Dolphin: A Cellular Voice Based Internet Shutdown Resistance System

Piyush Kumar Sharma
IIIT Delhi, imec-COSIC KU Leuven
India, Belgium
piyushs@iiitd.ac.in

Rishi Sharma
IIIT Delhi
India
rishi17260@iiitd.ac.in

Kartikey Singh
IIIT Delhi
India
kartikey17242@iiitd.ac.in

Mukulika Maity
IIIT Delhi
India
mukulika@iiitd.ac.in

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

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 outside the 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).

KEYWORDS

Internet shutdown, cellular, censorship

1 INTRODUCTION

The original idea of the Internet was to provide a platform to facilitate free flow of information across the globe. This unhindered access to information has promulgated 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 [81]. But on the other hand, many censoring nations attempt to disrupt the free flow of information (as per convenience), opposing the original idea of the Internet. As a result, in the past decade, there has been an exponential rise in the events of Internet censorship globally [37, 63, 79]. This has led to an ongoing arms race between adversaries and free speech activists across the globe; adversaries continue to evolve various censorship techniques [28, 64, 69, 82, 83], whereas civil liberty activists counter them with wide range of novel circumvention systems [26, 29, 38, 40, 80].

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 [2]). 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 the Internet *e.g.,* access to news, reporting power failures and outages, sending and receiving important emails *etc.* The recent COVID-19 pandemic further exacerbates the impact—a large population of the globe has moved to working online, both for professional and personal tasks. Thus, the regions with such shutdowns have been adversely impacted in these trying times. *E.g.,* due to Internet shutdown in Myanmar, some rural areas were not even aware of the pandemic for many months [11]. What is even more alarming is that more and more countries are opting for Internet shutdowns. *E.g.,* the number of countries that 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 [2]. Considering that such trends are becoming common, it is plausible that more nation states opt for such measures [1, 77]. Thus, it becomes imperative to explore solutions using which people living in Internet shutdown regions could access basic Internet services like email, accessing news articles, tweets, *etc.*

Proceedings on Privacy Enhancing Technologies 2023(1), 589–607

https://doi.org/10.56553/popets-2023-0034

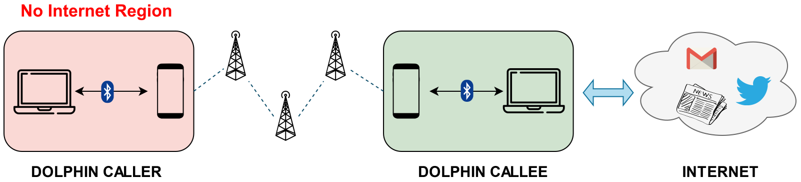

Figure 1: Overview of Dolphin's architecture.

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 regions, over regular voice calls. However, this 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 [3, 21]. Further, approaches like setting up separate ad-hoc networks [23, 31], using low Earth orbit satellites [47] and satellite phones have merit but may encounter infrastructural and deployment challenges.

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 [8, 9, 12, 19].

Novel use of cellular voice channel: The voice channel is by design not suitable 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 [25, 52, 67], 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 inherent challenges.

Dolphin overview: Dolphin user requires running a Dolphin client utility on its host, while also requiring a peer (*e.g.*, a friend) outside the 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 into 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 thereafter demodulates (and decrypts) 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.

Does Dolphin emulate dial-up modems? At a first glance, Dolphin seems similar to legacy dial-up modems, leading to believe that such 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 speech, rendering the channel unsuitable for transmitting data using legacy modems [70].

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 this condition. However, most of the Internet applications are built with reliability in consideration. Thus, in order to run these applications with 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.

It must be noted that Dolphin has a modular design that 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.

Additionally, we tested Dolphin during a real shutdown event [76] and confirmed that 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). Further, we also provide a way for users to access the 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 [22]) 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).

2 INTERNET SHUTDOWNS 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) [2].

Various projects keep track of these shutdowns a country as well as on a global scale. *E.g.,* the `accessnow` project [2] categorically reports incidents of shutdowns occurring across the globe, presenting detailed statistics of such events. Further, there are country-specific projects such as [10] which maintain a record of all the shutdowns that happen in India (a country with the highest number of shutdowns). Some projects even attempt to estimate the economic losses inflicted due to Internet shutdowns *e.g.,* `internetsociety` [14].

Other projects attempt to identify Internet outages in general. *E.g.,* IODA [6] 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) [7] 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, none provide solutions to circumvent them. Thus, we present Dolphin, a novel system that 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).

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) 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.

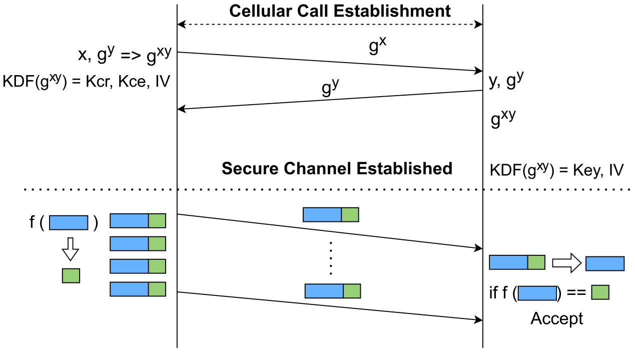

Figure 2: Dolphin's secure channel and data transmission phases. f() computes HMAC tag (green) and f_v() verifies it.

Once a cellular call is established, Dolphin then operates in two phases. The 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 the actual transmission of data (refer to Dolphin's overall design in Fig. 2). The details of these two phases are as follows:

Secure channel establishment phase:

(1) In this phase, the caller and callee establish a shared secret to encrypt the data bits, for which they rely on a Diffie-Hellman (DH) key exchange.
(2) The caller's client utility first selects a DH private part x, and derives the shared secret g^{xy}, using the already known g^{y} of the callee. Then the encryption/decryption key (K_{cr}, K_{ce}), and the initializing vector (IV) are derived from the shared secret using a key derivation function (KDF) by the caller. We use AES-128 in GCM mode (an AEAD cipher [74]) 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).

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 a performance standpoint as the 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. Using error detection/correction techniques is not suitable either as discussed in Appendix. A.2.2.

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 localize the impact of data corruptions 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 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 acknowledgments (or other control messages) at a relatively slower rate of 16 bps. The control information is sent at a lower 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 long, transmitting them at low rates does not hamper the overall performance.

