16 May

- Joined SecGroup Slack.

- Looked for download/source code for SILK codec and found September 2010 version in IETF document cited by Muffler paper (https://tools.ietf.org/html/draft-vos-silk-02), as well as March 2012 version 1.0.9 on GitHub (https://github.com/ploverlake/silk). Current internet standard codec, Opus, is based on September 2010 version of SILK, and the original Muffler work was done with that version, so it is probably better to use even though it is older. Unfortunately, it will be a pain to copy from the IETF document and build since it is no longer available in a nice package to download from anywhere. Alternative would be to use Opus, but then we have to re-do earlier Muffler work.

- Started reading technical details of SILK in IETF document. Trying to figure out where the default 8 different packet sizes come from, as noted in the Muffler paper. Looks to me like 5 different packet sizes, each 20 ms frame, between 1-5 frames per payload, every 20-100 ms.

- SILK applies two high-pass filters to the input, attenuating frequencies below 70 Hz, so any low frequency noises added will be stripped away.

- Looked up a lot of digital signal processing terminology on Wikipedia.

- Connected to muffler.cs.georgetown.edu

- Whitening filter might strip away high frequency noises that aren't associated with voice. This process has something to do with identifying segments that contain speech and segments that don't, so the obvious solution of adding a steady background tone at any frequency is probably not going to succeed.

17 May

- Continued reading SILK technical details.

- Cloned and built SILK version 1.0.9 from GitHub just to get started with something for now.

- Looked at LDC dataset, unzipped into my directory. Noticed that dataset is all .wav files but SILK reads .pcm files as input (although .wav is an implementation of PCM) so I might have to look into converting them… Apparently just renaming the file extension from .wav to .pcm creates an acceptable input file, so that would be easier. I tested this by renaming the file, opening it in Audacity, and listening to it, and that worked OK, and SILK didn’t complain when I used it as input.

- Trying to decode the output now, and Audacity throws an error saying that it did not recognize the type of file, and suggests importing it as raw data. When I import it, the graph looks right but it sounds like chipmunks when I play it. Importing it as Signed 16-bit PCM is best (I can hear the words) but the speaker still sounds like a chipmunk. Using a playback speed of 0.5 fixed this, and I could hear the original audio clearly. But what does that mean? Importing with a sampling rate of 22050 (half the default option of 44100) fixed this, and playback was fine at 1.0 speed.

- Now that I can successfully encode/decode files with SILK, the next target is figuring out how to analyze its output in terms of packet sizes. I also need to look at ways to manipulate the .wav/.pcm files.

18 May

- Waiting to see Brad Moore's code, hoping it will shine light on how to see SILK output packet sizes.

- Experimented with Audacity to hear what different audio effects do. Accomplished most basic task of creating a white noise track, mixing it with a speech sample, and saving the new mix track to try running through SILK.

- We can apply a high-pass filter in Audacity, so that might help us preview what SILK will do internally to input audio that we might modify.

- Read through a quick tutorial on using SOX, and skimmed a tutorial on the Java Sound API to see what might be useful.

22 May

- Set up ssh-key authentication from my machine to muffler.cs.georgetown.edu

- Continued working through Java Sound API tutorial to figure out how to load a file from our dataset and start doing things with it programmatically; still working on playback for now

23 May

- Started looking through Brad Moore's old code that Tavish found to figure out how he changed .bit files from SILK output into packet size distributions

- During meeting, decided to refocus efforts on examining the Opus codec instead of SILK due to its currency and available tools, also to continue exploring the Java Sound API as the possible tool of choice for manipulating input

- Wrote to Brad to ask about converting SILK output to abcdefgh format, since we'll have to do the same thing anyway with Opus

- Started reading IETF document on Opus (https://tools.ietf.org/html/rfc6716) as well as referenced document about security of VBR codecs (https://tools.ietf.org/html/rfc6562). Lots there to think about.

24 May

- Heard back from Brad who thinks ffmpeg had a tool to see the packet sizes of encoded audio files. Looked through ffmpeg and didn't see anything, but related tool ffprobe has a show_packets option that seems to be the key. Encoded one of our speech samples with Opus, ran the ffprobe tool, and got the size of each packet. Continuing to explore this.

- Wrote program to extract packet sizes from ffprobe output on Opus file, shown in sortedout.sizes in my directory opuswork, and it's not as simple as 8 discrete packet sizes. It appears to be continuous from 80 to 120 bytes, and then one packet with 143 bytes. This might be the same thing that Micah saw when running the RTP tool on an Opus file.

- ffmpeg no longer seems to support the output format from SILK, so I downloaded a deprecated release from 2013 that Brad might have used successfully back then. Not going well; it still doesn't appear to support SILK's output. Also tried downloading an older release of ffprobe separately, but that depends on libz which apt-get can't find.

25 May

- Asked Brad if ffprobe did in fact support SILK's bitstream format back when he was working on this. He does not remember.

- Trying to open .bit file using Java Sound API says it is not a valid format, so no clues about it there

- If Micah was able to see the same Opus file packets sizes with the RTP tool that I saw using ffprobe, maybe I can use the RTP tool to analyze the SILK .bit file. Looks like it was an option in opusenc (save-range), so that won't help with SILK files.

- Looking at modifying the SILK source code to print out each packet size before it writes to the output file. Trying to find the correct variable to print.

26 May

- Continuing to explore SILK source code. Everything that seems like the correct value (namely nBytes in SKP_Silk_range_coder.c, SKP_Silk_encode_frame_FIX.c, and SKP_Silk_decode_API.c) prints a range from about 18 to 84, nearly continuous in between, so not the 8 distinct packets sizes that we expect.

- I will continue trying to solve the SILK packet size mystery, but it seems that we might have to accept that packet sizes range from 18-84 (with SILK), or 80-120 (with Opus), assume that the previously described re-identification attack still works, attempt to redo the earlier work of encoding a bunch of samples and calculating a superdistribution over all those possible packet sizes, and then continue experimenting to make an arbitrary encoding match the new superdistribution.

30 May

- Switched gears to continue working through the Java Sound API tutorial

- Decided during team meeting to focus on replicating the re-identification attack on Opus files, choosing arbitrary bins for packet sizes. E.g. A = 80-90, B = 90-100, C = 100-110, D = 110-120, etc. I will batch encode all the samples with save range option, extract the packet sizes from that output, assign them to bins, and then create unigram/bigram/trigram feature vectors to feed into Weka and see if it can correctly identify male vs. female speakers. Later we will see if we can affect the distribution of A, B, C packets and map samples to a superdistribution (hopefully reusing the original Muffler code at that point).

