Hermes: Data Transmission over Unknown Voice Channels

Aditya Dhananjay, Ashlesh Sharma, Michael Paik, Jay Chen, Trishank Karthik Kuppusamy, Jinyang Li and Lakshminarayanan Subramanian
Courant Institute of Mathematical Sciences, New York University
{aditya, ashlesh, mpaik, jchen, tk883, jinyang, lakshmi}@cs.nyu.edu

ABSTRACT

While the cellular revolution has made voice connectivity ubiquitous in the developing world, data services are largely absent or are prohibitively expensive. In this paper, we present *Hermes* [1], a point-to-point data connectivity solution that works by modulating data onto acoustic signals that are sent over a cellular voice call. The main challenge is that most voice codecs greatly distort signals that are not *voice-like*; furthermore, the backhaul can be highly heterogeneous and of low quality, thereby introducing unpredictable distortions. *Hermes* modulates data over the extremely narrow-band (approximately 3kHz bandwidth) acoustic carrier, while being severely constrained by the requirement that the resulting sound signals are voice-like, as far as the voice codecs are concerned. *Hermes* uses a robust data transcoding and modulation scheme to detect and correct errors in the face of bit flips, insertions and deletions; it also adapts the modulation parameters to the observed bit error rate on the actual voice channel. Through real-world experiments, we show that *Hermes* achieves approximately 1.2 kbps goodput which when compared to SMS, improves throughput by a factor of 5× and reduces the cost-per-byte by over a factor of 50×.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Network communications

General Terms

Design, Experimentation, Measurement, Performance

1. INTRODUCTION

The penetration of data connectivity services in the developing world has largely remained an urban phenomenon with extremely poor connectivity in several rural and underdeveloped regions [15]. The fundamental roadblock is one of economics: due to low purchasing power and low active user densities (real user-density and not population density) coupled with the lack of reliable power and the high capital and operational costs for the provider, providing connectivity still remains a loss-making venture. Due to these factors, traditional wired network connectivity solutions have made very little in-roads in rural areas.

In recent years, many developing countries have undergone a cellular revolution with a significant proliferation of digital cellular networks in rural areas [7, 3]. In addition to voice calls, these networks provide SMS text messaging services that can serve as a rudimentary data layer. While it is technically easy to provide enhanced data services on top of a digital cellular network, economic factors prevent the cellular carriers from providing it; the cost of upgrading the cellular infrastructure with data-capable hardware and software is very high, given the low user densities and low purchasing power. As a result, very few carriers provide data connectivity through GSM Circuit Switched Data (also known as GSM Fax) and General Packet Radio Service (GPRS); even fewer carriers provide Enhanced Data rates for GSM Evolution (EDGE) and 3G services.

There are a large number of important services and applications that can be enabled by having just a low bandwidth point-to-point data connectivity solution; these services and applications broadly lie in the domains of microfinance, supply-chain management, agriculture and healthcare provisioning. Unfortunately due to the small market for data services among end-users, there exists little incentive for cellular providers to provide enhanced data services of any kind.

Under these extreme conditions, we ask the question: *Given the ubiquity of cellular coverage, can we provide any form of data connectivity services over the cellular network?* While SMS is available as a possible data channel, it is extremely low-bandwidth, where every SMS message is limited to 140 bytes and furthermore, the cost per bit is quite high [1, 4]. In most areas, a single SMS costs between $0.05 to $0.25 which is comparable to per-minute voice call rates [4]. Hence, if there exists an efficient data connectivity service over the cellular *voice* channel, one can enable a new class of mobile data services and also significantly reduce the cost per bit for data connectivity.

Unlike the traditional modem setting, providing data connectivity over the cellular voice channel is a challenging problem due to several constraints. First, the acoustic channel is

[1]Named after the great messenger of the gods, the guide to the underworld and the god of thieves, in Greek mythology.

MobiCom'10, September 20–24, 2010, Chicago, Illinois, USA.

extremely narrow-band, which inherently limits the achievable data rates. Second, the voice codes used in cellular networks introduce several unpredictable distortions due to the use of memoryful codecs; in contrast, commonly used modulation techniques operate only over memoryless channels. In addition, the cellular network uses several optimizations like voice activity detection and automatic gain control which further distort the underlying channel. Third, the underlying voice codecs are hard to model or predict since they are themselves adaptive and continuously change parameters. Finally, a typical end-to-end voice call path may traverse a heterogeneous set of network links, thereby making the underlying channel effectively unknown. This is especially true in rural areas where the mobile backhaul networks are point-to-point wireless links built using WiMax, micro-wave, satellite or other proprietary technologies.

This problem has received relatively little attention. The few prior efforts [9, 13, 5, 14, 17, 18] in this space have primarily concentrated on the case where the underlying voice codec is known. One of our explicit design goals is that our modulation algorithm should adapt itself to the unknown acoustic channel and not depend on any specific voice codec.

In this paper, we describe the design, implementation and evaluation of *Hermes*, a data transmission mechanism that uses the voice channel as an acoustic modem that achieves 1.2 kbps with very low bit error rates, while operating over unknown voice channels. To motivate the design of *Hermes*, we show that most of the known modulation techniques completely fall apart over cellular voice channels. *Hermes* modulates data over the extremely narrowband (approximately 3kHz bandwidth) acoustic carrier, while being severely constrained by the requirement that the resulting sound signals need to be voice-like. *Hermes* uses a robust transcoding and frame recovery scheme to detect and correct errors in the face of bit flips, insertions and deletions. *Hermes* can also adapt the modulation parameters according to the observed bit error rate on the actual voice channel.

We demonstrate the effectiveness of *Hermes* through extensive evaluation of our techniques over different real-world cellular networks. Achieving 1.2 kbps with very-low error rates (bit error rates in the order of 10^{-5}, with frame error rates $< 1\%$) over these unknown voice channel conditions is fairly significant, especially when standard modulation techniques give extremely low throughputs and high bit error rates. *Hermes* can enable a wide-range of network services in rural settings which have not been possible before including rural information services, mobile health-care and new forms of mobile banking. We had previously built many of these applications purely using SMS as a data layer; when compared to SMS, *Hermes* can provide improved throughput by a factor of 5×, while lowering the corresponding cost-per-byte by over a factor of 50×.

The rest of this paper is organized as follows. In § 2, we examine the cellular voice channel characteristics and explain why the acoustic characteristics (and hence, the data carrying capabilities) of a voice call are so unpredictable. In § 3, we describe traditionally used modulation techniques for transmitting data over an analog carrier, and examine why they under-perform over the cellular voice channel. In § 4, we describe the framing, transcoding and modulation techniques in *Hermes*. We evaluate the performance of *Hermes* in § 5, discuss related work in § 6 and present our conclusions in § 7.

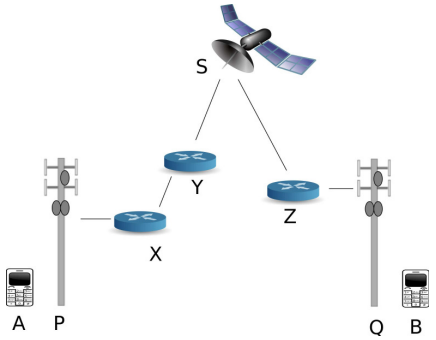

Figure 1: Simplified network topology, showing the routing path between cell phone A and B.

2. UNPREDICTABLE VOICE CHANNELS

In this section, we study the challenges involved with transmitting data over a cellular voice channel. We first present an overview of the cellular infrastructure and then discuss the sources of various unpredictable transformations that the audio signal can undergo along its end-to-end path.

2.1 Basic Setting

Fig. 1 shows a simplified routing path between two cellular phones. Let us consider only the unidirectional flow from phone A to phone B. The voice stream from the user (the analog sound waveform) is digitized by A, using a particular voice codec (§ 2.2.2). This digital representation of voice is transmitted to the base station P, which reconstructs an approximation of the analog sound waveforms. Since the digital channel between the mobile phone and the base station carries real-time voice, there are no retransmissions; this channel is lossy. The voice codec is designed to tolerate a certain amount of loss, without a significant reduction in the sound quality as perceived by the ear.

