SqueakyChat: Ultrasonic communication using commercial notebook computers

Harvest Zhang
hlzhang@princeton.edu

Bonnie Eisenman
bmeisenm@princeton.edu

ABSTRACT

We explored the feasibility of using ultrasonic communication as a covert channel in consumer electronics products. We describe the implementation of an ultrasonic encoder and decoder, as well as a prototype chat client ("SqueakyChat") used to demonstrate this functionality. We were able to sustain a transmission rate of 1 byte per second for arbitrary binary data, and we obtained up to 99.5% accuracy when broadcasting across 0 feet, and up to 81.6% accuracy when broadcasting across 10 feet. While this throughput is still quite low, we have demonstrated that for small messages ultrasonic communication is quite feasible using standard microphones and speakers shipped in consumer electronics. We then discuss limitations of this type of communication as well as potential use cases.

Keywords

#squeeeeeeak #covert-channels #furby #whyismydogbarking

1. INTRODUCTION

Following media coverage of BadBIOS[3], a piece of malware which allegedly communicated via ultrasound, ultrasonic communication has received renewed attention. In this project, we sought to explore the potential use of ultrasonic communications using standard consumer electronics – in our case, a pair of mid-2010 Macbook Pros.

Our project consists of three main elements: an ultrasonic encoder, decoder, and SqueakyChat, a chat client built as a demo of our project, which sends and receives ultrasonic messages in real time. SqueakyChat is able to transmit messages with high reliability for distances of up to about 15 feet. Although SqueakyChat demonstrates transmission of readable text, our encoder and decoder can handle arbitrary binary data as well, as the data is sent as a binary bitstream.

1.1 Motivation

We were interested in exploring the feasibility of ultrasonic communication as a covert channel in standard consumer hardware. Consumer microphones are optimized (understandably) to detect audible noise, and so their performance at ultrasonic or near-ultrasonic ranges may be severely lacking. Similarly, consumer-grade speakers' ability to reliably broadcast high-frequency audio is also limited. Thus, in order to utilize ultrasonic communication as a covert channel, a very narrow range of frequencies are available.

This covert channel could be useful in a number of scenarios. On mobile phones, microphone and speaker permissions may be granted with much less scrutiny compared with standard network and data access requests. Additionally, ultrasonic could be used to evade standard airgapping practices, which may not include removing sound cards or microphones. Ultrasonic can also function in scenarios where network access is undesirable, such as at crowded conferences or events, or in cheap electronics hardware where some networking is required but full Bluetooth or WiFi compatibility is overkill. It could also be used to implement rudimentary location tracking, for example, by having an infected device routinely broadcast over ultrasonic until it encounters an ultrasonic receiver.

2. IMPLEMENTATION

2.1 Constraints

The available frequency range for ultrasonic communication in commercially available notebook computers is affected by several factors: range of human hearing, bitrate of audio recording and streaming, and the frequency response of both the sending speaker and the receiving microphone.

Humans are typically capable of hearing sounds roughly in the range of 20 Hz to 20 kHz, and as children grow up their upper limit for hearing decreases to typically around 18-19 kHz [12]. Thus, we set an initial lower limit of signal frequency at 19 kHz, and then adjusted the frequency until few people were capable of detecting it played at full volume on a Macbook Pro. Frequencies over about 19,500 Hz were generally inaudible.

The sampling rate of the microphone provides a hard upper bound to the frequencies we can accurately detect. While the Macbook Pro will support sampling rates up to 48 kHz, 44.1 kHz is a much more commonly used standard across devices and CDs. By the Nyquist-Shannon Sampling Theorem [13], this means that the highest frequency signal that

can be perfectly reconstructed without aliasing is a little over 22 kHz. As it turns out, this hard upper bound is not the limiting factor, since the design of speakers and microphones usually does not feature a frequency response worth mentioning in that frequency range.

We were not able to find any definitive measurements of the frequency response of either the Macbook Pro's speaker or built-in microphone; therefore, we experimented with the ability to send and receive signals at high frequencies and met with success until just over 20 kHz, where either the speaker or microphone sharply falls off in response. Therefore, we treated 20 kHz as the upper bound of frequencies at which we could transmit.

