Master of Technology, is an original research work carried out by him under my supervision. In my opinion, the thesis has reached the standards 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/diploma.

Dr. Sambuddho Chakravarty

Dr. Mukulika Maity

July, 2022

Indraprastha Institute of Information Technology Delhi

New Delhi 110020

Acknowledgements

I would like to sincerely thank my advisors, Dr. Sambuddho Chakravartyand Dr. Mukulika Maity, whose constant guidance, availability, and support have enabled me to navigate every obstacle encountered in the course of this project. I have learned and grown immensely through their constructive feedback on my work.

I am also thankful to former PhD students Dr. Devashish Gosain and Dr. Piyush Sharma for their patient guidance during my journey.

I would also like to acknowledge my family and friends for being a consistent source of support through exceptionally trying times in my life.

Lastly, I am grateful for my time at IIIT Delhi, which has proved to be a turning point in my life and has provided me with the opportunity to learn from exceptionally knowledgeable professors to better myself at my craft. Every lecture has been a privilege.

Abstract

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 shutdown, leading to complete Internet disconnection, severely impacting lives in such regions. Naturally, these shutdowns render all existing circumvention schemes unusable.

Thus, 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 peer 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 that works atop any underlying encoding and modulation scheme. Further, to evade eavesdroppers over the insecure voice channel, we provide end-to-end confidentiality. Also, Dolphin can function 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 received in under 2 minutes).

Contents

List of Figures

List of Tables

Chapter 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 [2–4]. This has lead to an ongoing arms race between adversaries and free speech activists across the globe; adversaries continue to evolve various censorship techniques [5–9], whereas civil liberty activists counter them with wide range of novel circumvention systems [10–13].

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 [14]). Due to the complete Internet disconnection, none of the available circumvention tools work.

Moreover, such a step has severe impact on the lives of people residing in shutdown regions. They are even devoid of accessing essential services over Internet *e.g.,* access to news, reporting

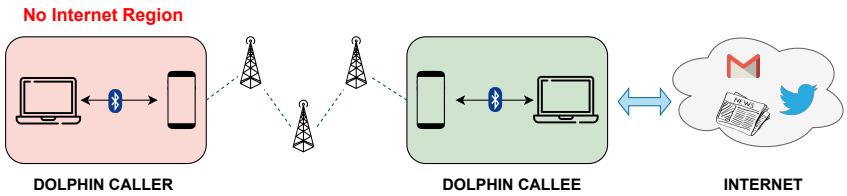

Figure 1.1: Overview of Dolphin's architecture.

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 rural areas were not even aware of the pandemic for many months [15]. 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 [14]. Considering that such trends are becoming common, it is plausible that more nation states opt for such measures [16,17]. 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.*

There may be multiple alternatives to exchange information during Internet shutdowns. A naïve solution may involve users in shutdown regions directly speaking to their friends and acquaintances in non-shutdown region, over regular voice calls. However, it does not assure confidentiality and is prone to eavesdropping by the cellular provider. Same is also true for SMS messages, besides being capped in several countries [18, 19]. Further, approaches like setting up separate ad-hoc networks [20, 21], using low earth satellites [22] and satellite phones have merit, but may encounter infrastructural and deployment challenges (*e.g.* requiring exceptional authorizations).

Thus, we introduce *Dolphin*, a novel system that can provide access to lightweight and delay-tolerant Internet applications by simply using the existing cellular voice channel to transmit encoded data bits. 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 [23–26].

Novel use of cellular voice channel: The voice channel is by design not built for running Internet applications. It is unreliable, lossy, insecure and highly bandwidth constrained. In the past researchers have explored the feasibility of transmitting data over cellular calls [27–29], but primarily through simulations and thus may not be representative of the challenges one might face when employing those schemes in real world scenarios. To the best of our knowledge none of the prior work attempted to practically use the cellular channel to access Internet applications and counter the challenges.

Dolphin overview: Dolphin user requires running a Dolphin client utility on its host, while also requiring a peer (*e.g.*, a friend) in a non-shutdown region to run a server utility. Both the peers also require mobile phones, paired to their respective hosts, through which the cellular call will be placed. Dolphin client's utility initiates a cellular call to the peer, that the Dolphin server program automatically receives. Once the user has some data to send (email, tweet *etc.*), it provides it to the Dolphin client which encodes (and encrypts) 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 recovers 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. 1.1.

Does Dolphin emulate dial-up modems? At a first glance, Dolphin seems similar to legacy dial-up modems. Thus, one may believe that the same voice modems could also be used in Dolphin. But such voice modems worked largely for landline connections, and the few that supported cellular channels are now obsolete. With the exponential growth of cellular users, service providers now use extreme compression and psycho-acoustic techniques that filter audio features that are not essential for humans to perceive. This renders the channel unsuitable for transmitting data using legacy modems [30].

Major challenges for Dolphin: We now enlist the three major challenges in sending data bits

using the cellular voice channel. First, the voice encoded data (that is to be transmitted over the 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 such frequencies 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 reliable and in order delivery of data. We discuss in Sec. 3.2, why especially for Dolphin, standard TCP is not a good option with respect to performance.

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.

Dolphin's proof-of-concept implementation: We successfully demonstrate that using Dolphin 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.[1]

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) ensuring reliable *end-to-end* transfer of data. Thus, it can be easily extended to support other lightweight applications as well.

Furthermore, Users can transmit/download a large file by leveraging multiple parallel calls. Dolphin can securely transmit a 2000 character file in 9 minutes just by using 2 parallel calls. Also, in case of call disconnections, Dolphin allows its users to resume the previous transmis-

[1]This duration to send an email might seem very large. Thus one can argue that a user can simply call the trusted friend and request him to send the email directly on his behalf. However, this dependence on the human peer could hamper the usability, as the peer has to be available whenever the user wishes to access the Internet applications. Dolphin, being a completely automated system addresses these concerns.

sion/downloading session (section 3.3).

Additionally, we tested Dolphin during a real shutdown event [31] that occurred in Delhi, India and confirmed that there also Dolphin worked with similar performance. Moreover, by design, Dolphin is easy to adopt and use—it requires access to a computer and a bluetooth enabled smartphone, and relies on commonly available open source libraries. It is agnostic to the underlying cellular technology (2G/3G/4G voice). Additionally, we also provide a way for users to access Internet, even with a fully-automated peer, that *requires no human support* after an initial setup. This is achieved using cellular voice automation services (such as Twilio [32]) that enables hosting the Dolphin server program on a cloud, while providing a local number that users could call. (ref. Sec. 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 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 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).
- For usability purpose, Dolphin provides pause and resume feature. Further, using multiple parallel calls, Dolphin can speed up the

Chapter 2

Internet shutdown and outages

Internet shutdowns are deliberate acts of turning off the Internet connectivity in a particular region (city, state or even a country) 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. These shutdowns could last for less than a day to over a year in some cases (472 days in Chad) [14]

Various projects keep track of these shutdowns at country as well as global scale. *E.g.,* `accessnow` project [14] categorically reports incidents of shutdowns occurring across the globe, presenting detailed statistics of such events. Further, there are country specific projects such as [33] 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` [34].

Other projects attempt to identify Internet outages in general. *E.g.,* IODA [35] 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) [36] 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).

Chapter 3

Dolphin System Design

We now describe the overall design of Dolphin. We begin by describing the individual components of Dolphin (depicted in Fig. 1.1) 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:

- Dolphin caller machine: This machine runs the Dolphin client utility. It accepts input from the user (*e.g.,* text) that it wishes to send over the Internet (*e.g.* as an email or a tweet). The client utility inputs the text to an audio encoder (explained in detail ahead in Sec. 4.2), which encodes the text to audio format. 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. 4.1). The audio received from the host is relayed to the Dolphin callee mobile over a standard voice call.

Dolphin callee infrastructure consists of:

- Dolphin callee mobile phone: This phone receives the call from the caller‘s phone and forwards the received audio to the server machine, via 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.*).