At base station P, these analog waveforms are reconverted to a digital representation, for transmission over link L_{PX}. This digital format is determined by the capabilities of P and X, as well as the type of communication link L_{PX}. This process (digital → analog → digital conversion) is repeated across potentially very hop until it reaches the destination base station Q (§ 2.2.4). At Q, the voice is encoded using the appropriate codec and transmitted to B. Note that this codec could be different from that used in the communication between A and P. Finally at B, the analog waveform is reconstructed and is played to the user.

2.2 Challenges

Next, we study the properties of the cellular voice channel (such as the one between A and B in Figure 1). In particular, we examine the various sources of distortion in the audio signal in order to understand why it is difficult to modulate digital information over such channels.

2.2.1 Narrowband Channels

Cellular voice channels are designed to carry human speech, specifically within the frequency range of $[300Hz, 3400Hz]$. The audio signal is band-pass filtered at the source phone (and potentially, at intermediate hops) where components outside of this frequency range are removed; this filtering introduces distortions into the transmitted audio stream.

Furthermore, a $3kHz$ channel is *very* narrowband; in accordance with Shannon's capacity formula, this severely limits the achievable data rate.

2.2.2 Voice Codec Distortions

Signal distortions introduced by voice codecs pose one of the biggest challenges to data modulation. Voice codecs digitize the speech at the sender and re-convert to analog at the receiver. Modern codecs, as employed by cellular networks including GSM and CDMA, aggressively exploit certain speech signal properties to achieve great compression while still maintaining good speech quality as experienced by human users. As a result, audio signals that are not *voice-like* are greatly distorted because they violate speech signal properties assumed by the underlying codecs. Below, we describe how voice codecs leverage human speech properties.

Memoryful Codecs: These codecs are based on psycho-acoustic audio processing techniques that model the speech input, such that only those characteristics relevant to the human speech and auditory system are transmitted. It is known that speech waveforms show only small variations over short time scales; in fact, speech can be modeled as a periodic waveform with a single fundamental frequency, with occasional bursts of hissing *(sibilants)* and popping *(plosive)* sounds. A large class of voice codecs are based on this model of speech, and have their foundations in linear predictive coding (LPC). LPC-based codecs estimate various speech parameters and represent them digitally. Based on this model of speech described above, LPC approximates the current output sample as a linear function of a few previous input samples. This *memoryful* nature leads to the output waveforms being different from the input waveforms, even though they might be perceived to be similar by the human ear. Audio signals with large variations over short time scales do not fit the speech model assumed by LPC and therefore are distorted even more.

Automatic Gain Control (AGC): In order to maintain the sound *volume* in a phone conversation, AGC uses the average output signal level in a feedback loop to control the amount of amplification for subsequent input samples. This implies that the amplitude of the received signal might be quite different from that of the input.

Voice Activity Detection (VAD): Human conversations are typically half-duplex, i.e. only one person tends to speak at a time. VAD detects the presence of speech activity in an audio stream, so that data transmission can be deactivated during the silence periods to save bandwidth and battery power. VAD distinguishes human voice from background noise by exploiting the fact that human voice *pulses* over time intervals on the order of a second, while background noise does not exhibit such pulsing. As a result, if the input audio signal does not show pulsing in amplitude, it may be filtered out by VAD as background noise.

2.2.3 Adaptive Behavior of Codecs

Since voice codecs are based on known algorithms, it is tempting to explicitly model and compensate for the distortions introduced by these codecs in the data modulation process. In other words: *Given an input audio signal, can we model and predict the distortions that the codec introduces?* Unfortunately, such an approach is not practical because many codecs change their operational parameters and modes on the fly. Consider the GSM AMR codec: it has 14 modes of operation and chooses the best mode according to the current wireless link quality and capacity requirements. For instance, if the underlying wireless conditions are bad, it reduces source coding and increases channel coding to send a lower-quality speech signal more robustly over the wireless link. Cellular network providers can also increase the number of calls that can be handled by a base station by forcing phones to switch to a low bitrate codec mode. Since the codec mode can change on the fly and each mode has its own characteristics, it is infeasible to accurately predict the distortions that the codec introduces in a given audio signal.

2.2.4 Heterogeneous network links

As explained in § 2.1, voice travels across multiple hops from the source to the destination base station. Each intermediate link may further transform the encoded voice signal according to its capabilities and bandwidth constraints, thus introducing additional distortions. For example in Figure 1, intermediate hop Y may transform the received signal into a lower bitrate to send over the expensive satellite link L_{YS}. These transformations are performed across potentially every hop, and this is another reason why distortions in received audio stream are so unpredictable.

3. TRADITIONAL MODULATION

In this section, we examine the fundamental question: *How do we modulate binary data over an analog acoustic carrier?* This process is a digital to analog conversion, where the input bit-stream is converted to sounds, which are transmitted over the voice call.

The obvious first question is: *Can we simply interface a traditional modem (or an acoustic coupler) to use the audio channel provided by the cellular voice call?* The answer unfortunately, is no. Traditional telephone lines operate on a memoryless channel, where the concept of signal and noise are intuitively clear. The signal is transmitted from one end to another; along the way, it gets attenuated, and a certain amount of noise gets *added* to it. Unfortunately, cellular networks do not operate under the additive noise model. The very concept of *noise* is unclear in this context, as most of the distortion is introduced by the voice codec, even before it is transmitted from the source cellular phone (§ 2.2.2). As a result, the observed SNR is a direct function of the currently and previously transmitted data bits. Traditional modems fail because their modulation techniques operate on channels that exhibit only additive noise properties.

In this section, we explore two sets of broad techniques used to perform modulation: *keying* and *multiplexing*. As we shall see, the codec introduces so much noise, that the resulting bit error rates are too high to be useful.

3.1 Keying

There are three fundamental properties of the acoustic carrier signal that can be modified (or keyed) in accordance with the input bits: amplitude, frequency and phase. While more complex modulation schemes can be constructed by combining two or more of these techniques, we consider them independently.

3.1.1 Amplitude Shift Keying (ASK)

This is a modulation technique, where the amplitude (or *strength*) of the carrier signal (of frequency fixed at f_c) is varied in accordance with the data bits being sent. The simplest

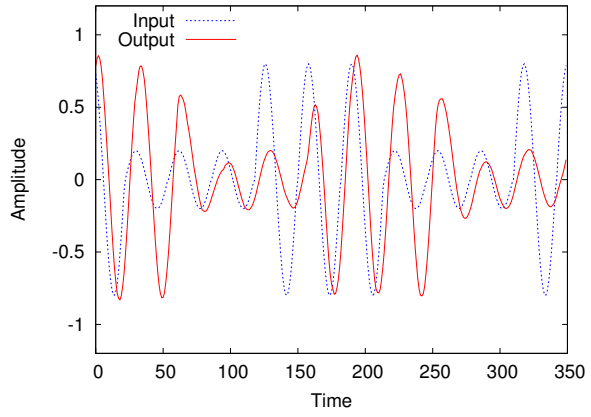

Figure 2: Codec induced distortions: ASK at 1 bit per symbol ($a_0 = 0.2$, $a_1 = 0.8$, $f_c = 1400Hz$). Since the input waveform shows large variations in amplitude over short time scales, the output waveform is significantly distorted by the codec.

method is binary ASK, where one sinusoid is transmitted per bit; a 0 and 1 are transmitted as sinusoids of amplitude a_0 and a_1 respectively. While this approach can be generalized to transmit n bits per sinusoid using 2^n unique sinusoids of different amplitudes, we consider only binary ASK.

Fig. 2 shows a simulation of performing ASK over 1 input bit per sinusoid, using a software implementation of the GSM Adaptive Multi Rate (AMR) codec. Intuitively, we would like a_0 and a_1 to be *far-apart* (amplitudes a_0 and a_1 being quite different), since channel noise will have have a smaller probability of modifying the waveforms sufficiently to cause bit-flipping. However, we observe that the amplitude of an output wave can be quite different from that of the corresponding input wave; this is due to the "memoryful" nature of the codec (see § 2.2.2). In other words, simply having symbols with *far-apart* constellation points does not necessarily lead to better performance, since the codec itself distorts the waveforms.

Suppose we choose parameters such that a_0 and a_1 are nearer to each other. While the amount of noise introduced by the codec is smaller, the waveforms are less resilient to channel noise (this figure is not shown). This illustrates the tension between the need to keep symbols close together (to please the codec) and the need to keep the symbols far apart (to maintain resilience against channel noise).