2.2 Encoding

2.2.1 *Frequency Shift Keying*

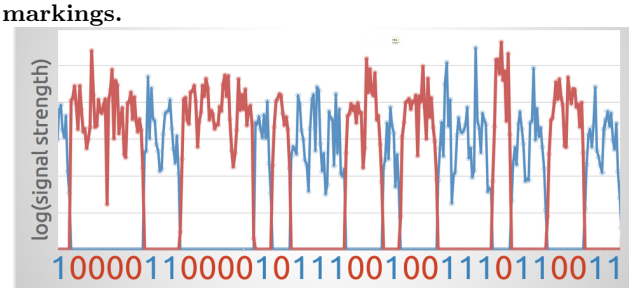
Frequency Shift Keying (FSK) is an encoding scheme wherein each bit value is represented by a specific frequency [12], and the modulation in the signal between the frequency corresponding to 0 and the frequency corresponding to 1 is decoded by the receiver to reconstruct the original bitstream.

Our encoding scheme uses a modified version of FSK. We add an additional frequency used to mark the start of a new character, since our application is more interested in bytes rather than specific bits. Thus, at the beginning of each sequence of eight bits, we insert a byte start signal. This addition was added due to the empirical difficulty of character segmentation; as soon as one bit is missed while decoding or an erroneous bit is inserted, all future bytes will be shifted off from the correct segmentation of bits into bytes, and so the rest of the message will produce gibberish bytes. The byte start signal thus also provides a form of error correction; if we do end up missing a bit or adding a bit here or there, the next presence of the byte start signal indicates that we should start the next byte at that point, even if the last byte has not been fully filled yet. Other methods for error correction may be applied for applications that do not need this kind of per-byte recovery.

The frequencies chosen after much experimentation were 19,600 Hz for the character-start marker; 19,800 Hz for 0 values; and 20,000 Hz for 1 values. This is largely above the range of human hearing and under the point where speakers and microphones can no longer send and transmit sound reliably; the separation of 200 Hz was also chosen as the minimum frequency separation required such that the signals do not bleed power into each other. Audacity, which has the ability to examine a recorded section of audio's frequency transform, shows the level to which signals sent at frequencies bleed into neighboring frequencies when recorded and reconstructed; smaller separations, like 150 Hz, led to coupling between the signals and therefore extremely unreliable signal reconstruction.

To encode data, a `.wav` file is constructed using sine waves of length 0.1 seconds generated at each signal frequency, scaled up to an amplitude of -32,000 to 32,000, which almost maxes out the volume of a 16-bit audio signal. This signal is then concatenated and either streamed or saved to a .wav file on disk, where it can be played back as a sound file later or broadcast immediately within realtime "chat" style applications, like our prototype SqueakyChat client.

Figure 1: An example of a raw bitstream encoded by our program, showing the relative powers of 0- and 1-bit signals as interpreted by the Goetzel algorithm. This one does not include character-start markings.



2.2.2 *Audible click elimination*

Initially, when attempting to play back encoded ultrasonic sound files, we noted that each transition between different ultrasonic frequencies (especially when transitioning from a tone back to silence) produced a highly audible clicking noise. This was obviously a problem, since it defeats the purpose of using ultrasonic frequencies to ensure silent data transmission. This phenomenon occurs due to the abrupt changing of the speaker's motion during such a transition, creating either a cusp in the waveform or a complete discontinuity. The click is broadband noise caused by this cusp; the sharper the cusp or larger the discontinuity, the louder and more obvious the click. Merely making sure that the end of the sine wave occurs at a signal level near 0 before transitioning to the next frequency or silence does not help, since despite removing the discontinuity (responsible for the loudest clicks), there remains a cusp in the speaker's motion that causes the speaker to generate a quieter but still audible clicking noise.

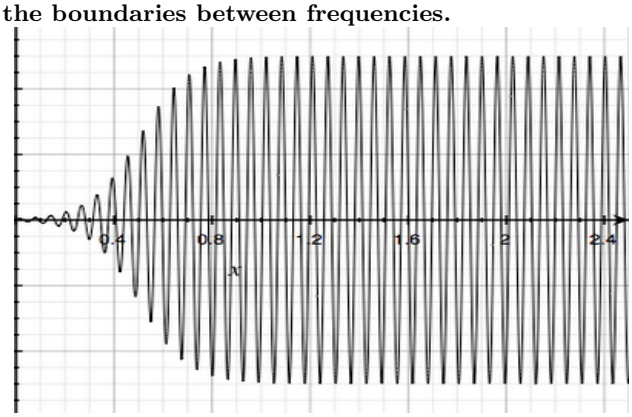
To mitigate this effect, we applied a fade in and fade out for each tone using a sigmoid function as the envelope. A linear ramp or something simpler may have also worked for longer ramp times, but a sigmoid function is asymptotic at 0 and 1 and provides a very smooth transition that can be achieved in fewer samples. Thus, the beginning and end of each bit both feature a sigmoid envelope. In practice, this approach successfully eliminated the clicking noise, and we were able to then minimize the fadeout time to a negligibly small period (27 milliseconds end to end) while still avoiding any audible artifacts.