3.1 Dolphin communication protocol

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

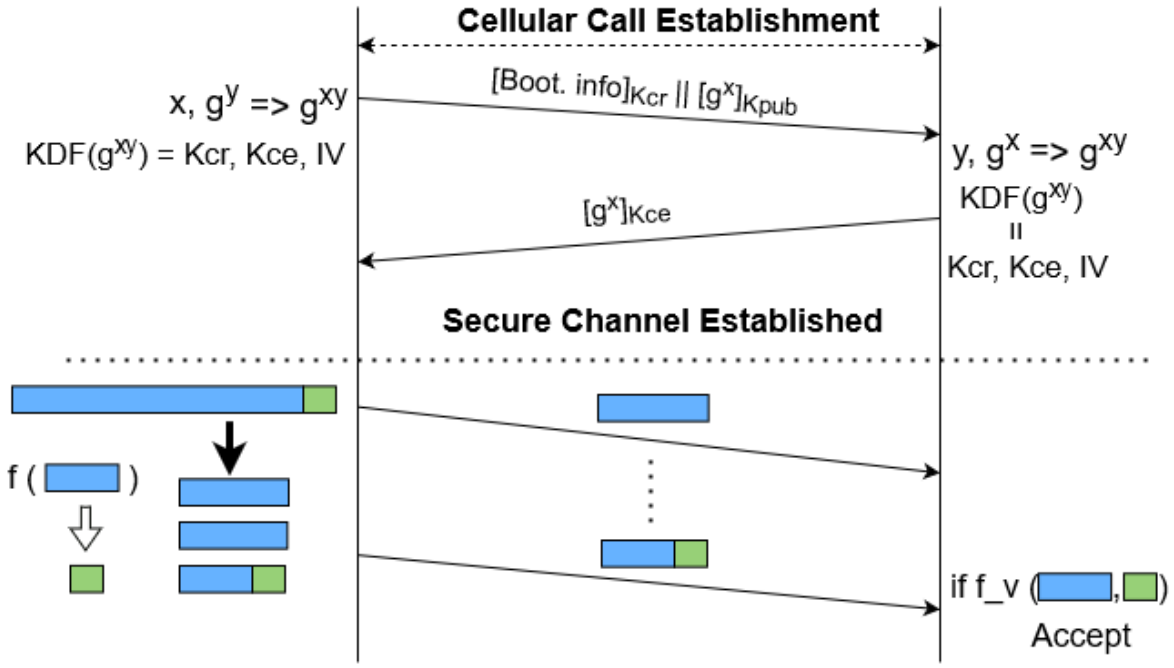

Figure 3.1: Dolphin's secure channel and data transmission phases. f() computes HMAC tag (green) and f_v() verifies it.

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. 3.1). 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 [37]) 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 magic 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 magic 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.

   In Sec. 6 we discuss our threat model in detail along with an analysis of possible attacks.

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 ensures integrity and authenticity of the data

bits. For efficient capacity utilization, the plaintext data bits are first compressed, before being encrypted and encoded.

2. The resulting data is divided into data frames and is transmitted sequentially 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).

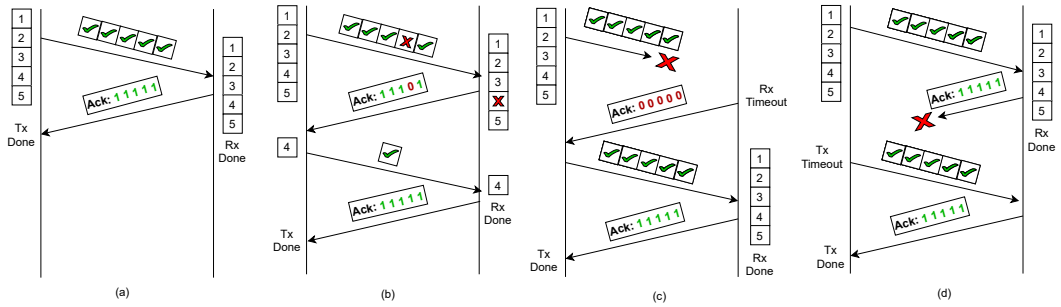

Figure 3.2: Some representative scenarios that are handled by Dolphin's reliability protocol: (a) represents the best case where no data is corrupted/lost, (b) depicts the case where one (or more) chunks are corrupted/lost, (c) is the case where a complete batch of chunks is corrupted/lost, and in (d) 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.

3.2 Dolphin reliability protocol

The above walkthrough raises several important questions *i.e.,* how is the data flow controlled, how is the data integrity preserved and verified *etc.* Moreover, it is known that the voice channel is lossy. Thus, a natural question is how to ensure reliable data transfer over the lossy cellular voice channel?

One approach is to directly use the standard TCP protocol between the caller and callee to ensure reliability. However, using standard TCP directly would lead to performance degradation.

This is because, in practice, we are able to transmit data at low transfer rates of about 64 bps over the voice channel, with tolerable errors (ref. Sec. 5.1). With such limited bandwidth, the overheads of the headers itself severely impact the overall performance. *E.g.,* a TCP ACK packet has a minimum header size of 40 bytes (i.e.) 320 bits, thus, even if there was no error, it would take atleast 5 seconds just to transfer a single ACK packet. Moreover, sending standard MTU sized packets will be detrimental from performance perspective as larger the amount of data transmitted , the more the chances of encountering errors during transmission (due to lossy nature of real-time voice), thus making it prone to many re-transmissions. Overall, it is not feasible to use standard TCP for Dolphin as it can severely impact performance.

Thus, to achieve reliable 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, while minimizing the overheads for such operations to a bare minimum. Moreover, as described ahead (ref. Sec. 5.1), we select a fixed bit rate for transmitting data and thus do not require congestion control mechanisms of TCP.

Our protocol involves dividing the data into small fixed sized chunks and transmitting each of them with their respective checksums. Small sized chunks help in localizing the impact of any data corruption or losses. 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 large sized data. This scheme helps in reducing the number of possible re-transmissions while transferring data. *E.g.,* one way to transmit 100 bytes data is to send it as a single chunk. An alternate way is to divide this data into smaller chunk sizes of say 20 bytes each before sending it. In the 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 a single 20 bytes chunk. This potentially leads to a five fold decrease in the amount of data to be re-transmitted. Thus dividing the data into smaller size chunks helps us in minimizing the amount of data to be re-transmitted. Also, our scheme requires transferring only 1 bit for acknowledging each individual chunk (ref. Sec. 3.2). In comparison to direct TCP, this is about 320 times reduction in the overhead.

Moreover, we transmit the data at a bit rate of 64 bps, and the acknowledgements (or other control messages) at a relatively slower rate of 16 bps. The control information is sent at a low rate to minimize the chances of its corruption so that we do not have to re-transmit this information again, as it does not contribute to overall data transmission. Moreover, since the control information is only a few bytes, transmitting them at low rates does not hamper the overall performance.

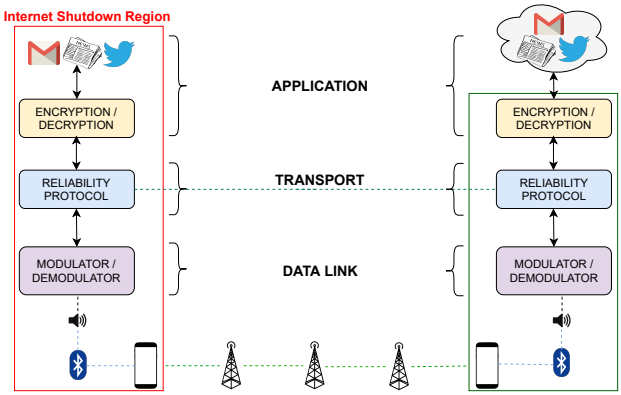

Figure 3.3: Dolphin's block diagram depicting its different functionalities (end-to-end).

Having discussed the major motivation and driving factors behind the reliability protocol, we now describe the end-to-end functioning of **Dolphin's reliability protocol**.

1. In order to transfer data in either direction, first the data is divided into smaller chunks of fixed size. Each chunk consists of data bits and the corresponding integrity check (CRC). These chunks are also prepended with a sequence number for managing their order (ref. Fig. 3.4).

2. Thereafter, the sender transmits a batch of chunks sequentially. The exact number of chunks in a batch are fixed and known to both the parties beforehand (with the help of bootstrapping information). 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 corrupted chunks (thereby soliciting re-transmission).

5. The sender receives the acknowledgement, verifies its integrity, identifies the corrupted chunks, and 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. Moreover, once the complete data has been received, the HMAC tag appended at the end is used as an additional mechanism to verify the integrity and authenticity of the complete received data.

8. Once there is no more subsequent data to be sent or application to access, the call is disconnected.

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. 3.2 and the overall working of Dolphin along with how the different components interact is depicted in Fig. 3.3.

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.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* [38]. 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) [39] 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.

It must be noted that we also handle the case when the delimiter itself gets corrupted. *E.g.,* if five chunks were transferred (numbered 1 to 5) but the delimiter of the second chunk was lost or corrupted. In this case, the delineation would detect four chunks numbered 1, 2, 4 and 5. Further, checksum validation would mark chunk 2 as corrupted (as it essentially contains data of both chunk 2 and 3) and mark chunk 1, 4 and 5 as correctly received. Thus, appropriate acknowledgements would be generated so that chunk 2 and 3 can be re-transmitted.

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. 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.