To quantitatively study the performance of ASK, we perform modulation using different values of a_0 and a_1. Demodulation is performed using the standard closest neighbor search algorithm: if the received sinusoid has amplitude $a \leq \frac{a_0+a_1}{2}$, the output is 0; else, the output is 1. The results are tabulated in Table. 1, and show that the bit error rates are very high.

The problem is further compounded with larger values of n, since the distance between certain pairs of symbols can be too small to be resilient against channel noise, while the distance between certain other pairs of symbols might be too large, thereby causing the codec itself to distort the waves.

3.1.2 Frequency Shift Keying (FSK)

a_0	a_1	f_c (Hz)	BER	a_0	a_1	f_c (Hz)	BER
0.2	0.4	1700	2.7×10^{-1}	0.4	0.6	2000	3.4×10^{-1}
0.2	0.6	1700	4.5×10^{-2}	0.4	0.8	2000	2.2×10^{-1}
0.2	0.8	1700	1.6×10^{-2}	0.6	0.8	2000	3.8×10^{-1}
0.4	0.6	1700	1.9×10^{-1}	0.2	0.4	2300	1.9×10^{-1}
0.4	0.8	1700	2.4×10^{-1}	0.2	0.6	2300	6.2×10^{-2}
0.6	0.8	1700	3.0×10^{-1}	0.2	0.8	2300	9.9×10^{-2}
0.2	0.4	2000	2.6×10^{-1}	0.4	0.6	2300	3.6×10^{-1}
0.2	0.6	2000	5.4×10^{-2}	0.4	0.8	2300	1.4×10^{-1}
0.2	0.8	2000	4.0×10^{-2}	0.6	0.8	2300	3.6×10^{-1}

Table 1: Performance of ASK (at 1 bit per symbol) for different a_0 and a_1 values. Distortions introduced by the voice codec lead to very high bit error rates.

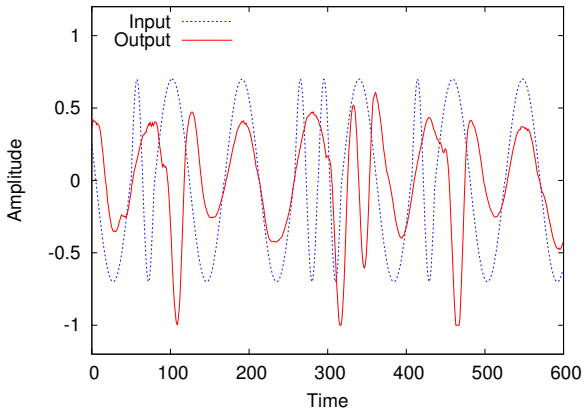

Figure 3: Codec induced distortions: FSK at 1 bit per symbol ($f_0 = 500Hz$, $f_1 = 1500Hz$, $a_c = 0.7$). Since the input waveform shows large variations in frequency over short time scales, the output waveform is significantly distorted by the codec.

This is a modulation technique where data is transmitted through discrete frequency changes of a carrier wave (of amplitude fixed at a_c). The simplest method of performing FSK is binary FSK (BSFK), where we transmit one sine wave per bit. A 0 or 1 is transmitted as a sinusoid of frequency f_0 or f_1 respectively. While this approach can also be generalized to transmit n bits per sinusoid using 2^n unique sinusoids of different frequencies, we consider only BFSK.

Fig. 3 shows a simulation of performing BFSK, using the GSM AMR codec. Intuitively, we want f_0 and f_1 to be *far-apart*, in order to be able to distinguish the symbols. However, this leads to abrupt and large changes in frequency over short time scales, rendering the audio stream to become non voice-like; the voice codec therefore significantly distorts to the audio stream. Therefore, we also need to keep the symbols close enough (in the frequency domain) to please the voice codec.

To quantitatively study the performance of FSK, we simulate the modulation using different values of f_0 and f_1. Demodulation is performed using the standard closest neighbor search algorithm: if the received symbol has frequency $f \leq \frac{f_0+f_1}{2}$, the output is 0; else, the output is 1. The results are tabulated in Table. 2, and show that the bit error rates are approximately an order of magnitude lower than in

f_0 (Hz)	f_1 (Hz)	BER	f_0 (Hz)	f_1 (Hz)	BER
1600	1700	1.0×10^{-2}	1700	1900	4.7×10^{-2}
1600	1800	1.0×10^{-2}	1700	2000	5.0×10^{-3}
1600	1900	4.4×10^{-3}	1800	1900	4.0×10^{-1}
1600	2000	1.9×10^{-2}	1800	2000	1.4×10^{-1}
1700	1800	4.7×10^{-2}	1900	2000	3.0×10^{-1}

Table 2: Performance of FSK (at 1 bit per symbol) for different f_0 and f_1 values. While distortions introduced by the voice codec still lead to high bit error rates, they are an order of magnitude lower than in ASK.

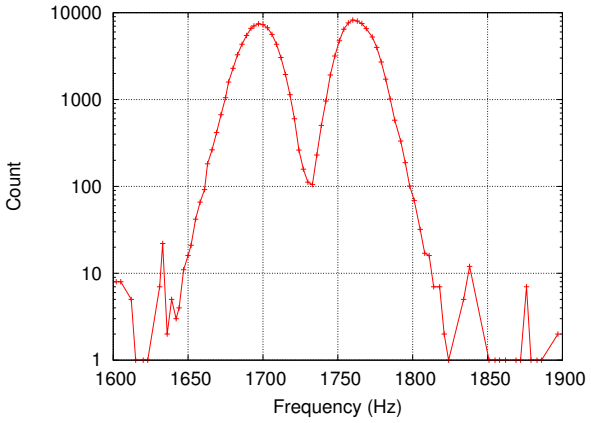

Figure 4: Frequency distribution of received symbols for FSK at 1 bit per symbol ($f_0 = 1700Hz$, $f_1 = 1800Hz$). Observe that the two inverted bell curves significantly overlap, and are not centered correctly.

ASK. The intuition behind this difference lies in the fundamental principles of memoryful codecs (§ 2.2.2). Since LPC approximates the current sample (samples are in the amplitude domain) as a linear function of previous samples, it *remembers* amplitude components better, leading to greater inter-symbol interference in ASK. Due to the better performance of FSK, it is a building block in the modulation layer of *Hermes* (§ 4.1).

In order to better understand the behavior of the underlying voice channel, we examined the frequency distribution of received sinusoids ($f_0 = 1700Hz$ and $f_1 = 1800Hz$). As shown in Fig. 4, the inverted bell curves for the $1700Hz$ and $1800Hz$ sinusoids significantly overlap, which obviously makes decoding harder. Furthermore, while the bell curve for $1700Hz$ is centered correctly, the curve for the $1800Hz$ sinusoid is centered around $1760Hz$; this is the reason that the closest neighbor matching demodulator at the receiver makes so many errors. Finally, we observed that the point at which the bell curves are centered varies for different codecs; thus if different codecs (or codec modes) are used at the two end points, it is very likely that the demodulation process will yield a high number of bit errors. The important implication of this behavior is that our choice of symbols and demodulator parameters should not be based on any particular codec.

3.1.3 Phase Shift Keying (PSK)

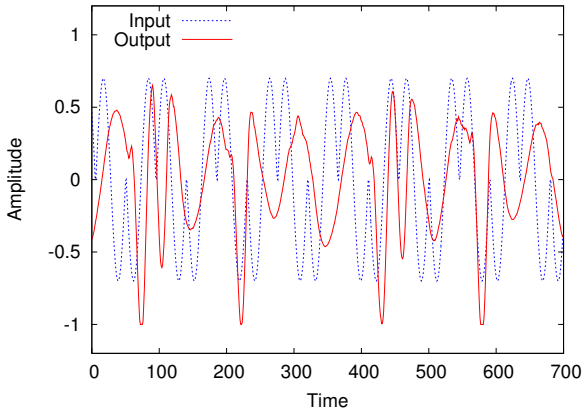

Figure 5: Codec induced distortions: PSK at 1 bit per symbol ($p_0 = 0$, $p_1 = \pi$, $a_c = 0.7$). Since the input waveform shows abrupt transitions at the symbol boundaries, the output waveform is significantly distorted by the codec.