2.3 Decoding

PyAudio [4] was used to establish an input audio stream, using a buffer size of 2048 samples. We read in 512 samples at a time, where each sample is represented as a 16-bit signed integer between -32768 and 32767. A Hamming window is applied in order to prevent artifacts from distorting frequency space transforms [15].

We next determine the signal power at each frequency of interest by implementing the Goertzel algorithm [11], which efficiently returns the component of the Discrete Fourier

Figure 2: A sigmoid envelope used to round off each sine wave's beginning and end. This was extremely effective at eliminating clicking sounds on the boundaries between frequencies.



Transform at a specified frequency. A Python implementation of the Goertzel transform is as follows:

```
def goertzel(frequency, audio):
  prev1 = 0.0
  prev2 = 0.0
  norm_freq = frequency / bitrate
  coeff = 2 * math.cos(2 * pi * norm_freq)
  for sample in audio:
    s = sample + (coeff * prev1) - prev2
    prev2 = prev1
    prev1 = s
  power = (prev2 * prev2) + (prev1 * prev1)
          - (coeff * prev1 * prev2)
  return int(power)
```

For each chunk of 512 samples, we compute the signal power using the Goertzel algorithm for a baseline frequency (19,400 Hz), the byte start frequency (19,600 Hz), the 0 signal frequency (19,800 Hz), and the 1 signal frequency (20,000 Hz). The three latter frequency powers are then normalized, divided by the baseline frequency power in order to cancel out a significant amount of noise in the signal, since noise at 19,400 Hz tends to be similar to the noise at the three other frequencies of interest.

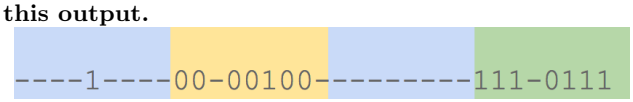
We experimentally determined thresholds for which, if any, signal was present during each chunk. As the power of each signal rose dramatically (several orders of magnitude in a quiet environment) when that signal was on, the threshold was generally highly effective; however, noisy environments would sometimes cause the recorded signal at high frequencies to partially drop out. The result is a value in `{0, 1, byte start, no signal}` for each chunk of 512 samples, which at a bitrate of of 44,100 Hz results in one value found approximately every hundredth of a second. This Goertzel output stream is now passed through a sliding window bit extraction algorithm to recover the original bitstream.

2.3.1 *Sliding window bit extraction*

We utilize a segmentation algorithm developed in Bonnie's Independent Work project in spring 2013, originally designed

Figure 3: The decoder pipeline. The raw audio signal is passed through the Goertzel algorithm and then through a sliding window bit extraction algorithm, coming out the other end as the reconstructed bitstream originally sent – hopefully.

Figure 4: Example of data from the Goertzel output stream and color-coded ideal bit segmentation for this output.



to detect lines in images of text from a 14th-century French manuscript, in order to recover the original bitstream from the Goertzel output stream. This algorithm utilizes a greedy approach based on knowledge of how large each segment ought to be. It attempts to find the window with the "purest" (e.g. most consistent) content at each step, then consumes that portion of data and proceeds.

Character-start markers, as discussed in the encoding section, are crucial to the success of this method. Without character-start markers, a single error in segmentation can cause a "chain reaction" of misaligned bits within characters.

