Kazooformation

An engaging teaching tool for the lowest layer of the OSI model

Joshua Jerred

https://kazooformation.joshuajer.red

Utah Tech University

CS-4600 Senior Project

24 April 2025

1 The Problem

Computer Science students are familiar with the OSI model, but often the details of the lowest layer, the physical layer, are glossed over. This is normally left to the electrical or computer engineering students. In the modern world we are seeing a shift towards software defined networking, where the physical layer can be controlled by flexible software instead of rigid hardware[3]. An example of this is the Joint Tactical Radio System (JTRS), which is a software defined radio program that was started by the US Department of Defense in the 1990s and is still being developed today[1]. It's clear that there is a need for a better understanding of the physical layer as software solutions are becoming more common.

Unfortunately, the broad fields of Digital Signal Processing (DSP) and Information Theory are riddled with trigonometry and dense with acronyms that can be difficult to learn. This results in a challenging path towards developing the base intuition that is required to learn advanced DSP topics.

Generally, the more engaging a teaching tool is, the more value it can provide as it's easier for early learners to connect with the topic. Therefore, an engaging physical layer teaching tool can be created based on the following assumptions:

1. DSP and information theory are often perceived as difficult and unengaging topics.
2. Kazoos are not boring.

2 The Solution

The choice to utilize the kazoo is a simple one - it makes the topic a lot more fun. This was inspired by the paper "Harder Drive: Hard drives we didn't want or need", by Dr. Tom Murphy VII, which enabled learning about topics that many otherwise wouldn't have learned about if it wasn't for a "needless"[2] approach.

The goal of this project was to create a new physical layer that can be heard and felt by the user, for the purpose of explaining fundamental DSP and signal processing topics. This was done by creating a software system that uses kazoo sounds to encode, store, and decode data as a series of discrete kazoo signals. This system is called Kazooformation.

Kazooformation uses recorded kazoo sounds to encode and decode data. Different sounds are used to represent different bits, and the series of sounds can be used to represent a byte stream. This is all audible, so the user can hear the data being transmitted, making it possible to see the physical layer in action.

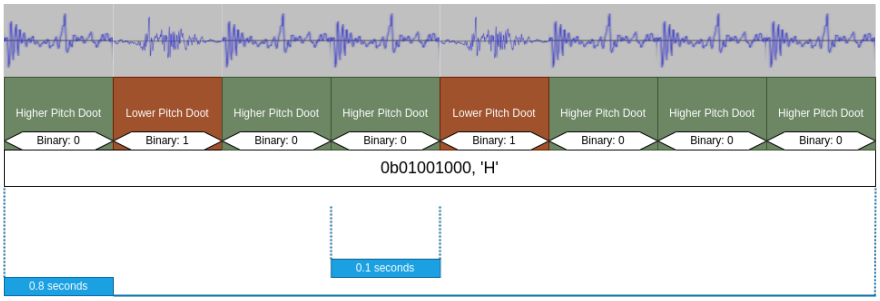

Figure 1: The ASCII letter "H" encoded as a kazoo signal using a binary model.

Ultimately, Kazooformation is based on Audio Frequency Shift Keying modulation, which means that the concepts directly map to real world modulation applications.

2.1 Kazoo Translation Layer (KTL)

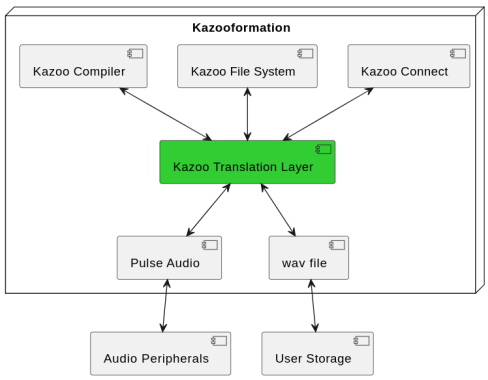

Figure 2: The Kazooformation system architecture.

KTL is a software library that is the core of the Kazooformation system. It is responsible for converting between serialized data and kazoo signals. As an abstract layer, it can be applied to a wide range of applications; if you have a byte stream, it can be converted to kazoo signals and back.

The most important detail of the KTL is the model system which is leveraged to use different symbols to represent different bits depending on the application. This enables flexibility in choosing symbol sets, symbol rates (baud), and bit rates, depending on channel limitations (e.g., audio cable vs. open air playback).

Although the focus was on the physical layer, the KTL offers a simple frame/packet structure that can be used as a simple data link layer.