This is a modulation technique in which digital information is transmitted through discrete phase changes of a carrier wave (of amplitude and frequency fixed at a_c and f_c respectively). Similar to ASK and FSK, the number of discrete phases used across the symbol space determines the number of unique symbols. Since the phase of the waveform changes for every single sinusoid that is transmitted, this creates a large number of discontinuities (abrupt jumps in amplitude) in the time domain signal. Even very small changes in phase are sufficient to cause discontinuities in the input signal. Since such audio signals are not voice-like, they get severely distorted by the codec, as illustrated in Fig. 5.

Performing PSK demodulation requires that all symbols are of the same length; in particular, we need to be able to accurately determine symbol boundaries. Once each symbol has been isolated, it is multiplied by a known sinusoid, and the resulting signal is low-pass filtered to give the original data. From Fig. 5, we observe that the sinusoids show *highly* varying time-periods, thereby making it virtually impossible to perform time synchronization and isolate the individual symbols. In effect, this renders PSK impossible to realize over memoryful codecs.

3.2 Multiplexing

The principle behind Orthogonal Frequency Division Multiplexing is to divide a channel into several non-interfering narrowband *sub-channels* and encode data at a low rate on each sub-channel. The aggregate capacity of the channel is therefore the sum of the capacities of all the sub-channels. Any modulation technique can be used on the sub-channels. In the context of using OFDM on the $3kHz$ cellular voice channel, we ask: *How many sub-channels can we create, and what data rate can each sub-channel achieve?*

Assume that the frame size is m bits and we transmit one frame in every epoch of time. Further assume that we can divide the $3kHz$ channel into k orthogonal sub-channels, each carrying an independent data stream of $b = \frac{m}{k}$ bits per epoch. Unfortunately, performing OFDM demodulation at the receiver on such a stream is very hard. The computational resources required to perform fine-grained analog

filtering on the received audio signal (in order to recover the data on each sub-channel) far exceeds the computational power of cellular phones, even for small values of k.

The only way around the problem is to ensure that $b = 1$; in other words, each sub-channel carries 1 bit per epoch. The modulation technique on each sub-channel is in effect *On-Off Keying*, which is a special case of ASK, where a_0 and a_1 are 0 and 1 respectively. The receiver determines that the n^{th} bit of the frame ($n \leq k$) is 1 if it can detect the presence of the n^{th} sub-channel carrier; else, this bit is determined to be 0. This technique is computationally feasible, since the receiver can perform a simple Fast Fourier Transform (FFT) on the received signal over the epoch; the frequency components of the FFT output give us the bits of the frame. This would not have been possible with $b > 1$, since it would have necessitated fine-grained and computationally intensive filtering to recover the individual sub-channels.

Unfortunately, the achievable data rates using multiplexing techniques is very low. For example, we experimentally determined that for an epoch size of $100ms$, we can transmit about 20 bits of data (20 sub-channels; 1 bit per epoch per sub-channel). If we increase (decrease) the epoch length, then we can use a larger (smaller) number of sub-channels as well.

Furthermore due to the memoryful nature of the codecs, we want to prevent frequency components (sub-channels) of one symbol from interfering with those in the next symbol. In order to achieve this, we need to leave a guard time of approximately $40ms$ to let the codec *forget* components from the previous symbol. Adding a guard interval of $40ms$ between epochs and using the previous example, this works out to about 150 bits per second, which is very low.

This problem is compounded due to the fact that the cellular voice channel has a frequency response behavior, such that it carries through only a certain fraction of the $3kHz$ bandwidth reliably (as shown in Fig. 6 and explained in § 4.3.1). This further reduces the practically achievable value of k, thereby reducing the throughput even more. We therefore conclude that the data rates are too low to be useful.

4. DESIGN

In this section, we explain the design of *Hermes*. User data is first fed into a module that performs framing and transcoding. In traditional networks, these two components are separated, but in *Hermes*, they are combined in order to improve error recovery (§ 4.2). This module breaks the data into fixed size chunks and creates a stream of frames from it. The resulting stream of frames is sent to the modulator (§ 4.1), which converts these bits into sounds that are sent over the actual phone call. On the receiver side, the reverse process is carried out. We now examine each of these stages in detail.

4.1 Modulation

For memoryless codecs, the generated audio signal does not need to have any special *voice-like* characteristics. Memoryful codecs on the other hand require the input audio signals to be voice like (§ 2). The design of symbols in *Hermes* must therefore satisfy the following requirements:

- The generated waveforms should pass the bandpass filter of $[300Hz, 3400Hz]$, which is the relevant band within which the human ear can understand the speech signals.
- Human speech has a fundamental frequency over long durations of time. When arbitrary symbols (sinusoids) are concatenated and sent over the air, the resulting sound stream should also have a fixed fundamental frequency.
- Any pair of symbols must not be too *far-apart*; otherwise, the codec will introduce distortions. Similarly, any pair of symbols must not be too *close together*; otherwise, the symbols will not be resilient against channel noise. Formally, the distance between the constellation points of any pair of symbols in the trellis diagram must be upper and lower bounded.
- Due to AGC (§ 2.2.2), different frequency components might be amplified (or attenuated) by different amounts by intermediate hops in the network; we should therefore not rely on the amplitude of the output waveforms to be similar to that of the input. Furthermore, the generated waveforms should exhibit voice-like *pulsing*, or it may be filtered out by VAD (§ 2.2.2).
- The acoustic symbols generated by the modulator should not depend on any particular codec or codec mode. An explicit design principle is that we should not train the modulation process offline to optimize for any specific codec/mode; it should be generic enough to work over any codec/mode.
- Algorithms used for coding and modulation should be simple enough to be run on even the most basic cellular phones.

We have designed a simple algorithm (called *IncDec*) to perform modulation in *Hermes*. It is explained in Algorithm 1; it can be thought of as being a constant-differential frequency shift keying technique.

Algorithm 1 Convert binary data to sound signals to be sent over a voice call.

```
Given: base frequency f_base, delta frequency δ.
f = f_base
for each bit b in the input string do
  if b = 0 then
    f = f − δ
  else
    f = f + δ
  end if
  Generate a sinusoid of frequency f
end for
```

The basic idea of the algorithm is that we have a frequency f that is initialized to the base frequency f_{base}. If we read a 0 (or a 1) from the input stream, we decrement (or increment) f by a fixed δ, and transmit a sinusoid of frequency f. This modulation has the advantage that all transitions are bounded by δ, and hence large jumps can be prevented.

Unfortunately, long strings of 0s or 1s can push the current frequency f over the permissible limit. Furthermore, since the frequency of transmitted sinusoids can vary quite a lot, it is hard to guarantee a fixed fundamental frequency of the resulting acoustic signal. Also, if there is a long string

of 0s, then the modulator begins to operate in the low frequency range, thereby reducing the number of sinusoids that can be sent per unit time (and hence the throughput). Finally, the frequency response of the channel is quite different in different frequency ranges (§ 5); the input string to the modulator could force it to operate in a frequency range that the channel does not carry very well. To work around these weaknesses, we need to *transcode* the input stream before feeding it to the modulation layer (explained in § 4.2).

On the receiver side, the modulator needs to convert the sound signals back into bits. This demodulation algorithm is also quite simple and is explained in Algorithm 2.

Algorithm 2 Convert received sound signals back into binary data.

```
Given: Input sound signal.
for each sinusoid in the input sound signal do
  Let f_curr = frequency of current sinusoid
  Let f_prev = frequency of previous sinusoid
  if f_curr ≤ f_prev then
    Output 0
  else
    Output 1
  end if
end for
```

At first glance, the intuition behind the demodulation algorithm might seem vague and unclear. To analyze it, we show that it can cope with both memoryless codecs that are used in traditional wireline phone networks as well as memoryful codecs that are used in cellular networks.

4.1.1 Memoryless Codec

In this case, the codec does not assume the sound signal to be voice-like, and therefore does not significantly modify the frequency components of the signal; in other words, the noise is *additive*. Hence with a very high probability, the received waveforms will also have frequencies approximately equal to those of the transmitted waveforms. Clearly, a transition from $f-\delta \rightarrow f$ or a transition from $f \rightarrow f+\delta$ will be decoded as a 1 by Algorithm 2. Similarly, a transition from $f+\delta \rightarrow f$ or a transition from $f \rightarrow f-\delta$ will be decoded as a 0. Unless the additive noise introduced by the network is extremely high, bit errors do not take place and the demodulation is correct.

