Acoustic Network with Coroutines

Fall 2024

Abstract

This report presents the development and implementation of an acoustic communication network using sound waves as the physical medium. The project involves designing and implementing the physical, link, and network layers of the communication stack. Key techniques such as modulation schemes, error correction, clock synchronization, CSMA were employed to ensure reliable data transmission. Challenges such as phase inversion, clock drift, and latency were addressed, and solutions were implemented to mitigate their effects.

Contents

1 Introduction

Acoustic networks offer a unique alternative to traditional wireless communication systems. This project explores the implementation of an acoustic communication network using sound waves as the medium. The goal is to develop a complete communication stack, from the physical layer to the network layer, and to evaluate its performance.

2 Physical Layer

Physical layer is the lowest layer of the network stack. It is responsible for transmitting raw bits over the physical medium. In our project, the physical medium is the air (or wire), and the raw bits are modulated into sound waves.

Breifly, our physical receives bits from the above layer, modulates them into sound waves, and call the audio framework to play the sound waves. There are several important techniques in the physical layer, including modulation schemes, frame structure, forward error correction, clock synchronization, etc. We will discuss them in the following sections. We will also discuss some hardware issues, which critically affect the performance of the physical layer.

2.1 Sound Representation

Sound is a wave that propagates through the air. The sound wave can be represented by a sequence of samples. More specifically, the sound wave can be represented by a sequence of floating point numbers, which are the amplitude of the sound wave at different time points.

2.2 Audio Framework

The audio framework is responsible for playing and recording sound waves. It provides APIs for the physical layer to play and record sound waves. The physical layer play sound waves by calling the audio framework's play API, and record sound waves by calling the audio framework's record API.

There are many audio frameworks available on the market. We have tried several of them, including JACK, miniaudio. Both of them are cross-platform audio frameworks, which can be used on both Windows and Linux.

We find that the different audio frameworks share similar API. Basically, we need to provide a callback function, which will be called by the audio framework. The type signature of the callback function is usually like this:

```
void callback(float* in, float* out, int frames, void* userData);
```

In every second, the audio framework will call the callback function multiple times. The time interval between two calls is determined by the frame size and the sample rate.

We didn't tune the frame size in our project and we keep the sample rate to be 48000 Hz.

2.3 Hardware Issues

In this section, we will discuss various hardware issues that we encountered during the development of our acoustic communication system. We will describe the nature of these issues, their causes, and the solutions we implemented to mitigate their effects.

2.3.1 Phase Inversion

One of the issues encountered in our project was phase inversion. When a sinusoidal signal $\sin(t)$ is played and subsequently recorded, the recorded waveform appears as $-\sin(t)$, indicating a phase inversion.

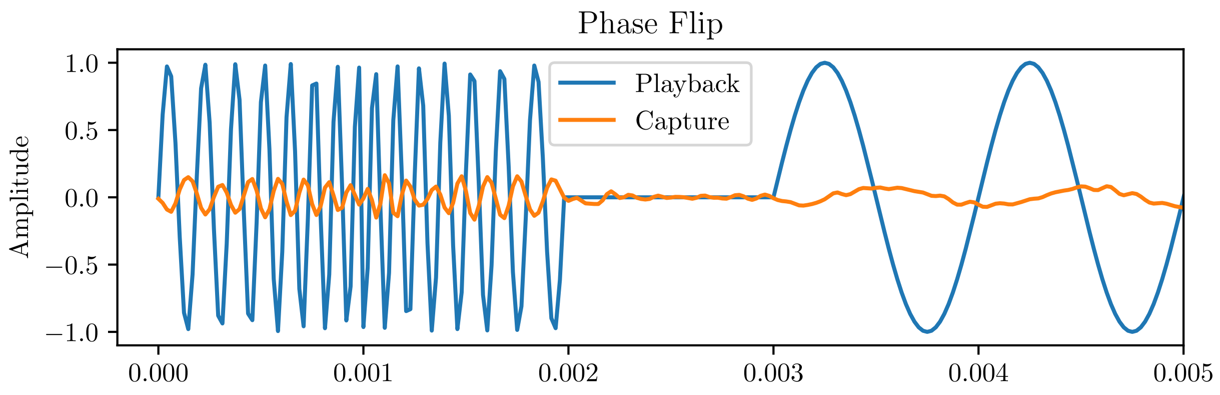

Figure 1: Illustration of phase inversion in recorded sound wave.