Dolphin assign sequence numbers from the set { 0, 1, 2, ... W } where W is a wraparound value < 256 (Since we are restricting sequence numbers to a single byte). Ideally this wraparound

value should at-least be more than twice the batch size to avoid synchronization issues between the sender and the receiver. In our experimental setup we have fixed it at 64 as we were usually dealing with batch sizes of 8 chunks.

Acknowledgements: Acknowledgements identify the correctly and incorrectly received (or lost) chunks. Each chunk corresponding to its seq. no. is assigned either a bit 1 (correctly received) or 0 (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.,* "11110011". The acknowledgement will also contain 1 byte for integrity verification.

One might argue as to why do we not send only the negative acknowledgements for the missing chunks. This could ideally further reduce the overheads. However, then we would require more than 1 bit per chunk as acknowledgement, since it would involve indicating the position of the missing chunk to the sender. In the current scheme the positioning information is implicitly handled. Moreover, sending just the negative acknowledgements would also make the size of acknowledgements variable, making it difficult to calculate appropriate timeout values and thus would not be beneficial.

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 (known beforehand 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. The selection of the delta value is backed by the observation that ITU [40] mandates the one way delay in a voice call to be strictly under 400ms. Thus selecting a value of 500 ms is reasonable. However, this parameter could be tuned as required.

Data compression: We perform the compression of data before encrypting it, as text data can be compressed with high compression ratio, as compared to cipher-text [41]. In our experiments, compression reduced the data size by 20%-60%, leading to overall lesser data being transferred over the voice channel.

Integrity check: The integrity for each chunk is calculated using the CRC algorithm (CRC-

8) [42]. 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 [43], as we generally select a much smaller chunk size *i.e.,* 20 bytes. Moreover, we also transmit a one time HMAC tag with the complete data to additionally verify the overall integrity of received bytes.

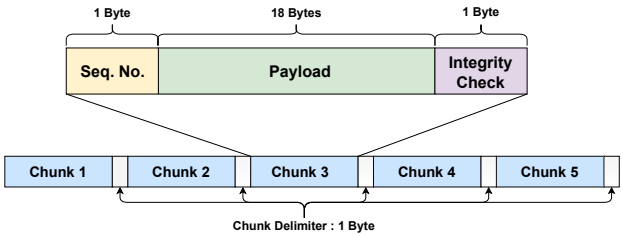

Figure 3.4: Details about individual chunks and how they are stacked before sending.

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. 3.4). 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.

3.3 Pause and Resume Protocol

We will now describe the Pause and Resume protocol used to download large files over multiple dolphin sessions. The Fig 3.5 depicts the protocol. For the sake of simplicity, we assume that the dolphin caller is requesting the file download. The pause and resume protocol builds over the data transmission protocol hence we can assume reliable in-order transmission of data on both

ends.

1. A cellular connection is made and a secure channel is established as described in section 3.1.

2. The caller requests a file using it's uri (uniform resource indicator) and suffixes the number of bytes it has received previously (hereafter referred to as seek). The seek value is initialised with 0.

3. The receiver (callee) would fetch the file using it's uri. If the uri is wrong it sends an error message. Otherwise it would start transmitting the part of the file requested based on the seek value.

4. In case the call is disconnected, the caller starts again from step 1 but with an updated seek value.

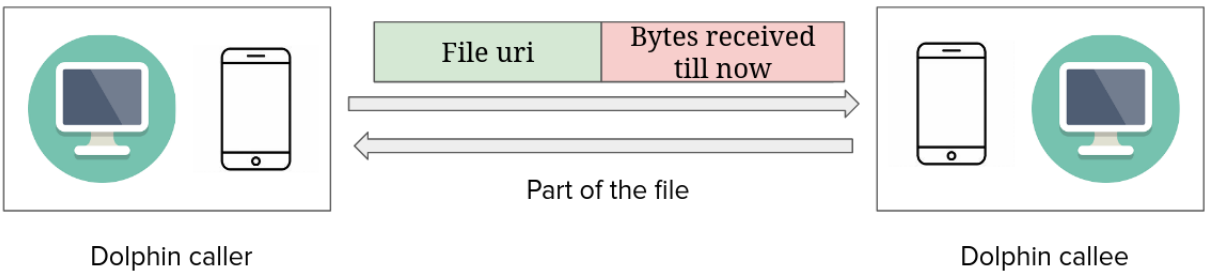

Figure 3.5: The pause and resume protocol.

3.4 Parallel Connections

We will now describe how we can utilize parallel dolphin connections to speedup file downloads. All parallel connections use the pause and resume protocol (Sec 3.3) to download the file even if one of the connection is disconnected. Figure Fig 3.6 depicts the protocol.

For the sake of simplicity, we assume that the dolphin caller is requesting the file from the internet and only utilizing two parallel connections. Also, the different dolphin callee's are independent and hence the caller might very well be connected to two different peers. Only

the two dolphin caller are assumed to coupled even though they are using two different mobile phones. A main controller (hereafter referred to as client) would be controlling the different caller's and using each of them to receive parts of data. The final file would be reconstructed by concatenating the parts.

Having discussed the major assumptions about the system, we now list the exact steps taken under the protocol:

1. The client would setup two different dolphin connections and establish secure channels over both of them.

2. It would assign a static weight (w_c) and a index to each channel with the following constraints, $0 \leq w_c \leq 1$ and $\sum w_c = 1$. This set of weights is called a distribution. These weights are used by the callee to appropriately divide the final file in parts and transmit over the corresponding connection.

3. When a file is requested, the client sends the file uri and the distribution to the callers. The caller sends this information over the connection and the callee send backs the requested part of the file.

4. Once all the caller's have received the respective parts, the client collates all the parts and reconstruct the final file.

3.5 Modes of operation

We now enlist the two operating modes of Dolphin:

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

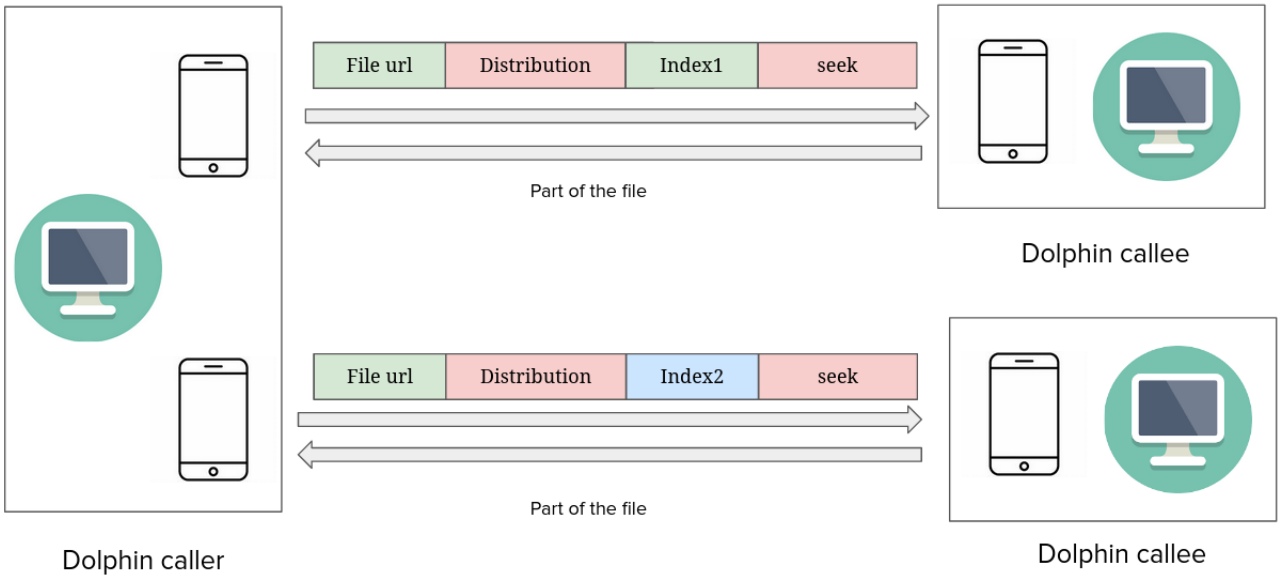

Figure 3.6: The parallel connection protocol.

model is not always conducive — what if one cannot find peers in non-shutdown regions? To answer this question, we introduce the second mode of operation.

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 [32]). 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. 4.3). However, unlike the human callee mode, such services would incur periodic subscription fee.

Chapter 4

Implementation Details

In this chapter we describe how we implemented the above protocol to easily use it in real-world scenarios.

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. 1.1). 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.