- As an aside, I should also encode some files with padding/cbr option to see just how much larger the output is compared to vbr encoding.

31 May

- Started learning Python and writing script to encode all the ldc files, parse range files, and create .dat files with packet sizes mapped to 8 different bins abcdefgh in the naive way, just increments of 5 bytes between 80 and 120. Easily done, and ready to move on to producing .csv Weka input

1 June

- Network outage at my house for most of the afternoon, so I focused on reading and homework for my class

2 June

- Created Weka input file for entire ldc dataset with just unigrams to start trying something. Generated .csv file with Python script, and then converted it to .arff at http://ikuz.eu/csv2arff/. First testing accuracy for identifying speaker with logistic regression, 10-fold cross validation. Result = 16% correct vs <1% random chance (143 speakers). For male vs. female classification, accuracy was 76% vs. 50% random chance.

- Comparing cbr to vbr… Encoded one file and the cbr version was actually smaller (albeit almost negligibly) than the vbr version. Viewing packet sizes, all cbr packets were 97 bytes whereas vbr packets varied in the 80-120 range. It seems that the point of vbr is not to save space/bandwidth over default cbr, but rather to attain higher quality without

using more space/bandwidth. When using cbr with a higher target bitrate (160), the file size balloons to 4x the size of the vbr output. I don't notice any difference during playback in Audacity between the three encoded samples.

5 June

- Enhanced Weka input to include bigrams as well as unigrams, and for male vs. female classification, accuracy increased from 76% to 79%

6 June

- Simple test of mixing audible white noise with a speech sample shows a significant change to the distribution of packet sizes…

  input.range total packets : 238
  a: xxxx30
  b: xx22
  c: x9
  d: xx17
  e: xxxxxx48
  f: xxxxxxxxx69
  g: xxxx30
  h: x13

  noisyinput.range total packets : 238
  a: 0
  b: x9
  c: xxxxxxxxxxxxxx105
  d: xxxxxxxx63
  e: xxxx37
  f: xx19
  g: 4
  h: 1

- Even with a very low amplitude, barely audible white noise in the background the distribution changes by a surprising amount, but not in a very orderly or predictable way.

  low-noise-input.range total packets : 238
  a: xxxxxxx55
  b: xxxxxxxx61
  c: xx16
  d: x14
  e: xxxxx41
  f: xxxx30
  g: x11
  h: x10

- Working with Python on these scripting tasks so far has been great, so now exploring the pydub library for manipulating .wav files. Looks like you can easily do things like mix audio down to the millisecond. Wrote program to mix 20 ms of low amplitude white noise with the first 20 ms of a speech sample. Results were surprising in that Opus encoded packets of the mixed sample were all much larger than the original sample… mostly 200+ bytes. Now I see that mixed .wav file is also about twice as large for some reason. When I play back the mixed file it sounds the same. The single 20 ms frame of white noise generated by Audacity is also a 299 byte encoded packet.

7 June

- Focused on reading and homework for my class, but also somewhat stuck until we can further discuss yesterday's results at our meeting tomorrow.

8 June

- Confirmed in meeting that one "packet" is the same as one 20 ms frame, by simple division of sample length by 20 ms. So we are looking at the right values using opusenc --save_range and ffprobe --show_packets.

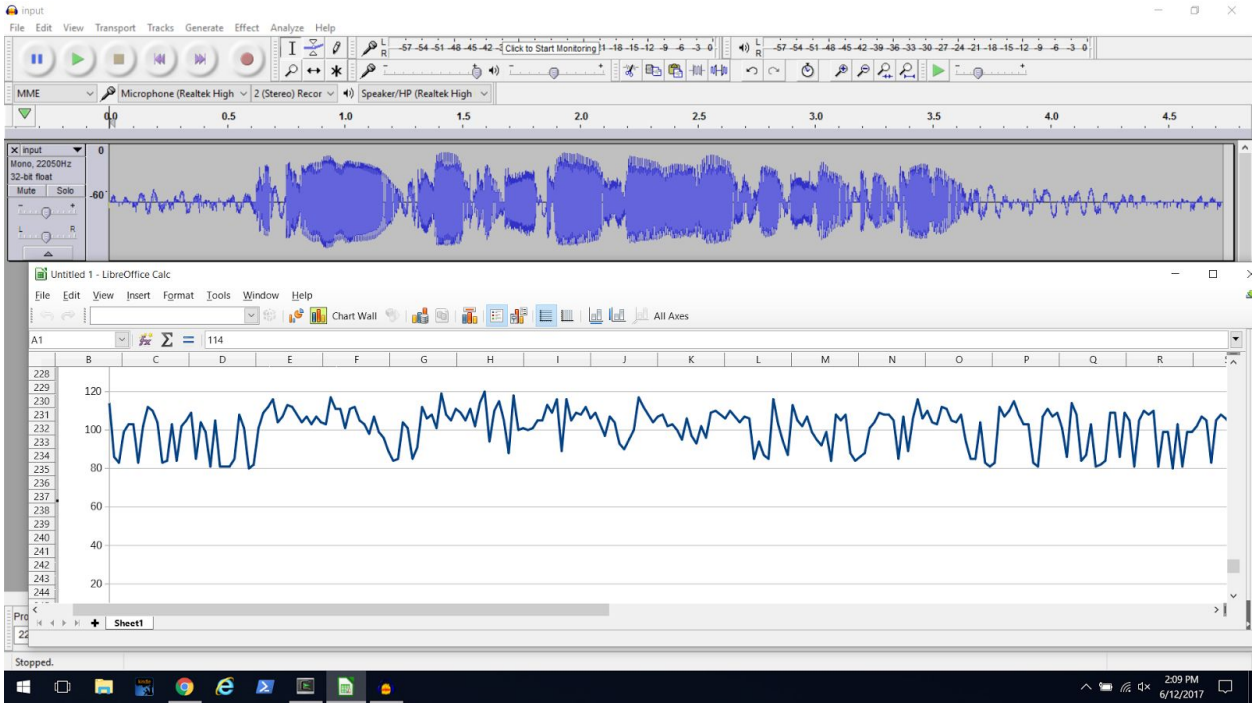
- Decided in meeting to refine replication of attack by adding trigrams and using svm function instead of logistic regression to improve accuracy. Using svm (functions.SMO in Weka) with bigrams did not increase accuracy. Several other functions performed slightly worse. Adding trigrams improved accuracy only slightly to 81%.