As illustrated in Figure 1, the amplitude of the recorded sound wave is the negative of the original played sound wave.

The underlying cause of this phase inversion was not extensively investigated. However, this issue can be readily corrected by multiplying the recorded sound wave by -1.

2.3.2 Clock Drift

Clock drift occurs when the clock of the recording device is not perfectly synchronized with the clock of the playback device. As a result, the recorded sound wave will gradually drift out of sync with the played sound wave over time. This can cause significant problems in our communication systems, as the timing of the received signal is crucial for correct demodulation and decoding.

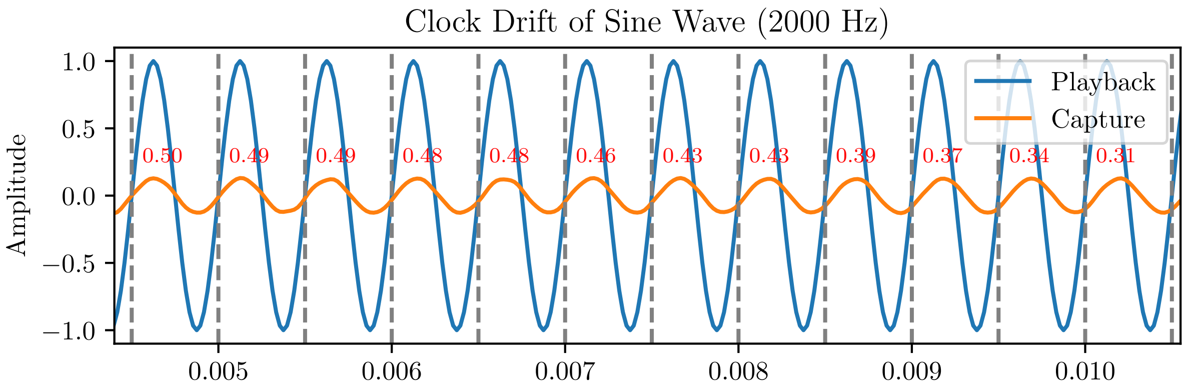

Figure 2: The plot to show the clock drift.

The red text is the similarity score between the played sound wave and the recorded sound wave, which is calculated by dot product, max is 0.5.

As illustrated in Figure 2, at the beginning of the recording, the recorded sound wave is in sync with the played sound wave. However, as time progresses, the recorded sound wave gradually drifts out of sync with the played sound wave. **Around 30 cycles**, the recorded sound wave is almost $\pi/2$ out of phase with the played sound wave.

This issue has a significant impact on Phase Shift Keying (PSK) modulation, as the phase of the received signal is crucial for correct demodulation.

2.3.3 Speaker to Microphone Latency

Latency is the time delay between the played sound wave and the recorded sound wave. Latency is crucial in our communication system, as we implement a CSMA protocol, which requires to sense the medium before transmitting. If the latency is too large, the sender may not be able to sense the medium correctly.

