Project Report

韩乐桐 hanlt@shanghaitech.edu.cn 杜昊宇 duhy@shanghaitech.edu.cn

Project 1. Acoustic Link

Project 1 requires us to build a reliable point-to-point link with microphone and speaker.

In short, the requirements are

1. Modulation and demodulation
2. Error correction
3. OFDM (Orthogonal Frequency Division Multiplexing)
4. Range Challenge
5. MIMO (Multiple-Input Multiple-Output)

Requirements 2. to 5. are optional tasks.

1.1 Modulation and demodulation

The first task involves both hardware and software, and require students have some basic knowledge of multi-threading.

Some microphones or speakers may not work as expected, however. I will take my own experiment as an example.

When I started doing this project, I have to devices available: One desktop computer with Windows 11 and one Apple laptop with macOS. Since my desktop do not have a microphone, I bought a UGREEN CM564 microphone (https://item.jd.com/100027630663.html).

The CM564 has a 3.5mm headphone jack, but it seems to have poor quality and is unable to accurately produce the specified waveform.

Thus, inspecting the waveform directly is **very** important and it is the most effect debugging tool throughout the project 1.

My method is as follow:

The receiver automatically dump received samples to a dedicated file, then use python notebook for 1. plotting the waveform (if the device is broken, then comparing the reference waveform with received one should reveal this issue), 2. write prototype algorithms in python for detecting preambles and demodulation, thus, there is no need to restart program many times to test algorithms, just do experiments on the recorded samples. This method can greatly boost debugging speed and after verifying the integrity of tools and effectiveness of the algorithm, converting it to C++/Rust/Java/... takes little work.

Currently, this kind of method is suggested in the Tip section (https://sist-cs120.github.io/project-doc/project1.html#task-2-2-points-generating-sound-waves-at-will). And I think this kind of method may worth a separate compulsory task for it, since I couldn't imagine someone finishing the whole Project 1 without creating some similar tools. Also, doing experiments on the recorded sound can serve as a great introduction to the Aethernet, helping some students get started on more challenging tasks below.

Tip

- You can record the sound into a file and then utilize tools like Matlab/NumPy to load, plot, and replay it. Debugging algorithms with offline files can be convenient as processes are reproducible.

1.2 Error correction

I really enjoyed error correction part! I used LT Code (Luby Transform Code) for error correction, its implementation is simple (I wrote my own implementation) and its theory is beautiful. However, it only works fine under situations with low BER (Bit Error Rate), otherwise most of our channel's capacity would be used for packet header and LT Code's overhead.

I have implemented the a variant of LT Code from [1], this variant reduces LT Code's overhead while maintaining it effectiveness. For fast and effective decoding, I have implemented "On the Fly Gaussian Elimination for LT Codes" [2].

1.3 OFDM

OFDM is relatively easy to implement. But choosing the right frequencies for specific devices requires some experiments.

1.4 Range Challenge

Range challenge, as its name suggests, it very challenging even for the 0.5m part. OFDM-based methods, such as Dhwarni [3]

, does not work for range > 20cm.

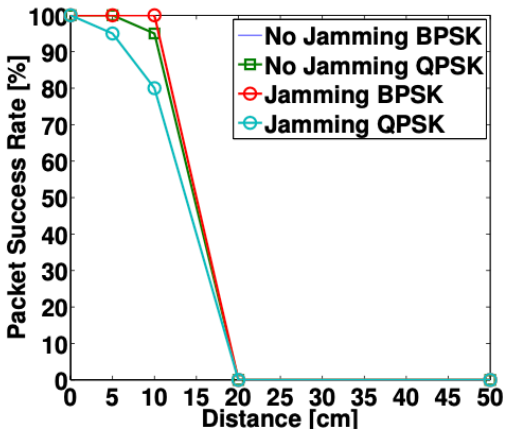

Figure 13: Communication Range of Dhwani

Actually, to get full points (>2m, 10000 bits in 15s without error), the only method I know is HRCSS (High-speed, long-range, and Robust Chirp Spread Spectrum) [4].

TABLE 1: Performance comparison of existing proposals.

Proposals	Modulation techniques	Frequency range	Audibility	Operational range	Max. data rate
[13]	Tone modulation	735-4410 Hz/18.4 kHz	Audible/inaudible	< 2 m	5.6/1.4 kbps
Digital voice [14]	M-arry FSK	0-12 kHz	Audible	< 2 m	2.4 kbps
Dhwani [8]	OFDM	0-24 kHz	Audible	< 1 m	2.4 kbps
[12]	FHSS	4.1-21 kHz	Audible	< 21 m	20 bps
Dolphin [18]	OFDM	6.4-8 kHz	Inaudible	< 2 m	240 bps
[19]	Complex Lapped Transform	6.4-8 kHz	Inaudible	< 3 m	600 bps
[6]	OFDM	8-20 kHz	Inaudible	< 1 m	500 bps
[9], [20], [21]	CSS-BOK	> 18 kHz	Inaudible	< 25 m	16 bps
[7]	CSS-QOK	> 18 kHz	Inaudible	NA (> 2.7 m)	16 bps
HRCSS	**CSS-variants**	$\mathbf{> 18}$ **kHz**	**Inaudible**	$\mathbf{< 20}$ **m**	**1 kbps**

Implementing HRCSS is not hard. However, it takes me some time to find the HRCSS paper. I started from the Dhwani paper [3] and checked the papers that cited it. Finally, after some fruitless attempts, our performance met the requirements.

