The development of the audio cleaning and compression tools.

Brett Alexander Preston

January-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. We deal with this problem 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 toolset was created for use with the Katzenpost/Echomix software, and other mix networks but it should be noted that any project interested in low-footprint, efficiently computed and good quality voice recordings may be interested in using it.

The code, as well as debugging and testing GUI is available at https://github.com/brettpreston/Rustic_Audio and a lean version without the GUI at 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 in the latter repo.

Contents

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 a feature-rich ecosystem with better community support. 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 fundamental frequency analysis and mathematical tools out of scope for this work. Rust also has more appropriate native decoders, which was crucial to the security of this toolset.

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 45kBs. This is ideal for use in modern low-bandwidth communications, and in particular in Katzen messenger, where the target packet size is 50kBs, with up to 5kBs available for the header and other cryptographic material.

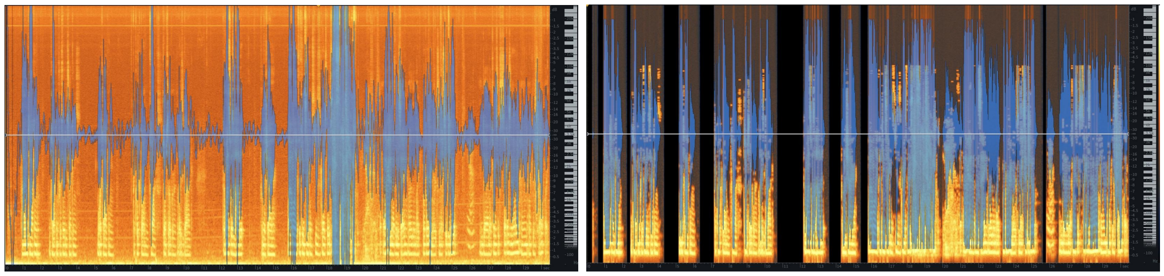
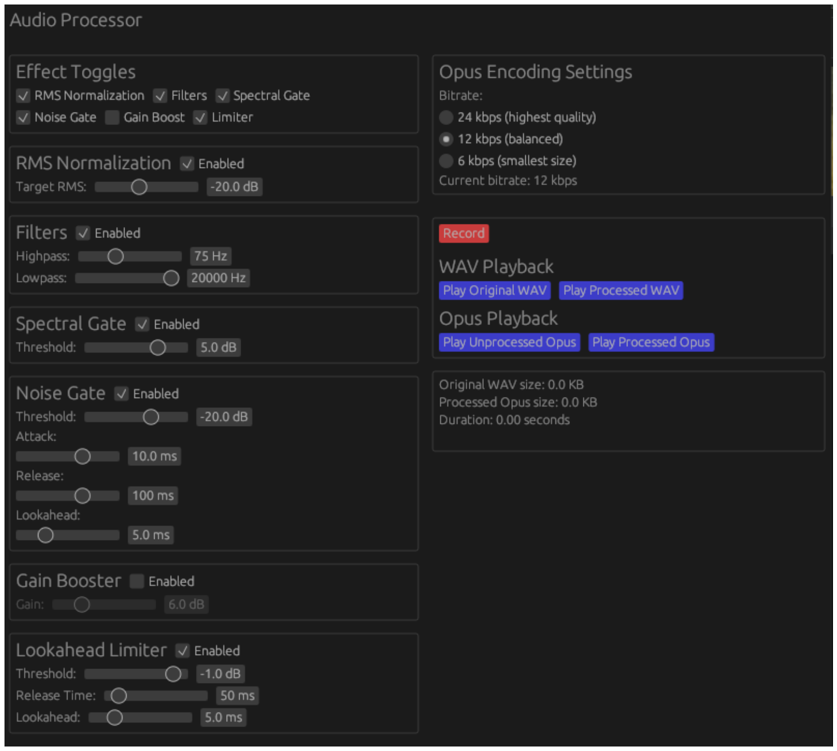


Figure 1: A graph of spectral information of a sample audio recording, with the waveform overlaid on top for clarity. Before the processing (left) and after (right).

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 optimal outcome. The specific effects are discussed in the DSP section of this document. The GUI can be accesses directly in https://github.com/brettpreston/Rustic_Audio/Builds.



Audio Processor

Effect Toggles
✓ RMS Normalization ✓ Filters ✓ Spectral Gate
✓ Noise Gate ☐ Gain Boost ✓ Limiter

RMS Normalization ✓ Enabled
Target RMS: -20.0 dB

Filters ✓ Enabled
Highpass: 75 Hz
Lowpass: 20000 Hz

Spectral Gate ✓ Enabled
Threshold: 5.0 dB

Noise Gate ✓ Enabled
Threshold: -20.0 dB
Attack: 10.0 ms
Release: 100 ms
Lookahead: 5.0 ms

Gain Booster ☐ Enabled
Gain: 6.0 dB

Lookahead Limiter ✓ Enabled
Threshold: -1.0 dB
Release Time: 50 ms
Lookahead: 5.0 ms

Opus Encoding Settings
Bitrate:
○ 24 kbps (highest quality)
● 12 kbps (balanced)
○ 6 kbps (smallest size)
Current bitrate: 12 kbps

Original WAV size: 0.0 KB
Processed Opus size: 0.0 KB
Duration: 0.00 seconds

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, 3.2kBs 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, 45kBs, we will be able to send it in a single packet.