4.1.2 Memoryful Codecs

As explained in § 2.2.2, memoryful codecs produce their output as a linear function of the previous input samples. There is a greater weightage given to more recent samples as opposed to older samples.

Let us assume that the last symbol transmitted over the air had an input frequency f_i, and the corresponding output sinusoid had frequency f_o. This sinusoid f_o was created by the codec by taking into consideration the input sinusoid f_i as well as the previous few sinusoids that preceded f_i. Assume that the next input sinusoid has frequency $f_i + \alpha$, where $\alpha = \pm\delta$. We ask the question: *What can we expect the output frequency of the next sinusoid to be, relative to the frequency f_o of the current output symbol?*

The answer is that if $\alpha = +\delta$, we expect the output frequency to be greater than f_o. Similarly, if $\alpha = -\delta$, then we expect the output frequency to be lesser than f_o. Any generic memoryful codec gives a greater weightage to the current input sample, and a linearly decreasing weightage to the preceding samples. Therefore, we expect that when compared to the previous output symbol, we expect the new output symbol to be biased *in the direction* of the new input symbol. This argument is formalized below.

Let us say that $t_0, t_1, t_2, \cdots, t_k$ are the samples belonging to the k previous input symbols. Let the corresponding output samples be $s_0, s_1, s_2, \cdots, s_k$. Without loss of generality, let us say that the memoryful codec uses a weighted average of the current symbol, along with the previous n symbols. Therefore when input symbol t_k was being transmitted as output symbol s_k, the codec used a weighted average of $t_k, t_{k-1}, t_{k-2}, \cdots, t_{k-n}$, in decreasing order of weights. Now when the codec receives symbol t_{k+1}, it uses a weighted average of $t_{k+1}, t_k, t_{k-1}, t_{k-2}, \cdots, t_{k-n+1}$, again in decreasing order of weights; this can be approximated as a weighted average of t_{k+1}, s_k. This is why the new output symbol is biased in the direction of the new input symbol, relative to the previous symbol output.

We note that none of the properties of the modulation layer are dependent on any particular codec or implementation. The explanation given above can be generalized to *any* memoryful codec or codec mode. Irrespective of the length of history considered or the biasing constants, the general principles still hold.

4.2 Framing Transcoding

Now that we have examined how the underlying modulation and demodulation layer works, we explain the higher layer in the stack: framing and transcoding.

This module takes in a stream of bits from the user and breaks them into fixed size chunks. For each chunk, it calculates and appends a 32-bit CRC to it, thereby creating a frame. It also inserts a fixed preamble between every pair of frames. The user data can be easily coded using escape sequences to ensure that the preamble pattern does not appear in it. This stream of frames and preambles is then transcoded.

Transcoding is defined as a digital to digital conversion of one encoding to another. In *Hermes*, we transcode the input stream into another representation before passing it to the modulation layer to be converted into sounds. The goals of the transcoding layer are summarized as follows:

- The stream fed by the transcoder to the modulator should be such that the modulator is able to guarantee a fixed fundamental frequency.
- The modulation layer should use a minimum number of unique frequencies over the air; this minimizes the distortions. Also, the modulator should keep the value of f within acceptable limits.
- The receiver should be able to recover the original bit stream with a very high probability, even in the face of bit insertions, deletions and flips in the post-transcoded stream of data that is modulated and sent over the air.

Transcoding of the data is performed using a very simple algorithm. It is a 1/2 code, meaning that it takes in one input bit and produces two output bits, as shown below:

$$0 \rightarrow 01$$
$$1 \rightarrow 10$$

We make a few observations about the interaction of the transcoding layer with the modulation layer. Whatever the original input string, the transcoded stream will have an equal number of 0s and 1s. In particular, *any* substring containing an even number (≥ 4) of bits in the post-transcoded stream will have an equal number of 0s and 1s. Furthermore it is easy to see that the maximum number of consecutive 0s or 1s in the transcoded string is 2. As a result, the output sound of the modulator has a fundamental frequency fixed at f_{base}, making it more voice-like. In fact, there are only 3 frequencies that are ever used: $f_{base} - \delta$, f_{base} and $f_{base} + \delta$.

At the receiver side, the task of the transcoder is to receive $2n$ bits from the demodulator and estimate the n original bits that were sent at the transmitter. In the simplest case where there are no alignment issues, we can simply use the following rules:

$$01 \rightarrow 0$$
$$10 \rightarrow 1$$

However, in the presence of alignment errors (the received bits could have bits inserted and deleted), we can run into trouble. Suppose the original pre-transcoded input stream was all 0s; the corresponding post-transcoding stream was therefore, $0101\cdots$. Now suppose that due to noise, the first sinusoid is lost at the receiver. As a result, the demodulator will output $1010\ \cdots$, which the reverse transcoder will incorrectly decode as a string of all 1s. Now, there is no way of detecting the error. The best we can do is to ensure that the pre-transcoded input stream does not have long strings of all 0s or all 1s, by using special escape sequences. Bit-stuffing techniques like these are very commonly used, and hence have been omitted from this discussion. For now, assume that the maximum of consecutive 0s or 1s permissible is k_{max_rep}.

Let us now consider a stream of bits that the demodulator outputs. We still need to determine where the correct boundaries lie; a wrong decision at this point will lead the rest of the stream being incorrectly decoded. Consider the pre transcoded input stream 01, which leads to the transcoded stream 0110; notice that a $0 \rightarrow 1$ transition in the pre transcoded input stream leads to a pair of consecutive 1s in the post transcoded stream. Similarly, a $1 \rightarrow 0$ transition in the pre-transcoded input stream leads to a pair of consecutive 0s in the post-transcoded stream. Therefore, all the reverse transcoder needs to do is identify a 00 or 11 and place the alignment boundary between the two.

This problem becomes trickier in the face of bit insertions and deletions. Even if the reverse transcoder has achieved alignment at some point of time, a bit insertion or deletion can throw it completely off, leading it to incorrectly decoding the rest of the stream. The problem is further complicated by the fact that bits can also get flipped; For example, a pre-transcoded 0 (post transcoded 01) might get flipped to 00 at the demodulator output; this in turn might force the reverse transcoder into incorrectly believing that it has lost alignment. The intuition behind differentiating an insertion/deletion error from a flip error is that former actually leads to a loss of alignment, whereas the latter does not. A bit flip leads to only one block of 2 bits being decoded incorrectly, whereas an insertion/deletion error leads to subsequent blocks (of 2 bits each) also being decoded incorrectly. The algorithm to detect (post-transcoded) bit flips, insertions and deletions is based on this intuition, and is shown in Algorithm 3.

Algorithm 3 Reverse transcode the data at the demodulator output.

```
Given: Demodulated string s, consisting of post-transcoded bits.
Given: k_max_rep, the maximum permissible number of consecutive 0s or 1s in the input.
for i = 1; i ≤ length(s); i = i + 2 do
  b_1 = s[i]
  b_2 = s[i + 1]
  if b_1 b_2 = 01 or b_1 b_2 = 10 then
    Output 0 or 1, respectively.
  else
    There is either a bit flip or an insertion/deletion error.
    m_i = ErrorMetric(i)
    m_{i+1} = ErrorMetric(i + 1);
    if m_i ≤ m_{i+1} then
      The alignment is correct, but there is a bit flip.
      Output X
    else
      We have lost alignment by one bit
      i = i + 1
      Output X on up to k_max_rep previously decoded output bits.
    end if
  end if
end for
```

For simplicity, assume that the input to the reverse transcoder is initially aligned; we will drop this assumption later. Let us now examine why this algorithm works. In the simplest case where there are no bit flips, insertions or deletions, it is easy to see that the reverse decoder gives us the correct output.

Now, let us consider what happens in the case of single bit flips. We first need to define the function $ErrorMetric(i)$. It is a look-ahead function that determines if the alignment boundary is at point i. It reads the input in blocks of 2 bits from i to $i + m$, where m is the maximum allowable look-ahead. It simply outputs the number of blocks of 2 bits it was unable to decode. For example, consider the input: 011010100110. In this case, m_0 is equal to 0, since the 2-bit blocks $01, 10, 10, 10, 01$ and 10 are all valid. On the other hand, m_1 is equal to 3, since there are 3 decoding errors in the 2-bit blocks $11, 01, 01, 00$ and 11. Since $m_0 \leq m_1$, it is more likely that the alignment boundary lies at 0, as opposed to the boundary lying at 1.

