

The development of the audio cleaning and compression tools.

Brett Alexander Preston

April 2025

This report is to be submitted to the NLnet Foundation.

I set out to create a way to send voice communication of best possible quality at low bandwidth, with the help of modern audio engineering knowledge. In a contemporary setting, where devices are likely to have ample processing power, but with limited bandwidth as is the case in anonymous communications protocols that provide strong anonymity. This tool set was created for use with the Katzenpost/Echomix software, and other mix networks but it should be noted that any project interested in having low-bandwidth, efficiently computed and good quality voice recordings may be interested in using it.

Contents

1	Work execution overview	1
1.1	Create a prototype app with audio record and play [3A]	2
1.2	Determine target packet size for katzenpost mixnet parameters and duration of message. [3F]	3
1.3	Evaluate and audit audio compression libraries, consider security history of library / codec choice, and applicability of codec for mixnet parameters, implement the best compression. [3B]	4
1.4	Evaluate and implement a secure decoder [3C]	4
1.5	Codec customization [3E]	4
1.6	Implement efficient signal processing, including noise reduction to achieve best sound quality at low bandwidth [3D]	5
1.6.1	High pass and low pass	5
1.6.2	Spectral gate threshold detection	5
1.6.3	Noise gate threshold detection	5
1.6.4	Gain booster	5
1.6.5	Lookahead limiter	5
1.7	A separate repo which includes the audio cleaning, processing, encoding and decoding tools with documentation. [5C]	5

1 Work execution overview

Despite initially starting this work using Golang, and spending a considerable amount of time on it, I have decided to ultimately do it in Rust. It is a more appropriate tool both because of its efficiency, and because it has great partial tools for implementing Fast Fourier Transforms and other frequency analysis tricks. This allowed me to engineer much more powerful noise reduction than I may be able to achieve in Golang, short of building a plethora of mathematical tools out of scope for this work. Rust also has good tools for encoding and decoding, which were a great starting point for achieving what I set out to do.

In the end, I succeeded at creating an effects chain that reliably produces clean, good quality voice audio, and guarantees that a 30sec recording will result in a file no larger than 45kB. This is ideal for use in modern low-bandwidth communications, and in particular in Katzen messenger, where the target packet size is 50kB, with up to 5kB available for the header and other cryptographic material.

The code, as well as debugging and testing GUI is available at https://github.com/brettpreston/Rustic_Audio and <https://github.com/katzenpost/echo>.

This was also an opportunity to take advantage of the new Rust interface for the Katzenpost API (part of milestone group 5), and so a minimalist voice-over-mixnet service was implemented.

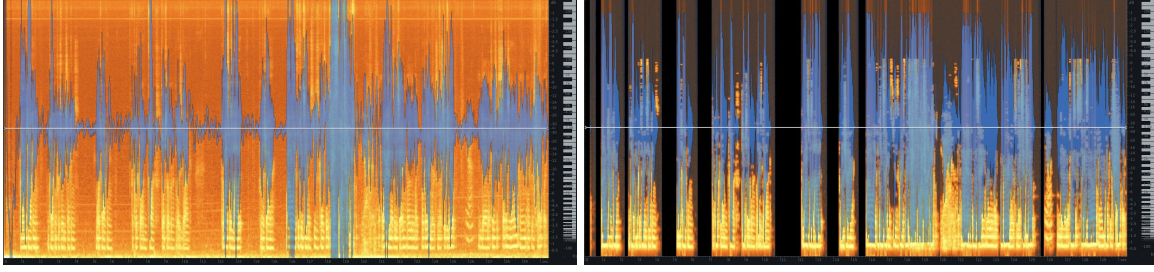


Figure 1: A graph of spectral information of a sample audio recording, with the waveform overlaid on top for clarity. Before the cleaning (left) and after (right).