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 [44] 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` [45] 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) [46] 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` [47], 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. 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 DH over elliptic curve group (ECDH), instead of DH over finite cyclic group. ECDH

keys are 32 bytes long 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 [48]. The established shared secret along with the bootstrapping information is used by the peers as input to the PBKDF (password based key derivation function) to derive the key and IV. We used `pycrypto` [49] and `coincurve` [50] to perform the crypto operations.

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, the caller can enable two-factor authentication (SMS based) on its password protected accounts so that every new access requires the caller to provide an OTP, preventing unintended access. Second, caller can use OAuth token based access schemes. These 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 incorporates 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 [51] could be used that relies on multiple parties to derive the password, with no one party having complete information.

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` [52], and Twitter using `twython` [53] library. The news application is automated using `newsapi` [54], which returns concise news snippets based on a keyword query.

Transmitting files: Dolphin can securely transmit/download a file (like a website's html) essentially by making the callee act as proxy for the caller. The dolphin client would send the url with the number of equal size parts that the data needs to be divided and the index of the part that the server needs to send back.The server would download and cache the url and respond back with the appropriate data.

Accessing web pages: Dolphin can also access web pages that can be rendered in a browser in a semi real-time fashion.The client start a https server on the localhost to which the browser connects as a proxy. It will also setup multiple dolphin calls during the bootstrap phase. After the user types the url and hits enter, the browser will redirect the request to dolphin client, which would appropriately divide the data among servers (based on number of active connections), download it, and send it back to the server. To further decrease the size of the data to be downloaded, the server would use the textize api [55] to download a text only version of the website.

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 human 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] The details of the implementation can be referred to in Appendix 8.3.

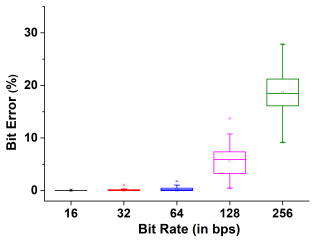

Figure 4.1: Bit error rate variation for different bit rates for 100B transfer.

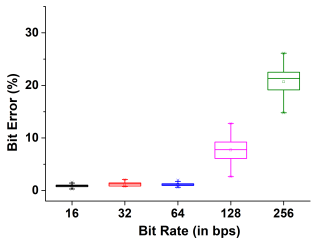

Figure 4.2: Bit error rate variation for different bit rates for 1000B transfer.

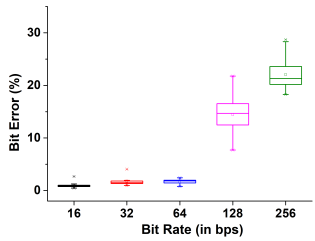

Figure 4.3: Bit error rate variation for different bit rates for 5000B transfer.

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

Chapter 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.

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	**Bit Rate (bps)**				
(Bytes)	**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 5.1: 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. 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 [56] 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 the bits transmitted yields the BER for data transmitted. In all our experiments, we computed the 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. 5.1 and present the complete error distributions in Fig. 4.1, Fig. 4.2 and Fig. 4.3 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.,* above 256 bps). This would reduce the overall latency. However, as demonstrated through our extensive experiments, higher data rate results in more error, 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.,* under 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.

It must be noted that we also explored several transmission rates between 64 and 128 bps. The error rates were directly proportionate to the increase in transmission rates (from 64 to 128 bps). We noticed an overall improvement in performance (average reliable transfer time) when using rates as high as 90 bps; but increasing it higher provided no significant performance gains. However, we also observed larger variance in error rates and download times when we used such relatively higher bit rates. At 64 bps, the data transmission was much more stable and consistent during our experiments. Thus, we used a rate of 64 bps for our experiments.

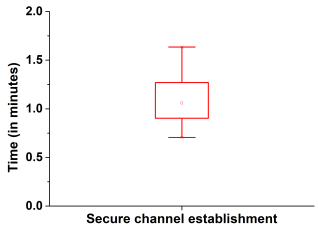

Figure 5.1: Dolphin's secure channel establishment time.

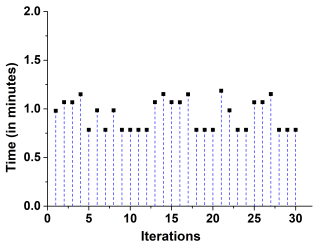

Figure 5.2: Time taken to tweet 280 characters (max. limit) using Dolphin.

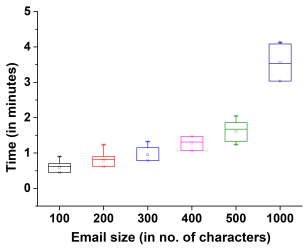

Figure 5.3: Time taken to send an email of varying sizes using Dolphin.

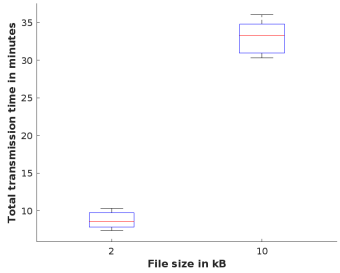

Figure 5.4: Time taken to send a file of varying length with two parallel connections.

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 manually switched to

use only 2G voice (representing bare minimum cellular connectivity) and the callee's phone was enabled with 4G voice connectivity. However, the peers may not always have such connectivity due to various reasons (such as intermittent signal or proximity to base station *etc.*).

Thus, to test the feasibility of all such 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. The BER received in these scenarios (*e.g.,* caller (2G) callee (4G), caller (3G) callee (4G) *etc.*) did not vary much (ref. Appendix 8.1 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 ($< 2\%$ error for bit rates < 64 bps) when we tested Dolphin for four popular providers which serve majority ($\approx 90\%$) of the users in their region [57]. Thus, one can infer that Dolphin functions well across different 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 (≈ 1100 miles apart) within the same country, as well as different countries (one in Asia and other in Europe, being≈ 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 2% error for 64 bps. Moreover, we also 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.

Impact of Bluetooth: One may argue, that bluetooth used for transferring data between the phone and laptop, might impact the transmission rates we observed. Thus, we conducted experiments to test if bluetooth impedes the achievable data rates. These experiments involved isolating the errors introduced by bluetooth transmission (if any) and comparing them to the

errors introduced by the cellular channel. We used the same setup as in Fig. 1.1. However, in this case, we recorded the audio at the callee's laptop as well as the caller's mobile phone. The latter bear the errors introduced due to bluetooth while the former contain the errors introduced by both the bluetooth as well as cellular channel. For all data rates (upto 256 bps), we observed that no errors were introduced by bluetooth (BER of zero). At the same time, the cellular channel introduced significant errors (Bit error of atleast 1%, max at about 20% for data rate of 256 bps). This confirms that bluetooth did not contributed to errors in audio, they were introduced only by the cellular voice channel.

Justifying Dolphin's observed data rates: It may be argued that the bit rate achieved by Dolphin is low and that it could be improved if existing modulation techniques that claim to provide a higher data rate are employed. To that end, we attempted to test these existing modulation techniques on our setup by encoding and sending data over the cellular channel, and measuring the corresponding error rates. Notably, there are very few modulation schemes that are made resistant to multiple distortions and processing, and thus, can actually be tested.

One such modulation scheme was proposed in Hermes [58]. It was developed in 2010 and claimed to obtain close to 1 Kbps data encoding rate with low bit error percentage ($< 1\%$). However, their code is not publicly available and the design of their demodulator is very complex with some missing details, making it difficult to implement. Another such research, authloop [30] developed in 2016, also tried to implement Hermes demodulator, but failed to do so. But, authloop build and implemented their own modulatotion scheme based on the ideas used in Hermes. Their modem claimed to have obtained about 500 bps encoding rate in simulation. Thus, we obtained Authloop modem code and used it to encode and sent data over real cellular channel to measure the error rates. We observed that authloop suffered about **55%** bit error rates when sent over the cellular channel. Thus, the previously developed modulation schemes that also extensively employed channel coding theory, were also not able to perform well when tested over the current cellular voice channel. We thus currently employ Dolphin's FSK based modulation and build the system with the practically achievable data rate of 64 bps. However, due to Dolphin's modular design, if in future some modulation scheme could provide better data rates, we could easily integrate that to improve Dolphin's performance.

Note: One may further question as to why the previous studies that claimed higher bit rates are not able to perform well. We investigated the reason for such difference and found out that the studies focused on the evaluation using only one of the many available modes of the popular AMR cellular codec. We performed simulation based evaluation and found out that for the mode which they tested, their modulation scheme performs well. However, for other modes their scheme faces large errors. Notably, Dolphin's modulation also performed at par or better when tested against the AMR codec mode used by previous studies, offering about 1% error at 1024 bps data encoding rate. We believe that the current cellular channel provides the large error inducing mode of AMR and thus makes it difficult to achieve high data encoding rates by Hermes, Authloop and Dolphin alike. (refer App. 8.2.5 for details)

5.2 Performance of Internet applications

In this subsection we quantify the performance when Internet applications (email, twitter *etc.*) are used over Dolphin.