The following table is a comparison of sample file sizes in kBs 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 $30 \times 12/8 = 45$kBs, 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].

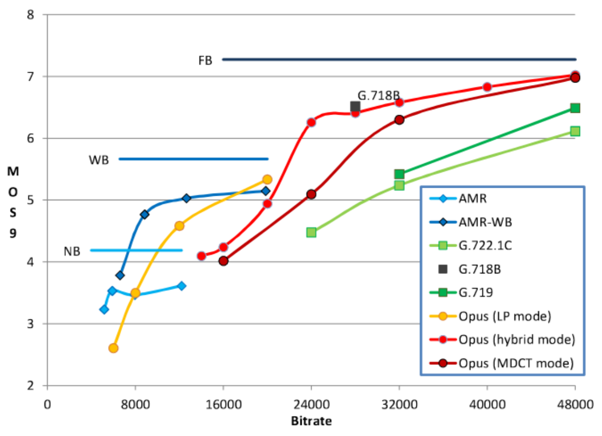


Figure 1: *Voice quality evaluation results in clean speech*

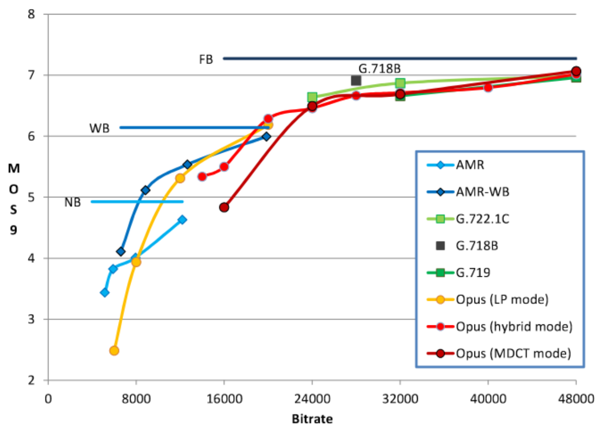


Figure 2: *Voice quality evaluation results in noisy speech*

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]

If we were building these tools for real-time streaming, we wold have to account for variable bitrate encoding potentially exposing the content of communications [4, 5] and therefore endangering the privacy properties of the system. However, since we're expecting to send these files in chunks, and since strong anonymity systems use padding to ensure all packets are the same size, we are free to use the more efficient VBR encoding.

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.

Thanks to the properties of Opus encoding, many of the tweaks I considered making in the encoding could be moved to the pre-processing effects chain with equivalent results. These include

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

1.6.1 RMS loudness normalization

I implemented Root mean square loudness normalization as I found users have a wide range of Mic gain input levels, I needed to start from a standard level to better predict the noise floor level of a voice recording for the sake of setting threshold values in further along the DSP chain.

1.6.2 High pass and low pass

Mic recordings may contain a lot of frequencies information that is unnecessary, we first filter out sub bass frequencies or potential DC offsets. Then we remove any high frequencies beyond human speech range.

1.6.3 Spectral gate threshold detection

using fast fortifier transforms we essentially split the signal into many multiple frequency bands and noise gate the band individually before recombining them.

1.6.4 Noise gate threshold detection

Final noise gate pass to remove audio below a set amplitude threshold.

1.6.5 Gain booster

Not usually necessary if using RMS. In many circumstances a microphone recording is too quiet and needs to be amplified for audible playback especially on sub par speakers (laptop or phone speakers for example). Increasing gain levels can lead to distortion without the use of a limiter.

1.6.6 Lookahead limiter

Reduces potential of distortion by anticipating drastic changes in signal amplitude and compensating.

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 Toukomaa. Voice quality characterization of ietf opus codec. In *Interspeech*, 2011.

[3] Philip Deljanov. Symphonia. `https://github.com/pdeljanov/Symphonia`, 2022.

[4] Andrew M. White, Austin R. Matthews, Kevin Z. Snow, and Fabian Monrose. Phonotactic reconstruction of encrypted voip conversations: Hookt on fon-iks, 2011.

[5] Chenggang Wang, Sean Kennedy, Haipeng Li, King Hudson, Gowtham Atluri, Xuetao Wei, Wenhai Sun, and Boyang Wang. Fingerprinting encrypted voice traffic on smart speakers with deep learning, 07 2020.