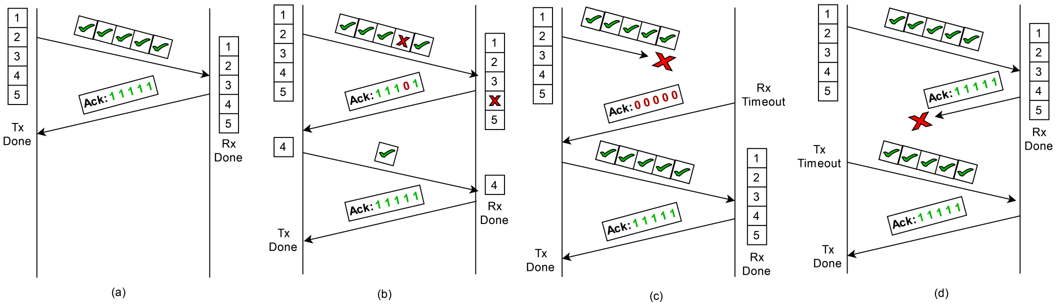

Figure 3: Some representative scenarios that are handled by Dolphin's reliability protocol: (a) represents the best case where no data is corrupted/lost, (b) depicts the case where one (or more) chunks are corrupted/lost, (c) is the case where a complete batch of chunks is corrupted/lost, and in (d) the acknowledgement(s) are corrupted/lost. All other scenarios that exist are the variation of these base cases and are thus handled by our reliability protocol.

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. 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 (thus 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 and the overall working of Dolphin along with how the different components interact is depicted in Fig. 28.

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* [73]. This technique allows us to use a character (say *e.g.,* the null character), 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) [32] 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, implying that a batch can have at most 256 chunks. Dolphin can be configured to transmit more chunks per batch, by reserving multiple (sequence number) bytes per chunk. Selecting a single byte for sequence number minimizes the overhead.

Each chunk within every batch is assigned a relative sequence number, *i.e.,* the first chunk of every batch has sequence number 1, the second has 2 and so on.

Acknowledgements: Acknowledgements identify the correctly and incorrectly received (or lost) chunks. Each chunk corresponding to its seq. no. is assigned either a bit 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 relative 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) ÷ 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 [43] 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 compress the data before encrypting it, as textual data can be compressed with a higher compression ratio, as compared to cipher-text [50]. 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) [78]. 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 [4], 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.

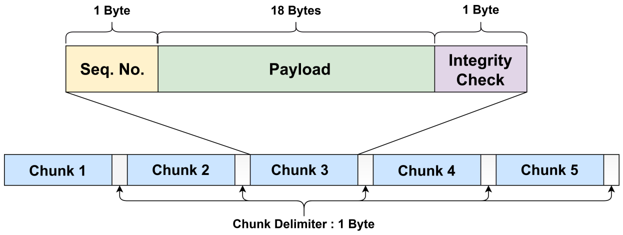

Figure 4: Details about individual chunks and how they are stacked before sending.

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. 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 Modes of Operation

We now enlist the two operating modes of Dolphin:

1.) Human callee mode: This mode requires the user in Internet shutdown region to find a trusted peer (or friend) in a region with uninterrupted Internet connectivity. The Dolphin user (caller) would then request this peer to setup the Dolphin callee infrastructure, for accessing Internet applications (such as Twitter, email *etc.*). This is similar to users running circumvention systems in non-censoring countries, to support those living under repressive regimes. However, this model is not always conducive — what if one cannot find peers in non-shutdown regions? To answer this question, we introduce the second mode of operation.

2.) 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 [22]). 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 fees.

4 IMPLEMENTATION DETAILS

In this subsection we describe how we implemented different components of Dolphin. [1]

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). 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 its 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 the `ofono` framework [13] for call automation as it helps manage various calling features via Bluetooth – dialing and disconnecting, tracking call related events (call established/missed etc.). `Ofono` is accessed using a `dbus` interface (using `pydbus` [17] 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) [58] 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` [60], 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. We include details about how Dolphin's API can be used to access Internet applications and websites in Appendix. A.2.1.

Establishing secure channel: We aim to establish a shared secret between the caller and the callee using DH key exchange. Traditionally DH uses 128 byte public 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 [41]. 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` [15] and `coincurve` [54] 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 outside the 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 [56] could be used, *e.g.,* rely on multiple parties to derive the password, with no one party having complete information.

Running Internet applications: In Dolphin the callee accesses Internet 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` [16], and Twitter using `twython` [18] library. The news application is automated using `newsapi` [20], which returns concise news snippets based on a keyword query.

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.[2] The details of the implementation can be referred to in Appendix A.3.

[1] The code of the proof-of-concept implementation is publicly available [36].

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

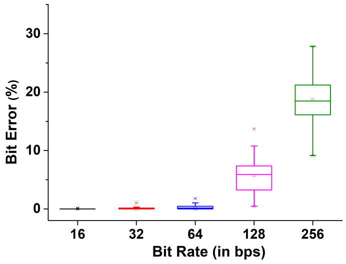

Figure 5: Bit error rate variation for different bit rates for 100B transfer.

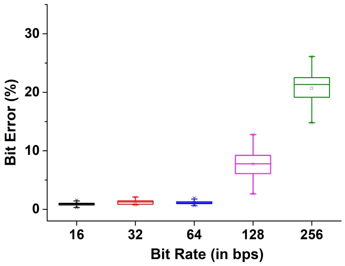

Figure 6: Bit error rate variation for different bit rates for 1000B transfer.

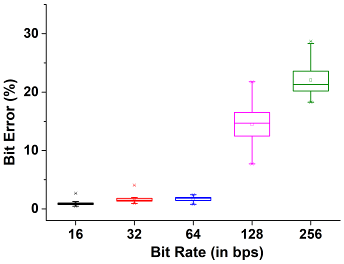

Figure 7: Bit error rate variation for different bit rates for 5000B transfer.

5 DATA COLLECTION AND RESULTS

We now present the details of various experiments performed for evaluating Dolphin along with their corresponding results. Broadly, we divided our experiments into two categories. The first set of experiments are devised to test the viability of sending data at different bit rates over the cellular voice channel. The 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 Achievable Encoding Rates in Dolphin

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 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 [71] 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. 1 and present the complete error distributions in Fig. 5, Fig. 6 and Fig. 7 for 100, 1000 and 5000 bytes respectively. It is evident from the table and the graphs that up to 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 errors, 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.

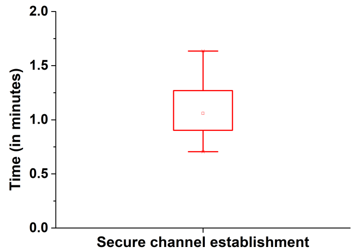

Figure 8: Dolphin's secure channel establishment time.

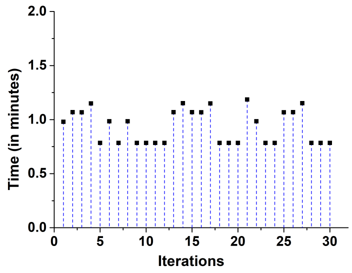

Figure 9: Time taken to tweet 280 characters (max. limit) using Dolphin.

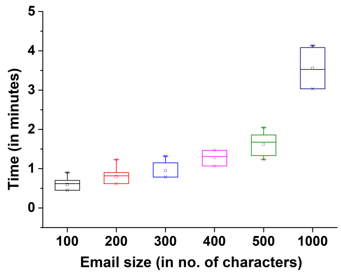

Figure 10: Time taken to send an email of varying sizes using Dolphin.