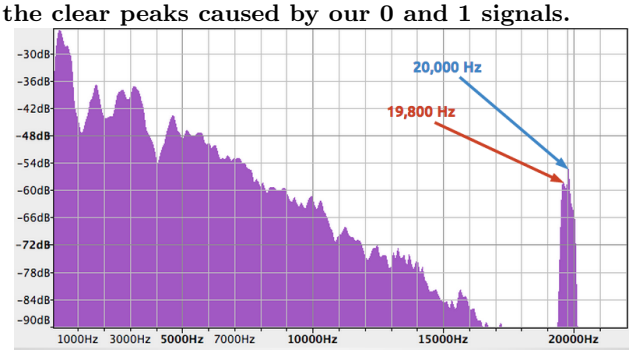
```
Start @ beginning
  for i in range(duration/2):
    Calculate "pureness" of window beginning @ i
Pick best window, slide forward, repeat
```

This approach to segmentation has many benefits. First, it

allows us to operate with a very small buffer, enabling us to process incoming audio data in a streaming fashion without the end user noting any processing delay. Secondly, it has extremely high accuracy and a very quick recovery time. Theoretically, because the potential window alignment range is set to be half the window size, erroneous window selection should be recovered from in 1-2 cycles; in practice, we saw typical recovery times of 1 iteration. Additionally, unlike some other segmentation schemes, it does not require us to insert silence or a third frequency between re-transmission of identical bits (i.e. '11' or '00'), which helps us increase our transmission rate. This is made possible because we have foreknowledge of what the bit duration ought to be.

3. EVALUATION

Figure 5: Result of a Fourier Transform on audio recorded during an ultrasonic transmission. Note the lack of noise in the range we are targeting, and the clear peaks caused by our 0 and 1 signals.



We evaluated our ultrasonic encoder and decoder using Squeaky-Chat, a command-line application which takes user text as input and prints received transmissions. We used Levenshtein distance [14] to measure our accuracy, and tested transmission using a 631-character sample of text taken from the introduction to A Tale of Two Cities, on the premise that this represented fairly typical English text. We considered transmitting characters as an acceptable proxy for transmitting other types of arbitrary binary data, because errors manifest similarly. Additionally, using English text allows us to easily apply error-counting algorithms, such as Levenshtein distance, which we use here.

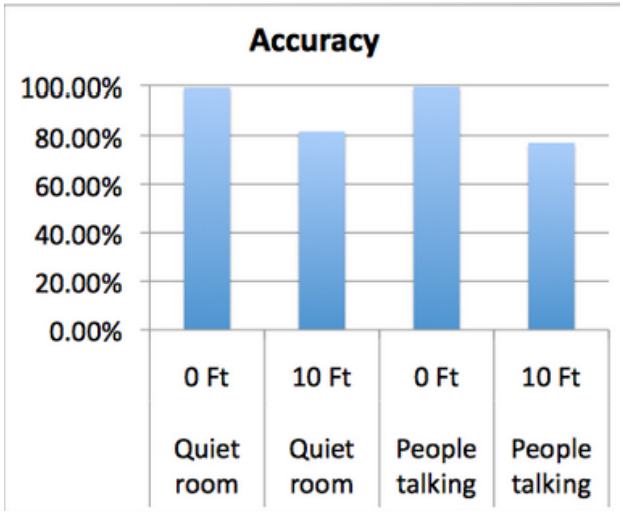
3.1 Throughput and reliability

We were able to reliably transmit data at a rate of 1 byte per second without too many errors. Lower bitrates do demonstrate lower error rates, but not significantly enough to be worth the decrease in throughput, while higher bitrates showed a precipitous drop in reliability.

When broadcasting to a quiet room, we observed accuracy rates of 99.1% with a distance of 0 feet, and 81.6% at 10 feet. To test performance in a noisy environment, we used http://coffitivity.com[1] to play back recorded coffee shop noise at a high volume (occasionally interspersed with actual conversation by several people in the room) in a room with a high amount of echo. This had only a modest impact on our accuracy rates. At 0 feet in the noisy room, we demonstrated 99.5% accuracy (better than the values for the quiet room, oddly); at 10 feet, the rate dropped to 77.0%. "0 feet" represents results from recording and broadcasting audio using the same laptop, and provides a good baseline for measuring other results.

We also observed that it was very important to turn off ambient noise reduction on our laptops. This included feature is designed to eliminate "noise" by segmenting what it believes is conversation from periods of silence, from which it takes its samples of background noise; given that all of our transmission would be considered background noise since it is not within the range the algorithm considers to be signal, the ambient noise reduction profile likely featured elements of our transmitted signal and then filtered them out accordingly. This is yet only a working hypothesis and more work would have to be done in order to determine the precise cause of the drop in reliability caused by this setting.

Figure 6: Results of our evaluation, demonstrating our accuracy when transmitting a 631-character piece of English text.