As already described (Sec. 3.1), Dolphin works in two phases *i.e.,* the secure channel establishment phase and data transmission phase. Thus, first we quantify the time taken to establish a secure channel between the caller and callee. As depicted in Fig. 5.1, 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, for better security guarantees, 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. Using a single dolphin connection 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. 5.2). 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. 5.3. It is evident that it takes ≈ 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). It took us on an average 2 minutes to receive to receive the 10 snippets. Using two parallel dolphin connections we tested the file downloading application. We took two files of sizes 2kB and 10kB and downloaded them over two parallel connections. Each connection was given the same weight i.e through each connection 1kb and 5kb of data was transmitted. We repeated these experiments 30 times for both the cases. From the Fig. 5.4 it can be seen that a 2kB files takes an ≈ 8.7 minutes to download, whereas a 5kB file is downloaded in ≈ 33.02 minutes.

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. And in case of larger file transmissions, dolphin can deliver in a semi-reasonable time using 2 or more parallel connections.

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 5.2: Error percentage for varying bit rates and file sizes (100 B and 1000 B) 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. 5.2, 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.

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

5.4 Anecdotes

While conducting the experiments, we observed an Internet shutdown in the region of one of the authors (Delhi, India) [31]. 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 character email transferred reliably in about a minute), further establishing Dolphin's efficacy.

Chapter 6

Security Aspects of Dolphin

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

6.1 Threat model:

It is known that shutdowns are carried out by ISPs on the orders of some higher authorities. Thus we assume that when shutdown resistance systems like Dolphin would become popular among the masses the same authorities could direct the telecom operators to identify and (or) block such systems. This practically deems the telecom operators as adversaries. However, to the best of our knowledge, no prior research has explored the possibilities of telecom operators as censors. Thus, we try to characterize their capabilities. Unlike regular network eavesdroppers, cellular voice channels cannot be trivially analyzed by capturing packets; cellular voice networks (except VoLTE) do not work on the regular Internet's store-and-forward model. Thus, we believe that it will be difficult to perform real-time traffic analysis on ongoing calls for telecom operators. But operators may intercept and record audio calls. We confirmed this by communicating with a major telecom provider operating in a developing country with frequent shutdowns.

Thus, 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. Although, it may attempt to perform offline analysis on recorded cellular calls to identify Dolphin calls. However, performing analysis

on all the calls could be resource intensive and practically daunting for a cellular provider. Thus, it may instead opt for some "smart ways" to disrupt Dolphin. *E.g.,* adversary may add noise or perturbations in voice calls with an aim to completely disrupt Dolphin while refraining 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 [23–26].

6.2 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 be very powerful because the adversary can aim to completely block Dolphin without having to even detect if Dolphin is under use. There are largely two ways by which an adversary can try to induce these perturbations. One way is to just drop or disrupt voice samples of short duration (say 0.1s or 0.2s) at every fixed or random interval. The other way is to add a constant disturbance (*e.g.,* 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 at randomly chosen intervals. However, the reliability layer in Dolphin helps recover from random data losses and thus 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. To do so, the adversary would need to induce perturbations at very small interval. *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 periodic disturbances affect perceivable voice quality. Thus to measure voice quality, we used PESQ (Perceptual Evaluation of Speech Quality) [59], 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 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 an average PESQ score of 1.6, clearly establishing that the audio in the cellular channel would become perceivably distorted 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 $\approx$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 achieved a score 3.6, demonstrating that such a disruption will lead to complete disruption of Dolphin without severely impacting the perceptual quality for normal calls. But, as a countermeasure to this attack, we can slightly alter Dolphin's default configuration by soliciting acknowledgements 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, would unlikely be implemented 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%.[1] Thus, it would prove to be a futile exercise for the adversary to disrupt Dolphin with continuous noise as well.

6.3 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 he/she received a call with some gibberish tones. 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 [60]. 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.

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

6.4 Replay Attacks

An adversary can attempt to replay a part of (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 corresponding to 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. 3) must include fresh timestamps (otherwise they are silently dropped by the callee and responded with as described in the active probing attacks).

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 using Dolphin. One easy approach that can be adopted 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 data contains valid CRC checksums, periodically after every few bytes. The signature of periodic checksums, being unique to Dolphin, could lead to its detection. We remove this detectable feature by XOR-ing the CRC values with some extra random bytes derived from KDF while establishing the secure key. Since the KDF is given the same input by both peers, they will be able to derive the same random bytes to invert the XOR.

However, the adversary may use advanced signal processing techniques to differentiate regular audio, from that generated by Dolphin. Broadly, an audio signal can either be studied in the time or frequency domain. We start by describing the time domain analysis.

In order to distinguish Dolphin, the adversary can analyze the time domain waveforms of Dolphin encoded audio and compare them with normal human audio. As depicted in Fig. 6.1,

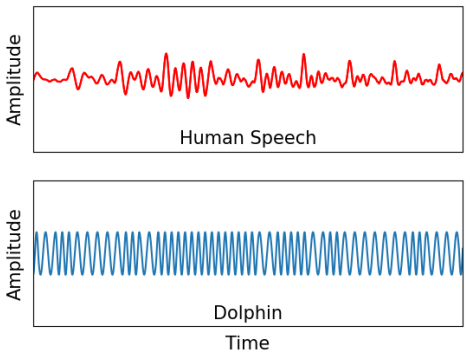

Figure 6.1: Signal waveform of 50 ms for normal human speech audio and Dolphin encoded audio.

one can visually differentiate Dolphin audio from normal human audio. This is because in Dolphin the amplitude and frequency of the signal has low variation in comparison to that of human speech. This distinguishing behaviour can also be characterized by a statistical analysis that records the change in amplitude and frequency of a signal across multiple time intervals, and if the change is relatively low, the waveform can be classified as Dolphin encoded. Using this analysis, we were able to distinguish Dolphin calls. The mean and standard deviation across all intervals (of 2 ms) for amplitude was 0.4 and 224 respectively for Dolphin. In comparison the mean of amplitude for normal human audio extracted from a large speech database [61] was 205 and 1590 respectively (about an order of magnitude difference in comparison to Dolphin).

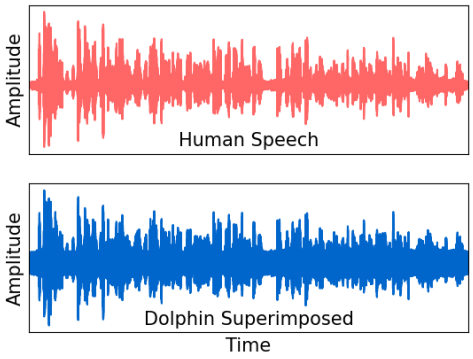

Figure 6.2: Signal waveform of 70s for normal human speech audio and Dolphin encoded audio superimposed over normal human speech.

As a countermeasure for such an analysis, we need to transform the Dolphin encoded audio to resemble more like normal human speech so that it cannot be distinguished. This can be achieved by superimposing Dolphin's voice encoded data with human speech. While superimposing, Dolphin's encoded data component should be suppressed as much as possible in order to make

detection harder. However, the suppression should be such that Dolphin's data can be decoded at the receiver with reasonable error rates. To find the sweet spot between suppression and decoding errors we performed experiments by varying the suppression values of Dolphin encoded audio and measured the corresponding error rates when sent over the cellular channel. We found that suppressing the Dolphin's audio encoded data by 20 dB gave us reasonable error rates ($\approx 1.5\%$); suppressing any further resulted in high errors ($> 5\%$). It can be seen from Fig. 6.2, that Dolphin superimposed audio looks almost identical to the normal human audio even when analyzed for a large time duration, with some additional noise component. Further, upon evaluation, this approach prevented us from distinguishing human speech from Dolphin's superimposed audio, based on frequency and amplitude variation. This is because, such variation was dominated by the normal human audio on which Dolphin was superimposed.

Alternatively, the adversary can analyze the audio signal in the frequency domain for finding distinguishing features between regular human speech and superimposed Dolphin audio. To convert the waveform into frequency domain, we employed the standard discrete fourier transform technique and analyzed the audio in this transformed domain. The frequency response of superimposed Dolphin audio was very similar to that of normal human speech extracted from the normal speech database [61] both visually and statistically. The mean and standard deviation of the frequency response for Dolphin superimposed audio was found out to be - 55.85 dB and 11.00 dB respectively. In comparison, for normal human audio we observed a mean and standard deviation of - 56.25 dB and 10.46 dB respectively. Thus, we believe, it will be difficult for the adversary to distinguish superimposed Dolphin to normal human audio even in the frequency domain.