Varying cellular connectivity: Notably, the above set of experiments assumes the caller to be in a shutdown region, and the callee to be outside. The caller phone was manually switched to use only 2G voice (representing bare minimum cellular connectivity) and the callee's phone was enabled with 4G voice connectivity. However, peers may not always have such connectivity due to various reasons (such as intermittent signal, 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 A.1 for details), indicating similar performance for different connectivity scenarios. Thus we continued using 64 bps as our default data transmission 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 on four popular providers which serve majority ($\approx$ 90%) of the users in their region [30]. 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 close proximity (in the same building) and may essentially be connected to the same cellular tower. One can argue that the results may vary if the geographical distance between the caller and callee is increased as it would involve data to travel over multiple cellular towers. Thus, we repeated the experiments with the caller and callee in different cities ($\approx$ 1100 miles apart) within the same country, as well as different countries (one in Asia and other in Europe, being$\approx$ 3600 miles apart). We observed similar BER for downloading 100 to 500 byte files at different bit rates (16, 32,..., 256 bps), with less than 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 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. 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 contains the errors introduced by both Bluetooth as well as the cellular channel. For all data rates (up to 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 at least 1%, max at about 20% for data rate of 256 bps). This confirms that Bluetooth did not contribute to errors in audio, but that such errors were introduced only by the cellular voice channel.

On Dolphin's observed data rates: Since Dolphin achieves low data encoding rates, we explored the possibility of improving it using existing modulation techniques that claim to achieve higher rates. Note that only a handful of such techniques have been developed to be resistant to distortions and processing artefacts. One such modulation scheme was proposed in Hermes [34]. It was developed in 2010 and claimed to achieve close to 1 Kbps encoding rate with low bit errors (< 1%). However, the unavailability of Hermes' source code, along with a few missing implementation details of the design of the complex demodulator, made it difficult for us to implement it. Another such technique, *Authloop* [70] developed in 2016, also tried re-implementing Hermes' demodulator but failed to do so. Thus, Authloop developed and implemented its own modulation scheme based on the ideas proposed by Hermes. Their modem achieved about 500 bps encoding rate in a simulation environment. Thus, we used Authloop's modem's code to encode and send data over the real cellular channel. However, the transferred data suffered about **55%** bit error rates. To sum up, encoding data using schemes that use sophisticated channel coding techniques do not seem to compensate for the losses experienced over cellular voice channels. Thus, we currently employ Dolphin's FSK-based scheme that achieves moderate data rates with low bit errors and is practical for tasks with low-performance requirements — email, news feeds and tweets. Most importantly, Dolphin's modular design allows for easy adoption of encoding schemes in the future that may provide higher data rates while assuring low errors.

We further explored the reason behind the poor performance of previous modulation techniques. We found out that these studies evaluated only one of the many available modes of the popular AMR cellular codec [44]. We recreated their setup in simulation using that specific mode of AMR and observed that these techniques achieve low bit errors at a high data rates. However, in other AMR modes, their scheme faces large errors. Notably, Dolphin's modulation also performed on par, or better, in the same simulation setup when tested against the AMR codec mode used by previous studies, offering 1024 bps data encoding rate with about 1% error.

We believe that the current cellular channels (that we came across in all our evaluations) use modes of AMR that lead to large bit errors and hence make it difficult to obtain high data encoding rates for Hermes, Authloop, and Dolphin alike (ref. Appendix. A.2.5 for further details).

Moreover, we discuss the possibility of using VoLTE to achieve higher encoding rates (in Appendix. A.2.3) and also highlight the differences in using VoIP and cellular channels with respect to the ease (or difficulty) in encoding data in Appendix A.2.7.

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. 8, we observe, on 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 a secure channel every time it sends or accesses some content. However, in case the user wishes to reduce the overall latency, the Dolphin caller can be configured to establish a key once and use it for all data transfers for a longer duration *e.g.,* a day.

Next, we measured the time taken by Dolphin to access (or send) content using different Internet applications. We tested three applications *viz.,* email, Twitter, and news. For each application, we measured the time taken by the caller to send (or receive) the complete data *reliably.* First, we tested the time taken to tweet a 280 character message (maximum size for a single tweet). We repeated this experiment 30 times and observed that on average it took under a minute to tweet this message (ref. Fig. 9). Similarly, we sent emails of varying sizes (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. 10. It is evident that it takes ≈ 1.7 minutes (102 seconds) to send an email of 500 characters. Lastly, it took on an average 2 minutes to retrieve 10 concise news snippets (around 60 characters each).

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.

5.3 Automated Callee Mode Performance

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. 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 the feasibility of using lightweight and delay tolerant Internet applications via Dolphin, in shutdown regions with transmission times in the range of a few minutes. We also present an analysis of the call costs incurred on Dolphin users in Appendix. A.2.8.

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

5.4 Anecdotes

While conducting the experiments, we observed an Internet shutdown in the region of one of the authors (Delhi, India) [76]. 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.

6 SECURITY ASPECTS OF DOLPHIN

We start 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 for telecom operators to perform real-time traffic analysis on ongoing calls. However, 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 channels to actively detect Dolphin. Still, 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.,* the 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 (for details ref. Appendix A.2.6). This is already observed in multiple recent Internet shutdowns [8, 9, 12, 19].

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 usual disturbance encountered while conversing. This attack may turn out to be very powerful because the adversary could completely block Dolphin without even having to detect if it 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 call.

Case I: We start by exploring how the adversary can use the first method 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 retransmission of data between the Dolphin peers. To do so, the adversary would need to induce perturbations at very small intervals. *E.g.,* the adversary may need to introduce perturbations every 2.5 s to corrupt each 20 byte chunk transmitted at 64 bps. But, in practice, this attack could render cellular voice unusable for regular callers due to the unpleasant periodic disturbance (after every 2.5 s) throughout the call.

To quantitatively verify this, we conducted experiments to determine how periodic disturbances affect perceivable voice quality. To measure voice quality, we used PESQ (Perceptual Evaluation of Speech Quality) [72], 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 above 3 is considered good whereas a score less than 3 is not considered ideal. Moreover, a score below 2 is considered 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 acknowledgments instead of the chunks. This way the adversary would require to drop samples after every ≈12.5s, since in Dolphin's default configuration, we transmit five chunks before transmitting an acknowledgment. Moreover, we calculated the PESQ score for this scenario (*i.e.,* disruptions after every 12.5s) and achieved a score of 3.6, demonstrating that such a disruption will lead to the complete disruption of Dolphin without severely impacting the perceptual quality of normal calls. But, as a countermeasure to this attack, we can slightly alter Dolphin's default configuration by soliciting acknowledgments 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 that 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, making it unsuitable for regular calls. However, the error percentage of Dolphin is still not affected much and is 2.1%.[3] Thus, it would prove to be a futile exercise for the adversary to disrupt Dolphin with continuous noise as well.

6.3 Active Probing Attack

The aim of this attack is to enumerate possible Dolphin callee numbers and eventually drop all calls made to them. To do so, the adversary can itself pretend to be 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 the required data encrypted with this key, or the provided data is incorrect, the callee program just plays an audio containing the traditional

[3] 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.

"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 [39]. 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.

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. To this end, 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, the suspicious audio may be noisy and probably contain arbitrary data due to a lossy channel and multiple retransmissions, significantly decreasing the chance of successfully detecting the callee. Even in the unlikely scenario where the adversary obtains 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 Sec. 6.3).