This implies that volume and distance are far greater limitations on our transmission than background noise, which has a much more modest impact. More interesting was the finding that most continuous ambient noise, such as people chattering, does not present issues to reliability, probably since most of these sounds occur in a frequency range well lower than our signal range. Rather, noises that are loud and brief, such as people dropping items or knocking on doors, caused far greater disruption. One hypothesis for this phenomenon is that these noises more easily cause clipping, where the microphone is driven to its maximum excitation and therefore the signal waveform is "chopped off." Much like the broadband excitation caused by the cusps and discontinuities in the output waveform, this chopping off of the input waveform results in a perceived broadband excitation and distortion across the entire frequency range.

One potential solution to this would be to turn down the sensitivity level on the microphone, thus making it less likely that clipping and distortion will occur. However, with our current signal transmission volume (already set as high as possible on the Macbook Pro), turning the sensitivity down to levels where dropping something heavy on the floor won't

cause clipping means that we also can't detect the ultrasonic signal strongly enough. There are commercially available microphones designed for high quality audio recording capable of withstanding a greater dynamic range without clipping; these are very seldom seen on laptops or phones, however. Speakers that can also broadcast more loudly at ultrasonic frequencies would also help.

3.1.1 The bitrate-reliability tradeoff

There are a few major factors that cause the current design to be limited to about a byte per second of throughput: the accuracy of the Goertzel algorithm, the sampling rate of the recording stream, and the need for a sufficient window size in order to successfully extract the bitstream from the Goertzel output stream.

The Goertzel algorithm needs a sufficient number of samples to be able to accurately determine the signal power at a given frequency. In our testing, giving it 512 samples reliably returned powers that corresponded with the signal currently being broadcast, while using smaller values such as 256 samples resulted in much higher variation. As long as we feed the Goertzel output to the sliding window algorithm, which also serves to denoise the signal, there is no noticeable advantage to providing the sliding window algorithm a much noisier but more frequent signal compared to a less frequent signal that is usually correct. Therefore, in this configuration, each Goertzel output bit takes a little over a hundredth of a second given a sampling rate of 44,100 Hz, and as discussed, the maximum rate at which an input stream can sample is not much more than that; using 48,000 Hz may provide incremental improvements for devices capable of sampling at that frequency, but it will not cause a significant increase in bitrate.

The size of the window in the sliding window bit extraction algorithm does, however, play a significant role in the bitrate of the overall system, since the window size is dependent on the length of each signal tone. For example, a window of size 8 requires 8 Goertzel output bits, or approximately 90 milliseconds, of continuous signal; we are providing just a little bit more than that, at 100 milliseconds per continuous signal. If we wished to speed up the transmission rate – for example, by lowering the length of each signal to 0.05 seconds – we would only have a sliding window of size 4, which experimentally produced entirely nonsensical results.

3.2 Limitations

The most obvious limitation is the extremely low transmission rate of 1 byte per second, which makes our current implementation infeasible for transmitting most file formats or even of supporting higher level layers of the networking stack. Additionally, the low throughput inhibits our ability to include error-checking codes or other features which could compensate for transmission and decoding errors. Performance is also impacted by each device's capabilities; we derived many of our values, including thresholds, empirically, making it likely that per-device calibration is necessary for different computers and phones.

Another limitation is that broadcasts by multiple parties are difficult to coordinate. Frequency Division Multiple Access (FDMA) techniques are not useful due to the narrowness of the usable frequency band and the enforced separation between signals needed to reduce interference between adjacent signal frequencies, and Code Division Multiple Access (CDMA) would require higher bitrates as well as distributed coordination in order to function. However, it is possible to create a one-to-many Time Division Multiple Access (TDMA) system [10], where – like in TCP – all devices listen and wait while they hear someone else broadcasting, and then once a "done signal" is sent, each device waits a random length of time, after which if someone else has not started broadcasting, the device may start broadcasting. With our bitrates, this would be a slow and arduous method of multiparty ultrasonic communication.

Ambient noise reduction, a built-in feature in many systems, also eliminates our ability to transmit data effectively; on systems where this is turned on by default, an attacker would need to disable it. For non-malicious use cases, the user would have to be made aware of this requirement.

Utilizing ultrasonic as a means of covert communication also relies on security through obscurity. It is extremely easy to detect the presence of ultrasonic communication once one wants to look for it; if it ever became a reasonably common method, it would be a simple task to create a tiny standalone hardware ultrasound detector or implement one in software in a mobile phone or laptop. Also, it should be noted that the frequencies we used are technically on the edge of human hearing, and are likely barely audible by small children as well as animals such as dogs, making it more likely that a bystander could notice the presence of our data transmission.