- Encoded silence generated in Audacity to see what packets look like (being careful to generate at 22500 hz to match ldc samples instead of 44100) and this showed that all packets fell into the naive "b" bucket, so maybe we should tweak bucket sizes

- Opus only compresses our ldc sample files by 16% with vbr (80-120 byte packets) over cbr of the same quality (120 byte packets), probably because the sample rate is only at 22500. Decided to look for .wav datasets sampled at 44100 to use instead, to show higher cost of using cbr to achieve security… must check out http://research.nii.ac.jp/src/en/list.html

12 June

- Graphed packet sizes over time to see how they compare to Audacity waveform (db) and there is clearly some relationship where we see few to no small packets during periods of very audible speech, but packets sizes are very erratic and often larger than expected during periods of relative silence. This seems to show high sensitivity of codec in picking up possible speech activity - sort of like many false positives, so tricking it to create larger packets seems like it should be quite doable.



- Continued working with pydub to manipulate samples. Fixed problem encountered on 6 June by generating the white noise in Audacity again at the correct 22500 rate. Targeted mixing the 6th frame of a sample with the frame of low amplitude white noise, and achieved imprecise effect. Targeted frame changed size, but so did some of the immediately following frames.

  tw614@muffler:~/opuswork/whitenoisetest/pydub-test$ head input.range
  960, 114, [[1, 113], MDCT, SWB, M, 960, 233030144]
  960, 86, [[1, 85], MDCT, SWB, M, 960, 221996288]
  960, 83, [[1, 82], MDCT, SWB, M, 960, 1426242304]
  960, 99, [[1, 98], MDCT, SWB, M, 960, 763957248]
  960, 103, [[1, 102], MDCT, SWB, M, 960, 456500992]
  960, 103, [[1, 102], MDCT, SWB, M, 960, 589827584]
  960, 83, [[1, 82], MDCT, SWB, M, 960, 1272446208]
  960, 102, [[1, 101], MDCT, SWB, M, 960, 62840832]
  960, 112, [[1, 111], MDCT, SWB, M, 960, 1312075520]
  960, 110, [[1, 109], MDCT, SWB, M, 960, 225551616]

tw614@muffler:~/opuswork/whitenoisetest/pydub-test$ head input-mixed.range
960, 114, [[1, 113], MDCT, SWB, M, 960, 233030144]
960, 86, [[1, 85], MDCT, SWB, M, 960, 221996288]
960, 83, [[1, 82], MDCT, SWB, M, 960, 1426242304]
960, 99, [[1, 98], MDCT, SWB, M, 960, 763957248]
960, 103, [[1, 102], MDCT, SWB, M, 960, 456500992]
960, 110, [[1, 109], MDCT, SWB, M, 960, 13507696]
960, 85, [[1, 84], MDCT, SWB, M, 960, 294991104]
960, 101, [[1, 100], MDCT, SWB, M, 960, 40022197]
960, 112, [[1, 111], MDCT, SWB, M, 960, 11730944]
960, 111, [[1, 110], MDCT, SWB, M, 960, 118809856]

- Converted a 44100 rate mp3 sample from accents-gmu into .wav and ran opusenc to determine that vbr output is 44% smaller than equivalent quality cbr, so compression does getter better with sample size/quality. Packet sizes ranged from 123 to 326.

- Ran svm (functions.SMO) against trigram input for speaker identification but the program seemed to hang overnight on my machine.

13 June

- Attempted to run svm against trigram input for speaker id on Muffler VM but ran into heap overflow, and unable to allocate memory when increasing the JVM heap to 4096mb. Micah increased the memory for the VM to 7.9gb. That worked. Accuracy increased only slightly to 23%.

14 June

- Converted accent_gmu dataset from mp3 to wav to try running Weka attack since packet sizes on the 44100 rate samples are in a wider range. Noticed that some vbr output from Opus is as much as 53% than equivalent cbr. Some is less than 20% smaller. Realized about:blankthat now I need to label speakers as male/female or something for the recognition task.

- Must look through newer ldc datasets for additional 44.1khz audio samples that we can use… https://catalog.ldc.upenn.edu/

- Must record myself for now at 44.1khz to get a good sample and begin the automating task of mixing different frames and getting reliable changes to output from codec… think binary search

15 June

- Found only one dataset from ldc that was recorded at 44100 hz and it is in Arabic which makes it difficult for me to listen to samples for sanity checks

- Built new 44100 dataset of 21 different speakers on voxforge, each with at least 10 or so samples of at least 3 seconds… http://www.repository.voxforge1.org/downloads/SpeechCorpus/Trunk/Audio/Original/44.1kHz_16bit/

- Weka's accuracy on the new dataset for speaker identification jumped to 78%. One concern: in this dataset, and in the original Muffler paper, each speaker was reading a different book. With the West Point ldc dataset, each speaker was reading the same sentences, and recording in the same environment. I wonder if the classifier is picking up more on the speaker, or what the speaker is saying (author's style, dialogue vs. poetry, etc.) or the environment where they are saying it. In any case, I think we've more-or-less replicated prior results now.

16 June

- Began automating experiment to mix white noise and tones of various frequencies, all at various amplitudes, with a full range of packet sizes. Tones were 10, 50, 1000, and 2000 hz. Amplitude ranged from nearly inaudible to +7 db. After a first run with one sample, it appears that white noise most reliably changes packet sizes, with greater amplitudes resulting in greater changes. Steady high and low frequency tones seem to produce very weak changes. Some initial results are still surprising, such as raising the amplitude and actually seeing packet sizes drop in some cases.

19 June

- Fixed bug in program that invalidated some of Friday's empirical results (I was importing already adjusted noise samples and then adjusting them again programmatically). Ran another experiment targeting only 110-135 size packets with just the white noise at amplitudes from nearly inaudible to +9 db. This time Increase in amplitude only increased packet sizes (no decreasing anomalies) but at +9 db the increase was only 6-9 bytes.

20 June

- At +20 db, the packet sizes for 4 of my 5 targets increased by 100-150%. Upon further reflection and subjective listening, I think this might be acceptable. +20 db is still not very loud, and relatively few of the packets would have to change by such a magnitude. I found similar results by applying a sine wave tone at 170 Hz (middle of human speech) instead of white noise. Interestingly, 1 of my 5 targets was barely affected by these experiments, increasing by only 10%.

- Starting to write program that performs larger, more general experiment given a speech sample that successively targets every packet in the sample with the full range of white noise amplitudes and computes average change for each noise mix on all original packet sizes.

22 June

- Continued working on program to mix every frame of a sample file with white noise ranging from +0 to +20 db and computing the average packet size increase for each combination across the sample input. This should produce a recipe to make any packet size, on average, into some other packet size. There will still be a lot of seemingly random variation, but it should get us close to the desired super distribution while sounding like snap-crackle-pop in the background. Will extend this to run over all .wav files in the voxforge directory.