1.1 Create a prototype app with audio record and play [3A]

In order to facilitate evaluation and testing of this work, I created a GUI that allows for recording of a voice sample, and playing it back both unprocessed and processed. It uses the egui framework, and allows for changing of the DSP parameters interactively to achieve the desired effect. The default settings are what I have determined to achieve the best possible outcome. The specific effects are discussed in the DSP section of this document. The GUI can be accessed directly in https://github.com/brettpreston/Rustic_Audio/Builds.

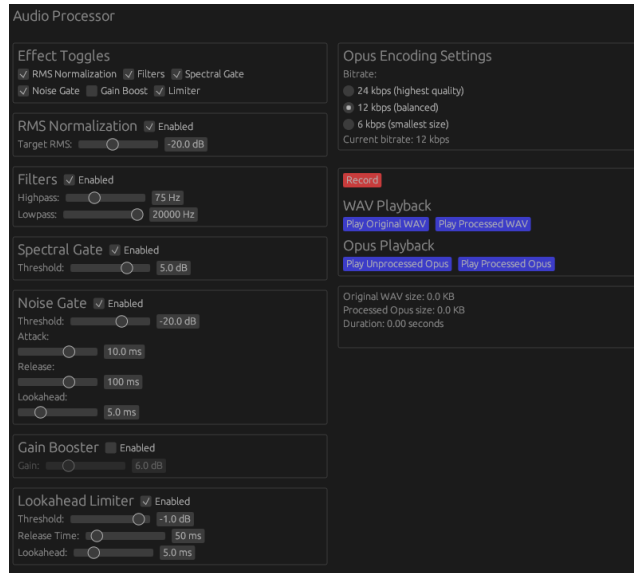


Figure 2: The testing and debugging GUI, with parameters set to the recommended presets.

1.2 Determine target packet size for katzenpost mixnet parameters and duration of message. [3F]

A typical implementation of the Katzenpost mix network has a packet size of 50kB, and sends and receives on average 2.5 packets per second. The mix network by default implements an average latency of 0.2s per hop, which results in an average round-trip latency of 1.8s. According to [1], this latency has a 2% chance of exceeding 4s. Up to 98% of a packet's size are usable payloads if the encryption used is X25519. With hybrid CTIDH1024-X448 NIKE, 3.2kB of those are taken up by a header, with over 93% of the bandwidth still being usable payload. So, if we keep an audio file under, say, 45kB, we will be able to send it in a single packet.

The following table is a comparison of sample file sizes in kB generated by mono VBR encoding in various codecs, starting with a lossless wav file. The cells are clickable, so the reader can verify the sound quality. For the HTML version of this table, visit <https://brettpreston.github.io/mixnet-samples>. If we use VBR, with a target of 12kbps, we may be able to send up to 30s of voice audio in a single packet.

Codec	Bitrate	Frequency band width	Complexity	Deep voice (32sec)	Deep voice + noise 35 sec	High Voice	High voice+noise 49 sec	Generic rock 51 sec
wav				2930	2990	4560	4330	6620
opus	64 kbps	wide	10	304	271	456	390	293
opus	32 kbps	wide	10	113	131	170	190	137
opus	16 kbps	wide	10	59.1	68.5	89.7	98.4	71.6
opus	12 kbps	wide	10	45.4	51.8	69.4	74.7	54.5
opus	12 kbps	wide	5	45	50.4	68.4	72.5	53.5
opus	12 kbps	narrow	5	49	54.5	76.2	78.6	60.2
opus	6 kbps	wide	10	25.9	26.9	41	39.3	30.6
opus	6 kbps	wide	5	25.6	26.4	40.1	38.6	30.2
opus	6 kbps	narrow	5	25.4	28.2	36.4	40.7	31.2
speex	64 kbps	wide	10	165	166	256	242	126
speex	32 kbps	wide	10	95.2	109	145	158	108
speex	12 kbps	wide	10	36.3	42.4	55.7	51.1	54.3
speex	4 kbps	wide	10	22.5	25	34.9	36	35.3
codec2	3200bps	default	n/a	12.7	14.1	19.9	20.4	15.5
codec2	1200bps	default	n/a	4.74	fail	7.47	fail	fail
codec2	450bps	default	n/a	2.4	fail	3.73	fail	fail

Table 1: A comparison of compression efficiency, with audio samples demonstrating quality available on click.