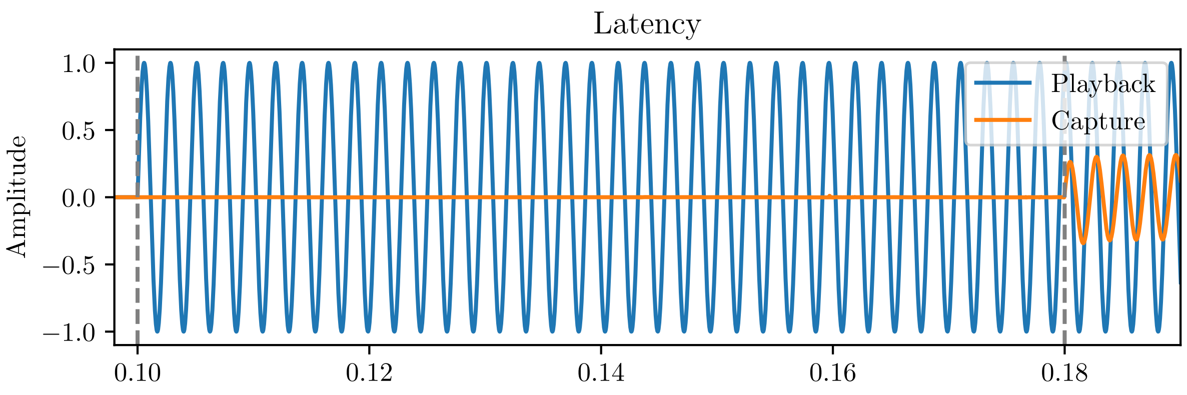

Figure 3: The plot to show the latency on the provided sound card.

We measure the latency of the provided sound card by playing a sound wave and recording it at the same time. As illustrated in Figure 3, the latency of the provided sound card is around 0.08 seconds. The experiment environment is 1) Linux, 2) miniaudio framework, 3) use a wire to connect the output and input of the sound card.

The latency of 0.08 seconds is unacceptable for our communication system. We find that the latency is much smaller when we use Windows.

2.4 Clock Synchronization

Clock synchronization is the process of synchronizing the clocks of the recording device and the playback device. This is crucial for our communication system, as the timing of the received signal is crucial for correct demodulation and decoding, no matter the modulation scheme.

We use chirp to synchronize the clocks. A chirp signal also denotes the beginning of a frame.

Chirp signal is described by the following equation, where f_0 and f_1 are the start and end frequencies, respectively, and T is the duration of the chirp signal:

$$f(t) = ct + f_0$$
$$c = \frac{f_1 - f_0}{T}$$
$$x(t) = \sin\left(2\pi\left(\frac{c}{2}t^2 + f_0 t\right)\right)$$

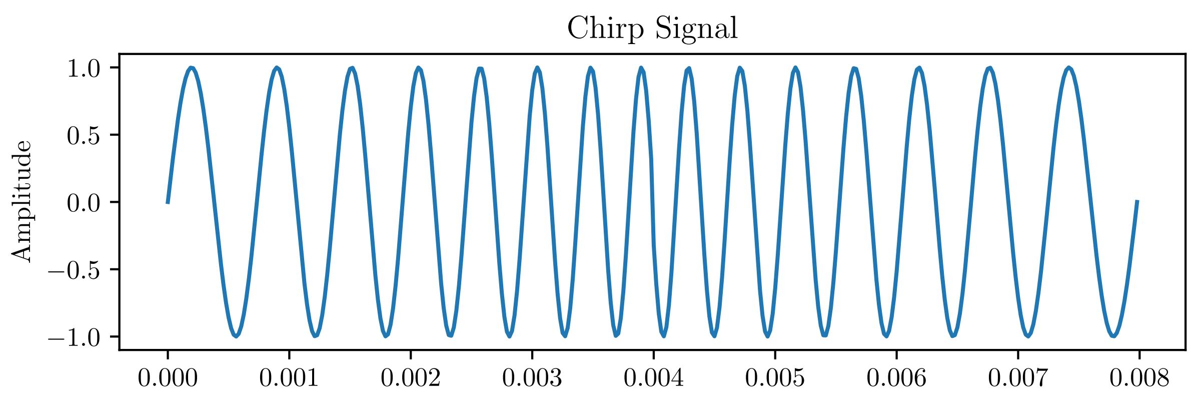

Figure 4: The plot of a chirp signal.

The Figure 4 shows the plot of a chirp signal. The actual chrip signal we use is $x(t)$ concatenated with $-x(-t)$, which has a better performance in practice.

To find the chirp signal, we do a sliding dot product between the received signal and the chirp signal, and if the similarity score is larger than a threshold, we find the chirp signal.