3.3 Potential use cases

Given these limitations, ultrasonic communication appears to be a feasible, but limited, covert channel for data transmission. For transmitting malicious payloads, for example, both the speed and reliability are currently low enough that this seems like a measure of last resort – not to mention the fact that ultrasonic data transmission can be easily thwarted by either generating interfering ultrasonic signals or simply loud background noises. On the other hand, if an attacker merely needed to "activate" some existing payload with an ultrasonic "trigger," it seems that this would be extremely easy to implement without being detected, as this could be done with a very short message.

Ultrasonic communication could also have applications in non-malicious scenarios where messages need to be broadcast to many people. For example, in a conference where there is too much stress on the network for WiFi to work reliably, organizers could distribute an app that can detect messages broadcast over the PA system using ultrasonic encoding.

This could also be an extremely useful technology for "Internet of Things" (IoT)[8] products. IoT envisions many interconnected, everyday devices which communicate with both each other and the Internet, ranging from lights to toasters to toys. Audio receiving and transmitting equipment are extremely cheap, especially compared with expensive Bluetooth and WiFi hardware. Additionally, most IoT products don't require high fidelity or throughput, making them a

perfect potential use case for ultrasonic data transmission.

4. FUTURE WORK

Improving the transmission rate would be a priority in order to make ultrasonic communication practical for use. One potential way to do this would be to very carefully adapt existing multiple-frequency transmission protocols. The most well known example would be Dual-Tone Multi-Frequency signaling, which is used in touch-tone telephones [9] for dialing; however, in our experiments we had difficulties differentiating between signals when using frequencies fewer than 200 Hz apart, so we may need to experiment with the number of possibly simultaneous signals we can cram into a small range of frequencies. Also, simply playing two signals simultaneously – say, 19,200 Hz and 20,000 Hz – will cause a very audible beat frequency to occur due to constructive and destructive interference between the multiple signals, so a feasible multiple-frequency scheme would have to either use signals without audible beat frequencies (which are thus too far apart for use) or do some other trickery (e.g. quickly alternating between all simultaneous frequencies for short bursts) to prevent any audible sound from being produced. This appears quite difficult to achieve without creating audible artifacts.

Future work could also focus on improving the reliability of transmission using techniques appropriate to the type of data being transmitted. For example, in the case of our SqueakyChat prototype client, odds are high that English text is being transmitted; we could utilize a Markov Model or other, more complex predictive text models to correct spelling errors on the fly. One could also add standard error-checking codes and experiment with existing data-transfer protocols which include redundancy and fault-tolerance.

Our current implementation uses Python, making it unsuitable for use on mobile devices. It would be interesting to test the limits of common Android phones and iPhones to determine how well they can detect and generate ultrasonic signals. It would also be interesting to conduct a social experiment to determine how readily common users as well as security-conscious experts grant microphone and speaker permissions compared to network and data permissions.

5. RELATED WORK

A variety of projects have demonstrated limited ability to communicate via ultrasound, though we observed that most open-source projects have focused on strictly limiting the range of possible transmissions in order to achieve greater accuracy.

Shopkick is a commercial product designed for retail stores. Customers download the Shopkick app to their smartphone, which then initiates ultrasonic communication with devices installed in the retail store. The customer then receives discounts and other special offers based on their physical visits to the store. Interestingly, because the user needn't remember to open the app to receive these notifications [6], the app is likely operating constantly in the background of the user's phone.

BadBIOS is a much-debated piece of malware that allegedly evaded standard airgapping techniques using ultrasonic communication. Researcher Dragos Ruiu claimed that after removing networking equipment from two computers, the malware used ultrasonic networking to repair itself as he attempted to remove it [3]. While this claim has not been verified by other security researchers and is still much-debated, the case set off a burst of interest in ultrasonic communication.

QuietNet, a similar project to ours featuring an ultrasonic-based chat client, debuted on HackerNews last week. QuietNet also uses numpy and pyaudio to generate and decode data. However, unlike our implementation, which is able to encode and send arbitrary binary data, QuietNet is limited to a fixed alphabet thanks to its encoding scheme [2]. This alphabet is limited to a small set of English letters, numbers, and symbols.

