Kazooformation: An engaging data representation model

1 Problem Description

The broad fields of Digital Signal Processing (DSP) and Information Theory are a trigonometry and acronym riddled landscape that can be scary to learn. This results in a painful path towards developing the base intuition that is required to learn advanced DSP topics. Thankfully, modern algorithms paired with modern processors has allowed for more flexibility with DSP teaching tools.

This project is based on two assumptions:

1. DSP and information theory are inherently boring topics
2. Kazoos are not boring

Generally, the more engaging a teaching tool is, the more value it can provide as it's easier for learners to connect with the topic.

The choice to utilize the kazoo is a simple one - it makes the topic a lot more fun. This topic was inspired by the paper "Harder Drive: Hard drives we didn't want or need", by Dr. Tom Murphy VII, which led to me learning about topics that I otherwise wouldn't have if it wasn't for a "needless" approach. Another motivation is the fact that Kazoo sounds are an inherently challenging to interpret, creating an engaging project to develop!

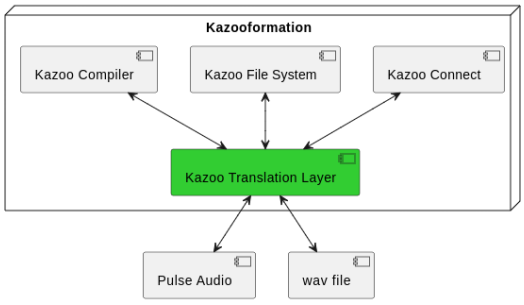
2 Proposed Solution

I propose the project **Kazooformation**. Kazooformation is a software system that uses kazoo sounds to encode, store, and decode data as a series of discrete kazoo signals. In order to transcode kazoo signals to and from binary data, a set of unique symbols must be created which represent some number of bits. The system for converting between data and kazoo signals will be called the "Kazoo Translation Layer". Applications that a user would interact with can be built on top of the translation layer.

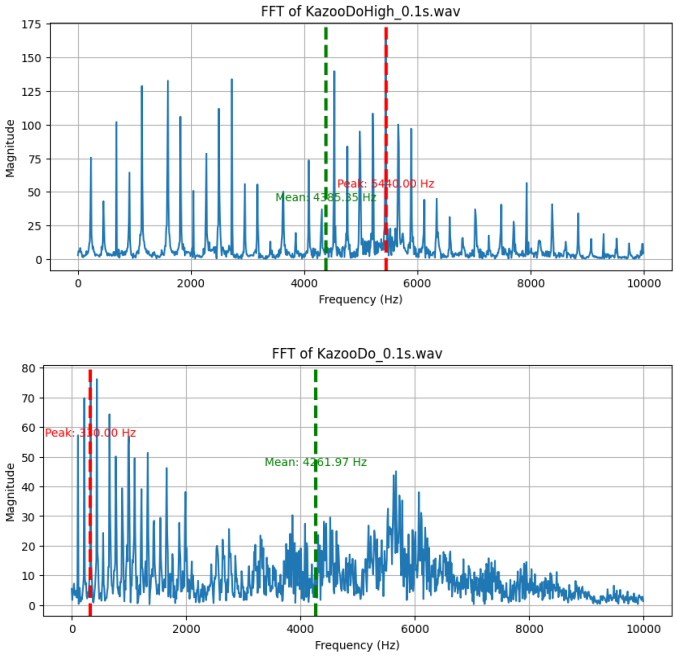
Once data can be converted to and from kazoo sounds, there are a limitless number of applications that can be built on top of this. I've narrowed it down to three primary applications.

1. Kazoo Connect - A terminal based chat client which connects to another client via audio input and outputs. This allows for kazooformation to work across a room or via a voice call.
2. Kazoo File System - Another terminal based client, similar to tar it packs and unpacks files into kazoo signals stored in an audio file.
3. Kazoo Compiler - This is a stretch/"just for fun" goal - add to the compiler made in CS-4550 to allow for kazoo based executables.