In the case of single bit flips, the reverse transcoder will receive either 00 or 11, so it knows that something has gone wrong. It checks to see whether it has lost alignment; if not, then the only explanation for the 00 or 11 is bit flipping. In this case, the reverse transcoder outputs an X, signifying that it does not know how to decode that bit; a higher layer handles these Xs. If both bits are flipped, then 01 gets converted 10 and vice versa; these errors are undetectable at this layer; such cases are handled by a higher layer CRC.

In the case of a bit insertion or deletion, the decoder might not determine that an error has taken place immediately; in fact, the reverse transcoder could potentially continue outputting up to k_{max_rep} bits. Take for example the input stream $001111110\cdots$, which is transcoded to $010110101010101001\cdots$; assume that $k_{max\_rep} = 6$ pre-transcoded bits. Suppose this bit stream is demodulated at the receiver with a 0 bit insertion after the first 4 bits: $0101010101010101001\cdots$. The reverse transcoder will read

0101010101010101 to output 00000000. At this point, it encounters a 00 and determines by looking ahead that it has in fact, lost alignment. At this point, it loses confidence in the previous k_{max_rep} bits that it output, and replaces them all with X. In reality, we need not X-out all k_{max_rep} output bits; it can be proved that we only need to X-out only the previous p output bits, where p is the number of output bits ago, where a $0 \rightarrow 1$ or $1 \rightarrow 0$ transition took place in the output of the reverse transcoder. If there were no such transitions in the last k_{max_rep} output bits, then we need to X-out all of the previous k_{max_rep} output bits.

We now examine the rationale behind why *Hermes* merges the functionality of the framer and the transcoder. Consider the final output of the reverse transcoder, a string $s \in \{0, 1, X\}^*$. It breaks the stream into individual frames using the predefined preamble as a delimiter. Unfortunately, if the delimiter itself has an X, then the frames on either side of it will be lost.

Consider a frame (data + CRC) in $\{0, 1, X\}^*$. The next step is to attempt to recover the frame by filling up the Xs with the correct bits. An X could be 0 or 1 (bit flipping), $00, 01, 10$ or 11 (bit deletion) or the $NULL$ character (bit insertion). It brute-force searches through all possible values that all the Xs could take; with a very high probability, only one of them will satisfy the checksum; this is the frame that is passed up to the higher layer. We could also attempt to recover frames that are on either side of a corrupted delimiter; currently, this is beyond the scope of our solution.

4.3 Parameter Choice

We ask the question: *What are the best values of f_{base} and δ for the best performance? Hermes* uses a very simple search and estimation algorithm to find initial values of f_{base} and δ. During the progress of the actual call, these parameters can be fine-tuned on the fly.

4.3.1 Initial Parameter Set

The first step is to estimate the *frequency response* of the underlying voice channel. To do this, we choose a small number (say, 50) of unique frequencies that are uniformly spaced between the range [$300Hz$,$3400Hz$]. These frequencies are multiplexed and sent over the air with an epoch of $200ms$. Fig. 6 shows the input and output FFTs superimposed over each other.

From Fig. 6, we observe a very interesting pattern in the frequency response of a particular voice channel that we tested. In particular, in the band of approximately [$500Hz$, $1500Hz$], the data is almost completely lost. On the other hand, the range of [$2000Hz$, $2500Hz$] is carried through quite reliably. Above $2500Hz$, we can observe a few false peaks as well. The power of this observation is that with a simple test of $200ms$, we can probe the frequency response of the channel to understand what frequency components are carried through cleanly.

Let us say that f_{max} is the frequency that had the highest peak in the FFT output; this is (albeit, approximately), the frequency that the channel carries through the best. In *Hermes*, we set the initial value of f_{base} to f_{max}. To find a good initial value of δ, we use the following intuition. If δ is too large, then the memoryful codec will severely distort the waves; if we use very small values of δ, then channel noise might cause decoding errors. We find that setting δ to be approximately within $10 - 25\%$ of f_{base} works well. As long as we find reasonable initial values, *Hermes* can fine tune the parameter set on the fly.

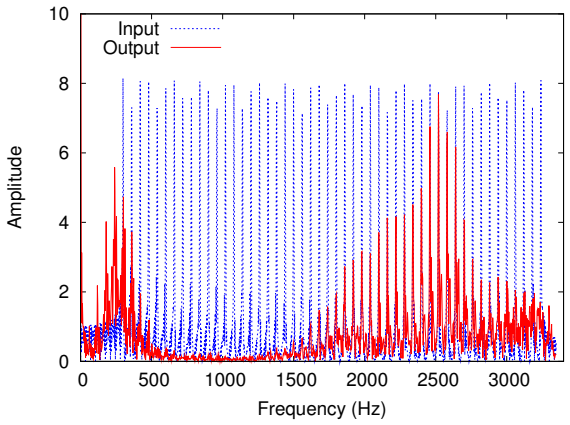

Figure 6: FFT of the input and output audio signals. Peaks are expected at $300, 360, 420, 480, \cdots Hz$. There are a few false peaks, and many missing peaks. This experiment probes the frequency response of the channel.

4.3.2 Fine Tuning

In order to understand the rationale behind our fine tuning algorithm, we need to examine how the error rate varies as a function of f_{base} and δ. We conduct an experiment over an actual phone call, where the data has been modulated using different values of f_{base} and δ. Fig. 7 plots the resulting bit error rates, as a function of f_{base} and δ. Assume that the optimal parameter values are $\hat{f_{base}}$ and $\hat{\delta}$. If we keep f_{base} fixed at $\hat{f_{base}}$ and vary δ, we observe that as δ moves away from $\hat{\delta}$, the error rate increases. Similarly, as f_{base} moves away from $\hat{f_{base}}$, the error rate increases. It illustrates the fact that once we have chosen an initial parameter set, we can vary f_{base} and δ and observe how the error rate changes. We can then use any standard *hill climbing* algorithm to find the optimal parameter set.

The advantage with this approach is that as the sender varies its parameter set, the receiver can continue to run its demodulation algorithm unchanged. The demodulator does not depend on any constants or parameters; its output only depends on the relative difference between the frequency of the current sinusoid and that of the previous sinusoid.

4.4 Overcoming VAD and AGC

To overcome VAD, we need our audio stream to *pulsing* in amplitude over long time scales. In *Hermes*, we employ a simple trick to fool VAD into believing that there is human voice on the channel. Each second of the audio stream is broken up into 2 pieces of $0.5s$ each. The first piece is left untouched; the amplitude in second piece is scaled down to 0.7 of the original. To overcome AGC, we need to make sure that the amplitude in the audio stream is not very low. The amplification factors mentioned above (1.0 and 0.7) are high enough that we do not expect AGC to kick in. However we note that in our algorithm, we do not modulate data based on the amplitude of the carrier. Therefore, even if AGC kicks in and clips some samples, the correctness of *Hermes* is not compromised. As far as VAD and AGC are concerned, the audio stream transmitted by *Hermes* is normal speech.

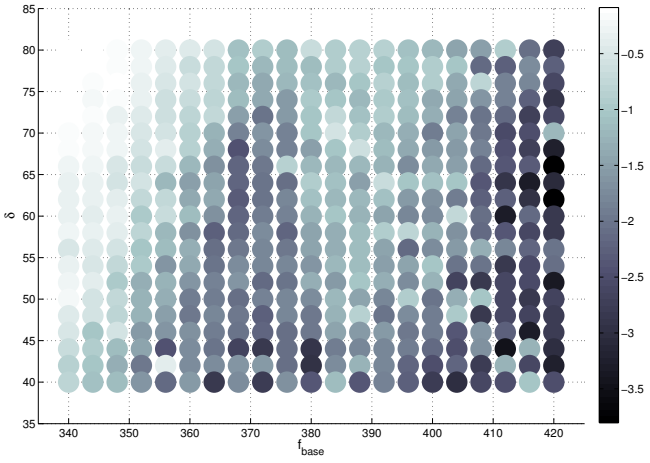

Figure 7: Logarithm (to base 10) of the error rate as a function of f_{base} (340-420Hz) and δ (40-80Hz). The landscape shows that any standard hill climbing algorithm could be applied to find local minima.