As can be heard in the samples above, as well as quantified in [2], we experience diminishing returns above 12kbps with Opus when it comes to speech compression. As long as minimizing the file size is a priority, either 12 or 16kbps appears to be a fine choice of bitrate. While we are encoding speech, it also doesn't make sense to encode multiple channels. So, the most efficient encoding comes with 12kbps VBR, mono. Since we are using VBR, the bitrate corresponds to the maximum file size. We can therefore be sure that a 30 sec recording will not exceed $30 \times 12/8 = 45$ kB, and in most cases will be smaller - especially after we applying all the noise reduction tricks described in the following sections.

The following figure comes from [2].

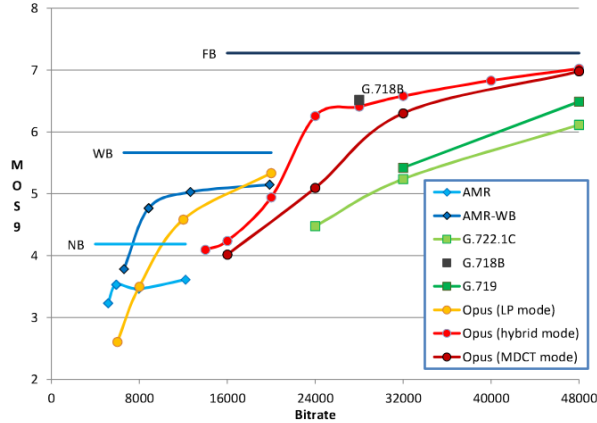


Figure 1: Voice quality evaluation results in clean speech

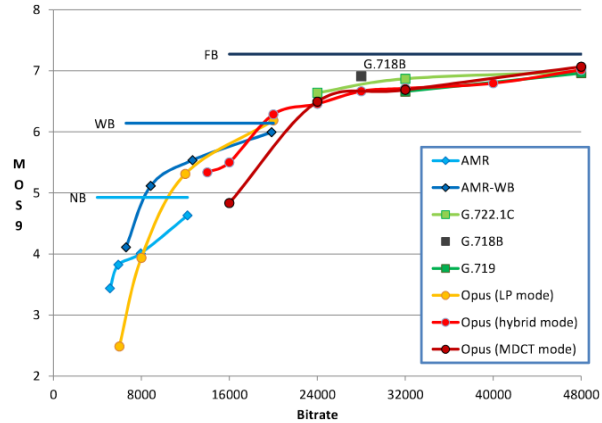


Figure 2: Voice quality evaluation results in noisy speech

It illustrates that in the case of Opus encoding, we experience diminishing returns in quality above the range of 12-16kbps.

1.3 Evaluate and audit audio compression libraries, consider security history of library / codec choice, and applicability of codec for mixnet parameters, implement the best compression. [3B]

After evaluating leading codecs in use for voice communication today, we find that Opus yields the best quality at lowest bandwidth. We therefore implement Opus encoding. It is worth noting that Codec2 is capable of extremely efficient encoding. Its efficiency relies heavily on sinusoidal coding and a narrow frequency band, which means that we quickly lose clarity and some distinguishing features of the original speech recording. The simplified harmonic content encodes less information compared to the popular codecs.

Here are a few Rust-based Opus encoders I considered:

- Magnum-Opus
- Opus-Native
- Ogg-Opus, which promises to create an Opus file in an Ogg container.

1.4 Evaluate and implement a secure decoder [3C]

I have evaluated the following decoders:

- Symphonia [3]

1.5 Codec customization [3E]

We use wide band compression as it has little impact on file size, and improves quality, and complexity 10 since we expect to use modern devices. Encoding with Opus, frames lengths under 20ms at low bit rates have audible distortions (as well as frames sizes over 80ms.) We will therefore stick to frame size of 20ms.