2.2 Kazoo Connect (Chat Client)

Kazoo Connect is a simple terminal and/or web based chat client that uses the KTL to send and receive text messages using kazoo sounds. Its purpose is to demonstrate the KTL in a real world application. It uses the user's default audio input and output devices as the communication channel.

There is also a basic HTTP server that keeps track of received messages, allowing for a simple web interface to be used. The web client is a minimal React-based web client that provides a friendly interface for non-technical users.

2.3 The Code and How to Build It

Kazooformation repo: `https://github.com/joshua-jerred/kazooformation`

All build and use instructions can be found in the file README.md, located in the root of the repository. Kazoo Connect works on Linux and MacOS.

3 Technical Overview

This project is dependent on the Kazoo Translation Layer, all applications are built on top of it. It builds and works on both Linux and MacOS, but the development was done on Linux. The CMakeLists.txt file in the root of the repository is the best place to look to understand file structure.

Term Definitions

- **Symbol**: A symbol is a discrete kazoo sound that can be used to represent a bit or a series of bits.
- **Model**: A model contains a unique set of symbols, their corresponding bit values, and the modulation scheme used to encode the symbols.
- **Baud Rate or Symbol Rate**: The rate at which symbols are transmitted and received. Baud and symbol rate are used interchangeably. This is *not* the same as bit rate as multiple bits can be represented by a single symbol.
- **Bit Rate**: The rate at which bits are transmitted and received. This is defined as Baud Rate × Number of Bits per Symbol.
- **Symbol Level Alignment**: The process of detecting the start and end of a symbol in a stream of audio data. Required for decoding the symbols from unaligned audio data. WAV files are assumed to have symbol-aligned data, so alignment issues are minimal in that context.
- **FFT**: Fast Fourier Transform, an algorithm that converts a signal from the time domain to the frequency domain. See the image below.
- **Frequency Peak**: If an FFT is taken across audio samples, the peak frequencies can be detected. These are the audio frequencies that are most present in the audio samples.

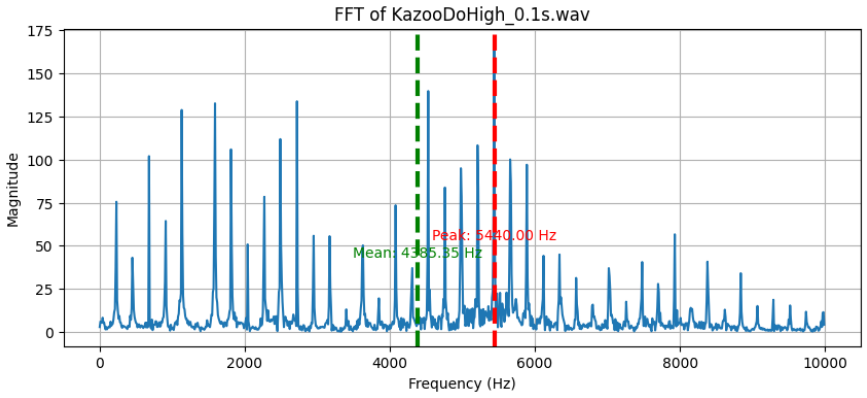

Figure 3: An FFT of a kazoo sound, with the peak frequency marked by a red line.

3.1 The Kazoo Translation Layer

The KTL was created as a C++ library using the CMake build system. Being a CMake interface library, it can easily be used in other CMake projects. A user of this library only needs to include the header ktl/translation_layer.hpp

Software Stack

- **Toolchain**: C++17, CMake, GNU/GCC.
- **FFTW-3**: A C library for Fast Fourier Transforms/audio signal analysis.
- **PulseAudio**: A C library for audio input/output.
- **GTest/GMock**: A C++ library for unit testing.

Key Software Components

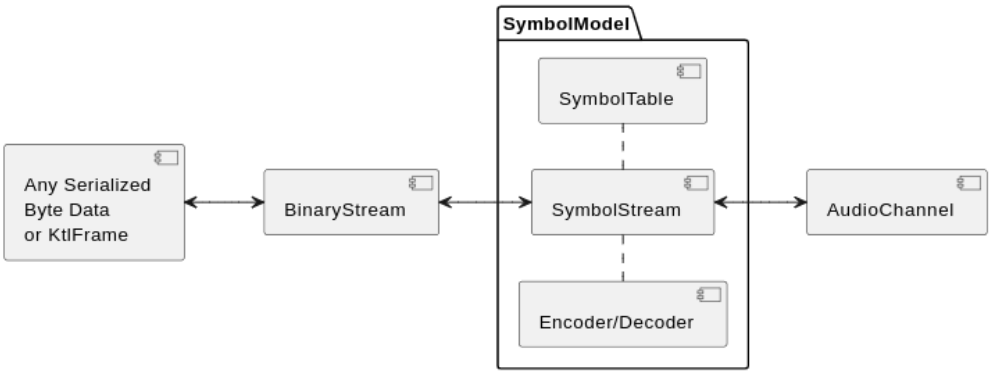

Figure 4: The data flow of the Kazoo Translation Layer.