6.5 Offline Analysis

Overall, it is challenging to build a completely undetectable anticensorship system. A determined and capable enough adversary can eventually detect it, and Dolphin is no exception. However, we still provide a thorough analysis below, demonstrating how we can make the task of detection non-trivial for the adversary. 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 straightforward approach that the adversary can adopt 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, unique to Dolphin, could lead to its detection. We remove this detectable feature by XOR-ing the CRC values with extra random bytes derived from KDF while establishing the secure key. Since both peers provide the same input to the KDF, they can derive the same bytes to invert the XOR.

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

To distinguish, the adversary can analyze the time domain waveforms of Dolphin encoded audio, and thereafter compare them with standard human audio. As depicted in Fig. 11, one can visually differentiate Dolphin audio from standard human audio. This is because in Dolphin, the signal's amplitude and frequency have low variation compared 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 the signal across multiple time intervals. 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 amplitude for normal human audio extracted from a large speech database [65] was 205 and 1590, respectively, which is about an order of magnitude higher.

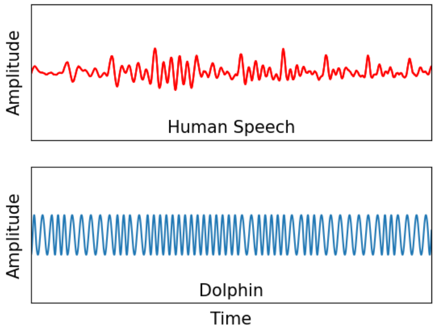

Figure 11: Signal waveform of 50 ms for normal human speech audio and Dolphin encoded audio.

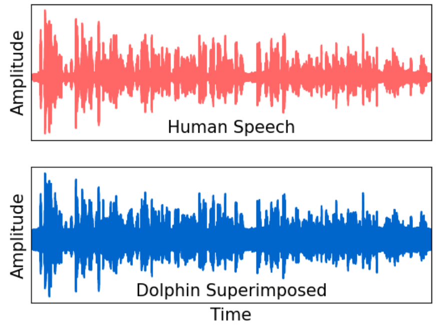

Figure 12: Signal waveform of 70s for normal human speech audio and Dolphin encoded audio superimposed over normal human speech.

As a countermeasure for such an analysis, we need to transform the Dolphin encoded audio to resemble normal human speech so that the adversary cannot distinguish both kinds of audio signals. 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 to make detection harder. However, the suppression should allow Dolphin's data to 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. 12, that Dolphin superimposed audio looks almost identical to the normal human audio even when analyzed for a long duration. 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 standard human audio on which Dolphin was superimposed. We performed this experiment for varying sets of human speech (150 samples) collected from multiple sources, including sample audios from speech database [65] and online sources such as audio extracted from YouTube.

Alternatively, the adversary can analyze the audio signal in the frequency domain to find distinguishing features between regular human speech and superimposed Dolphin audio. We employed the standard discrete Fourier transform technique to convert the waveform into the frequency domain and analyzed the audio in this transformed domain. Superimposed Dolphin audio frequency response was very similar to that of normal human speech extracted from a standard speech database [65] both visually and statistically. The mean and standard deviation of the frequency response for Dolphin superimposed audio was found to be - 55.85 dB and 11.00 dB, respectively. In comparison, we observed a mean and standard deviation of - 56.25 dB and 10.46 dB for normal human audio, respectively. Thus, we believe, it would be non-trivial for an adversary to distinguish superimposed Dolphin from 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, the 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 [85], 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 instances 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. Thus, overall, a complete defence of Dolphin would include superimposition and silence introduction. These defences will further impact the performance, albeit making detection non-trivial. The performance degradation due to superimposition may be minimized by applying adequate suppression. Moreover, the degradation due to silence intervals depends on the cover audio. On testing about 13K audio samples in the speech database [65], we find that on average human audio contains 31% silence. Thus, one should expect to observe similar overheads on average.

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.

6.6 Active Targeted Disruption Attacks

Here we consider the possibilities of an adversary performing active attacks to disrupt Dolphin usage. The adversary can attempt to perform such analysis on some suspicious calls, which may be challenging to identify if they are superimposed using the normal human voice. Thus, the adversary may try to perturb all *suspicious* calls to corrupt data or acknowledgment chunks. The impact of such efforts would be similar to what was described earlier in Sec. 6.2.

Specifically, the adversary could attempt to actively inject forged messages and/or acknowledgments. Such modifications would eventually be detected and recovered due to the integrity mechanisms (CRC for chunks + HMAC on complete data) and reliability protocol. But if the adversary aims to disrupt the Dolphin service consistently, it would need to forge/modify *all* acknowledgments (making the cellular channel practically unusable). However, disrupting calls merely based on suspicion could have considerable collateral damages if such calls are benign (as discussed in Sec. 6.2).

7 RELEVANT WORK

In the traditional censorship circumvention literature various systems have been proposed to send data using the underlying VoIP or video channel over the Internet [27, 33, 42, 51, 59, 84]. However, the cellular voice channel is different in the usage of codecs and channel transmission constraints (ref. Appendix. A.2.7) and thus has received separate attention with several prior research efforts exploring the feasibility of sending data bits over the cellular voice channel [24, 25, 34, 35, 48, 52, 66, 67]. However, most of them are simulation studies that attempt to provide theoretical bounds for sending data over voice, propose new modulation schemes, or focus on a specific codec *etc* (more details in Appendix. A.2.9). 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.

8 CONCLUDING REMARKS

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 within a shutdown region by relying on trusted peers outside such regions who 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,s and news. Across all our experiments for these applications we observed that it takes only a few minutes to successfully use 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 propel development of more sophisticated systems.

ACKNOWLEDGMENTS

We would like to thank our shepherd Diogo Barradas and other anonymous reviewers for their valuable feedback. This research is partially supported by the Research Council KU Leuven under the grant C24/18/049, by CyberSecurity Research Flanders with reference number VR20192203, and by DARPA FA8750-19-C-0502. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any of the funding agencies.

REFERENCES

[1] 100 hours in the dark: How an election internet blackout hit poor Ugandans, Reuters, January 2021. https://tinyurl.com/y49d7shm.

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

[3] Bulk SMS country wise restrictions. https://www.sendmode.com/bulk-sms-country-restrictions-infographic.

[4] Cyclic Redundancy Check. https://en.wikipedia.org/wiki/Cyclic_redundancy_check.

[5] Fast API web framework. https://fastapi.tiangolo.com/.

[6] Internet Outage Detection and Analysis (IODA) Project. https://ioda.caida.org/.

[7] Internet Outages Map by Cisco. https://www.thousandeyes.com/outages/.

[8] Internet services restricted in 13 districts of Haryana, India, NDTV, November 2017. https://tinyurl.com/y5646kz9.

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

[10] Internet shutdown tracker in India. https://internetshutdowns.in.

[11] Myanmar citizens not aware of COVID-19 and its human rights impact. https://tinyurl.com/yxuzab5q.

