CS120 Project Report

Zhelin Zhu, Bangzheng Fu

{zhuzhl, fubzh }@shanghaitech.edu.cn

Disclosure Preference: Disclose

0 Project 0

What We Did

Project 0 serves as a warm-up assignment in our course. There are no restrictions on the choice of programming languages. Opting for C++ and utilizing the ASIO-based library JUCE, we embark on developing our Aethernet program.

The assignment requires the implementation of two functionalities. Firstly, we are tasked with recording a period of sound and playing it back. Due to our unfamiliarity with the JUCE library, we approach this task through trial and error. It becomes apparent that JUCE does not provide low-level interfaces, and understanding how JUCE invokes the ASIO driver proves challenging. Complicating matters further, the audio channel is shared by both the input and output buffers, leading to interference in the stream. To mitigate this, we ensure the output buffer is cleared before each loop iteration, preventing echoes and ensuring the recorded sound remains clear.

The second task involves playing a predefined sound while recording, and we choose the song *Astronomia* for this purpose. The performance outcome is amusing, adding an enjoyable element to the exercise.

Adjusting hardware and drivers presents its own set of challenges. Notably, one of our computers, a Lenovo Legion Laptop, appears to be incompatible with the ASIO4ALL driver. Consequently, the JUCE program fails to recognize the sound card, necessitating the use of the original sound card driver. This issue is resolved upon obtaining an UGREEN external sound card.

A notable hurdle in the development process involves setting up the JUCE project in Visual Studio. Each time it is opened, manual addition of ASIO-SDK files to the project, including the path, is required. This

inconvenience is reported on the course GitHub. We find a solution. By copying the files from asio-sdk/common/ to JUCE/modules/, which is the JUCE installation path, we eliminate the need to manually add the path in Visual Studio.

1 Project 1

What We Did

Among the four projects, Project 1 emerges as the most formidable, primarily due to its high device dependency. The project presents a series of challenges, with the success of our implementation being heavily influenced by environmental factors. Notably, a change in the testing location necessitates repeated tuning of parameters. The vulnerability of audio transmission becomes evident, as even slight noise can compromise the signal. Factors such as table stability, computer height, stereo power, and the distance between the transmitter (Tx) and receiver (Rx) all play pivotal roles in the success of the transmission.

Our chosen preamble is the up-chirp signal spanning from 1000Hz to 8000Hz. Initially, we employ fast Fourier transform (FFT) to detect the preamble. However, when calculating the convolution of up-chirp and down-chirp and implementing FFT, the frequency spectrum exhibits two peaks at a specific position. The program scans these positions to determine if the preamble is detected. Unfortunately, the algorithm does not perform optimally, lacking sufficient distinguishability in the spectrum to precisely identify the preamble's start position.

To address this issue, we refer to the Matlab template code provided by the professor. This code utilizes correlation coefficients to determine the

detection of the preamble. Additionally, a squeak signal is addressed through sliding-window smoothing. For bit modulation, we experiment with various methods. Amplitude Shift Keying (ASK) proves effective only in short distances between the microphone and the stereo, as greater distances increase susceptibility to interference from the environment, resulting in poor transmission accuracy. While ASK facilitates a high Baud rate, it relies on volume differentiation to distinguish symbols.

Frequency Shift Keying (FSK) necessitates Fast Fourier Transform, a concept covered in the course EE150. Each symbol corresponds to a particular frequency of a sine wave. The receiver (Rx) processes the acoustic signal, applies FFT, and extracts frequency information, with the main frequency representing a symbol. However, the frequency stability is inadequate for a 15-second transmission requirement, and the overall available bandwidth is constrained. The received signal exhibits low sharpness of separation in the frequency spectrum, resulting in a high error rate of approximately 50 percent.

Ultimately, we settle on Phase Shift Keying (PSK) as our chosen modulation method. Using a 1000Hz sine wave as the carrier wave and another sine wave with a phase offset of , PSK outperforms ASK and FSK. A significant challenge we encounter is mitigating environmental disturbances. Transmitting information through audio in the presence of disturbances proves challenging, especially without a background in relevant courses. We continually fine-tune parameters such as convolution sum threshold, smoothing weight, and match length, often facing frustration when a definitive solution eludes us, forcing us to rely on trial and error to identify a set of viable parameters. The need for parameter adjustments persists when running the program in different locations.