- Initial results over one input sample (sci0222.wav) shows expected increases for packet sizes that I previously tested, but unexpected behavior for larger packet sizes around 200 bytes; adding white noise in some cases continuously decreased the size and amplitude increased. Running this over a wider set of samples will be interesting.

- Decided in meeting to continue to focus on how to manipulate samples instead of repeating the attack to reach previous levels of accuracy.

- Agreed to try automating the process as much as possible to work on an arbitrary vbr codec. Vision is to be able to take a codec and a large dataset and run our program to automatically try a wide range of noises to mix with each frame of each sample and parse the results to produce the optimal mapping which could then be loaded into the Muffler layer and used. This would include automatically finding the optimal bucketing scheme to defend against the worst case attacker scenario.

27 June

- Continued to extend program by automating it to run over the entire voxforge dataset. It's taking a long time to run (I think mainly the system calls to opusenc) so I'm running it in tmux (https://askubuntu.com/questions/8653/how-to-keep-processes-running-after-ending-ssh-session). I might also try removing some other system calls (lines that do clean up) to try speeding this up. Or maybe I can explicitly parallelize the main body of the program somehow.

28 June

- As of 1255 today, the program that started running last night at about 1830 is still running on the VM. I am curious to see if the results will provide any insight, but clearly this is not going to be a feasible way to keep working. I will rewrite the program to mix all packets in each sample, and then analyze the whole output file rather than doing one frame at a time. Some initial testing will be required to ensure that this doesn't completely change the results.

29 June

- As of 1020 this morning, the program is still running.

- Decided in meeting to re-run the current program on a smaller dataset with more intermediate and final outputs for better analysis.

- Decided to also try mixing low amplitude frames taken from speech samples that we already know encode to various packet sizes (i.e. injecting exactly the type of information that the encoder is looking for instead of synthetic tones and white noise).

- Decided to try injecting tones at various frequencies and audible amplitudes but very short durations (e.g. 1 ms) which hopefully speakers cannot physically output, so this would be totally transparent to users.

- Decided to try listening to mixed samples first to see how much quality is really affected before proceeding with exhaustive experimentation.

30 June

- Realized that my white noise .wav that I was using to mix was only 2 ms in length, not 20 ms, so I killed the previously running program. Now trying to re-run a streamlined version using the desired 20 ms .wav. First attempt at streamlining consists of mixing and encoding the entire file at once instead of frame-by-frame, but I also need to check that this doesn't produce totally different results for an individual frame. The 2 ms file is also audible, so that might mean that Audacity can't produce anything short enough to trip up my laptop speakers.

- New version of the program crashed at some point due to an anomaly when parsing a range file, but results up to that point showed that after encountering hundreds or thousands of each packet size, increasing white noise amplitude showed monotonic decrease on larger packets and virtually no change on smaller packets.

- Comparison of two experimental approaches (mixing individual frames with noise vs. mixing the entire file with noise) seems to show significant differences. It appears that the encoder might be able to detect and strip out the noise when it is mixed over the entire file. See miniAnalysis2.txt (generated by miniExperiment.py).

3 July

- Created sample with white noise at random amplitudes (between +0 and +20 gain) every 20 ms, and it is noticeable but not really distracting from the speech. If we could get the security we want with that sound effect, I think we'd be happy.

- Tried extracting one frame (number 18 from sci0222.wav) from a sample that encoded at a higher sampling rate (producing packet size of about 200 bytes) and repeated the

experiment to compare mixing individual frames vs. the entire file, and both approaches dramatically increased the size of the target packets (going from about 130 to 300 bytes). The resulting speech sample sounded terrible (although speech was still fully understandable) as this was done with the frame at the original amplitude, so now I will make it quieter and try again to use amplitude as a knob.

- When lowering the amplitude and applying it across the file at random levels of gain, the resulting speech sample sounds good with little distraction.

- Repeated the big experimental analysis across the whole dataset using 18sci0222. Initial results appear to be very similar to using white noise. Small monotonic increases to small packet sizes, and larger monotonic decreases to larger packet sizes.

5 July

- Learned that sound really is just a combination of frequencies and amplitudes. Different instruments have different timbre because of the array of frequencies they generate on a given note/pitch, called overtones, which are usually multiples of the *fundamental*, called *second and third harmonics* and so on. See http://meandering-through-mathematics.blogspot.com/2013/09/why-do-different-musical-instruments.html. The spectrogram view in Audacity allows us to visualize this for a wav file. http://manual.audacityteam.org/man/spectrogram_view.html

- Examined some spectrograms vs. packet size charts to look for insight… not much help there. Some packet size spikes seem to correspond with spikes of higher energy across all frequency bands (like white noise) but not when that energy is sustained. So maybe sustained white noise is being filtered out, which would explain why targeting individual frames with white noise seems to work while whitewashing the whole thing doesn't.

- Started looking at Cython to be able to try again with the experiment to mix and encode all individual packets across the whole dataset with the operating system context switching overhead that will take too long. Existing Python wrappers for Opus all appear to be out of date. http://docs.cython.org/en/latest/src/tutorial/clibraries.html

- Started looking at Opus source code and documentation to figure out what library functions I need to pull into Cython. I note that the encoder is stateful, so we can't necessarily treat each input frame independently, and this further confirms the results I've seen with targeting individual frames vs. mixing an entire file.

6 July

- Repeated the big experimental analysis across a subset of samples using 18sci0222, but this time mixing individual packets instead of the whole file. Results appear very similar to all previous experiments. Interesting that at very large packet sizes, +0 can lead to a huge decrease and then increasing gain of mixed noise makes packets smaller but less so.

- The pitch of a sound is not the same as the fundamental frequency. Pitch is the harmonic that is most heard by the human ear. Pitch is subjective, but most humans agree on frequencies that sound higher or lower.

10 July

- Collected thoughts for explanation of "what is sound…" Last time we talked about frequency, amplitude, and time as asked what is missing to create timbre. I've learned that pitch and timbre are the subjective result that humans hear when presented with a combination of many different frequencies at different amplitudes. Spectrogram is the best visual of this. Then it occurred to me that human speech occurs in only a very narrow band of the spectrogram. One suggestion is that we almost fully capture it between 50 and 1,000 hz, so that's where I should look to generate or extract noises to use for overlay.

- Installed opus-tools from source, still trying to figure out how I can call the API.

11 July

- Decided in meeting to continue working on Opus API integration while repeating experiments with louder noises focused inside the range of human speech. Team will continue trying to obtain samples that can't play over speakers, and also look at generating noises like this through tools to directly modify or create .wav files.

12 July

- Started running program to test +0 to +40 using 18sci0222 on the smaller voxforge subset using the individual frame approach. Also listened to random +0 to +40 example and it sounded terrible, but I could still understand the speaker.