Our clock synchronization algorithm is extremely accurate, we can archive sample level synchronization, which means the error is less than 1 sample (1/48000 seconds).

2.5 Modulation Schemes

We have experimented with several modulation schemes in our project, including Amplitude Shift Keying (ASK), Frequency Shift Keying (FSK), and Phase Shift Keying (PSK).

Thanks to Python, we can easily visualize the signals, and iterate fast.

2.5.1 ASK

ASK is a modulation scheme that encodes symbols in the amplitude of the wave. Since we are reminded that transmitting sound waves through the air is not reliable, we didn't use ASK in our project 1. **We use FSK in Project 1, and use ASK in Project 2.**

However, for cable connection, ASK is a good choice. We find that we can tranmit a bit every 2 samples, where one $= [0, 1]$, zero $= [0, -1]$.

By ASK, we archive a 24000 bps transmission rate. (There are some overheads in the frame structure, so the actual transmission rate is less than 24000 bps.)

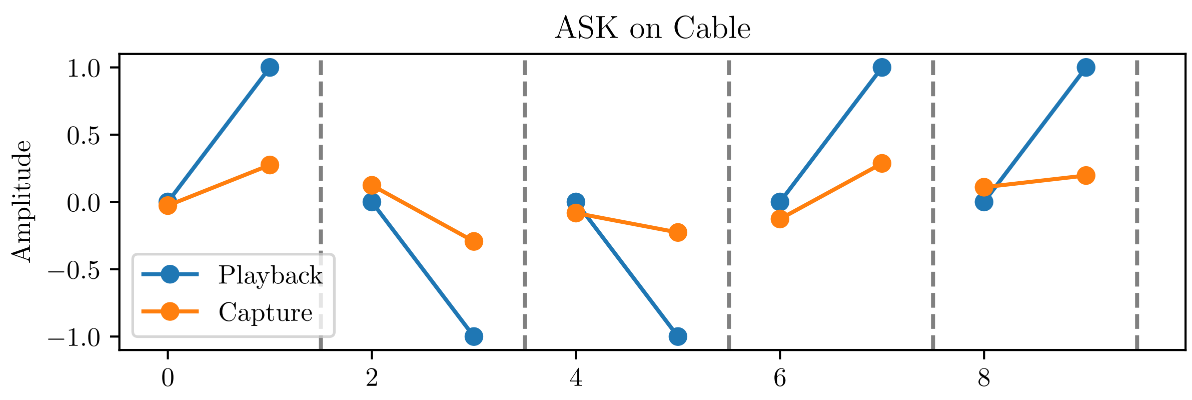

Figure 5: The plot of ASK modulation.

The Figure 5 shows the plot of ASK modulation. The bit sequence is 1, 0, 0, 1, 1.

To modulate the bit sequence, we simply concate the signal of each bit.

To demodulate the signal, we do a dot product between the received signal and the signal of each bit, and choose the bit with the largest similarity score.

2.5.2 FSK

FSK is a modulation scheme that encodes symbols in the frequency of the wave.

We find that FSK is a good choice for our project 1. Because the frequency of the sound wave is more reliable than the amplitude of the sound wave when transmitting through the air.

Although the issue of clock drift is still a problem, we can still achieve a good performance with FSK. Because it has a higher tolerance to clock drift than PSK.