Similarly, sonicnet.js is a Javascript library for ultrasonic communication; however, like QuietNet, it mandates that users limit themselves to a very limited library and is not meant to be used for arbitrary data [7]. The demo application is a web interface which allows the user to send one of six emoticons over ultrasound – hardly an extensive means of communication!

Recent editions of the Furby toy also use ultrasonic communication to interact with the user's Android phone. Apparently this communication isn't truly ultrasonic, because the FAQ for the Furby includes a soothing message targeted at annoyed users who can hear the messages [5].

6. CONCLUSIONS

We have successfully demonstrated that standard consumer electronics are capable of both transmitting and receiving data encoded at ultrasonic frequencies, at least up until around 20,000 Hz, and that some sort of ultrasonic communication between devices never intended for such use is indeed possible. We have also explored the limitations of this medium, which are quite extensive; we are limited in our range of usable frequencies as well as by the sampling rate of our devices, causing a low transmission bitrate, non-negligible error rates, and non-ideal multiparty communication mechanisms despite fairly good multicast performance by its nature. Thus, while some improvements can be made, particularly to the reconstruction algorithms used by our decoder, we believe that it is unlikely that long-range, high-throughput, low-error ultrasonic communication using common consumer electronics with current mobile speaker and microphone technology will be useful. Neither does it seem useful for covert communication for any serious purposes, since it is quite easily and obviously detected if anyone is looking for it.

The limitations don't mean that ultrasonic communication is useless. Ultrasonic communication is a highly effective one-way, broadcast medium for devices in close proximity, a use case where existing data transfer protocols suffer from excess wireless network congestion due to the lack of useful multicast implementations. It may also prove to be an exploitable if not particularly robust way for malware on devices like phones to communicate with malicious parties without requesting suspicious permissions; this is also fairly simple to detect if searched for specifically, but it may be

harder to detect if it only broadcasts infrequently.

While our analysis is by no means exhaustive, we have demonstrated that ultrasonic communication in consumer devices is feasible, and thus presents an interesting and dynamic problem space for future commercial and academic research.

6.1 Availability

The code for this project is available publicly on Github. We also intend to publicize our work once we have cleaned up the codebase.

`https://github.com/bonniee/ultrasonic`

7. ACKNOWLEDGEMENTS

We would like to acknowledge Professor Edward Felten, as well as Harvest's roommates, David and Dodam, for enduring our tests as well as providing some helpful background noise during the relevant portion of the evaluation.

8. REFERENCES

[1] "Coffitivity". `http://coffitivity.com/`. Accessed: 2014-01-14.

[2] "GitHub: quietnet". `https://github.com/Katee/quietnet`. "Accessed: 2014-01-14".

[3] "Meet badBIOS, the mysterious Mac and PC malware that jumps airgaps". `http://arstechnica.com/security/2013/10/meet-badbios-the-mysterious-mac-and-pc-malware-that-jumps-airgaps/`. Accessed: 2014-01-14.

[4] "PyAudio". `http://people.csail.mit.edu/hubert/pyaudio/`. Accessed: 2014-01-14.

[5] "Reverse Engineering a Furby". `http://poppopret.org/2013/12/18/reverse-engineering-a-furby/`. Accessed: 2014-01-14.

[6] "Shopkick: ShopBeacon". `http://www.shopkick.com/shopbeacon`. Accessed: 2014-01-14.

[7] "Sonicnet.js". `https://github.com/borismus/sonicnet.js`. "Accessed: 2014-01-14".

[8] K. Ashton. "That 'Internet of Things' Thing, in the real world things matter more than ideas". *RFID Journal*, 2009.

[9] A. Z. Dodd. *"The essentail guide to telecommunications."*. Prentice Hall PTR, 2002.

[10] D. D. Falconer. Time division multiple access methods for wireless communications. *IEEE Communications Magazine*, January 1995.

[11] G. Goertzel. An algorithm for the evaluation of finite trigonometric series. *The American Mathematical Monthly*, 65(1):pp. 34–35, 1958.

[12] H. F. A. Olson. *Music, Physics and Engineering.* Dover Publishing, 1967.

[13] C. E. Shannon. Communication in the presence of noise. *Proc. Institute of Radio Engineers*, 37(1), 1949.

[14] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, Jan. 1974.

[15] E. W. Weisstein. "Hamming Function." From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/HammingFunction.html`. Accessed: 2014-01-14.