[12] Myanmar shuts down Internet but allows cellular services to function, Telenor, may 2020. https://tinyurl.com/up2ytfo.

[13] Ofono telephony management framework. https://01.org/ofono.

[14] Policy brief: Internet shutdowns, Internet Society, December 2019. https://www.internetsociety.org/policybriefs/internet-shutdowns.

[15] Python Crypto Library. https://pycryptodome.readthedocs.io/en/latest/.

[16] Python SMTP Library. https://docs.python.org/3/library/smtplib.html.

[17] Python dbus Library. https://pydbus.readthedocs.io/en/latest/legacydocs.

[18] Python wrapper for the Twitter API. https://pypi.org/project/twython/.

[19] Russian Authorities "Secretly" Shutdown Moscow's Mobile Internet: Report, Forbes, August 2019. https://tinyurl.com/47pnarm2.

[20] A simple HTTP REST API for searching and retrieving live articles from all over the web. https://newsapi.org/.

[21] Trai extends the 100 SMS per day per SIM limit to 200 SMSs per day per SIM. https://www.trai.gov.in/sites/default/files/press_release_for_8th_amendmenet.pdf.

[22] Twilio - communication API for SMS, voice and video automation. https://www.twilio.com/.

[23] Agarwal, S., and De, S. Rural broadband access via clustered collaborative communication. *IEEE/ACM Transactions on Networking (ToN) 26*, 5 (2018), 2160–2173.

[24] Ahmad, T., Reed-Sanchez, E., Zarinni, F., Afutu, A., Adjaho, K., Nyarko, Y., and Subramanian, L. Greenapps: A platform for cellular edge applications. In *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies (COMPASS)* (2018), pp. 1–5.

[25] Ali, B. T., Baudoin, G., and Venard, O. Data transmission over mobile voice channel based on m-fsk modulation. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)* (2013), IEEE, pp. 4416–4421.

[26] Barradas, D., Santos, N., Rodrigues, L., and Nunes, V. Poking a hole in the wall: Efficient censorship-resistant internet communications by parasitizing on webrtc. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2020), pp. 35–48.

[27] Barradas, D., Santos, N., and Rodrigues, L. E. Deltashaper: Enabling unobservable censorship-resistant tcp tunneling over videoconferencing streams. *Proceedings of Privacy Enhancing Technologies' Symposium (PoPETS) 2017*, 4 (2017), 5–22.

[28] Beznazwy, J., and Houmansadr, A. How China Detects and Blocks Shadowsocks. In *Proceedings of the ACM Internet Measurement Conference (IMC)* (2020), pp. 111–124.

[29] Bock, K., Hughey, G., Merino, L.-H., Arya, T., Liscinsky, D., Pogosian, R., and Levin, D. Come as you are: Helping unmodified clients bypass censorship with server-side evasion. In *Proceedings of Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)* (2020), pp. 586–598.

[30] Coverage data on cellular subscribers in India by Telecom Regulatory Authority. https://www.trai.gov.in/sites/default/files/PR_No.45of2021_0.pdf.

[31] Chebrolu, K., and Raman, B. FRACTEL: A fresh perspective on (rural) mesh networks. In *Proceedings of the Workshop on Networked Systems for Developing Regions* (2007), pp. 1–6.

[32] Cheshire, S., and Baker, M. Consistent overhead byte stuffing. *IEEE/ACM Transactions on Networking (ToN) 7*, 2 (1999), 159–172.

[33] Connolly, C., Lincoln, P., Mason, I., and Yegneswaran, V. TRIST: Circumventing Censorship with Transcoding-Resistant Image Steganography. In *Proceedings of Workshop on Free and Open Communications on the Internet (FOCI)* (2014).

[34] Dhananjay, A., Sharma, A., Paik, M., Chen, J., Kuppusamy, T. K., Li, J., and Subramanian, L. Hermes: Data transmission over unknown voice channels. In *Proceedings of the ACM International Conference on Mobile Computing and Networking (MobiCom)* (2010), pp. 113–124.

[35] Dogar, F. R., Qazi, I. A., Tariq, A. R., Murtaza, G., Ahmad, A., and Stocking, N. Missit: Using missed calls for free, extremely low bit-rate communication in developing regions. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI)* (2020), pp. 1–12.

[36] Dolphin code. https://github.com/pi-yush/Dolphin-code.

[37] Filasto, A., and Appelbaum, J. OONI: Open Observatory of Network Interference. In *Proceedings of the Workship on Free and Open Communication over the InternetFOCI* (2012).

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

[39] Frolov, S., Wampler, J., and Wustrow, E. Detecting probe-resistant proxies. In *Proceeding of the Symposium on Networks and Distributed Systems Security (NDSS)* (2020).

[40] Frolov, S., and Wustrow, E. $httpt$: A probe-resistant proxy. In *Proceedings of Workshop on Free and Open Communications on the Internet* ($FOCI$) (2020).

[41] Gura, N., Patel, A., Wander, A., Eberle, H., and Shantz, S. C. *Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 119–132.

[42] Houmansadr, A., Riedl, T. J., Borisov, N., and Singer, A. C. I want my voice to be heard: IP over Voice-over-IP for unobservable censorship circumvention. In *Proceedings of the Symposium on Networks and Distributed Systems Security (NDSS)* (2013).

[43] ITU-T, I. Recommendation g. 114. *One-Way Transmission Time, Standard G 114* (2003), 84.

[44] Järvinen, K. Standardisation of the adaptive multi-rate codec. In *2000 10th European Signal Processing Conference* (2000), IEEE, pp. 1–4.

[45] Jio cellular provider international calling rates, india. https://www.jio.com/en-in/faq/isd-plan/what-are-the-charges-for-international-isd-or-ild-calls.html#/.

[46] Karlin, J., Ellard, D., Jackson, A. W., Jones, C. E., Lauer, G., Mankins, D., and Strayer, W. T. Decoy routing: Toward unblockable internet communication. In *Proceedings of the Workshop on Free Open Communication over the Internet (FOCI)* (2011).

[47] Kassing, S., Bhattacherjee, D., Águas, A. B., Saethre, J. E., and Singla, A. Exploring the "Internet from space" with Hypatia. In *Proceedings of the ACM Internet Measurement Conference* (2020), pp. 214–229.

[48] Kazemi, R., Boloursaz, M., Etemadi, S. M., and Behnia, F. Capacity bounds and detection schemes for data over voice. *IEEE Transactions on Vehicular Technology 65*, 11 (2016), 8964–8977.

[49] Kim, H., Kim, D., Kwon, M., Han, H., Jang, Y., Han, D., Kim, T., and Kim, Y. Breaking and Fixing VoLTE: Exploiting hidden data channels and mis-implementations. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2015), pp. 328–339.

[50] Klinc, D., Hazay, C., Jagmohan, A., Krawczyk, H., and Rabin, T. On compression of data encrypted with block ciphers. *IEEE transactions on information theory 58*, 11 (2012), 6989–7001.

