Project Report

——By Shi Junyan and Xu Chenhao

To be honest, my teammate and I have spent a huge considerable amount of time and have encountered numerous difficulties in those four projects, and we put in a lot of effort to overcome them. We think the project might have be the most difficult part in this course, which makes this course the most difficult one among all course in SIST.

In the first project, we thought the project would be very simple initially, just as it appeared on the surface. But the following experience told us that we were totally wrong. Firstly, the language problem, we have not learn java language before and TAs recommended us to use Java language because they didn't know what unexpected problems would happen with C language, so we spent a lot of time to get familiar with java language. Java has a more verbose syntax compared to some other programming languages. It requires a lot of boilerplate code, which can be overwhelming for beginners. Secondly, unlike all the projects we have worked on before, this project has no code foundation at all. All framework should be made by hand. We have to learn the sound card driver protocol asio by ourselves and attempt to search online for user manuals that hardly be found at all. These difficulties exist even before we actually start writing code. We have downloaded and tested asio4all, JASIOhost, Audacity and other sound card driver. The resources for using these programs are not easy to search on website.

After overcome the language problem and unknown library, we started to writing the code for first project. We drew inspiration from an example provided by Jasiohost itself, after modifying and attempt to make the computer hear a segment of sound and then play them back. Then a strange problem happened in the code: when there is only input channel or output channel, it worked very well, but if the input channel and the output channel are opened simultaneously, the output channel worked very well but the input channel will be skipped directly without any errors. After we have traversed the code of this library, we found it in a rather inconspicuous corner that if two channels exist simultaneously there is a variable called "isinput" must be judged before you use this channel, otherwise it would be considered as calling the output channel. After overcome this, we finally can send and receive sounds at the same time.

Then we began to think about ways to transmit data with Amplitude Shift Keying, Frequency Shift Keying, or Phase Shift Keying. Three methods are varied. We have tried the amplitude shift keying and find that even two microphones are very close, the attenuation of sound amplitude during transmission is still very unstable. After comparing these advantages and disadvantages, we decided to use frequency shift keying. Then, another very strange problem occurred. Through time comparison, I found that calling the jasiohost library between two data reads to ensure cache availability requires a significant amount of time, even much longer than the time required for reading and calculating which causes the problem of disappearing during the reading process. Finally, we had to ask our classmates before we found out that the first time we used it after downloading asio4all, we must use the built-in sound system of the computer which has 44100Hz rate. If we use a microphone with lower than 44100 Hz, this program will encounter problems, even if it was set to 44100Hz sampling rate, it would not actually achieve 44100Hz sampling rate. After the first download, we used headphones with a sampling rate of 8000Hz for testing, then the problem happened. This issue was resolved after reinstalling asio4all.

Then we went to project 1 task 3, which led to a big headache for us. We immediately realized the need to use Fourier transform but how to align the obtained acoustic signals? How to narrow the range of values of the Fourier transform sound wave signal to the original range? How can the transmission be completed within the specified time? How can we ensure accuracy when we need to transmit so much information in such a short amount of time? For the first question, we found that using the “1010” start alignment method is very unstable. So, we took a clever move: due to the large bandwidth, we directly used a high-frequency signal as the alignment signal. After receiving the alignment signal, the receiver will enter alignment mode and move the signal bit by bit backwards until the high-frequency peak obtained from the Fourier transform is less than a certain amount. After finishing alignment mode, the receiver begins the data analysis mode. For the second question, we use four different frequencies of sound waves to simultaneously transmit two bits. To balance transmission rate and the accuracy, we use 128 points to transmit a basic information. Then we found that there is a slight difference between the speed at which the sender sends sound and the speed at which the receiver receives sound. The sender will be slightly faster, which leads to even if the beginning is aligned, the back will gradually deviate, causing all errors to start at a certain point in the middle. Due to this error occurring approximately after 200 transmission units, we can only package 100 data into one frame and add a header in front of the frame to help us align each frame.

Even though the theoretical requirements can be met now, it still has a significant gap from theory to physical practice. The above code sometimes still has an error rate of over 50% during runtime, as if our previous efforts were completely ineffective. To be honest, this has had a significant impact on us. After several comparisons between the data converted by the receiver and the original data, we found that the probability of being recognized as a lower frequency at higher frequencies is greater than that of being recognized as a higher frequency at lower frequencies. This situation is particularly evident when switching from low frequencies to high frequencies. We finally realized that due to the attenuation speed of high-frequency signals in the air being greater than that of low-frequency signals, and it is impossible for the sound devices of two computers to come into contact at zero distance. So we have to add different weights in the high-frequency region to balance the different attenuation speed.