5. EVALUATION

5.1 Methodology and Implementation

The algorithms used for transcoding, modulation, demodulation and decoding are all quite simple, since we would like them to run directly on cellular phones, which are not necessarily high-end. However, since we are still prototyping and testing the protocols, these algorithms are currently implemented on regular desktop computers that interface with and use the cellular phones as the underlying physical communication channel. We are in the process of implementing the algorithms on the Android [6] platform.

5.2 Raw Channel Throughput and Error Rate

We first would like to understand what the *raw* bit error rates are for the post-transcoded stream, which is transmitted over the actual voice call. In this experiment, we operate both end points of the voice call over the cellular service provider AT&T (data from experiments conducted across heterogeneous cellular service providers is presented in § 5.2.1). Fig. 7 plots these error rates, as a function of f_{base} (340-420Hz) and δ (40-80Hz). We repeat this experiment for f_{base} in [2200Hz, 2400Hz] and δ in [300-800Hz], and observe a very similar error landscape with a large number of data points showing error rates in the order of 10^{-4} to 10^{-5}. Due to its similarity to Fig. 7, it has been omitted.

Since a $1:2$ transcoding is performed before modulation and transmission over the voice call, the approximate goodput (data rate of the original pre-transcoded stream) is $\frac{f_{base}}{2}$ bits per second. This shows that we can achieve a goodput of up to 1200 bits per second quite reliably over the voice call. Furthermore, the end-to-end frame success rate is almost 100% (see § 5.3 for a deeper analysis on frame success rates, as a function of the underlying bit error model and rate).

5.2.1 Heterogeneous Cellular Service Providers

	f_{base}(Hz)	δ(Hz)	BER
AT&T → AT&T	2200	480	1×10^{-5}
T-Mobile → T-Mobile	2400	640	1×10^{-5}
AT&T → T-Mobile	2170	470	1×10^{-5}
T-Mobile → AT&T	2130	640	1×10^{-5}

Table 3: Performance impact of heterogeneous cellular service providers. In all cases, *Hermes* finds operational parameters that provide high throughput and extremely low error rates over actual voice calls.

We repeat these experiments across sender - receiver pairs on different cellular service providers. For each pair, we present the experimentally observed optimal f_{base} and δ values, along with the bit error rate. From Table. 3, we observe that *Hermes* is able to find parameter configurations that yield very low error rates, while providing high throughput. Furthermore in all cases, we observe a very similar error landscape to that illustrated in Fig. 7; due to its similarity, it has been omitted.

5.2.2 Error Patterns

In order to further understand the bit error patterns, we consider the experiment over an actual voice call, repeated over different ranges of f_{base} and δ. We consider those configurations ($< f_{base}, \delta >$ pairs) that result in bit error rates of 10^{-3} to 10^{-4} bits per second, and analyze when exactly those bit errors took place in time. We observed that bit errors occur uniformly and independently of each other; in other words, they do not show bursty behavior. We confirmed that this behavior held true across heterogeneous service providers as well. Furthermore, we observed that although bit insertions and deletions do take place, a significant majority of errors are simple bit flips.

5.3 Effect of the Transcoding and Framing Layers

Next, we examine how much of the pre-transcoded data can be recovered, in the face of bit insertions, deletions and flips into the post-transcoded stream. Since it is difficult to control the error rates from the underlying voice channel, we conduct a simulation as follows. Break the input stream into chunks of 26 bytes, which when concatenated with the 32-bit CRC, give us packets of 30 bytes. There is a 1-byte preamble between packets, consisting of all 0s. This stream of packets (with the accompanying preambles) constitute the stream s. There are totally 10^3 packets in this experiment. Perform the 1:2 transcoding on s, to get a bit stream t. Artificially introduce errors (insertions, deletions and flips) into t to get $\hat{t}$. Pass $\hat{t}$ through the reverse transcoder to get the bitstream $\hat{s}$. The next step is to use the delimiter (preamble) to recover to frames from this stream; note that these frames may contain Xs. Finally, use the frame recovery method to fill up the Xs and pass the frame up to the higher layer if the CRC check is successful. Note that we attempt to recover only those packets that have between 1 and 4 Xs in them; if the number of Xs is large, the computational complexity to attempt recovery is prohibitively high.

To study the effect of bit flips, insertions and deletions, we consider them separately. In § 5.2.2. we have observed that the underlying bit errors are independent of each other (in other words, they are not bursty); we therefore model

	Insertion		Deletion		Flips	
	No	Yes	No	Yes	No	Yes
1×10^{-2}	0.8	4.1	1.2	3.2	1.2	34.3
7×10^{-3}	3.5	10.4	3.0	10.5	3.0	58.9
4×10^{-3}	14.4	33.6	13.6	32.3	13.6	84.9
1×10^{-3}	60.7	81.7	60.0	78.5	60.0	96.1
7×10^{-4}	70.1	86.3	70.4	85.6	70.4	97.9
4×10^{-4}	81.3	91.6	81.4	91.6	81.4	99.1
1×10^{-4}	95.1	98.6	94.8	97.9	94.8	99.8

Table 4: Frame success rate (expressed as a percentage) for different underlying bit error rates. The transcoding layer is able to guess the X values to successfully reconstruct many damaged frames. We compare against the case where no recovery of Xs is attempted (labeled "No").

the bit errors as a uniform error distribution. We repeat each experiment for a set of different error rates (fraction of bits that are flipped, inserted or deleted), and the results are presented in Table. 4, from which we observe that even under high underlying bit error rates, *Hermes* is able to correctly recover a large fraction of damaged frames. The table also shows what the frame error rate would be, if there were no attempts to reconstruct damaged frames (labeled as "No" in the table).

The performance of the frame recovery layer in *Hermes* is better under bit flip errors, as opposed to insertion and deletion errors. This is due to the fact that each bit flip creates exactly one X, while each insertion or deletion error can create up to a maximum of k_{max_rep} Xs in the reverse transcoded stream (in our implementation, we set $k_{max\_rep} = 8$). Since we aim to recover frames that have ≤ 4 Xs, we do not even attempt to recover a large number of frames under the insertion and deletion error models. If *Hermes* is run on devices with higher processing capabilities, we could attempt to recover frames with a larger number of Xs. We have also observed that the frame recovery layer in *Hermes* is able to successfully recover almost all packets with ≤ 4 Xs in them.

As observed in § 5.2.2, a majority of errors are simple bit flips. From Table. 4, we can see that the transcoding and framing layer in *Hermes* significantly improves the frame success rates under the bit flip model, even when the underlying bit error rate is in the order of 10^{-3}. For example, for error rates of 1×10^{-2}, 7×10^{-3}, 4×10^{-3} and 1×10^{-3}, *Hermes* achieve an improvement in frame success rate by a factor of 28×, 19×, 6× and 1.5× respectively.

5.3.1 CRC Collisions

We ask the question: *What is the probability of a reconstructed frame passing the CRC check, while being different from the original frame?* We know that CRC32 has a Hamming distance of 4; this means that if we take a correct codeword (frame payload + corresponding CRC) and choose any $n \leq 4$ bits to replace with random values, we will always be able to detect the error. For $n > 4$ flips, this probability reduces with increasing n. During recovery in *Hermes*, each X can be expanded as 0, 1, 00, 01, 10, 11 or NULL. The simplest work-around is to upper bound the number of Xs in a frame to 2, should we attempt to recover it; the Hamming distance between any two valid codewords will be at most 4, and therefore all errors will be detected. However, by attempting to recover frames with up to 4 Xs in them, we significantly improve the frame success rate. In this case, the Hamming distance between any two valid code-words is at most 8. While we expect the CRC to catch a very high fraction of these errors, it is not guaranteed. We can implement any block coding mechanism (such as Reed-Solomon) at a higher layer to detect and correct such errors.

	f_{base}(Hz)	δ(Hz)	BER
AMR → AMR	2340	420	1.6×10^{-4}
AMR → EFR	2280	645	1.8×10^{-3}
EFR → AMR	2280	630	1.9×10^{-3}
EFR → EFR	2265	585	2.6×10^{-3}

Table 5: Performance impact of heterogeneous codecs on operational parameters and performance of *Hermes*. Irrespective of the codecs used, *Hermes* is able to achieve low error rates, with high throughput.