- Continued working to figure out the Opus API. Found necessary headers to get Opus type definitions. Wondering how to read in a frame of .wav audio and how to write an encoded packet back to a .wav file.

13 July

- Experiment for +0 to +40 is still running, but results so far continue to look unusable.

- Found example of how to use API at https://chromium.googlesource.com/chromium/deps/opus/+/1.1.1/doc/trivial_example.c but still trying to get it to compile and run

14 July

- Results for +0 to +40 experiment show same pattern as previous experiments but further extrapolated.

17 July

- Tried to understand Cython as well as "Extending Python with C/C++" to figure out which is more appropriate for our needs. Or maybe Pyrex or SWIG? I'm frustrated. Hopefully Tavish and I can at least get the Opus API to do something in native C tomorrow.

18 July

- Figured out how to compile with Opus.api... gcc opustest.c -I /usr/include/opus/ -L /usr/local/lib/*.so

- Talked through how Cython, pydub, Opus API pieces all fit together. Will look into raw_bytes function in pydub and "named pipe" as a way to pass data to the encoder.

20 July

- Drew first draft of diagram showing how all pieces will fit together

- Still trying to get Opus API "trivial example" to work now that it compiles. Output is blank so far. Converting .wav to 16-bit little-endian .pcm with ffmpeg didn't help.

21 July

- Got "trivial example" to work correctly. Realized that it encodes all but the last frame of its input because of its loop break condition, so I was getting nothing by feeding it only one frame. Output sounds just like input. Reading a .wav is fine.

- Still trying to figure out how to compile/link/run Cython example. Continuing to get undefined symbol error when importing the c-algorithms queue library.

28 July

- Successfully built and ran Cython “using libraries” example with C-Algorithms. Ready to start trying to link Opus API in Cython.

- Questions for discussion at next meeting: (1) How do we handle statefulness of the encoder when running the same frame through it multiple times? (2) What noises do we focus on trying to mix, and do we have the ones that can’t play over the air yet?

29 July

- Worked on building Cython interface and wrapper to Opus API encoding functions and type definitions. Able to compile, import, and instantiate an OpusEncoder.

31 July

- Got Cython wrapper for Opus that seems to work, outputting a reasonable integer for an encoded frame size. Working on additional sanity checks to ensure that process of importing a .wav with Pydub, taking a slice, converting to raw_data, and passing through the Cython wrapper to a opus_int16 buffer for encoding isn’t losing anything.

- I am noticing that running my test multiple times, I get a slightly different packet size with each run, and selecting VOIP vs AUDIO mode doesn’t seem to make a difference. It’s also not the same packet size I see when running opusenc on the input file.

- Not sure if we want to use raw_data or get_array_of_samples to convert Pydub AudioSegment to what OpusEncoder wants. Or maybe we want to convert the bytestring to an array ourselves?
  https://stackoverflow.com/questions/32373996/pydub-raw-audio-data

- Ran through encoding all of sci0222.wav trying get_array_of_samples and packet sizes were a little curious but not totally out of whack, but unfortunately I couldn’t listen to the resulting file that I output (just dumped into a file with extension .opus). Figuring out these internal representations is the next major technical challenge.

1 August

- Discussed how to handle encoder state in our meeting, and decided to pass a copy of the struct to the opus_encode function each time, and only update the good copy after a successful iteration. This should be easy to work into the Cython wrapper.

- Discussed what to try mixing once this framework is running, and came up with the idea to send white noise through the encoder, and then decode it, and use the decoded noise. In theory this would capture what the encoder is looking for, and it would be independent of the codec used. Micah also says he will work on finding or creating the noises that can’t be played over speakers.

2 August

- Incorporated what we talked about for copying and saving state of the encoder, although it is untested.

- Realized that opusenc and opusdec exclusively use the multistream implementations, which might explain the different numbers I'm seeing output, so I'll have to switch to that.

- Looks like I need to pull in some functionality from libogg to write and read encoded opus files. It will be easier, for sanity testing purposes, to immediately decode each packet and write out to a .wav just like trivial_example.c does it. We really don't ever need to play with .opus files.

3 August

- Got encoded and decoded audio back out into a file, and it has some distortion but I can hear the personal speaking clearly. Tested to see if the conversion from little-endian to big-endian and back (as shown in trivial_example.c) has something to do with it, but that just gave me pure static. Tested to see if application mode and bit-rate account for the distortion, but that didn't seem to change much either.

- Packet sizes output from trivial_example.c and my Cython implementation are similar, slightly different, and the trivial_example.c output is 267 bytes vs. 299 bytes from mine. Not sure what that means yet. Tomorrow I will try writing directly out to a file before encoding/decoding to see if the input gets distorted between Pydub and Cython.

4 August

- Writing out to a file after passing the audio data through Cython has the same distortion, before I even try to encode or decode it. So the problem is with the get_array_of_samples function, or maybe in the conversion from Python integers to int16_t values

- Figured out that the array of samples only had 882 entries instead of the full 960 for a 20 ms frame, so every frame written to file contained a bit of garbage on the end causing the distortion. Only writing len(frame) = 882 samples made the distortion go away, same with on the decoded end, but the encoder is also encoding that garbage because it can only be given a fixed set of values like 960. Must figure out why Pydub isn't giving us 960 samples.

- The 882 comes from taking 20 ms of a 44100 hz sample. 960 corresponds to 48000 hz which is how we initialize the encoder. I need to figure out how this isn't a problem in trivial_example.c. Maybe because when it reads in the raw bytes it just eats 960 samples regardless, so the frames are divided up differently(?)

- Opus internally resamples 44100 hz audio to 48000 before encoding, so I guess I don't have to worry about it encoding garbage. Maybe my sanity check written out to a file is irrelevant. But I still wonder why the trivial_example.c output is slightly smaller than my sanity check file from Pydub/Cython. I think because trivial_example.c is working over fewer frames since those are 960 samples instead of 882. https://hydrogenaud.io/index.php?PHPSESSID=p9iuu6ffo16522fb3c16vsb4u3&topic=97051.msg821334#msg821334

- Using ffmpeg to convert the .wav to a 16-bit little endian raw .pcm, and then telling Pydub to import it at a 48000 sample rate, seems to have fixed things. Now I'm getting exactly the same size output from my Pydub/Cython program as the trivial_example.c. Individual encoded packet sizes are still different though.

6 August

- Repeated experiment5 (trying to build static mapping between packet sizes over small Voxforge dataset, testing one frame a time) just to see how much faster it is with Cython and minimal file I/O. It's very fast. (Of course, that experiment is still a theoretical failure). The speed-up is about 10-20x.