However, a determined adversary can perform a deeper analysis that looks beyond signal characteristics. For instance, the adversary can try to see if the Dolphin superimposed audio resembles an actual human conversation. Usually, human conversation has random silences, which are absent in Dolphin's encoded data superimposed with voice. The absence of silences may help the adversary to identify Dolphin. Thus, we performed silence detection using short term analysis [62] which differentiates voice and silence based on the energy observed in a small interval. The voice frames would have much more energy than frames that contain silence.

As expected, we found no silences in Dolphin's encoded data superimposed with voice. In comparison, the normal human speech audio did contain random instance of silences.

As a countermeasure to such analysis, in Dolphin, silences can be introduced at appropriate intervals (e.g., at instances where silence is already present in the cover audio), keeping in mind that Dolphin's performance is not drastically impacted. Such silence introduction will make adversarial analysis difficult.

Lastly, the adversary may still be able to learn hidden features in Dolphin audio using advanced learning techniques. Analysis of such detection and proposing counter detection methods shall be explored in future.

6.5 Active Attacks

Here we consider possibilities of an adversary performing active attacks to disrupt Dolphin usage. Notably, performing active operations an all calls passing through the telecom operator will hamper the QoS of benign users, which is outside the scope of our threat model. However, the adversary can attempt to perform such analysis on some suspicious calls (which are difficult to identify if they are superimposed using normal human voice). That being said, the analysis will be similar to the one done in perturbation attacks, where adversary introduced noise in calls and aimed to disrupt Dolphin by corrupting the chunks or the acknowledgements.

However, here we assume that the adversary can act as a man in the middle in an ongoing cellular call and modify or inject data or acks to disrupt Dolphin sessions. Modifying or forging data or acks would eventually be detected and recovered due to the integrity mechanisms (CRC for chunks + HMAC on complete data) and reliability protocol. However, if the adversary aims to consistently disrupt Dolphin service, it would need to forge/modify all acks. As discussed in Sec. 6.2 doing so for a benign call would lead to poor QoS and make the cellular channel unusable. Thus, the adversary would not perform such operations. Notably, forging selective real-time voice data in an ongoing call with superimposed Dolphin audio may in itself be an extremely challenging task for an adversary.

Chapter 7

Relevant Work

In the traditional censorship circumvention literature various systems have been proposed in the past that send data using the underlying VoIP or video channel over the Internet [63–68]. However, the cellular voice channel is different and thus has received separate attention with several prior research efforts exploring the feasibility of sending data bits over the cellular voice channel [27–29, 58, 69–72]. But, these studies are either for a particular codec, provide just the theoretical analysis, or do not provide accessible code. Moreover, 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 unprecedented data distortion and losses (*e.g.,* due to poor connectivity of mobile devices with the base stations). This behavior may greatly hamper the functioning of existing schemes and has not been studied. Thus, with Dolphin we develop an end-to-end system and demonstrate the practicality and usability of sending data over voice by running actual Internet applications using the cellular voice channel. We solve various challenges and build a security and reliability layer, that can work with any underlying modulation scheme.

Chapter 8

Discussion

8.1 Results for Varied Cellular Connectivity

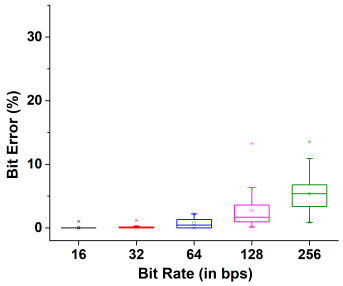

Figure 8.1: 4G-4G: Bit error rate variation for different bit rate for 100B content transfer.

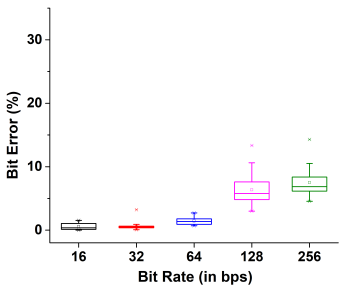

Figure 8.2: 4G-4G: Bit error rate variation for different bit rate for 1KB transfer.

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

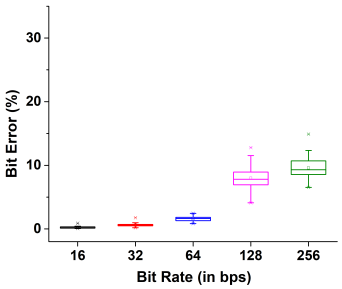

Figure 8.3: 4G-4G: Bit error rate variation for different bit rate for 5KB transfer.

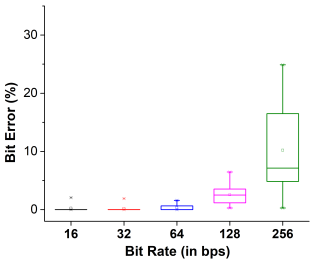

Figure 8.4: 3G-4G: Bit error rate variation for different bit rate for 100B transfer.

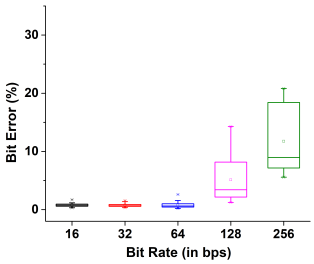

Figure 8.5: 3G-4G: Bit error rate variation for different bit rate for 1KB transfer.

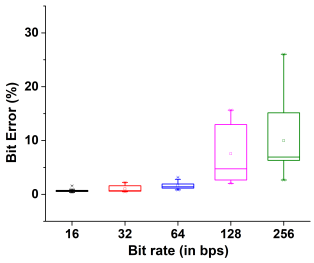

Figure 8.6: 3G-4G: Bit error rate variation for different bit rate for 5KB transfer.

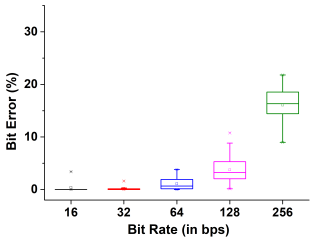

Figure 8.7: 3G-3G: Bit error rate variation for different bit rate for 100B transfer.

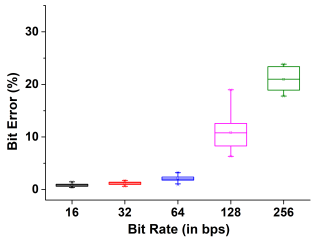

Figure 8.8: 3G-3G: Bit error rate variation for different bit rate for 1KB transfer.

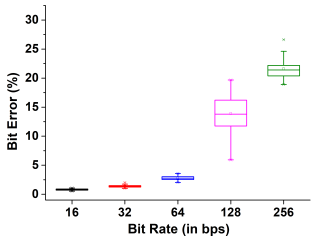

Figure 8.9: 3G-3G: Bit error rate variation for different bit rate for 5KB transfer.

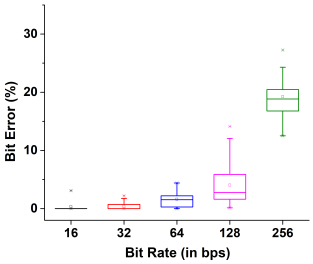

Figure 8.10: 2G-3G: Bit error rate variation for different bit rate for 100B transfer.

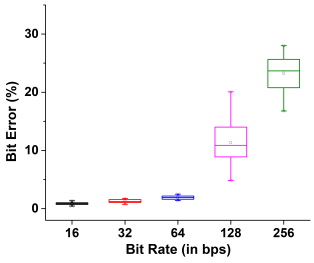

Figure 8.11: 2G-3G: Bit error rate variation for different bit rate for 1KB transfer.

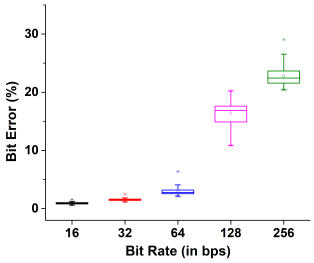

Figure 8.12: 2G-3G: Bit error rate variation for different bit rate for 5KB transfer.

the results of 2G-4G setting. Thus, here we present the results of remaining scenarios in Fig. 8.1 to Fig. 8.15.

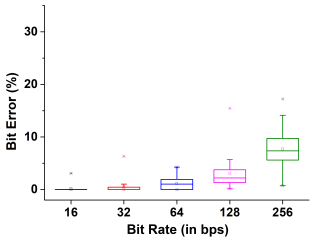

Figure 8.13: 2G-2G: Bit error rate variation for different bit rate for 100B transfer.