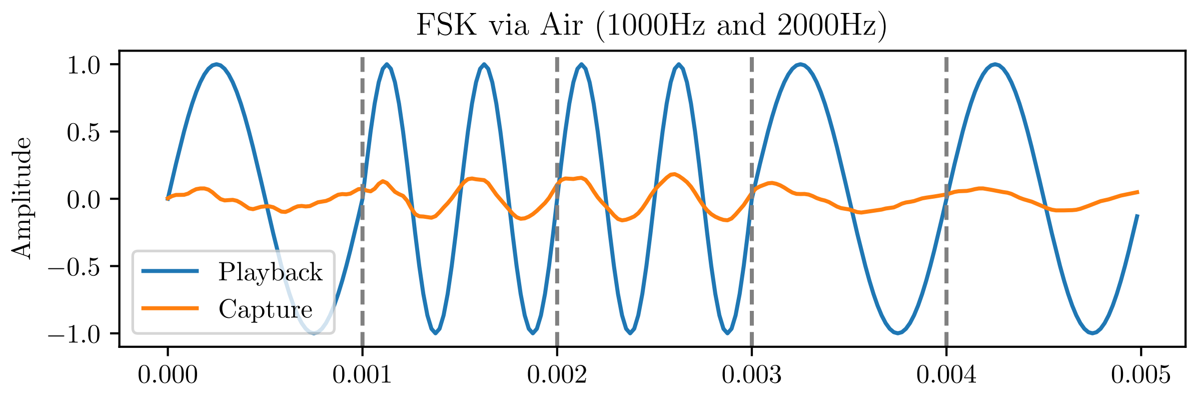

Figure 6: The plot of FSK modulation.

The Figure 6 shows the plot of FSK modulation. The bit sequence is 1, 0, 0, 1, 1. The frequency of 1 is 1000 Hz, and the frequency of 0 is 2000 Hz.

In the evaluation, we set the frequency of 1 to be 4000 Hz, and the frequency of 0 to be 8000 Hz. **FSK can directly pass the distance challenge without any complicated trick.**

2.5.3 PSK

Due to the clock drift issue, we didn't use PSK in our project 1, although we have implemented the PSK modulation and demodulation with OFDM.

2.6 Frame Structure

Frame Structure in Project 1 and Project 2

```
fielid | chirp | data |
bits   |       | 40   |
```

Since in Project 1 and Project 2, we are transmitting a fixed length of data, we don't need to include the length of the data in the frame structure. We use FEC to correct the error in the data. We do not have retranmission in Project 1 and Project 2 so there is no CRC.

Frame Structure in Project 3 and Project 4

```
field | chirp | len | src | dest | type | seq | data | crc |
bits  |       | 14  | 2   | 2    | 2    | 4   |      | 16  |
```

As we divide the network into layer, the frame structure is also divided into several fields. `chirp` and `len` is encoded and decoded by the physical layer. `src`, `dest`, `type`, `seq`, `data` and `crc` is encoded and decoded by the link layer. Therefore, `crc = crc16(src + dest + type + seq + data)`. To avoid the corruption in `len`, we limit the length of the frame.

2.7 Forward Error Correction

Forward Error Correction (FEC) is a technique used to detect and correct errors in transmitted data. By adding redundant information to the transmitted data, the receiver can correct errors in the received data without the need for retransmission.

First, we need to investigate the error model of the physical layer. The error model of the physical layer is burst error. For example, if we fail to locate the chirp signal, the received signal will be a burst error, probably the entire frame is corrupted. Therefore, our FEC should be able to correct burst errors.

We have tried two FEC schemes, Hamming code and Reed-Solomon code.

Hamming code is a simple FEC scheme that can correct single-bit errors and detect double-bit errors. But it is too weak to correct burst errors. Moreover, the overhead of Hamming code is too large.

Reed-Solomon code is a powerful FEC scheme that can correct burst errors. More specifically, Reed-Solomon code can correct t errors in a codeword of length n, where $t = \frac{n-k}{2}$, and k is the number of information bits in the codeword. **We adopt RS(255, 223) in our project**, which can correct 16 errors in a codeword of length 255. RS code has a much larger block size (255 bytes) than Hamming code (7 bytes), so we can handle burst errors more effectively.

Although RS code has a large block size, it is still not enough to correct the burst errors in the physical layer. We use another technique to correct the burst errors, which is **randomization**. The sender and receiver has a pre-defined random seed. The sender will reorder the bits in the frame according to the random seed, and the receiver will reorder the bits back according to the random seed. This technique can effectively correct burst errors.