The encoder as it came presented several issues.

RMS detection and normalization had to be implemented to account for people's different mic settings

1.6 Implement efficient signal processing, including noise reduction to achieve best sound quality at low bandwidth [3D]

1.6.1 High pass and low pass

1.6.2 Spectral gate threshold detection

1.6.3 Noise gate threshold detection

1.6.4 Gain booster

1.6.5 Lookahead limiter

Background noise tends to be an issue with voice recordings, we could implement a noise filter with a small processing footprint, as well as a dynamic range compressor/limiter before encoding the message. This will improve the likelihood of a clear and appropriately loudness balanced audio message.

Noise reduction is recommended pre-encoding for maximum clarity. State-of-the-art noise suppressors tend to be based on neural networks, such as [4] [5]. This is somewhat resource intensive, but not out the question on mobile devices. There is an analysis of XIPH's efficacy and efficiency at <https://jmvalin.ca/demo/rnnoise/>. A demonstration of this process can be found here. A potential alternative is an automated Fast Fourier Transform noise suppressor, however, that is likely to involve extensive customization of available tools.

When it comes to Codec2 at 1200bit/s to 3200bit/s, an EQ boost of several decibels at 3kHz is a simple and effective way to improve consonant clarity. Noise suppression and equalizing can make a big difference when making Codec2 viable as demonstrated here. Compare with the original sample here. Without these adjustments, Codec2 may not meet the quality expectations of today's users.

- Pre-processing: implementing a noise filter before encoding. A demonstration of the effect denoising can have on Codec2 can be found in section 3 of this analysis.
- If trade-offs in the codec itself are acceptable, a wider frequency band on the higher end and improved handling of noise, both in noise reduction and encoding of consonants, would go a long way to improve the sound quality. Out-of-the-box Codec2 uses a very narrow subset of human voice frequencies and doesn't handle consonants well, which means it could have trouble with some consonant-heavy languages. These adjustments would come at the cost of compression efficiency.
- Post-processing: as it stands now, we can mitigate the losses after decoding by boosting what little of the higher frequency range survived. An equalizer is the most resource-efficient way to address this problem.
- Implementing a neural network-based decoder, akin to WaveNet, for frequency "reconstruction." [6] This is very resource intensive, and so may not be feasible on most personal devices. One should also keep in mind that these tools don't reconstruct the original voice, they create a clean simulacrum of a human voice which may not sound like the original speaker.

1.7 A separate repo which includes the audio cleaning, processing, encoding and decoding tools with documentation. [5C]

In addition to submitting these tools for implementation in Katzen/Echo, I have created this repo

https://github.com/brettpreston/Rustic_Audio

along with a list of dependencies and instructions on how this code can be used in any project that might need it. The version in the katzenpost repo <https://github.com/katzenpost/echo> has fewer dependencies, since it doesn't include the GUI. That repo also includes Katzenpost's Rust thin client, through which the audio tools interact with the network.

References

- [1] Ewa J Infeld, David Stainton, Leif Ryge, and Threebit Hacker. Echomix: a strong anonymity system with messaging, 2025.
- [2] Anssi Rämö and Henri Toukoma. Voice quality characterization of ietf opus codec. In *Interspeech*, 2011.
- [3] Philip Deljanov. Symphonia. <https://github.com/pdeljanov/Symphonia>, 2022.
- [4] XIPH. Rnnoise. <https://github.com/xiph/rnnoise>, <https://jmvalin.ca/demo/rnnoise/>.
- [5] Werman. Noise suppression. <https://github.com/werman/noise-suppression-for-voice/>.
- [6] W. Bastiaan Kleijn, Felicia S. C. Lim, Alejandro Luebs, Jan Skoglund, Florian Stimberg, Quan Wang, and Thomas C. Walters. Wavenet based low rate speech coding. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 676–680, 2018.