7 August

- Working to switch over to the multi-stream and floating-point Opus functions as used in the opusenc tool. Multi-stream functions work fine now, but there is some weirdness with trying to convert from signed 16-bit to float. I get output, and I can sort of hear the speaker, but it is very heavily distorted and too fast. Maybe that just doesn't matter for our purposes, though, and we can stick with the fixed point stuff. I might just have to regenerate all the range files across our dataset.

- Next step seems to be getting an understanding of the superdistribution so I can complete the part of the framework for calculating and checking target packet sizes.

8 August

- Decided in meeting to continue experiments with mixing noises above and below the range of human hearing. Tested to see if opusenc would strip out 20.5 khz noise, and it did not. Encoded clip of 20.5 khz noise was larger than encoded clip of silence. Audacity spectrogram also showed that the noise was still present. Played over the speakers, this noise was not audible. This could help with hitting target packet sizes because we can turn up amplitude very loud whereas before we focused on low amplitude noises in the range of human speech. Also tested during the meeting if Slack would transmit a high frequency noise above the range of human speech, and it did. I will continue testing if it makes a difference whether the Opus encoder is run in VOIP mode vs. AUDIO.

- Also tested during the meeting if input was identical to output after being encoded and decoded by Opus. It seems like it should be different because VBR is lossy by definition, but file sizes are the same. Running sha1sum on the two files showed that they are different, however. It appears that file size is determined solely by the length of the audio clip; there is a fixed number of frames and samples per frame. The difference is how carefully(?) things are sampled by the encoder. So it may still be interesting to try mixing processed noises that should contain only things of interest to the encoder.

- Another idea would be trying to do something with echo cancellation, possibly mixing frames with some previously occurring frames, so that the mixed noise would be encoded but would not be audible after going through Skype or any other VOIP system. This is harder to test.

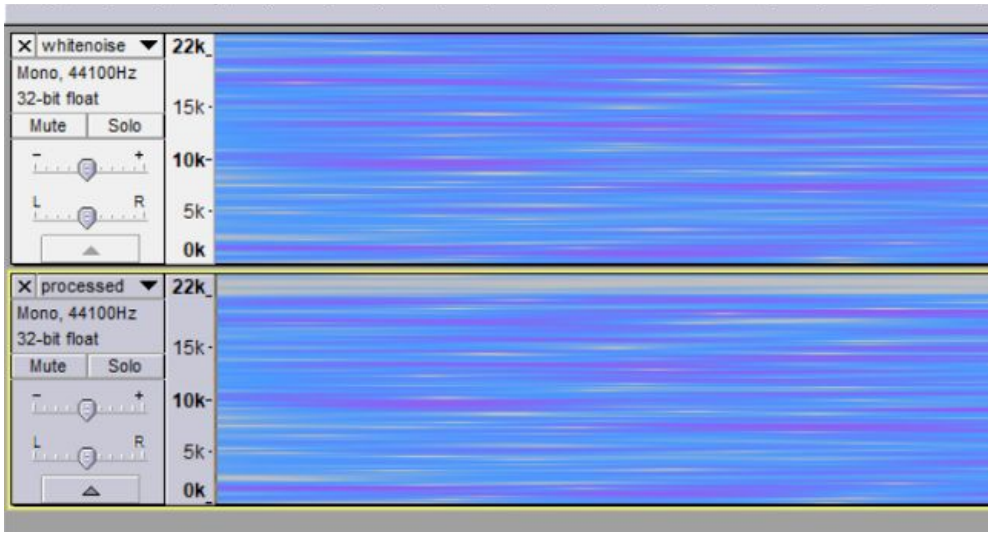
9 August

- Rewrote fasterExperiment5.py to recompute (because I'm now using the plain fixed point encoding function instead of opusenc) the original packet sizes for each sample in our dataset, and then mix the frames one-by-one with the 20.5 khz noise (20khzSinLoud) at gains from 0 to 40, being careful this time to update/not update the encoder state the way it should be done. This will just give us a better idea of the potency of this noise as mixing material.

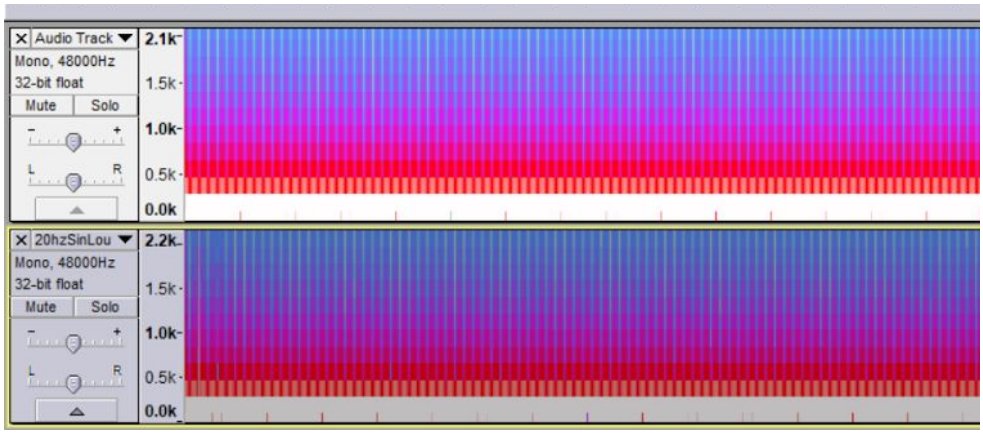
- Results of the above with both VOIP and AUDIO settings are less promising than the previously tested 18sci0222 noise (which at least monotonically increased packet sizes at the low end). Mode did not seem to make any meaningful difference, which is good since we don't have to make assumptions about Skype/etc. but doesn't shed any new light on where to go from here.

10 August

- Looked at spectrogram of one frame of Audacity-generated white noise compared with Opus-processed (encoded and decoded) white noise and found that the only thing systematically filtered out seemed to be above 20 khz, which makes sense if that's the limit of human hearing. So this might explain why the 20.5 khz tone did not have much effect when mixed.



- Will try now to look at very low frequency tones instead, because although Opus documentation says it uses a high-pass filter, I'm not seeing the effect of it. First attempt is a 20 hz sine wave. Results of fasterExperiment5.py are quite similar to the previous tests of low-amplitude noises in the speech range: good monotonic increases to smaller packet sizes, but becoming very ineffectual around packet sizes of 180+. Still, this is inaudible to cranking up the gain as high as we want is promising. Here's what the spectrograms look like for this noise: Slightly reduced amplitude, but not filtered out.



