CS378 - Computer Networks Lab

Lab 02 : Hands-On PHY Layer

Design Report

Omkar Shirpure (22B0910)
Krish Rakholiya (22B0927)
Suryansh Patidar (22B1036)
Parthiv Sen (22B1055)

Contents

1 Introduction

In this lab, we explore the implementation of a physical (PHY) layer for audio-based communication. The focus is on encoding, transmitting, and decoding data through audio signals. This document outlines the methods used for data transmission, the error detection and correction mechanisms implemented, and the results of our testing.

We employ the Cyclic Redundancy Check (CRC) method for error detection up to the specified limit of maximum errors. Our system is designed to correct up to 2-bit errors per transmission, requiring the addition of two extra points on the curve for correction.

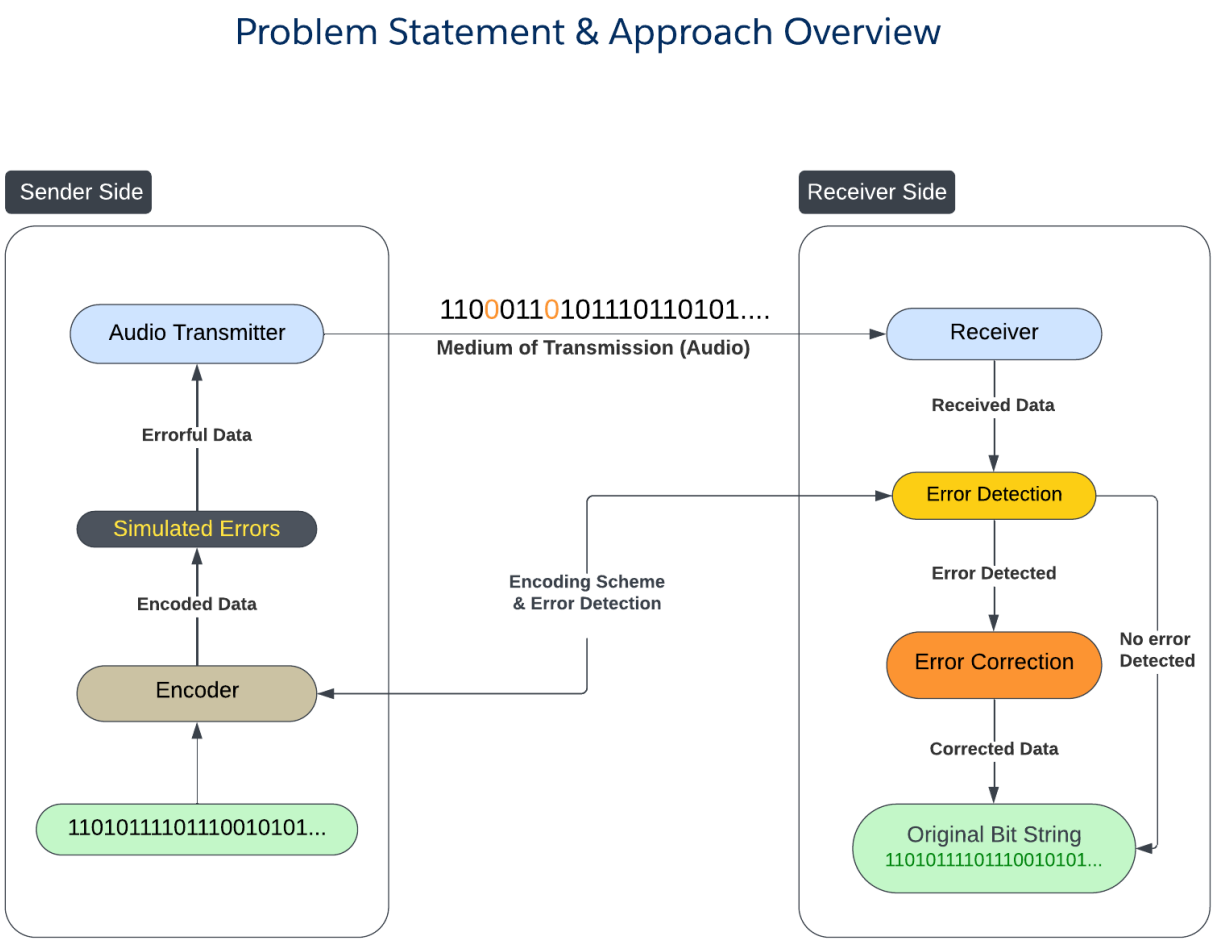

Figure 1: Overview of the Approach

2 Audio Transmission and Reception

2.1 Transmission Setup

The transmission process involves converting binary data into audio signals. We used the following frequency pairs for signal representation:

- '0' is represented by a 5000 Hz signal.
- '1' is represented by a 7000 Hz signal.