5.4 Effect of Heterogeneous Codecs

We observed that AT&T and T-Mobile force the phones to operate over the AMR codec. We suspect the reason is that since we are in a highly dense urban environment, the carriers can force the AMR codec to operate on lower bitrate (and hence, lower quality) modes, in order to optimize base station capacity. Our attempts to test *Hermes* over the EFR codec failed, since AT&T and T-Mobile have disabled the phone codes to change the codec. We therefore conduct a simulation using a software implementation of the EFR codec and the default mode of the AMR codec. For different pairs of codecs used at the senders and receivers, we present the optimal performance numbers in Table. 5. The first observation is that irrespective of the codecs used, *Hermes* is able to achieve low error rates, with high throughput. The next observation is that depending on the codecs used, the optimal operational parameters are different.

Another important observation we make is that the optimal parameter set for one set of codecs might significantly under-perform for another set of codecs. For example, for EFR → EFR, the optimal parameter set is ($f_{base} = 2265Hz$, $\delta = 585Hz$). If we use the same parameters for AMR → AMR, we get a high bit error rate, of approximately 5.6×10^{-2}. The implication is that we cannot train or determine the symbols beforehand, since we do not know what codecs (or codec modes) are going to be used.

It is important to observe that the real world performance numbers across heterogeneous cellular service providers (Table. 3) are different from the simulation results for heterogeneous codecs presented in Table. 5; in fact, we are able to achieve significantly lower bit error rates in the real world. For example, the AT&T → T-Mobile call was probably in fact an AMR → AMR call. This disparity can be explained due to the fact that we simulate over just one mode of AMR, whereas in the real world, the AMR codec might keep adjusting its parameters on the fly for optimal performance.

5.5 Economic Analysis

The average time taken to send an SMS message is about 5 seconds; since the payload of an SMS is 140 bytes, this works out to a data rate of 224 bits per second. When compared to SMS, *Hermes* therefore provides a 5× improvement

in throughput. In most regions in the developing world, the per-minute cost of a voice call is approximately equal to the cost of sending an SMS message. Considering framing overheads and retransmissions, *Hermes* still provides an improvement of more than 50× in terms of cost per byte.

6. RELATED WORK

There have been a few prior efforts on transmitting data over a GSM voice channel. Each of these approaches suffer from one or more of the following drawbacks: i) The symbols are trained over a particular codec and therefore will not work over unknown voice channels, ii) the details of the modulation process are not published in the public domain or iii) the throughput achieved is too low, iv) the resulting bit error rate is simply too high to be useful, or v) there are only simulation-based results without a real-world implementation.

Katugampala et. al., [11, 10, 9, 8] construct a modem for low bit rate speech channels that transforms data to speech like waveforms to pass through the GSM EFR voice channel while avoiding VAD. However, the details of their algorithms and modulation techniques are not published in the public domain. LaDue et. al., [13] design a data modem for the GSM voice channel by training over the EFR codec, where they use evolutionary optimization to construct a symbol space that is successfully sent over the air. The symbol dictionary is constructed by generating an initial symbol set; the data is fed into the vocoder and the symbols that can be decoded on the receiver side are considered as the fittest symbols. New symbols are produced to update the symbol set and the process is repeated until a robust set of symbols are constructed. Due to the training of symbols over EFR, their approach might not work over any generic and unknown voice channel. Chmayssani et. al., [5] use QAM to modulate the data and pass it through an EFR channel to achieve low error rates. However, their results are purely simulation-based. Kotnik et al [12, 14] modulate data based on the autoregressive speech model and pass it through HR, FR and EFR codecs. However, they achieve very low throughputs and high error rates; for example at 533*bps*, the BER is over 25%. Recently, Sayadiyan et. al., [16, 17] use speech parameters to design a symbol table and send the resulting data through the EFR channel. While they are able to achieve low bit error rates, the data rates are low; furthermore, theirs is a simulation-only result. Tyrberg [18] provides a detailed survey of the field of data transmission over speech coded voice channels. AqLink [2] (from Arbiquity) is the closest to *Hermes* in terms of operating over a wide variety of heterogeneous cellular channels. However, it is a closed and proprietary system; the algorithms and modulation techniques are not published in the public domain.

7. CONCLUSION

The problem of data transmission over unknown voice channels has several important implications on enabling new mobile services in rural developing regions where no data connectivity solution exists today. In this paper we have presented *Hermes*, a point-to-point data transmission protocol that provides 1.2 kbps with very low error rates (bit error rates in the order of 10^{-5} and frame error rates of $< 1\%$), over unknown voice channels. Using extensive analysis, we show that *Hermes* works well across a wide variety of voice channels and is robust in the face of bit flips, bit insertions and bit deletions caused by voice codec distortions. The frame recovery mechanism in *Hermes* also significantly improves frame success rate by up to a factor of 28×. To the best of our knowledge, this is among the first works in this space, which also provides a new data connectivity solution that when compared to SMS, improves throughput by a factor of 5×, while lowering the corresponding cost-per-byte by over a factor of 50×.

Acknowledgments

We would like to thank Kentaro Toyama, Bill Theis, Venkat Padmanabhan and Ram Ramjee from Microsoft Research India, Dennis Shasha, Russell Power and Yair Sovran from NYU and Lu Ruan from Iowa State University for their constant encouragement and fruitful discussions. Finally, we thank the anonymous reviewers and our shepherd Sachin Katti for their comments, which helped us substantially improve this work.

8. REFERENCES

[1] Africa's SMS crisis - and how to stop it. http://venturebeat.com/2009/09/25/africas-sms-crisis-and-how-to-stop-it/.

[2] Airbiquity's aqLink. http://www.airbiquity.com/pdf/whitepapers/aqLink_Overview.pdf.

[3] Cell phone statistics in africa. http://www.revivaltimes.com/cellphonestats.html.

[4] SMS costs in Africa. http://manypossibilities.net/2009/04/sms-costs-in-africa-2008/.

[5] Chmayssani, T., and Baudoin, G. Data transmission over voice dedicated channels using digital modulations. In *Radioelektronika 2008*.

[6] Google Android. http://www.google.com/android.

[7] International Telecommunications Union - Statistics. http://www.itu.int/ITU-D/ict/statistics/.

[8] Katugampala, N., Al-Naimi, K., Villette, S., and Kondoz, A. Data Transmission. http://www.wipo.int/pctdb/en/wo.jsp?WO=2005109923.

[9] Katugampala, N., Al-Naimi, K., Villette, S., and Kondoz, A. Real time end to end secure voice communications over gsm voice channel. In *13th European Signal Processing Conference*.

[10] Katugampala, N., Al-Naimi, K., Villette, S., and Kondoz, A. Real time data transmission over gsm voice channel for secure voice and data applications. In *2nd IEE Secure Mobile Communications Forum* (2004).

[11] Katugampala, N., Villette, S., and Kondoz, A. Secure voice over gsm and other low bit rate systems. *IEE Seminar Digests 2003*, 10059 (2003), 3–3.

[12] Kotnik, B., Mezgec, Z., Svečko, J., and Chowdhury, A. Data transmission over gsm voice channel using digital modulation technique based on autoregressive modeling of speech production. *Digit. Signal Process. 19*, 4 (2009), 612–627.

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

[14] Mezgec, Z., Chowdhury, A., Kotnik, B., and Svečko, R. Implementation of pccd-ofdm-ask robust data transmission over gsm speech channel. *Informatica 20*, 1 (2009), 51–78.

[15] Mishra, S. M., Hwang, J., Moazzami, R., Subramanian, L., and Du, T. Economic analysis of networking technologies for rural developing regions. In *In 1st Workshop on Internet and Network Economics* (2005).

[16] Rashidi, M., Sayadiyan, A., and Mowlaee, P. A harmonic approach to data transmission over gsm voice channel. In *ICTTA 2008*.

[17] Shahbazi, A., Rezaie, A., Sayadiyan, A., and Mosayyebpour, S. A novel speech-like symbol design for data transmission through gsm voice channel. In *ISSPIT 2009*.

[18] Tyrberg, A. Masters Thesis: Data Transmission over Speech Coded Voice Channels, 2006.