In the second project, we have to using data cables to transmit sound signals, which greatly reduces the possibility of errors but also increases the requirement for speed. The ACK mechanism is much more difficult than the physical layer in the project 1. This Project also required for a Mixer to simulate a practical physical layer, which enable sender to hear the sound sent by itself. Obviously, the probability of collision will increase significantly. A new method of transition is wanted!

The first problem is to translate binary code to text. We had use text format document before to deal with, so, this time, we translate and store them in array. In task 1, two nodes are connected with different characters. So, we don’t need to care about collision. But the transmit speed is challenged. We need to transmit 6250 Bytes in 10 seconds, which means 50000 “0”s or “1”s. This need to be much more faster than before. So we encode 2 “0” or “1” in one number to accelerate this period. The collision problem exists when enter into task 2. Because receiver need to send back acknowledgement to sender, the identifications of sender and receiver are not stable from now on. Both users need to change between two characters. We exchanged the codes and launched the getackdata method function. When receiver receive a pack, it will check the cyclic redundancy check. If the cyclic redundancy check is corresponding with the message transmitted, it will return a correct acknowledgement to sender, else, a wrong acknowledgement will be presented. But how to make sender know that and resend the package? An idea occurred. We use an array to record the bool value of receiving. And check it every turn when choosing which package to send. The change between sender and receiver took us a lot of time. Firstly, the users don't know whom the audio was send to. So, we tried to use different frequency to distinguish them, which can't avoid to use higher frequency. But after testing many times, the performance seemed good! Then, in order to be a sender and receiver at the same time, the parallel method is called. We set two threads for sending and receiving. The last thing we have done in project 2 is error detection. The method we use is to set an error_time, which can record the time that not receiving any data. So after 200 times, it will raise an error message to notify that the connection has been interrupted.

Project 2 realizes the wire link of the physical layer, so that the transmission no longer uses sound signals but electrical signals. It eliminates the problem of audio degradation during transmission and reduces the requirements for the transmission environment. In Project 1, we need to put two computers very close to each other, and even need a microphone to be a speaker, and the positions of the front, rear, left, and right need to be carefully adjusted. After replacing it with an audio cable, the transmission distance is determined by the length of the wire, and the environment does not need to be silent.

The first two projects built a local area network using audio signals. Now in Project 3, we need to interconnect local network with operational networks with IP, which need some preliminary preparation. At the very beginning, some software are needed for packet construction, sending, receiving, and parsing. We spent lots of time to get familiar with socket in java. Then for debugging packet encapsulation and network protocols, Wireshark is an essential tool. It can capture and analyze the network traffic of the specified network interface.

In task 0, we connected them through campus LAN and sent an IPv4 packet. In task 1, we encountered a problem. The input need to be as similar as the icmp in the real world. So we need to modify the surface in node1 to enable an input. Then it's hard to suit Aethernet with Internet. Because the JAVA language is not so friendly for other package to import. We spent a lot of time working on such kind of problems, which lead to stay up all night long to study that thing. Instead of using a smartphone as node3, we use one laptop to work as node1 and node3. This could be eliminate the number of devices, on the other hand, this could also make debugging easier.

The last project may be not the most difficult. But considering that it need the basis of all three projects before, it becomes different. The last project is to make node2 as a DNS server, which can receive ping request from node1 by cable connect, and ping the ip in real world. After trying several days, we abandoned it. The most challenging part is to achieve the transform from physical layer and internet layer. We also need to send back the ping result to node1. So the trouble is great.

All in all, we spent this semester in nervous, tension and fatigue. The project need great effort to design, test and debugging. The lectures indeed give some basic idea and paradigms of those concepts. But we think more practical instructions should be given to student to have a better start with project. Also, instructions for tools and libraries are needed. The difficulty of this course mostly comes from these project. It is undeniable that these projects do provide space for development for students with spare academic abilities, but for ordinary students, they bring a lot of mental and physical pressure. We hope this course will optimize more.