Suggestion

We propose that the TA provides more comprehensive guidance on communication technology within the course. We feel that the current content does not align well with the course objectives, and additional direction from the TA could significantly enhance our understanding of these concepts.

2 Project 2

What We Did

Thanks to the audio toolkit, we have a more reliable channel for transmitting acoustic signals. The project mandates a bit rate greater than approximately 6000 bits/s, prompting us to shorten the preamble. Initially, we employ a full Amplitude Shift Keying (ASK) method, utilizing eight sine waves with varying volumes. The Baud rate is set to (bit rate) / 3, which, though not as fast as anticipated, lays the foundation for combining ASK with Phase Shift Keying (PSK). This combination adjusts the Baud rate to (bit rate) / 4.

However, the modulation method proves sensitive to the stereo's power, leading to distinct output power between two connected computers. This discrepancy requires manual adjustments, involving manipulating the rotary knob on the mixer and adjusting the volume in the Windows system control center. This process not only proves time-consuming but also hinders bidirectional transmission. Achieving accuracy on one computer leads to failure on the other. Despite this limitation, we press on beyond the acknowledgment task.

Drawing inspiration from Ethernet encoding, which uses baseband

transmission, we adopt the Manchester code for bit encoding. Each bit is represented by four samples: ”0” as [0.95, 0.95, -0.95, -0.95] and ”1” as [-0.95, -0.95, 0.95, 0.95]. Decoding involves examining every four samples. The preamble mimics the Ethernet Frame preamble with 7 bytes of ”10101010” and 1 byte of ”10101011.” Achieving a bit rate of 11k bits/s, this approach effortlessly tackles subsequent tasks. Although accuracy is high, occasional errors prompt the implementation of CRC-32 for packet correctness checks. Additionally, packet numbering facilitates effective retransmission handling. Despite taking several days to complete, the bidirectional transmission task exceeds expectations, delivering consistently accurate results.

Suggestion

Some peers encountered driver issues, particularly with Intel Smart Audio Technology drivers disrupting ASIO4ALL when running the JUCE program. While TA assistance resolved the problem for some, variations in solutions arose. For instance, my old laptop faced a similar issue, and the method that worked for a friend did not yield the same results. In a scenario with only my old laptop, I might have received a score of 0 in the project. I recommend the course group explore ways to proactively address and minimize such driver-related problems for a more seamless project experience.

3 Project 3

What We Did

In Project 3, the audio component requires minimal modification as it inherites from Project 2. Only a few lines of code need adjustments. The

message transmission between Node 1 and Node 2 is swift and accurate. To enhance our project, we integrate pcap to capture packets. By utilizing pcap and Wireshark, we encode our packets following the ICMP protocol. In our program, we use pcap to sniff the packet, checking its type and destination address to determine the appropriate action. If the packet is destined for the router or the local LAN it belongs to, the program handles forwarding or replies accordingly. If the destination address is Node 1, we use ASIO and the sound card to send the data, wait for the acknowledgment, and reply back. In the case of a ping from Node 1 to Node 2, the router replies through the sound card.

When performing Network Address Translation (NAT), we follow the instructions. If the destination belongs to the LAN, we simply forward it. If it is destined for the WAN, we modify the source address as Node 2 and record the original source address in the NAT table. We also note the ICMP ID for later reference during the reply. This allows us to check the table and make forwarding decisions when the ICMP reply is received. During NAT traversal, we include the local address information in the data region. When Node 2 receives the ICMP, it checks the local address to determine the port for forwarding, receives the corresponding reply, and then replies back with the information.

Suggestion

I try the virtual network card Windows Loopback Adapter. And Microsoft provides no document about it. I suggest the course group could provide some instructions on it.

4 Project 4

What We Did

This project involves less work compared to previous ones. We focus on implementing DNS functionality. Inspired by the analysis of the command "ping www.baidu.com," we replicate the DNS steps. Node 1 sends the network address we want to ping, simulating a local DNS server. The process involves:

1. Sending a DNS request to the DNS server at Shanghaitech. 2. Upon receiving the DNS reply, attempting to ping the IP it provides. 3. Capturing the reply and sending it back to Node 1.

We consider the HTTP task to be more intricate than some bonus tasks. As a result, we opt to forgo Task 2 and instead concentrate on completing the bonus tasks in Project 3.

Suggestion

The project is good.