ShanghaiTech CS120 Project: Final Report

Huaqiu WANG
wanghq@shanghaitech.edu.cn

Yu SHI
shiyu1@shanghaitech.edu.cn

During this semester, our group has completed four projects in CS120. In project 1 and 2, our group implemented Aethernet, a network for transmitting signals based on sound. And based on this network, in project 3 and 4, we connected Aethernet to the actual network to form a complete network system.

Project 0

Firstly, in project 0: warm up, we started by getting familiar with the ways to sample and process audio using a program. Since java is the recommended language in this class, our group used java to complete the project. Project 0 required us to be able to record and play back what was recorded, as well as play back and record sound at the same time. For java, there is the Java sound library that comes with java, as well as the ASIO-related `Jasiohost`, and in the end our group chose Java sound directly to complete the project.

Project 1

After the warmup, we officially started building the audio network in project1. First, at the very basic level, we need to be able to define a sound wave of our own - a sine wave, for example. At the same time, it is crucial to set a fixed sampling rate: a higher sampling rate means that the sound wave can represent a wider range of frequencies. In project, we use 48000hz as the sample rate, which is the upper limit for mainstream sound cards, as it already covers the frequency range of human hearing. Also, it maximizes the use of the audio spectrum, and if other sampling frequencies are used, the sound card may implicitly resample, which introduces additional noise.

The signal we are transmitting is an electrical signal of 0s and 1s (in project 1 we need to transmit a 10000bit txt file), but we are transmitting it via sound, so we need to map the 0s and 1s into a sound wave, i.e. Modulation. Similarly, the receiver, after receiving the sound, needs to process the audio to get the original 0s or 1s, i.e. Demodulation. In class, we learned about modulation methods such as PSK, ASK, and FSK, and in project 1, our group used PSK directly to modulate. In addition, if the modulated signal is played directly, then it may be mixed with some noise, which may solve the wrong content in the decoding process. Therefore, we need a pre-defined preamble to start the signal, chirp signal is a common preamble, and a sliding window is needed to correlate the received signal with the preamble during the decoding process, so that the preamble can be recognized, and the decoding process can start. In addition, CRC code can also be added to check

whether there is any error in the decoded signal.

In the process of completing project1, we borrowed the MATLAB code provided by teacher ZHICE, YANG, and implemented the two parts of Sender and Receiver respectively. In the process of actual external implementation, we encountered a lot of difficulties. For example, the volume of the recorded audio was a big problem. The sound of the computer's own fan would turn into a murmur to be recorded, and the microphone of my computer (Lenovo LEGION) might not be very sensitive, so my computer could only take without an audio transmission cable. In addition, if the volume of the recording is too low, then the decoding process will not recognize the preamble or decode something wrong, while if the volume is too high, there will be a lot of noises. In addition to java sound, our group also uses a java library called `WavFileWriter`, which can save an `Arraylist` as a wav file, and also import wav files into an `Arraylist` and process them. When faced with the problem of varying volume, we normalize the received audio after converting it into an array to avoid the problem of too high or too low volume. After using this method, the decoding rate reaches 100%.

Project 2

In project 2, instead of the transmission through external ways, audio cable and USB sound card are used, and we need to transmit more bits in less time. In order to transmit more bits in less time with less noise, our group changed from PSK Modulation in project 1 to 16-QAM Modulation where orthogonal sinusoidal signals encode 4 bits in one sign (4 sin signals by 4 cos signals). The final bandwidth of the network is approximately 8kbps, but a maximum speed of 12 kbps (64-QAM) is reachable. We does not apply 64-QAM inconsideration of data alignment in our physical layer implement as the data are mainly byte aligned. However, due to the fact that the modulation in amplitude needs to be set to the correct threshold value when decoding (and at the same time, at different volumes the value always satisfies), our group has not been able to achieve 100% correctness and therefore struggled to get full marks in Task 2. Therefore it was difficult to get full marks in Task2.

The Task2 is to implement ACK, so it requires Sender to be able to receive ACK while sending packets and process the data instantly. However, the Java Sound we used is not able to process the received audio instantly, it can only record a segment and process this segment. At the same time, turning on the Recorder also takes a while, which cause problems and wastes much time. Our solution is to send and receive packages in cycles periodically: a cycle is about 2s, in the first period, Sender sends a package, Receiver receives and processes the information, in the remaining time, Receiver replies with an Ack, Sender receives an Ack, and then both parties enter standby state waiting for the next cycle to start. We synchronize by checking the received acknowledgement, however This solution is rather reluctant, since the best solution is still to change Sender and Receiver to be able to realize multi-threaded processing and receive and send instantly (thus we use the `JACK` in the later project).