- Also tried mixing with processed versions of whitenoise and 20hzSinLoud without any especially interesting results. Everything so far works better (allows for bigger packet size changes) than the 20.5 khz noise. The 20 hz noise seems best because at least it's inaudible (compared to the whitenoise or individual speech frames).

- Next I want to try to further turn amplitude up and down on the 20 hz noise. In particular, I wonder what negative adjustments will do at the top range of packet sizes where things start to invert. I also recall huge jumps in size in an earlier buggy version of the program that doubled down on gain increases (not good then with audible white noise, but now perfectly acceptable).

11 August

- Modified fasterExperiment5.py to try running gains in 5 db increments from -100 to +100 on the 20 hz noise. Spreads in packet sizes widened (following the same patterns as before), but maybe not as much as we need. Sill, this looks better than anything seen yet. Next I tried to double the range again, but there was no significant difference, so I guess the previous range maxed out what Pydub or Opus can handle.

14 August

- Did a listening experiment to see what a speech sample sounded like after being mixed with 20 hz noise, and it had a lot of distortion despite that the fact that the 20 hz noise by itself is inaudible (both initial .wav and encoded .opus formats). I am wondering if the amplitude gain function in Pydub is also affecting the frequency, and will now conduct an experiment to check this. I will take a longer duration 20 hz tone and randomly adjust the gain, frame by frame, up and down and see what that sounds like.

- The experiment described above makes it appear that Pydub's function for adjusting gain is somehow turning inaudible noise to be audible. Other possibilities to check include seeing what happens by simply opening and exporting the file, and maybe not slicing it into frames (perhaps some garbage is being introduced between frames, and that's what I'm hearing).

- So it's not any fault of Pydub. If I just turn up the amplitude by even 1 db in Audacity, the 20 hz noise becomes audible. Reducing the tone frequency to 10 hz improves the situation slightly, but it's still audible when gain is increased even a few db.

- Looking back at high frequency tones now, the 20.5 khz tone is also clearly audible when turned up a few db. 22 khz is better, but when turned up about +20 db it is actually audible at a low pitch. Now I wonder how my speakers distort sounds when they are physically unable to play them correctly.

- 2 hz sounds good when turned up in Audacity, but when I do the random gain overlaid on a speech sample it is even more distorted than what I heard at 20 hz.

- This might be part of my problem, because I definitely hear the ticks… "Also remember that starting and stopping pure tones invokes modulation effects which can be audible even if the tones themselves are not. For example, generate a pure tone of 440Hz, the oboe "A" at the top of the orchestra. Unless you take special steps, there will be a tick at the start and stop points. The two points generate infinite frequencies and are very clearly audible." https://forum.audacityteam.org/viewtopic.php?f=62&t=81241

16 August

- There is some difference between gain and volume that I need to understand. When increasing the gain, especially in Pydub which works on 16-bit integers, I might be cutting off the tops of sound waves and distorting the way the sound.

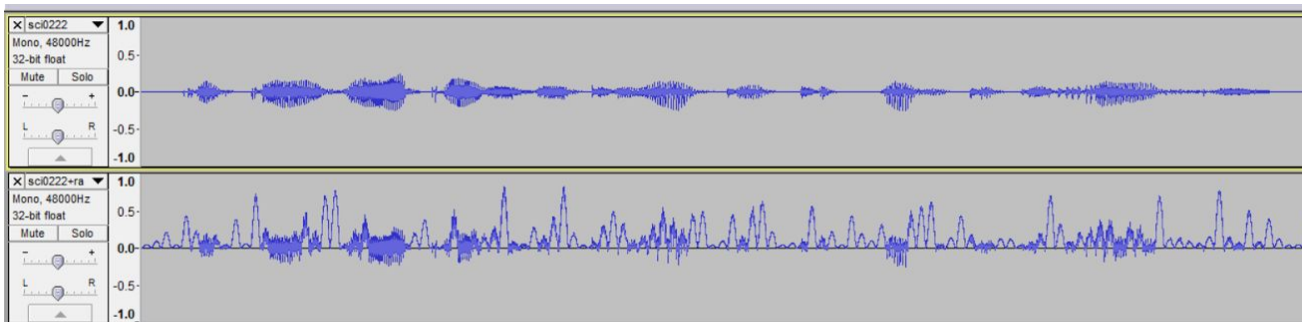
- So going over 0 db means that this "clipping" phenomenon occurs. When I generate a tone in Audacity with amplitude of 1, that means that the peaks are already at 0 db, which gives me no room to further amplify or increase gain. So I need to generate them at maybe 0.1 amplitude, and then I can dynamically scale them up in Pydub. https://www.quora.com/How-do-you-increase-the-volume-of-an-audio-track-using-Audacity

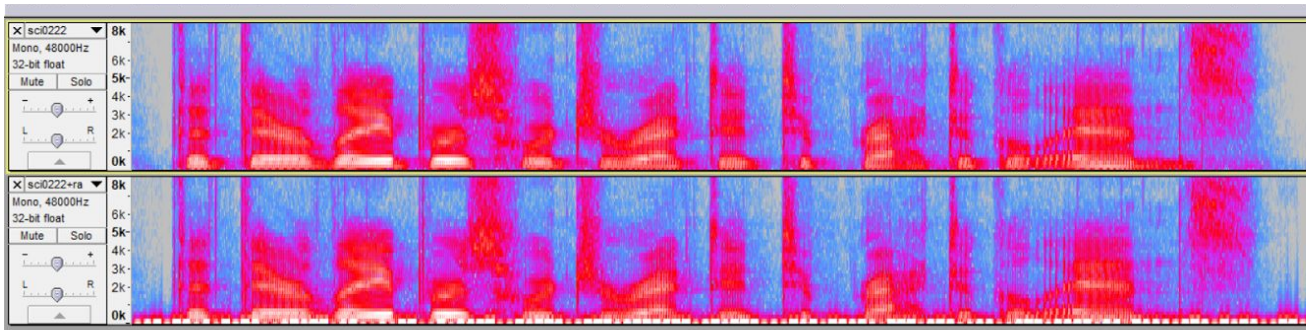
- Also of note, changing the gain using the slider on the left of the track in Audacity does not do anything to the underlying audio when exported. To make adjustments that stick, I need to use Effects -> Amplify.

- I regenerated my 20 hz noise at 0.01 amplitude, which Audacity shows can be scaled up 40 db without clipping, and performed the random adjustment listening experiment on a speech sample again, and it was still distorted.

