CS120 Project Report

Group 1

Tianyao YAN
yanty@shanghaitech.edu.cn

Wenfei YU
yuwf@shanghaitech.edu.cn

I. Introduction

For the CS120 projects, we developed Athernet, a network that utilizes acoustic signals or sound waves for transmitting information across audio connections. The implementation of Athernet was executed using C++ with the assistance of JUCE, ASIO, and Npcap.

II. Project 0. Warm up

A. Overview

- Task 1: (0/0 point) Self-assessment
- Task 2: (3/3 points) Understanding Your Tools

B. Stories

For the first encounter with JUCE, during the National Day holiday, we spent almost the entire break studying the use of JUCE. This also set the tone for the learning pressure of this course.

C. Designs

Refer to the following text.

D. Challenges

The documentation for JUCE is extensive and detailed, making it challenging to filter out the useful content.

E. Solutions

Sacrificing rest time, we devoted a significant amount of time to studying the use of JUCE.

III. Project 1. Acoustic Link

A. Overview

- Task 1: (3/3 points) Understanding Your Tools
- Task 2: (2/2 points) Generating Sound Waves at Will
- Task 3: (4/5 points) Transmitting Your First Bit
- *Task 4: (0/1 point) Error Correction
- *Task 5: (0/2 points) Higher Bandwidth
- *Task 6: (0/2 points) Chip Dream
- *Task 7: (0/1 point) MIMO
- *Task 8: (0/4 points) Range Challenge

B. Stories

We spent a massive amount of time on parameter tuning. In Task 3, we were able to achieve accuracy of over 90%, which would have earned a perfect score in previous years. However, this year, the standards were raised, and the project document had unclear descriptions, resulting in us receiving a score of only 80% accuracy.

C. Designs

- Task 1:
  Read the input audio from inbuffer and write the predefined audio to outbuffer.
- Task 2:
  As $y = Asin(\omega x + \varphi)$, we just set ω to the required frequency and added the two functions together.
- Task 3:
  As each bit is inherently either a 0 or 1, we employed two distinct waves to represent these values. The carrier wave signifies 1 through a sine wave, while another wave indicating 0 called zero wave is represented by a sine wave with a specific phase difference from the carrier. Consequently, the receiver can discern the bits based on the sign of the inner product between the received wave and the carrier wave.
  We employed a chirp as a preamble wave to enable the receiver to discern the initiation of packet transmission. It is generated by

$$\text{preamble}[i] = cos(\sum_{n=0}^{i} \frac{2\pi}{48000}(5000 + \frac{i(10000-5000)}{520-1})$$

  where The sample rate is set at 48000Hz, and the chirp frequency ranges from 5000Hz to 10000Hz, with the preamble length defined at 520 bits.
  We utilized the inner product of received samples and the preamble wave to identify the preamble wave. This approach allowed the receiver to pinpoint the starting point of actual bit transmission.

D. Challenges

Due to poor audio reception on the laptop, we choose to plug in headphones into the notebook. Air is an unstable transport medium. Placing two headphones together, one for capturing sound and the other for emitting sound, however, due to various unstable factors, the phase of the received audio is consistently unstable. Achieving complete accuracy is challenging, and in most cases, there is a significant occurrence of complete phase reversal.

E. Solutions

Unfortunately, we haven't found a suitable solution. In cases of extensive phase reversal. CRC and ECC are ineffective. In the end, we were unable to achieve 100% accuracy in Task 3.

IV. Project 2. Manage Multiple Access

A. Overview

- Task 0: (0/0 point) Audio Toolkit
- Task 1: (4/4 points) Cable Connection
- Task 2: (0/5 points) Acknowledgement
- Task 3: (0/2 points) Carrier Sense Multiple Access
- Task 4: (0/1 point) CSMA with Interference
- *Task 5: (0/3 points) Performance Rank
- *Task 6: (0/3 points) X

B. Stories

We faced some mysterious issues, and it wasn't until the final day before the deadline that we successfully resolved the problem through a series of iterations involving plugging and unplugging, changing sound cards, and modifying the method of reading input files. As a consequence, we were only able to complete Task 1. Despite this setback, we made an effort to complete Task 2 post-deadline in order to retain the 5% finish score. However, we were informed that only obtaining scores would enable us to maintain the scores.

C. Designs

- Task 1:
  With the transition to a wired connection offering enhanced stability, we reduced the number of audio samples required to transmit one bit. Additionally, we introduced packet division to enhance the overall stability of the transmission.
- Task 2:
  Upon receiving a packet, the receiver sends an acknowledgment in the form of a preamble chirp wave. This mechanism allows the sender to confirm successful reception by the receiver. Consequently, the sender can monitor the network's functionality, and in the absence of any acknowledgment within a defined timeframe, it can terminate the sending process, signaling a potential disconnection.

D. Challenges

If there is a cable malfunction or issues with the program, the accuracy of received bytes is approximately 50%, similar to random guessing.

E. Solutions

We spent a significant amount of time pinpointing issues and debugging.

V. Project 3. To the Internet

A. Overview

- Task 0: (0/0 point) Sending and Receiving IP Datagram
- Task 1: (3/3 points) ICMP Echo
- Task 2: (4/4 points) Router
- Task 3: (3/3 points) NAT
- *Task 4: (1/1 point) NAT Traversal
- *Task 5: (1/1 point) IP Fragmentation
- *Task 6: (0/0 point) Virtual Network Device
- *Task 7: (1/1 point) ICMP Echo #
- *Task 8: (1/1 point) Router #
- *Task 9: (0/1 point) NAT #
- *Task 10: (1/1 point) ARP
- *Task 11: (0/* points) Star

B. Stories

While at Berkeley, with a 16-hour time difference, we checked some of the optional tasks. Upon realizing that it was already midnight, Professor Yang Zhice advanced the check time of Task 10. However, it is precisely for this reason that we were unable to complete Task 11 as it requires collaboration between multiple groups.