To minimize noise interference, frequencies above 5000 Hz were chosen and coprimes with each other. Each bit would be transmitted over a duration of approximately 1000 ms (subject to change for the best setup in further labs but for now keeping it in safer range), balancing speed and accuracy.

We were thinking the use of multiple frequencies (≥ 3) but keeping it reserved for the MAU Lab. If not used there, we will optimize the transmission further using multiple frequencies.

2.2 Modulation of transmission data

In our encoding scheme, we directly map binary values to distinct frequencies (5000 Hz for '0' and 7000 Hz for '1'), eliminating the need for modulation techniques. Since we don't use any high-pass or low-pass filters to shape or counteract signal components, the fixed frequency approach simplifies implementation.

2.3 Reception Setup

During the reception phase[1], the audio signals are captured through a microphone setup that samples the incoming frequencies. The recorded signal is processed to identify the transmitted frequencies and reconstruct the original bitstream. To mitigate noise, we employ a noise reduction technique by subtracting the average intensities at equidistant frequency points between the minimum and maximum frequencies, under the assumption that the noise distribution across this frequency range is uniform.

The reconstructed bitstream is then encoded into binary, where specific frequencies are mapped to *0*s and *1*s.

The process begins by defining a threshold, which is calculated using the formula:

$$\text{threshold} = \text{mean} + \text{factor} \times \text{std\_dev} \tag{1}$$

Here, the 'mean' is the average amplitude of the filtered signal, and 'std_dev' represents the standard deviation, which quantifies the variation in the signal's amplitude. The 'factor' is a multiplier that adjusts the sensitivity of the threshold.

With the threshold established, the next step is to identify the starting point of the transmitted message in the audio signal. Since the transmission uses two frequencies—one representing a binary '0' (lower frequency) and the other representing a binary '1' (higher frequency)—the task is to pinpoint where the actual data transmission begins.

To do this, the code examines a section of the filtered signal, starting from a fixed index (1000 in this case) and moving forward. For each position 'i', it compares the average amplitude of the next 100 samples ('i' to 'i+100') with the average amplitude of the previous 100 samples ('i-100' to 'i'). This comparison is done by computing the absolute difference between these two averages.

[1]The reception techniques is subject to minor changes or enhancements according to the results of testing and make it robust in varying acoustic environments.

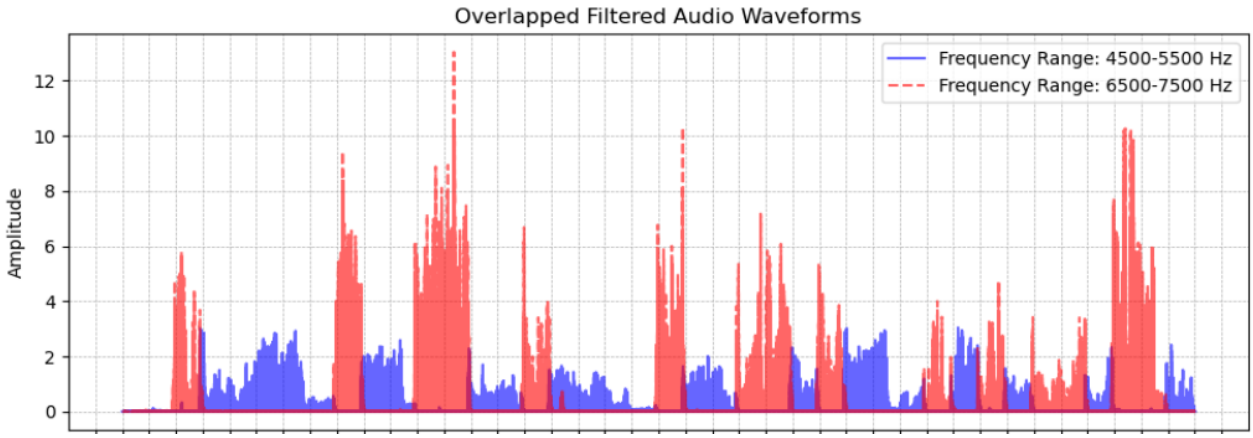

Figure 2: Spectogram that we get for the transmitted frequencies

This process is performed independently for both the lower and higher frequency bands. The goal is to detect a significant change in amplitude, which indicates the start of the actual data transmission. When the absolute difference exceeds the calculated threshold, it signals that the transmission likely started at that point. The code then records the corresponding index as the potential starting point for the message.