[51] Kohls, K., Holz, T., Kolossa, D., and Pöpper, C. Skypeline: Robust hidden data transmission for voip. In *Proceedings of the ACM Asia Conference on Computer and Communications Security (AsiaCCS)* (2016), pp. 877–888.

[52] LaDue, C. K., Sapozhnykov, V. V., and Fienberg, K. S. A data modem for GSM voice channel. *IEEE Transactions on Vehicular Technology 57*, 4 (2008), 2205–2218.

[53] Lerner, A., Fanti, G., Ben-David, Y., Garcia, J., Schmitt, P., and Raghavan, B. Rangzen: Anonymously getting the word out in a blackout. *arXiv preprint arXiv:1612.03371* (2016).

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

[55] Li, C.-Y., Tu, G.-H., Peng, C., Yuan, Z., Li, Y., Lu, S., and Wang, X. Insecurity of voice solution VoLTE in LTE mobile networks. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2015), pp. 316–327.

[56] Li, S., and Hopper, N. Mailet: Instant social networking under censorship. *Proceedings on Privacy Enhancing Technologies 2016*, 2 (2016), 175–192.

[57] Liu, Y., Bild, D. R., Adrian, D., Singh, G., Dick, R. P., Wallach, D. S., and Mao, Z. M. Performance and energy consumption analysis of a delay-tolerant network for censorship-resistant communication. In *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing* (2015), pp. 257–266.

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

[59] McPherson, R., Houmansadr, A., and Shmatikov, V. Covertcast. *Proceedings on Privacy Enhancing Technologies' Symposium (PoPETS) 3* (2016), 1–14.

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

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

[62] Nasr, M., Zolfaghari, H., Houmansadr, A., and Ghafari, A. Massbrowser: Unblocking the censored web for the masses, by the masses. In *Proceedings of Symposium on Networks and Distributed Systems Security (NDSS)* (2020).

[63] Niaki, A. A., Cho, S., Weinberg, Z., Hoang, N. P., Razaghpanah, A., Christin, N., and Gill, P. IClab: a global, longitudinal internet censorship measurement platform. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)* (2020), IEEE, pp. 135–151.

[64] Niaki, A. A., Hoang, N. P., Gill, P., Houmansadr, A., 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).

[65] Open Speech and Language Resources repository. https://www.openslr.org/resources.php.

[66] Özkan, M. A., and Örs, S. B. Data transmission via GSM voice channel for end to end security. In *2015 IEEE 5th International Conference on Consumer Electronics-Berlin (ICCE-Berlin)* (2015), IEEE, pp. 378–382.

[67] Perić, M., Milićević, P., Banjac, Z., and Todorović, B. M. 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)* (2015), IEEE, pp. 239–242.

[68] Pradeep, A., Javaid, H., Williams, R., Rault, A., Choffnes, D., Le Blond, S., and Ford, B. Moby: A Blackout-Resistant Anonymity Network for Mobile Devices. *Proceedings on Privacy Enhancing Technologies' Symposium (PoPETS) 3* (2022), 247–267.

[69] Ramesh, R., Raman, R. S., Bernhard, M., Ongkowijaya, V., Evdokimov, L., Edmundson, A., Sprecher, S., Ikram, M., and Ensafi, R. Decentralized control: A case study of Russia. In *Proceedings of Network and Distributed Systems Security (NDSS) Symposium 2020* (2020).

[70] Reaves, B., Blue, L., and Traynor, P. Authloop: End-to-end cryptographic authentication for telephony over voice channels. In *Proceedings of the USENIX Security Symposium (USENIX Security '16)* (2016), pp. 963–978.

[71] Ristad, E. S., and Yianilos, P. N. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20*, 5 (1998), 522–532.

[72] Rix, A. W., Hollier, M. P., Hekstra, A. P., and Beerends, J. G. 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 50*, 10 (2002), 755–764.

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

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

[75] Shadbolt, P. Firechat in hong kong: How an app tapped its way into the protests. *CNN, October 16* (2014).

[76] 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.

[77] Internet shutdown in Kazakastan amid unrest. https://blog.cloudflare.com/internet-shut-down-in-kazakhstan-amid-unrest/.

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

[79] Sundara Raman, R., Shenoy, P., Kohls, K., and Ensafi, R. 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.

[80] Tschantz, M. C., Afroz, S., Paxson, V., et al. SoK: Towards grounding censorship circumvention in empiricism. In *Proceedings of IEEE Symposium on Security and Privacy (S&P)* (2016), IEEE, pp. 914–933.

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

[82] Wang, Z., Cao, Y., Qian, Z., Song, C., and Krishnamurthy, S. V. 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.

[83] Yadav, T. K., Sinha, A., Gosain, D., Sharma, P. K., and Chakravarty, S. Where the light gets in: Analyzing web censorship mechanisms in India. In *Proceedings of the Internet Measurement Conference 2018* (2018), pp. 252–264.

[84] Zarras, A. Leveraging Internet services to evade censorship. In *International Conference on Information Security (ICIS)* (2016), Springer, pp. 253–270.

[85] Zhang, T., and Kuo, C.-C. J. Audio content analysis for online audiovisual data segmentation and classification. *IEEE Transactions on speech and audio processing 9*, 4 (2001), 441–457.

A APPENDIX

A.1 Results for Varied Cellular Connectivity

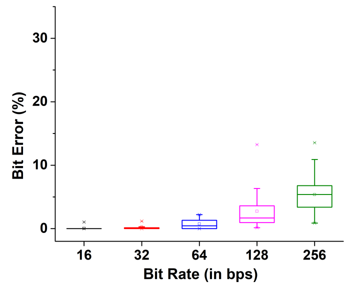

Figure 13: 4G-4G: Bit error rate variation for different bit rate for 100B content transfer.

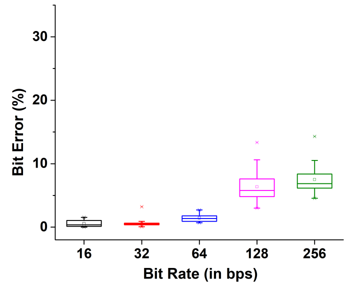

Figure 14: 4G-4G: Bit error rate variation for different bit rate for 1KB transfer.

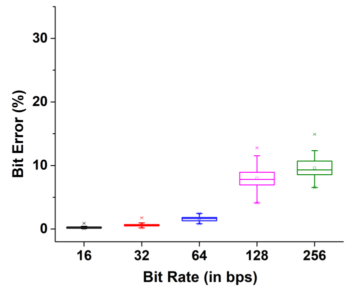

Figure 15: 4G-4G: Bit error rate variation for different bit rate for 5KB transfer.

In order to gauge the performance of Dolphin with varying cellular connectivity, we conducted experiments with different connectivity scenarios. Overall there are six possible combinations of caller and callee for 2G, 3G, and 4G voice connectivity *i.e.,* 4G-4G, 3G-4G, 2G-4G, 3G-3G, 2G-3G and 2G-2G. For each combination,

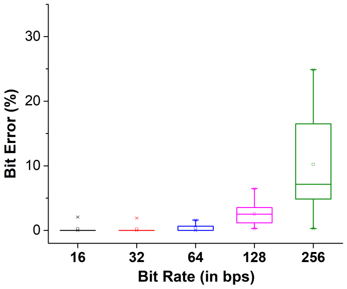

Figure 16: 3G-4G: Bit error rate variation for different bit rate for 100B transfer.