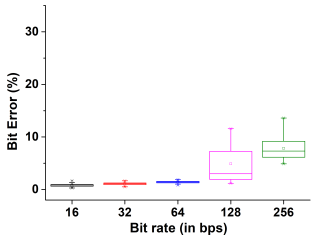

Figure 8.14: 2G-2G: Bit error rate variation for different bit rate for 1KB transfer.

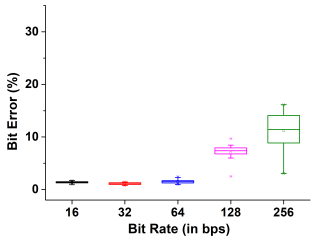

Figure 8.15: 2G-2G: Bit error rate variation for different bit rate for 5KB transfer.

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.

8.2 Additional Discussion Points

8.2.1 Dolphin Usability

Dolphin is an end-to-end system and has a modular architecture such that every layer can be developed independently. Dolphin supports bi-directional communication and can be easily used as an API for transferring any application's content. The API offers send(data) and recv() function calls (similar to the standard linux send() and recv() system calls) which can transfer or receive data via the underlying cellular channel using the Dolphin protocol.

We used this API and tested it for three popular Internet applications. Moreover, we also tested Dolphin for accessing websites. To do so, we built a TLS proxy that can be used with browsers such as firefox and chrome. The proxy listens on a local port for TCP connections from the browser and performs MITM between the browser and the website. The proxy only forwards the HTTP GET request using the send() call, which transfers the request to the Dolphin server (already listening using the recv() call), where the corresponding HTTP response is retrieved and sent back to the proxy. The proxy then forwards the HTTP response to the browser. The

above cycle is repeated for subsequent website requests. The current implementation of Dolphin and the proxy is functioning and tested to access static websites. Notably, we do not tunnel actual TCP and TLS packets between the client and the website as the overhead to do so via the bandwidth constraint cellular channel is prohibitively large.

8.2.2 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.

8.2.3 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. Exploring new techniques (such as modulation) to increasing the data rate further, in the face of such errors 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.

Additionally, there are researches that explored sending data at relatively higher bit rates in

packet based VoLTE systems [73, 74]. Such schemes exploited the access control implementation vulnerabilities to transfer data packets. These schemes may be beneficial in scenarios where VoLTE services are maintained in the shutdown region. However, an adversary can very easily restrict such schemes by disabling VoLTE and allowing only 2G and 3G voice services to function. Also, in developing countries where such shutdowns are most prevalent, VoLTE services are anyways not very widespread. Moreover, these schemes require rooting the phones and modifying the kernel to achieve data transfer which would not be very usable even for most tech savvy users. On the other hand, Dolphin's scheme works irrespective of the underlying cellular technology and is usable even for general non-tech users.

8.2.4 Using Dolphin as a covert channel:

The primary focus of designing Dolphin has been to provide basic Internet access in regions experiencing shutdowns.

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 a secondary secret communication channel to bootstrap various anti-censorship systems [75, 76] *etc.*

8.2.5 Potential reason for observing low bit rates:

In Dolphin, the feasibility study of encoding and sending data at different bit rates revealed that we can encode data at bit rates of around 64 bps with reasonable error rates. But, there are existing studies [30, 58] that show that data encoding rate can be much higher (500 bps to 1000 bps). When we tested them on real cellular call we observed poor performance for these studies. We believe that the potential reason for such differences can be explained as follows.

First, in the context of this work, it is important to understand the role of modulation and codec when encoding data bits to be sent over the cellular channel. Modulation is responsible for encoding/converting the data bits to analog signal such that this signal can be transmitted over

the physical medium. Similarly, demodulation is applied at the receiver to obtain the encoded data bits from the analog signal. A codec transforms this data encoded modulated signal such that it can be transmitted on a constrained and capacity limited physical medium (in terms of bandwidth available *etc.*). The codecs are optimized to preserve the audio features in the signal and do not care about if the audio contains encoded data bits instead, leading to large data errors. The modulation schemes try to anticipate and minimize the distortions due to codec to ensure good achievable data encoding rate. Thus, the success of the modulation scheme is dependent on the underlying codec.

One of the oldest and most widely used codec in cellular networks is the AMR codec. It is an adaptive multirate codec and provides different modes each working on different bandwidth or bit rates (4.75 to 12.2 kbps). The network operators selects one of these modes depending on the network condition, the client density *etc*. However, the few previous studies (Hermes and authloop) that proposed a codec independent modulation scheme assumed and evaluated their system for the maximum AMR bit rate mode *i.e.,* 12.2 kbps.

When we tested Dolphin against AMR's 12.2 kbps configuration codec in simulation, we observed that it offers bit error rates comparable to those reported in the previous studies. Dolphin's modulation observed a bit error rate of 1.02% for data encoding rate of 512 bps [1]. However, when we test Dolphin on a lower AMR mode of 7.4 kbps, the error observed for the same encoding rate increases to 15%. In comparison, we observed that authloop introduces an error of about 23%, on the same AMR mode of 7.4 Kbps, which is even higher than Dolphin.

Thus, overall we believe that while performing the experiments for Dolphin, we obtained one of the lower AMR rate modes from the service providers, leading to higher bit error rates. However, if we get a higher rate AMR codec modes or some other codec, the obtained data rate and Dolphin's performance can be drastically increased. Moreover, we tested Dolphin for one of the worst codec configurations and show that still it can be used to access lightweight applications. Thus, Dolphin should be able to obtain performance atleast similar to the one reported in this paper or higher. In future, we can introduce a module in Dolphin, which does some initial testing to get a sense of the maximum achievable data rate and performs the data

[1] Similar low error rate of about 1% was also observed for 1024 bps.

transfer according to the available data rates.

8.2.6 Motivation for not shutting down the cellular channel

Internet shutdowns are generally employed to stop the rapid broadcast of information and the organization of protests. Such broadcasts generally happen over social media apps such as Whatsapp, Facebook, etc. But, in the absence of Internet access, it becomes extremely difficult to use cellular channels for such purposes. Moreover, blocking the cellular voice channel, besides the Internet, would completely disconnect the masses and prevent them from availing critical services e.g. banking, emergency healthcare, civic helpline etc. Hence, the known censors do not have much motivation to block the cellular voice channel by default (as backed by multiple cited instances). Additionally, even with Dolphin, it is not trivial to spread information en-masse due to its practically achievable data rate, making it less appealing for the adversary to completely block the cellular channel. However, in the worst case, if the censor decides, than it could still block the cellular channel, but with serious collateral damages

8.2.7 VoIP vs cellular

The major distortions in the voice channel are generally introduced by codecs. The codecs used for VoIP communication are much less sophisticated as they work over the Internet and they are not very constarined about bandwidth. However, cellular codecs are designed for very bandwidth constrained operations and thus perform many psycoacoustic optimizations on the original audio leading to much more distortions. Thus, the solutions and analysis for VoIP systems are not easily applicable to the cellular channels. Moreover, we also performed some evaluation to see the achievable data rates over VoIP apps using Dolphin's modulation. We found out that we were able to achieve a data encoding rate of 1 Kbps with reasonable error rates (about 2We include a section on Page xx under the heading "VoIP vs cellular" highlighting the above.

8.2.8 Cellular call charges

It is difficult to do a cost analysis as it would be dependent on the region from which the call is being made and the region to which it is being made to. Internet shutdowns are generally performed in developing countries which usually have a low cost. For instance, we studied the incurred cost in India, which is the country with most shutdowns across the globe. International call rate in one of the most popular provider in India i.e., Reliance Jio, ranges from 0.0065 $ to 0.077 $ per minute to all countries around the world []. Thus, for a 1 hour operation of Dolphin, it would cost from a min of 0.39 $ to a max of 4.62 $. In Myanmar it costs about 1.62 $ for an hour. In ethiopia it would cost about 10.8 $. The call rates would be higher if the shutdown is being performed in developed countries.

8.2.9 Comparison with existing work

We discuss in detail in the previous subsection about how Hermes and Authloop's modulation schemes compare to that of Dolphin. But, there are a few other studies that also propose a modulation scheme for encoding data bits. LaDue et al. [27] proposed a modulation scheme that was designed specifically for the GSM-EFR codec. Similarly, Ozkan et al. [69] also designed and tested the modem for the GSM FR codec. Since, these modems are designed for a specific codec, they are not good candidates to be considered for integration in Dolphin. Another, modem was designed by Ali et al. [29] and was tested for AMR 12.2 kbps mode as well as the AMR 4.75 mode. However, for the lower rate AMR mode, the authors demonstrated a data encoding rate of close to 80 bps, which is similar to that of Dolphin.

8.2.10 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 [18, 19]. 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. These limitations make SMS less suitable for sending data in comparison to voice channel.