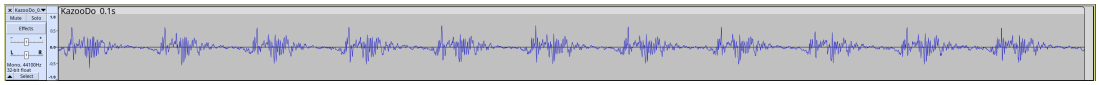
All of these will be built on top of either wav files for storage, or Pulse Audio for live input/output of audio data.



After some initial research, it's been shown that kazoos are one of the worst carriers for creating distinct audio signals. Below are output graphs from a Fast Fourier Transform (FFT) across two audibly distinct kazoo samples. It can be seen that the audio is very messy and some effort will need to be put into finding a reliable way to differentiate between distinct kazoo signals.



Another mathematical reality is the fact that at least 0.005 seconds of audio data is required to get the full scope of each kazoo sound, which can be seen in the waveform below (repeating patterns that are 0.005s long).



Due to this, the fundamental symbol rate limit of a kazoo based communication system is 200 symbols per second. With this symbol rate limitation, if there were only two distinct symbols, representing 1 and 0, the bit rate would be 200 bps which is nearly unusable. Thankfully, kazoos are capable of generating all sorts of different noises, so we can create as many distinct symbols as we want, allowing us to encode multiple bits at once. We are only limited by the ability to differentiate between distinct symbols. The more symbols we have, the higher the data rate, but the error rate will increase as well.

3 Technical Overview

The majority of this project will be a custom implementation, targeting Linux hosts. Due to this, few software tools will be used. This project will be created in C++17, utilizing a CMake build system. The audio integration will be done through WAV files or Pulse Audio. Signal analysis will primarily use the C++ library FFTW-3, or custom solutions when appropriate.

Although I have some experience with C++ and some with DSP, using kazoo signals creates a unique challenge which will test my abilities. The C++ FFT library in entirely new to me. I think the kazoo adds a unique challenge performance and reliability wise. This project will require for me to learn more about performant implementation while doing DSP.

Unit testing will prove to be very helpful for this project, I plan on using GTest and GMock as needed.
On the kazoo side of things, the website http://kazoologist.org/ has done the vast majority of the leg work when it comes to kazoo research. This is legitimately a helpful resource as it provides crucial information related to creating unique sounds. I sent an email to this person about a month ago and sadly did not receive a response.

4 Milestone List

Weeks 1-2 : Create the basic project structure, build basic targets with all libraries linked (FFTW-3 & GTest/GMock). Also, buy a kazoo and start creating symbols.

Weeks 3-4 : Start work on the Kazoo Transition Layer, basic unit tests. This will be the majority of the FFT work and will be the greatest challenge. Two unique symbols should be the initial goal.

Week 5-6 : Create a basic symbol identifier that can differentiate between at least 4 kazoo signal symbols.

Weeks 6-9 : Kazoo Translation Layer reliability and bit rate maximization - unit testing with introduced noise for later use over unreliable channels.

Weeks 10-11 : Kazoo Connect - unit test and manual testing using two separate computers in the same room.

Weeks 11-12 : Kazoo File System - Create the basic wav file wrapper, consider a nested structure.

Weeks 12-15 : Kazoo Translation Layer automatic channel quality monitoring and dynamic strategy management. Make the symbol rate/number of symbols non-static, so if the quality of the underlying communication channel worsens, the data rate will be slowed down to compensate for a high bit error rate and vice versa. There is a large chunk of time reserved for this as this period is focused on pushing the KTL to it's limits. Consider an RTP (Reliable Transport Protocol) approach depending on the time that is available.

Stretch Goal : Kazooxecutable: Integrate the kazoo translation layer into the compiler from CS-4550.

5 Validation Plan

Continual validation should be done via Unit Testing. The Kazoo Translation Layer should be fully unit tested as it is the basis of the whole system. This is a highly testable system, so there should be no problems achieving a target of 90% test coverage within the translation layer. The applications built on top of the translation layer can also be unit tested, but it may not be as valuable as they can be tested manually.
The other components should go through manual testing - it will be clear if they work or not.

- File System - Can a file be wrapped on one client, sent to another, and be unwrapped?
- Chat Client - Can two people have a typing-speed text conversation via a Discord call using the Kazoo chat client?
- Compiler - Can applications be built and then later run from wav files?