- Next I generated a 5-second 20 hz tone in Audacity, and tried to increase the amplitude in the 2nd and 4th seconds of the clip, and playback clearly revealed the audible start-stop modulation problem that I read about yesterday. Now I need to learn about the "special steps" to which that writer alluded… http://forum.audacityteam.org/viewtopic.php?p=217959&sid=43882222a9aba0c3832f34a123ecfe56#p217959
-

- Simply doing Effects -> Fade In and Fade Out on my noise clip might take care of this. A quick test on a longer duration clip confirms. I created a new version of my noise to use (20hzSinQuiet) and repeated the randomly adjusted listening experiment with success… no distortion! And the wave forms in Audacity of the original speech sample and the overlaid sample look totally different.





- Results of repeating fasterExp5.py with this noise appear to be on par with everything previously seen with white noise. This is as good as it gets right now: inaudible, and capable of moving packets sizes somewhat in a reasonably predictable way (namely, making smaller packets bigger, and bigger packets smaller).

17 August

- Encoded the sci0222 sample and the sci0222+random20hzQuiet with opusenc, and created histograms to show how the packet size distribution changed (still dealing with some annoyance between 44.1khz and 48khz and slightly longer/shorter outputs, etc)...

    sci0222.range total packets: 157

    a: 27 xxxxxxxxxxxxxxxxx
    b: 32 xxxxxxxxxxxxxxxxxxx
    c: 27 xxxxxxxxxxxxxxxxx
    d: 17 xxxxxxxxxx
    e: 25 xxxxxxxxxxxxxxx
    f: 18 xxxxxxxxxxx
    g: 8 xxxxx
    h: 3 x
    sci0222+random20hzQuiet.range total packets: 157

    a: 0
    b: 16 xxxxxxxxxx
    c: 27 xxxxxxxxxxxxxxxxx
    d: 17 xxxxxxxxxx
    e: 20 xxxxxxxxxxxx
    f: 16 xxxxxxxxxx
    g: 35 xxxxxxxxxxxxxxxxxxxxxxx
    h: 26 xxxxxxxxxxxxxxxx

- So now I can show two very different Audacity wave forms with two very different Opus-encoded packet distributions that sound identical. This is good news.

- The randomly mixed track is about 25% larger than the original track encoded in Opus (using --raw), but still 25% smaller than if we used --hard-cbr at the equivalent bit rate (192.4) to capture the same level of quality. I expect that we'll see more performance improvement over --hard-cbr when we're matching the superdistribution, because we probably won't be skewing the packet sizes toward the 'h' end so much.

22 August

- Looked again at Muffler paper and Brad's code to start thinking about trying to calculate and map to a superdistribution. Created .dat files over the voxforge dataset as a starting point, and now getting ready to run the attack again through Weka (since encoded packet sizes are different now, calling the API through Cython instead of using the opusenc tool, and we need the baseline accuracy). Created wekainput.csv. Next I need to convert to this .arff and figure out the commands for Weka again.

23 August

- Repeated Weka attack on new .dat files created with my Cython code, and found only 52% accuracy now (down from 78% when the .dat files were created from opusenc --save-range files). Now I need to look into this. I did notice that the .dat files seemed a lot more homogenous, compressed around the middle packet sizes for long stretches.

- Also learned how to convert .csv to .arff files without uploading/downloading… java -cp ~/opuswork/replicate-attack/weka-3-8-1/weka.jar weka.core.converters.CSVLoader wekainput_original.csv -H -B 10000 > wekainput_original.arff

- java -cp ~/opuswork/replicate-attack/weka-3-8-1/weka.jar weka.classifiers.functions.SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C 250007" -calibrator "weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4" -t wekainput_original.arff > attack_results.txt

- Target bitrate is 64kb/s for both opusenc tool and Cython implementation. I tried the multi-stream functions and those don't make any difference either. The fixed vs. floating point issue is all I can guess. 'Wav_read' function in audio-in.c is how file is turned into floating point samples for opusenc tool. It looks like division by 32768 (which appears in wav_read) might be all I need to do to get my values between 1.0 and -1.0 as described in the Opus API documentation. https://www.kvraudio.com/forum/viewtopic.php?f=33&t=414666#p5791365

- I created the .dat files calling the floating point encoding version, and repeated the Weka attack, and the accuracy actually got worse… 48%. I need to do some sanity checks to make sure I didn't destroy the input audio. When I write out the 32-bit floating point file and open it in Audacity, I hear the speaker OK by the speed is too fast… playback speed of maybe 0.66 sounds most like the original (but the pitch drops). What does that mean? Also, the encoded packet sizes using the floating point function seem to be not much different than using the fixed point function (including from trivial_example.c).

- Setting complexity level to 10 in my Cython program (like the default in opusenc) also did nothing new. All other default controls seem to be the same.

1 September

- Decided in meeting to continue working on replicating the attack. I will try to normalize floating point values (maybe dividing by the max in the integer array coming from Pydub), and maybe try to modify the opusenc tool code to run some checks on that for more insight. I might also be able to better extract what's going on with the wav_read function.

- Decided to also begin integrating the original Muffler algorithm to see what we can really accomplish with the 20 hz noise.

- Idea from the meeting: Is there some opportunity to modify the Muffler algorithm to take advantage of the fact that we can make large packets smaller, or does that just make CBR at a low bit-rate a better solution (since we'd essentially be degrading quality)?

11 September

- Tried normalizing floating point representation of shared_sample.pcm by dividing by the max value in the Pydub array_of_samples instead of the maximum integer value (per Tavish's suggesting), and it sounds reasonable but playback is still too fast and the resulting packet sizes still clump together around 161 bytes.

13 September

- Identified that my shared_sample floating point representation is exactly ½ the length of the original sci0222 sample. So somehow I'm dropping every other frame or reducing each one to 10 ms.

- Realized that I was only writing the first 2 bytes of each array value to the shared_sample file instead of the full 4 bytes for a 32-bit floating point value. Shared_sample output now sounds very good and has the correct playback length.

- Modified opusenc.c to write out sanity_check.pcm which is the floating point representation of its input before feeding to the encode_float function. Listened to it in Audacity, and it sounds good. Need to figure out how to compare it to my own floating point representation. This will tell me if the inconsistency lies in the way I'm opening/reading/preparing the input, or somewhere in the parameters of my encoder.

- As a starting point, I see that my shared_sample file is 549kb vs 602kb for the sanity_check I got from opusenc, but it looks like opusenc includes some padding to round off the last frame, at least. I'll try to remove that padding and see what happens.

- For some reason, the sanity_check produced by opusenc is 3/10ths of a second longer than the original sci0222.wav, just sort of stretched out. I wonder if that's because of something I did when writing the file or how I import things into Audacity, or if that is

intentional on the part of opusenc. Also, it does look like I should be normalizing with the max integer in the denominator, not the max of the array_of_samples.