8.3 Automated Callee Implementation

`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` [77] 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 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.

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

8.4 Accessing websites over dolphin

A big limitation of dolphin is it's inability to provide normal browsing capabilities to it's users. We fetched home pages of alexa top 1000 websites and found out that the average size is at around 100 KB and can easily be as high 5 MB for some news websites. Hence, browsing these websites would very much be non real time. To decrease the page size, we used the textize api [55]. Textize would extract all the text and serve a plain text version of the file. After textizing and compressing the resulting plain text file we were about to get an average file size of 10 KB with going to a maximum of 50 KB. To provide a further speedup we could leverage parallel dolphin connections as described in Sec 3.4.

Conventionally, Users utilize web browsers like Google chrome, FireFox, etc to browse the internet. We developed a HTTPS proxy to provide a user friendly interface. The user would configure their browsers to connect to this proxy and the proxy would configure and use all the dolphin connections and sessions. This way a user can configure a normal browsing session to run over dolphin.

Chapter 9

Conclusion

The world has recently 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. 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. Overall, there is a compelling need to build systems that provide basic Internet access during shutdowns. Dolphin is a first attempt in this direction and we hope it will further propel development of more such sophisticated systems.

Bibliography

[1] "United nations general assembly, human rights council thirty-second session, third item," https://www.article19.org/data/files/Internet_Statement_Adopted.pdf.

[2] 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.

[3] 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.

[4] A. Filasto and J. Appelbaum, "Ooni: Open observatory of network interference." in *FOCI*, 2012.

[5] 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*, 2017, pp. 114–127.

[6] 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, pp. 252–264.

[7] 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.

[8] J. Beznazwy and A. Houmansadr, "How china detects and blocks shadowsocks," in *Proceedings of the ACM Internet Measurement Conference*, 2020, pp. 111–124.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] "Myanmar citizens not aware of covid-19 and human rights impact," https://tinyurl.com/yxuzab5q.

[16] "Internet shutdown in kazakastan amid unrest." https://blog.cloudflare.com/internet-shut-down-in-kazakhstan-amid-unrest/.

[17] "100 hours in the dark: How an election internet blackout hit poor ugandans, reuters, january 2021," https://tinyurl.com/y49d7shm.

[18] "Bulk sms country wise restrictions," https://www.sendmode.com/bulk-sms-country-restrictions-infographic.

[19] "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.

[20] S. Agarwal and S. De, "Rural broadband access via clustered collaborative communication," *IEEE/ACM Transactions on Networking*, vol. 26, no. 5, pp. 2160–2173, 2018.

[21] K. Chebrolu and B. Raman, "Fractel: a fresh perspective on (rural) mesh networks," in *Proceedings of the 2007 workshop on Networked systems for developing regions*, 2007, pp. 1–6.

[22] S. Kassing, D. Bhattacherjee, A. B. Águas, J. E. Saethre, and A. Singla, "Exploring the" internet from space" with hypatia," in *Proceedings of the ACM Internet Measurement Conference*, 2020, pp. 214–229.

[23] "Myanmar shuts down internet but allows cellular services to function, telenor, may 2020," https://tinyurl.com/up2ytfo.

[24] "Russian authorities 'secretly' shut down moscow's mobile internet: Report, forbes, august 2019," https://tinyurl.com/47pnarm2.

[25] "Internet services restricted in 13 districts of haryana, indai, ndtv, november 2017," https://tinyurl.com/y5646kz9.

[26] "Internet shutdown in response to mega public gathering in india, business world, october 2020," https://tinyurl.com/yxa9e32u.

[27] 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.

[28] 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.

[29] 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.

[30] B. Reaves, L. Blue, and P. Traynor, "Authloop: End-to-end cryptographic authentication for telephony over voice channels," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 963–978.

[31] "Government orders internet shutdown in sites close to delhi and the national capital region." https://www.nationalheraldindia.com/national/govt-orders-internet-shutdown-in-areas-close-to-farmers-protest-sites-in-delhi.

[32] "Twilio - communication API for sms, voice and video automation," https://www.twilio.com/.

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

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

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

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

[37] J. Salowey, A. Choudhury, and D. McGrew, "Aes galois counter mode (gcm) cipher suites for tls," *Request for Comments*, vol. 5288, 2008.

[38] J. Romkey, "Rfc1055: Nonstandard for transmission of ip datagrams over serial lines: Slip," 1988.

[39] S. Cheshire and M. Baker, "Consistent overhead byte stuffing," *IEEE/ACM Transactions on networking*, vol. 7, no. 2, pp. 159–172, 1999.

[40] I. ITU-T, "Recommendation g. 114," *One-Way Transmission Time, Standard G*, vol. 114, p. 84, 2003.

[41] 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.

[42] J. S. Sobolewski, "Cyclic redundancy check," in *Encyclopedia of Computer Science*, 2003, pp. 476–479.

[43] "Cyclic redundancy check," https://en.wikipedia.org/wiki/Cyclic_redundancy_check.

[44] "Ofono telephony management framework," https://01.org/ofono.

[45] "Python dbus library," https://pydbus.readthedocs.io/en/latest/legacydocs.

[46] M. Masahisa, "Frequency-shift-keying phase-modulation code transmission system," May 21 1968, uS Patent 3,384,822.

[47] K. Mostafa, "minimodem - general-purpose software audio fsk modem," http://www.whence.com/minimodem/.

[48] 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

[49] "Python crypto library," https://pycryptodome.readthedocs.io/en/latest/.

[50] O. Lev, "Cross-platform python cffi bindings for libsecp256k1," https://pypi.org/project/coincurve/.

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

[52] "Python smtp library," https://docs.python.org/3/library/smtplib.html.

[53] "Python wrapper for the twitter api," https://pypi.org/project/twython/.

[54] "A simple http rest api for searching and retrieving live articles from all over the web," https://newsapi.org/.

[55] "Textize api," https://www.textise.net/.

[56] 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.

[57] "Coverage data on cellular subscribers in india by telecom regulatory authority," https://www.trai.gov.in/sites/default/files/PR_No.45of2021_0.pdf.

[58] 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.

[59] A. W. Rix, M. P. Hollier, A. P. Hekstra, and J. G. Beerends, "Perceptual evaluation of speech quality (pesq) the new itu standard for end-to-end speech quality assessment part i–time-delay compensation," *Journal of the Audio Engineering Society*, vol. 50, no. 10, pp. 755–764, 2002.

[60] S. Frolov, J. Wampler, and E. Wustrow, "Detecting probe-resistant proxies." in *NDSS*, 2020.

[61] "Open speech and language resources repository," https://www.openslr.org/resources.php.

[62] T. Zhang and C.-C. J. Kuo, "Audio content analysis for online audiovisual data segmentation and classification," *IEEE Transactions on speech and audio processing*, vol. 9, no. 4, pp. 441–457, 2001.

[63] 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.

[64] A. Zarras, "Leveraging internet services to evade censorship," in *International Conference on Information Security*. Springer, 2016, pp. 253–270.

[65] K. Kohls, T. Holz, D. Kolossa, and C. Pöpper, "Skypeline: Robust hidden data transmission for voip," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, 2016, pp. 877–888.

[66] C. Connolly, P. Lincoln, I. Mason, and V. Yegneswaran, "$trist$: Circumventing censorship with transcoding-resistant image steganography," in *4th $USENIX$ Workshop on Free and Open Communications on the Internet ($FOCI$ 14)*, 2014.

[67] R. McPherson, A. Houmansadr, and V. Shmatikov, "Covertcast," *Proceedings on Privacy Enhancing Technologies*, vol. 3, pp. 1–14, 2016.

[68] D. Barradas, N. Santos, and L. E. Rodrigues, "Deltashaper: Enabling unobservable censorship-resistant tcp tunneling over videoconferencing streams." *Proc. Priv. Enhancing Technol.*, vol. 2017, no. 4, pp. 5–22, 2017.

[69] 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.

[70] 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.

[71] 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.

[72] 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.

[73] H. Kim, D. Kim, M. Kwon, H. Han, Y. Jang, D. Han, T. Kim, and Y. Kim, "Breaking and fixing volte: Exploiting hidden data channels and mis-implementations," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 328–339.

[74] C.-Y. Li, G.-H. Tu, C. Peng, Z. Yuan, Y. Li, S. Lu, and X. Wang, "Insecurity of voice solution volte in lte mobile networks," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 316–327.

[75] J. Karlin, D. Ellard, A. W. Jackson, C. E. Jones, G. Lauer, D. Mankins, and W. T. Strayer, "Decoy routing: Toward unblockable internet communication." in *FOCI*, 2011.

[76] 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.

[77] "Fast api web framework," https://fastapi.tiangolo.com/.