The final starting index of the transmission is determined by taking the minimum of the detected indices from both the low and high-frequency bands. This ensures that the analysis accounts for both possible bit values ('0' and '1'), and the actual message string is extracted starting from this index.

3 Encoding

The encoding process involves converting input data into a format suitable for transmission over an audio channel. We used the Cyclic Redundancy Check (CRC) method to append a CRC bit-string to our original data. The CRC is computed by dividing the original data (extended with zeros) by a generator polynomial. The remainder of this division is appended to the original data, ensuring that the resulting encoded string is divisible by the generator polynomial.

3.1 CRC Encoding

For error detection and correction, we need to ensure some things,

1. 1-bit detection and correction with (100% accuracy)
2. 2-bit detection and correction with (100% accuracy)

*The case of 1-bit errors is inherently accounted for within the framework designed to handle 2-bit errors. To ensure the above requirements are met, we need a CRC polynomial such that their $H_d \geq 5@20$[2] and also for the particular polynomial we choose, the HW(4)[3] should be 0 until data length ≤ 20.

We would be potentially using the CRC polynomial : `(0xbae)` as it satisfies all the given constraints for 2-bit error detection and correction. The reference for the data of the same is given at the end.
An example below for the bitstring "101"
Original Data: 101
Encoded Data: 101110101110100

3.2 Framing

Each frame consists of a preamble, header, data, and trailing bits:

- **Header:** Potentially includes frame length, and payload length. The header is a 6-bit binary field that represents the length of the data being transmitted. This length information is crucial as it tells the receiver how many bits to expect in the payload, allowing it to properly extract and interpret the data. The length is calculated by converting the number of bits in the `corrupted_data` to a binary string and padding it to ensure it is always 6 bits long.

  This header is then prepended to the `corrupted_data`, forming the complete data packet for transmission. By including the length as a header, the receiver can accurately determine the start and end of the payload, ensuring the data is correctly interpreted and any errors can be properly detected and handled.

- **Payload:** Contains the encoded data and appended CRC.

This structure ensures accurate data encapsulation and error detection.

[2]The notation "$H_d \geq 5@20$" indicates a minimum Hamming distance of 5 for a code length of 20 bits.
[3]HW(4), for example, is the number of undetected 4-bit errors out of all possible 4-bit errors affecting the codeword (both corruptions to dataword and the check sequence are considered)

4 Decoding: Error Detection and Correction

Upon receiving the audio signal, the first step is to decode it back into its original binary format. Once in this binary format, we employ a CRC checks for error detection and brute-force approach for error corrections.

4.1 Error Detection

For error detection, as discussed, we ensured the $H_d \geq 5@21$, and hence we can detect 2 bit errors which would require $H_d \geq 3@21$ During correction, the maximum number of errors in the bit-string for any iteration would be four, and we require a Hamming distance (H_d) of 5 to locate the correct dataword within that vicinity of 4 bit errors, which we ensure while choosing the CRC polynomial (to support data length ≤ 20 and $H_d \geq 5$).

To identify the incorrect bits in the corrupted bitstring, we will be using a brute-force approach that involves flipping pairs of bits across all possible index combinations. The idea is to iterate over every possible pair of bit positions in the bitstring and flip those bits. For each flip, you then check the CRC remainder. If the remainder becomes zero after flipping a pair of bits, it indicates that those particular bits were corrupted, and flipping them back to their correct values fixes the error.

4.2 Error Correction

Given the constraints on our bit-string length, which is limited to 20 bits or fewer, the brute-force method becomes a practical choice. The relatively short length of the bit-string allows us to efficiently apply an algorithm with a time complexity of $O(n^2)$, where n represents the length of the bit-string. This quadratic time complexity is manageable within our constraints, making the brute-force approach both feasible and effective for our error detection and correction needs.

5 References

- Cyclic Redundancy Check : `https://en.wikipedia.org/wiki/Cyclic_redundancy_check`.
- CRC Polynomial Collection : `https://users.ece.cmu.edu/~koopman/crc/hd5.html`
- `0xbae` as CRC Polynomial Data : `https://users.ece.cmu.edu/~koopman/crc/c12/0xbae.txt`
- Python PyAudio Documentation : `https://people.csail.mit.edu/hubert/pyaudio/`