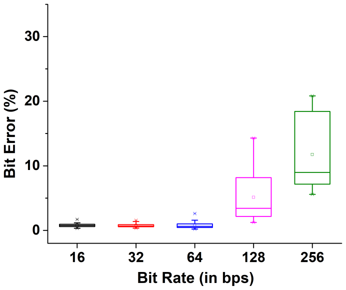

Figure 17: 3G-4G: Bit error rate variation for different bit rate for 1KB transfer.

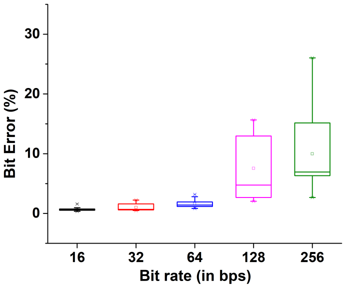

Figure 18: 3G-4G: Bit error rate variation for different bit rate for 5KB transfer.

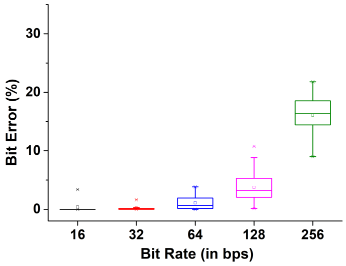

Figure 19: 3G-3G: Bit error rate variation for different bit rate for 100B transfer.

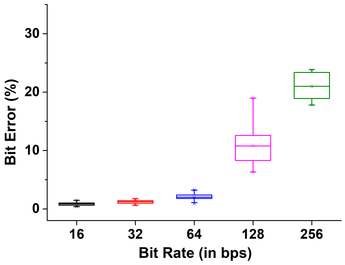

Figure 20: 3G-3G: Bit error rate variation for different bit rate for 1KB transfer.

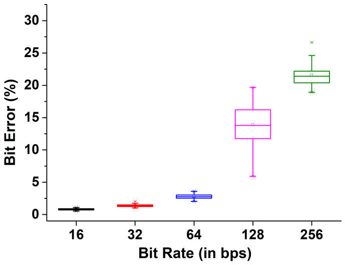

Figure 21: 3G-3G: Bit error rate variation for different bit rate for 5KB transfer.

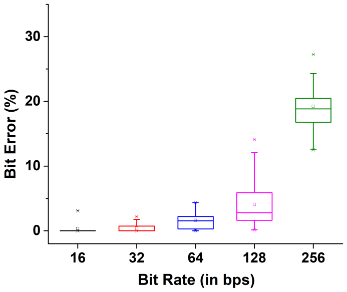

Figure 22: 2G-3G: Bit error rate variation for different bit rate for 100B transfer.

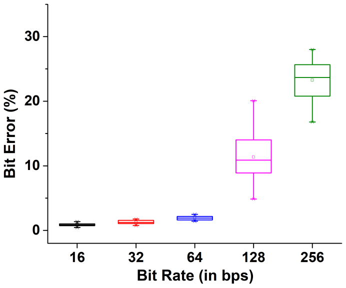

Figure 23: 2G-3G: Bit error rate variation for different bit rate for 1KB transfer.

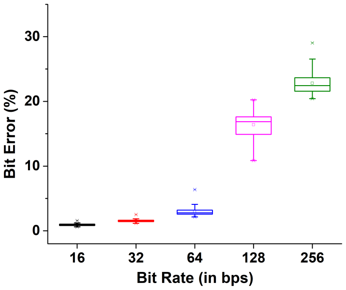

Figure 24: 2G-3G: Bit error rate variation for different bit rate for 5KB transfer.

we transferred data of different sizes (100, 1000 and 5000 bytes) at different bit rates (16,32,...,256), and record the BER. We have already depicted the results of 2G-4G setting. Thus, here we present the results of remaining scenarios in Fig. 13 to Fig. 27.

It is evident that for all scenarios, we obtain a 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 establish that Dolphin would work across all cellular connectivity scenarios. Notably, the observed error rates in all these experiments suggest that AMR codec was employed by the cellular channel during such evaluations (as discussed in Sec. 5.1 and Appendix. A.2.5).

A.2 Additional Discussion Points

A.2.1 Dolphin Usability: Dolphin supports bi-directional communication and can be easily used as an API for transferring any application's content. The API offers send() and recv() function calls (similar to the standard UNIX/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 *viz.,* email, news and Twitter. In addition, for accessing

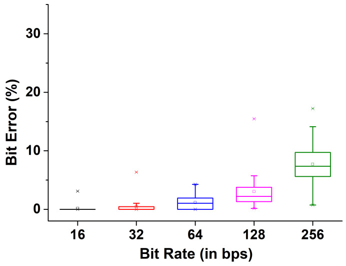

Figure 25: 2G-2G: Bit error rate variation for different bit rate for 100B transfer.

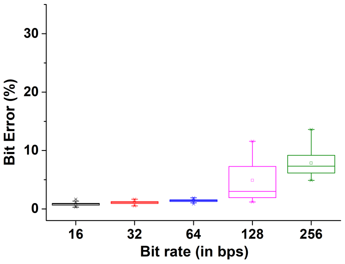

Figure 26: 2G-2G: Bit error rate variation for different bit rate for 1KB transfer.

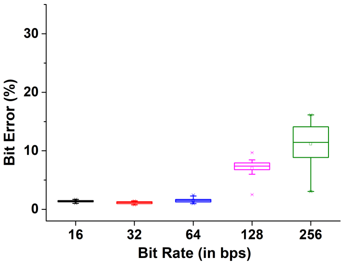

Figure 27: 2G-2G: Bit error rate variation for different bit rate for 5KB transfer.

websites, we built a proxy that can be used with browsers such as Firefox and Chrome (that is functioning and tested to access small static websites). The proxy listens on a local port for TCP connections from the browser. On receiving a website request, the proxy sends only the HTTP GET request to the Dolphin callee using the send() function call. The callee (already listening using the recv() function call) parses the received request and fetches the website data directly. The callee then sends the HTTP response to the proxy via the API, using the underlying cellular connection. Finally, the proxy then forwards the response to the browser. The above sequence is repeated for every website request. Notably, we do not transfer all TCP/TLS packets over the cellular channel, as doing so over the bandwidth-constrained cellular channel would be prohibitively costly. The callee is assumed to be trusted in our model and gains full visibility of the user content as there is no end-to-end TLS connection between the client and the application server. However, if one manages to perform a TCP and TLS connection, it will come at the cost of multiple orders of magnitude degradation in performance. Alternatively, if cellular operators employ more efficient codecs (*e.g.,* full rate AMR) or modulation techniques with higher encoding rates, the performance degradation could be minimized and end-to-end TLS based connections could also be employed.

A.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.

A.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 research works that explored sending data at relatively higher bit rates in packet based VoLTE systems [49, 55]. 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.