For error detection, we tried both basic parity code and CRC checksum on the decoded data. They both lead to effectively error detection with acceptable redundancy. However, 2 dimensional parity code as error correction code does not perform as expected due to unsatisfying accuracy of 16-QAM modulation & demodulation.

At the same time, with audio cable, the sound played by one side will also be recorded by itself, so

this is also a problem to consider, both Sender and Receiver need to recognize whether the recorded sound is transmitted by the other side. At last, due to time constraints, we were only able to complete parts of Tasks 1 and 2, and were not able to implement the CSMA content of Task 3.

Project 3

The project 3 requires the Aethernet to develop a local area network and aims to interconnect the Aethernet with operational networks, which relies on the IP protocol. In this project, we developed a multithread framework with JACK audio to parse the data while receive and send audio signal. JACK is proved to reduce the latency drastically. The framework include a IO processing thread updating the input buffer and output buffer and a data processing thread pending and demodulating packets. The data processing thread also switch the status of a node based on the data received, and whether the node sends certain data or decode certain data is based on the status of the node. The 2 threads form a typical producer-consumer scenario, thus we use 2 `BlockingQueues` to synchronize them with output and input requests, and data in blocking queues are transferred to a to-decode array or input buffer in each callback. When sending packages within the network (between node 2 and node 1), the framework is able to meet the limit of latency and decode packets with buffer size of 128 and sample rate of 48000 Hz.

We applied `Pcap4j` package to capture and send packages from Aethernet to Internet devices. The official and unofficial example code repositories pcap-4j and pcap4j are of great help when we generate ICMP packages in developing, it helps to solve basic problems such as how to send or monitor on certain packages. A challenge in project3 is that the packet sent by the Aethernet may fails to be detected by node 3 and Wireshark, this may because of certain faults in the form of ICMP package, and debugging on the ICMP package sent take a long time. The challenges we encountered in this process include that the network toolkit fails to correctly display the latency due to invalid timestamps in ICMP and package loss caused by inappropriate data format conversions, etc. The latter problem also confused us as the wrongly conversions on address would not lead to 100% package loss. We compare the packets we sent with the packets captured by Wireshark to find the inappropriate parts of the ICMP packets and send ICMP packets that can an be resolved by node3.

Project 4

The project 4 requires to implement a DNS router in our Aethernet. In this project, Node 1 send the DNS request to node 2, and node 2 work as the DNS server to resolve domain names. The tasks also requires the Aethernet in project 3 to be upgraded with TCP and UDP protocols. However developing a complete functional TCP or UDP protocol on our Aethernet may be time consuming, so we modified our existing network to meet the requirements on project 4. We added several control sequence on our network and tried some modern preamble implement which consists of also control sequence on the network. We also modified the state transition design in node 2. As we had not implement a complete TCP /UDP protocol, we only finished task 1 DNS Server in this project.

We use the net package in java to send DNS requests. In The project, when a DNS request packet is received by node2, it decodes the data and identify the data as a DNS request with certain domain name. Node 2 then use `inetAddress` function from net package to get the corresponding IP address

of the resolved domain and initiate an `InetAddress` object with the IP address. The node 2 then checks whether the IP address is reachable by `isReachable(ipaddress)` method. If the IP address is reachable, node 2 send the DNS packets back to node 1. Node 2 takes use of the DNS cache of the system, rather than caching DNS in a java data structure.

A challenge we encounter in this project is that the debugging with the status transition of node 2 is time-consuming and we put a lot of effort in this process. The challenges also include that the multithread framework based on JACK has some synchronization problem when sending big packets and thus the decoded data are not correctly resolved. The receiver may receive and decode only part of the data. As this project does not has a limit RTT, we uses the previous `javasound` framework to finish task 1.

Conclusion

This project is strongly correlated with specific hardware, and thus we debug with Aethernet while discovering several unexpected characteristics of the audio system of our devices. This takes up some difficulties among all 4 projects, for example, the output channel gradually tends to send a sign of large amplitude (1) and a sign of small amplitude (0) with the same amplitude when the index of the sign in the frame exceeds a certain value (e.g. a frame containing 60 signs). Also we ignores the technical information of the real networks leads to failure of some of the project tasks. The selection of basis modules and platform also influences our development as developing the network with `javasound` takes more effort for synchronization against the challenge about the floating of time when player and recorder send and receive data.

For the projects, we suggest that tasks testing the latency of physical layer be introduced to project 1, a more robust physical layer with higher bandwidth can be developed from the beginning.