3 Link Layer

Link layer is the second layer of the network stack. In contrast to the physical layer, which is responsible for transmitting raw bits, the link layer is responsible for transmitting frames, which are sequences of bits with a specific structure. The structure includes the source address, destination address, type, sequence number, data, and CRC.

Link layer in our project is responsible for reliablly transmitting bits to the receiver. It means that our link layer has meachanisms to retry the transmission if the transmission fails. We use ACK to confirm the successful transmission, and retransmit the frame if the ACK is not received.

3.1 CSMA Protocol

Carrier Sense Multiple Access (CSMA) is a protocol used to avoid collisions in the network.

We implement a CSMA protocol using a state machine. If the sender wants to transmit a frame, it first senses the medium. If the medium is idle, the sender transmits the frame. If the medium is busy, the sender waits for a certain time and senses the medium again.

But because of the latency issue, CSMA is not very effective in our project.

4 Network Layer

Network layer is the third layer of the network stack. It is responsible for routing packets from the source to the destination.

In our project, instead of implement the network layer protocol, we leverage the OS's network layer protocol implementation by using TUN.

4.1 TUN

TUN is a virtual network device that allows the user to send and receive IP packets.

In detail, we create a TUN device, and assign an IP address to the TUN device. The operating system will route the packets to the TUN device, and we can read the packets from the TUN device in our program, and send the sound waves to the physical layer. For the received sound waves, we can write them to the TUN device, and the operating system will route the packets to the destination or the application.

5 Above Network Layer

Above the network layer, we can implement any network protocol, such as TCP, UDP, etc.

Since we use TUN, we can use any network protocol that is supported by the operating system.

However, there are issues in the evaluation. Windows has many background applications that use the network, which overwhelms our acoustic network. We use a whitelist to filter the packets, and only forward the packets that are sent by our application.

6 Implementation

We use C++20 to implement the project. Our program launch two threads, one for audio framework, and one for the main program.

For the audio thread, which is almost a real-time program, we use a **lock-free queue** to communicate between the audio framework thread and the main program thread to avoid the unexpected latency.

For the rest of the program, we use **coroutines** to handle the asynchronous tasks. Coroutines are a powerful feature in C++20, which can be used to write asynchronous code in a synchronous way. It is much easier to write and understand than the traditional callback-based asynchronous code.

However, coroutines have some limitations. It is not designed for real-time programming, so we use a lock-free queue to communicate between the audio thread and the main program thread.

Theoretically, coroutines may have a performance bottleneck, because the scheduler of the coroutine is not designed for real-time programming. However, in our project, the performance bottleneck is the sample rate. The CPU in our laptop is powerful enough to handle 48000 samples per second.

7 Feedback

7.1 Interesting and Challenging

The project is very interesting and challenging. It is a great project. The documentation and evaluation is very clear and detailed.

The project introduces many interesting concepts that I have never learned before, such as the physical layer protocol, link layer protocol.

The setup of the project is pretty easy. Since sound applications are in user space. User space programs are easy to write and debug.

The project is very challenging. The physical layer is very hard to implement, especially with someone who has not learned *Signals and Systems* before. Fourier Transform, Convolution, etc. are crucial in the physical layer design.

7.2 Physical Layer Takes Too Much Time

One concern is that the physical layer takes too much time to implement. Most computer science students have no knowledge in communication systems.

Moreover, the physical layer is not very useful for computer science students. I believe that most of the computer science students will work on link layer and above, not the physical layer.

I suggest that the project should less focus on the physical layer. It takes me sometimes to finetune our implementation to archive the performance goal.

Futhermore, I hope we can add several advanced topics. One is high speed network, such as DPU, RDMA, etc. We can have a programming project using DPU, RDMA or DPDK.

8 Conclusion

This project successfully demonstrated the implementation of an acoustic communication network using sound waves as the physical medium. The physical layer was designed to handle modulation, demodulation, and synchronization, while the link layer ensured reliable frame transmission through mechanisms such as CSMA and acknowledgment-based retransmission. The network layer was integrated using TUN devices, allowing the use of standard network protocols. It is a great project, and we have learned a lot from it.