A.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-shutdown regions such as a low bit rate covert channel for exchanging secret information. Such solutions may be useful for bootstrapping various anti-censorship solutions [26, 46, 61, 62].

A.2.5 Potential Reason For Observing Low Bit Rates: We studied the feasibility of encoding and sending data at different rates and it revealed that we could achieve 64 bps data encoding rate with reasonable errors. But existing studies [34, 70] propose encoding schemes that achieve much higher rates (500 – 1000 bps). Contrary to such claims, we observed much poorer performance when we tested such schemes on real cellular calls. We believe the potential reason for such differences can be explained as follows.

In the context of Dolphin, it is essential to understand the role of modulation schemes and codecs when encoding data bits to be sent over the cellular channel. Modulation is responsible for encoding/converting the data bits to an 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 (*e.g.* in terms of available bandwidth). The codecs are optimized to preserve the audio features in the signal, such as pitch, timbre, rhythm, etc. Such optimizations, often involving compression, are agnostic to the signal's encoded data. While human perception compensates for distortions and losses in voice samples from using codecs, lost data bits cannot be recovered. In order to maximize goodput, the modulation schemes try to anticipate and minimize distortions arising from codecs. Thus, the modulation scheme's success depends on the underlying codec.

AMR is one of the cellular network's oldest and most widely used codecs. It is an adaptive multi-rate codec and provides different modes, each working on different data rates (4.75 to 12.2 kbps). The network operators select 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 evaluated their system assuming the full AMR bit rate mode (*i.e.,* 12.2 kbps).

In a simulation, we tested Dolphin's modulation and demodulation schemes against AMR's 12.2 kbps configuration codec. We used Authloop's simulation setup for the same and observed that Dolphin offered performance comparable to those reported in the previous studies. Dolphin's modulation experienced a bit error rate of 1.02% for a data encoding rate of 512 bps [4]. However, when we tested Dolphin on a lower AMR mode of 7.4 kbps, the error observed for the same encoding rate increased to 15%. In comparison, Authloop introduced an even higher error of about 23% for the same AMR mode of 7.4 kbps.

Thus, since Dolphin and Authloop observe high error rates over the real cellular channel, it indicates that the cellular channels provide lower bit rate AMR rate modes during the actual call. However, Dolphin's performance would drastically improve if cellular operators use higher AMR codec modes (or even other codecs). Moreover, in this work, we evaluated Dolphin for significant distortion inducing codec configurations and demonstrated that it can still be used to access lightweight applications. Thus, Dolphin should be able to obtain performance at least at par with those reported in this paper or higher. In future, we can augment Dolphin so that during bootstrap, it tests the medium to identify the maximum achievable data rate and performs transfers accordingly.

A.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. Broadcasters typically use 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 helplines etc. Hence, the censors usually do not have much motivation to block the cellular voice channel (as backed by multiple cited instances [8, 9, 12, 19]). Additionally, even with Dolphin, it is not trivial to spread information en-masse due to the modest data rates achieved. This makes it less appealing for the adversary to block the cellular channel completely. In the worst case, if the censor decides, it could still block the cellular channel; but this would have serious collateral damages.

Notably, in scenarios of complete communication blackout (Internet as well as cellular), it will be extremely difficult to design a system to access Internet applications. However, there are different sets of solutions (*e.g.,* Rangzen [53], Firechat [75], 1am [57], [68]) that are applicable for such a threat model which facilitate one-to-one or one-to-many communication between clients within the blackout region by assuming a mesh network.

A.2.7 VoIP vs Cellular Channel: Codecs generally introduce major distortions in the voice channel. The codecs used for VoIP communication are much less sophisticated as they work over the Internet and are not very constrained about bandwidth. However, cellular codecs are designed for very bandwidth constrained operations and thus perform many psychoacoustic optimizations on the original audio leading to much more distortions. Therefore, the solutions and analysis for VoIP systems are not trivially applicable to 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 achieved a data encoding rate of 1 Kbps with reasonable error rates (about 2%) over Whatsapp, Signal and Telegram.

A.2.8 Cellular Call Charges: Dolphin's functioning requires the callee to be in a non-shutdown region. Depending on the geographical area impacted by the shutdown, the callee may reside within the country (city-wide shutdown) or outside (nation-wide shutdown). The international cellular call charges may be applicable if the callee is outside the country. Thus, the user may want to know the cost of availing Dolphin service. Domestic cellular calls are generally cheaper compared to international ones. However, it may be challenging to do a cost analysis of international calls as call charges depend on the region. Internet shutdowns are generally performed in developing countries with cheap international calling rates. For instance, we computed the cost for India, a country suffering the most shutdowns globally. International call rate in one of the popular providers in India, i.e., Reliance Jio, ranges from $0.0065 to $0.077 per minute to any country [45]. Thus, for an hour of operation of Dolphin, it would cost between $0.39 to $4.62. In some shutdown-prone countries [2] like Myanmar and Uganda, it would cost roughly $1.62 and $4.8, respectively. Since domestic and international calls are costlier in developed countries, availing Dolphin would also incur relatively higher prices. Moreover, Dolphin can even be free for users if the Dolphin callee (or the proxy) initiates the call, instead of the caller. Such scenarios could even be practical when used to exchange bootstrapping information with existing anti-censorship systems, as discussed in Appendix. A.2.4.

A.2.9 Comparison With Other Relevant Modulation Schemes: In a previous subsection (ref. Sec. A.2.5), we discuss 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. [52] proposed a modulation

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

scheme explicitly designed for the GSM-EFR codec. Similarly, Ozkan et al. [66] also designed and tested the modulation scheme for the GSM FR codec. Since these modulation schemes 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. [25] and was tested for AMR 12.2 kbps mode as well as the AMR 4.75 mode. However, the authors demonstrated that for the lower rate AMR mode, a data encoding rate of about 80 bps was achievable. This is similar to what was observed for Dolphin.

A.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 [3, 21]. 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.

A.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.[5]

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 [5] 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.

A.4 Dolphin End-to-end System

Dolphin involves multiple layers, each having a different function. This layered design as depicted in Fig. 28 lends robustness and stability to the end-to-end system. Each layer can evolve separately without affecting the others. For instance, if in the future any new modulation scheme can offer high data encoding rates, it would require only changing the data link layer; the upper layers could continue working agnostically.

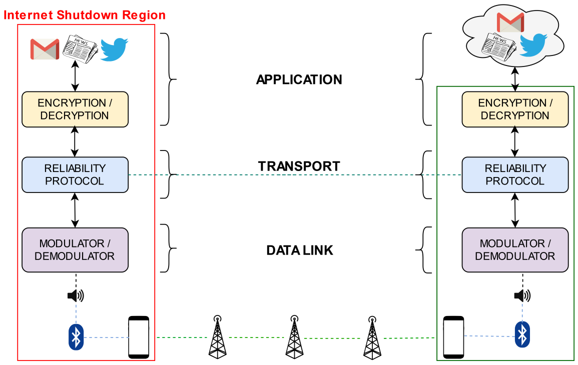

Figure 28: Dolphin's block diagram depicting its different functionalities (end-to-end).

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