- **AudioChannel**: A generic interface for continuous 16-bit PCM audio samples. This is used to allow for easy integration of different audio input/output software. It is used for both WAV files and PulseAudio. The audio channel interface is used by the model to encode and decode audio samples to/from the **SymbolStream**.
- **SymbolStream**: The symbol stream is a simple double ended queue that is used to store the symbols that are being transmitted and received. The symbol stream can be converted to and from a **BinaryStream**. Each model has a unique symbol stream (templated) as the symbols and their bit-width are defined by the model.
- **BinaryStream**: The binary stream is a class that is used to store raw binary data where the concept of a byte is abstracted away. This is important to keep the model system bit-width agnostic. It allows for easy serialization and deserialization of data.
- **SymbolModel**: This is the base class for all kazoo models. It contains the symbol set and stream, a mapping of symbols to bits, and the modulation scheme used to encode/decode the symbols. To implement a new model, a copy of an existing model can be made and added to the model list.
- **KtlFrame/Deframer**: KTL isn't supposed to go past the physical layer, but a simple frame structure is provided to allow for easy integration. The frame has unique start and end bytes (outside of 7-bit ASCII) to allow for easy detection of the start and end of a frame. The frame is converted to and from a **BinaryStream**. The frame structure is accessible to the users of the KTL.

Test Driven Development

As KTL is primarily an abstract library, test driven development is necessary. The majority of the KTL is unit tested using GTest and GMock.

The Models & Modulation Schemes

A few models have been created inside of the KTL. They can be found in the `src/ktl/models` directory. All models use the FFT wrapper created in the file `src/ktl/audio/fft.hpp` for signal analysis.

When it comes to encoding, the kazoo models take the provided BinaryStream and convert it to a symbol stream. The symbol stream is then processed in order; each symbol is mapped to PCM audio samples that are used to populate the generic audio channel.

All models operate at a fixed symbol rate of 10 symbols per second; this rate ensures the resulting signal remains audibly distinguishable and pedagogically effective. This is a somewhat arbitrary choice, individual models can use different symbol rates at the cost of reliability.

- **TestingModel**: A non-kazoo binary model (two symbols, representing 0 and 1) that is used for unit testing. It does not provide symbol level alignment. It encodes data as simple sine waves.
- **K1 Model**: The first kazoo model to use actual kazoo sounds. It is a binary model without symbol level alignment. It simply checks if the peak frequency of the signal is above or below a center frequency. This model is best suited for pre-recorded kazoo signal .wav files.
- **K2 Peak Model**: A 4-symbol kazoo model that uses a basic frequency peak detection algorithm to differentiate between the symbols. This model provides basic symbol level alignment. This is the most efficient model when using WAV files. Although it has symbol alignment, it is not very reliable due to the nature of kazoo sounds and the audio samples chosen for the symbols. This means that PulseAudio works, but the underlying audio channel must be near perfect (e.g. no open air communication).
- **K3 Reasonable Model**: A binary kazoo model that focuses on functionality over performance, allowing for open air communication between devices. Although the software is similar to the K2 model, the symbol audio samples were carefully recorded and chosen to be more distinct and consistent. This model has better symbol level alignment than the K2 model; across the length of a single symbol, multiple FFTs are taken to determine where one symbol ends and the next begins. This model has the most robust test coverage as well.

  The same peak detection algorithm is used in this model as even with carefully chosen audio samples, different peak frequencies can be detected in the same symbol. There is a good opportunity for improvement here.

3.2 Kazoo Connect

Kazoo Connect is just an application example of the KTL, it was intentionally kept simple. The terminal and web client/server are separate applications but they can communicate with each other.

Kazoo Connect Terminal

Kazoo Connect Terminal is a simple terminal client with a thread for KTL and a thread for the user input. All code for this is in the file `src/main.cpp`. It can be built and run using the target `kazoo_connect`.