1.5 MIMO

MIMO (Multiple-Input Multiple-Output) seems hard to implement at first sight.

1. Preamble

   Only one **Tx** can send preamble sequence. Or we cannot find the start of signal correctly.

2. Training sequence

   Traning sequence is a fixed sequence that is known to both **Tx** and **Rx** , it gives **Rx1/2** the CSI (Channel State Information) between **Tx1/2** and **Rx1/2**. CSI is a complex number denoting how the channel "transforms" the signal sent by **Tx**. With the CSI matrix, we can "reconstruct" the original signals with methods like zero forcing or maximum likelihood.

3. Zero Forcing v.s. Maximum Likelihood

   https://zhuanlan.zhihu.com/p/617999774 is a great explanation. In our experiments, ML gives much better performance than ZF.

We implemented MIMO with Sony D100 as **Rx1/2**, and Lenovo A200 speakers (https://item.jd.com/100054847566.html) as **Tx1/2**.

Project 2. Manage Multiple Access

Project 2 has two main requirements: 1. Implement CSMA correctly; 2. Speed.

2.1 CSMA

CSMA's parameters are also very hardware-specific, as collision detection requires us to be able to differentiate busy state and collision state. Again, we need to record the sound into a file and analyze how collision state is different from busy state.

2.2 Speed

Since there is a performance rank optional task, we applied various optimizations on our acoustic link.

1. From OFDM to 4B/5B code.

According to our experiments, OFDM offers a speed of about 12000 bits/s (5 sub carriers, one symbol is 16 samples, the cyclic prefix is 4 samples). To further boost performance, we switched to 4B/5B code, which offers a speed of 20000 bits/s.

2. Sliding window.

   The project document suggests the Stop and Wait method. Although this methods is easier to implement and is closer to real-life implementations, it pays one RTT per packet, which can be ~20ms. To make things worse, due to the frequency shift of DAC/ADC, the packet cannot to be too large. To mitigate this effect, we implemented sliding windows sender and receiver, and added a ACK field to packets. By setting windows size properly, this method eliminates nearly all the waitings compared with the stop-and-wait scheme in the setting of Task 4.

3. Fast-retransmit

   Since we are directly connect on a link, we can start retransmitting immediately upon receiving duplicate ACKs.

Project 3 To the Internet

Project 3 involves crafting and parsing packets and building a router.

3.1 Crafting & Parsing Packets

We used **pcapplusplus** (https://pcapplusplus.github.io), a cross-platform C++ wrapper of libpcap, WinPcap, NPcap for capturing and sending packets.

3.2 Router

With the help of pcapplusplus, implementing router is not hard. We only need to forward packets according to their destination address and modify packet contents according to the rules

of NAT.

3.3 Virtual Network Device

TAP-Windows provides easy to use virtual network devices out-of-the-box.

Project 4

Project 4 involves UDP/TCP protocols. However, we only implements the DNS task.

4.1 DNS

Since DNS uses UDP protocol, we only need to modify NAT in project 3 to support DNS requests/replies. This part is relatively simple.

---

1. C. Studholme and I. Blake, "Windowed Erasure Codes," 2006 IEEE International Symposium on Information Theory, Seattle, WA, USA, 2006, pp. 509-513, doi: 10.1109/ISIT.2006.261768. ↩

2. V. Bioglio, M. Grangetto, R. Gaeta and M. Sereno, "On the fly gaussian elimination for LT codes," in IEEE Communications Letters, vol. 13, no. 12, pp. 953-955, December 2009, doi: 10.1109/LCOMM.2009.12.091824. ↩

3. Rajalakshmi Nandakumar, Krishna Kant Chintalapudi, Venkat Padmanabhan, and Ramarathnam Venkatesan. 2013. Dhwani: secure peer-to-peer

acoustic NFC. In Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM (SIGCOMM '13). Association for Computing Machinery, New York, NY, USA, 63–74. https://doi.org/10.1145/2486001.2486037 ↩ ↩

4. C. Cai, Z. Chen, J. Luo, H. Pu, M. Hu and R. Zheng, "Boosting Chirp Signal Based Aerial Acoustic Communication Under Dynamic Channel Conditions," in IEEE Transactions on Mobile Computing, vol. 21, no. 9, pp. 3110-3121, 1 Sept. 2022, doi: 10.1109/TMC.2021.3051665. ↩