C. Designs

- Task 1:
  For the sake of simplicity, we opted not to transmit MAC addresses in Aethernet. Instead, an Aethernet ICMP packet comprises an IP header, an ICMP header, and ICMP data. Node 1 is assigned the IP address 172.18.1.2, while Node 2 is assigned 172.18.1.1. We generated Aethernet ICMP echo and echo reply packets in accordance with RFC 792. To record and calculate RTT and generate a "random" identification, a high-resolution clock in chrono was implemented.
- Task 2:
  In the Aethernet program of Node 2, the hotspot interface was activated and utilized for sending and receiving ICMP packets from Node 3, which is expected to be connected to the hotspot of Node 2.
- Task 3:
  The identification field of the ICMP packet was employed to capture the source port information. Upon receiving an ICMP echo packet in Aethernet from Node 1, Node 2 replaced the source IP of Aethernet Node 1 with its WLAN IP. Subsequently, it transmitted an ICMP echo packet to the public network server at 114.114.114.114. Finally, after identifying the port in the identification field of the ICMP echo reply message from the public server, Node 2 sent an ICMP echo reply to Node 1 in Aethernet.
- Task 4:
  We stored the IP address of Node 1, which Node 4 ultimately needs to ping, in the data of the ICMP packet using group-specified options. Therefore, Node 2 can retrieve the IP address it needs to ping Node 1 by reading the data and transmit ICMP echo and echo reply in Aethernet.
- Task 5:
  To enhance transmission accuracy in Aethernet, we established an MTU of 44 bytes, comprising 20 bytes for the IP header and 24 bytes for data. Adhering to RFC 791 guidelines, we accurately configured the flags and fragment offset in the IP header. Subsequently, we successfully fragmented and replied the long ICMP echo packet originating from Node 4.

- Task 7 and Task 8:
  We configured a virtual network adapter: Microsoft Loopback Adapter. We bound the IP and MAC addresses of Node 1 and Node 2 using

  ```
  netsh -c "i i" add neighbors
  ```

  Building upon Task 1 and Task 2, we obtained ICMP echo and echo reply packets through the interface of this virtual network adapter.
- Task 10:
  We created complete ARP and reply packets based on RFC 826 when Node 1 pings Node 2.

D. Challenges

- We were unable to customize the Ethernet and IP headers of socket in C++. Meanwhile, the documentation for Npcap is detailed and complex, making it challenging to find the needed information.
- The 'arp -s' command often associates an IP address with a randomly selected incorrect network interface.
- After Node 1 disconnected from WLAN, we couldn't locate the ICMP packet corresponding to the ping command in any network interface, making it impossible to proceed with further operations.

E. Solutions

- We spent a significant amount of time, dedicating oneself to studying the Npcap documentation, often neglecting sleep and meals.
- We used "netsh" command mentioned instead.
- Unfortunately, as of now, we still have not found a solution.

VI. Project 4. Above IP

A. Overview

- Task 1: (4/4 points) DNS
- Task 2: (1/1 point) HTTP
- Task 3: (?/3 points) Project Report
- Task 4: (0/0 point) Relaunch
- *Task 5: (0/1 point) Browsing the Web
- *Task 6: (1/1 point) Project Report #

B. Stories

While working on this project, the Spring semester of 2024 had already begun at the University of California, Berkeley. We stayed up very late, attempting to complete optional Task 5. However, due to the vague task requirements in the document, we used a completely incorrect and non-scoring method to finish Task 5, staying awake until 4 in the morning local time in vain.

C. Designs

- Task 1:
  DNS operates on the UDP protocol, and to implement this, we included a class UDPHeader strictly following the actual UDP header structure outlined in RFC 768. In this particular task, Node 2 consistently forwards requests from Node 1 to the Internet and relays replies from the Internet back to Node1, handling both DNS and ICMP traffic.
- Task 2:
  In this task, we implemented HTTP1.1 by extracting packet samples directly from Wireshark. Given that HTTP operates on the TCP protocol, we introduced a class TCPheader according to RFC 793. The task involves multiple states for the two nodes. Initially, DNS functionality is maintained, followed by Node 2 establishing a connection with the specified domain through a handshake process. After successful handshaking, Node 2 initiates an HTTP GET request to the domain and receives the corresponding reply. Given that website content often exceeds the upper limit of a single IP packet, Node 2 must patiently wait for the actual conclusion that usually marked by the packet with the PSH flag set to 1 of the HTML document. Subsequently, Node 2 consolidates all data fragments before forwarding the complete HTML document to Node 1 over Aethernet.

D. Challenges

The retrieved page contains a substantial number of bytes, and attempting to transmit all the bytes from Node 2 to Node 1 in a single Ethernet packet might result in inaccuracies, especially towards the end of the transmission.

E. Solutions

Node 2 must transmit the bytes to Node 1 in multiple packets through Aethernet.

VII. Suggestions

1) The task requirements in the document should be refined to prevent students from being led down an incorrect path in completing the task. We suffered greatly from it in Project 1 Task 3 and Project 4 Task 5.
2) We hope that the course can reduce its reliance on Aethernet as it has high hardware requirements. This indirectly encourages students who wish to achieve good grades to spend money on expensive high-performance equipment and may occur peculiar and unexpected bugs.