Kazoo Connect Server

Kazoo Connect Web is similar to the terminal client but it provides a simple HTTP server for use with a web client. The server uses basic TCP socket handling with minimal parsing of the HTTP protocol. The server has paths for getting all received messages and status, sending a message, and clearing the list of received messages. The server can be built and run using the target `kazoo_connect_server`.

The web client is a simple React app that is derived from the create-react-app template with a basic form to send messages and a list of received messages. It can be run like any other node js application from the `src/kazoo_connect_server/web_client` directory. API requests are proxied to the backend.

3.3 Using KTL in Other Projects

Thanks to CMake, KTL can easily be used in other projects. Here are two simple examples that demonstrate KTL integration into other applications:

Example: Creating a CMake target that uses the KTL

```
# An application that uses the Kazoo Translation Layer
add_executable(kazoo_connect src/main.cpp)
target_link_libraries(kazoo_connect kazoo_translation_layer)
```

Example: Writing data to a .wav kazoo file

```
// Any type that can be converted to a std::span can be used
const std::array<uint8_t, 4> data = {0, 1, 2, 3};

kazoo::TranslationLayer tl{kazoo::TranslationLayer::ModelType::K3_REASONABLE_MODEL};
tl.addData(data);
tl.encode();
tl.saveWav("test.wav");
```

3.4 Audio Analysis Work

There was a large amount of work done to analyze the audio samples that were recorded and used in the KTL. This can be found in the `doc/` directory.Most of the work was done using Python along with various audio processing libraries.

Kazoo sounds are awfully noisy and inconsistent, FFT visualizations were critical for selecting the best audio samples and for knowing where their frequency peaks were.

4 Research Summary

- **https://kazoologist.org** - This website is one of the only resources available for kazoo research. It was incredibly helpful for understanding what was actually going on at the sound level.
- **Meetings with Curtis Larsen** - Regular meetings with Curtis were critical for keeping the project on track and resolving project planning issues.

- **Meeting with Bart Stander** - After mentioning my project to Bart, he provided me with the mathematical background for understanding music notes within an octave. This was helpful for getting unstuck in the early signal analysis phase and it helped with planning signal differentiation.
- **Wikipedia (Modulation Pages)** - Wikipedia is seemingly the best place to read about information theory and DSP concepts. The signal modulation page maps out a large number of high level concepts. It helped me connect the necessary dots between the various concepts.
- **StackOverflow** - StackOverflow was most used for figuring out how to use the FFTW-3 library. It of course was a constant companion and deserves a mention.
- **NoiseCollector on freesound.org** - The user NoiseCollector on freesound.org has a few kazoo audio files that were used to create the initial models before custom samples were recorded.
- **https://notblackmagic.com** - This website provides written tutorials on various DSP concepts including filtering and modulation/demodulation.
- **My Past Work** - In the past I have used Pulse Audio, WAV files, and TCP sockets. I pulled the initial code for these components from my past work.
- **Dr. Tom Murphy VII** - Without the paper "Harder Drive: Hard drives we didn't want or need", I would have never considered a 'needless' approach to developing learning tools.
- `https://www.youtube.com/watch?v=h7apO7q16V0` - This video was a great refresher on FFTs and how they work, needed for FFTW-3 usage.
- **Monoprice** - Like all other networking equipment, Monoprice sells kazoos, they're decent quality and cheaper than Amazon.

5 Further Work

Kazooformation is a functional system, but there is certainly a lot of room for improvement within the KTL.

- The various models can be parameterized to allow for different symbol rates and bit rates. There is clearly an opportunity for a machine learning model here. For example, a model could be trained to maximize symbol separation in the frequency domain under various noise conditions.
- The demodulation algorithm should be improved. The current algorithm relies on a naive approach to peak detection and it is not very reliable. An approach that properly applies various filters to the signal should be used.
- An 8-symbol or 16-symbol model could be designed with new audio samples to improve bit rate.

References

[1] Michael C. Cox. "THE JOINT TACTICAL RADIO SYSTEM (JTRS)". In: *Army AL&T* March-April 2001.5 (2001), pp. 5–7. URL: `https://asc.army.mil/docs/pubs/alt/2001/2_MarApr/articles/05_Joint_Tactical_Radio_System_JTRS_200102.pdf`.

[2] Tom Murphy VII. "Harder Drive: Hard drives we didn't want or need". In: *A record of the proceedings of SIGBOVIK 2022*. `tom7.org/harder`. Pittsburgh, USA: ACH, Apr. 2022, pp. 259–277.

[3] *What Is Software-Defined Networking (SDN)? — IBM — ibm.com*. `https://www.ibm.com/think/topics/sdn`. Oct. 2022.