Athernet: Acoustic-based User Space Network Stack

Cheng Peng
ShanghaiTech University
`pengcheng2@shanghaitech.edu.cn`

Huizhe Su
ShanghaiTech University
`suhzh@shanghaitech.edu.cn`

Abstract

Athernet is a user-space TCP/IP stack with full functionality and handy utilities built on the acoustic channel. This report describes the main architecture design and the technical implementation of the Athernet.

Keywords: network stack, acoustic link, network protocol

1 Introduction

This work is submitted in partial fulfilment of the course CS120: Computer network at ShanghaiTech University. The network stack is an essential topic in computer networks and has various implementations. To comprehensively understand it, we are asked to implement our network stack Athernet.

Unlike traditional wireless or wired methods, Athernet uses sound waves to transmit data between devices. The stack has implemented the basic functionalities of the standard TCP/IP and can be used with existing network infrastructure. Additionally, the system has the potential to be used in areas where wireless communication is not feasible, such as underwater communications.

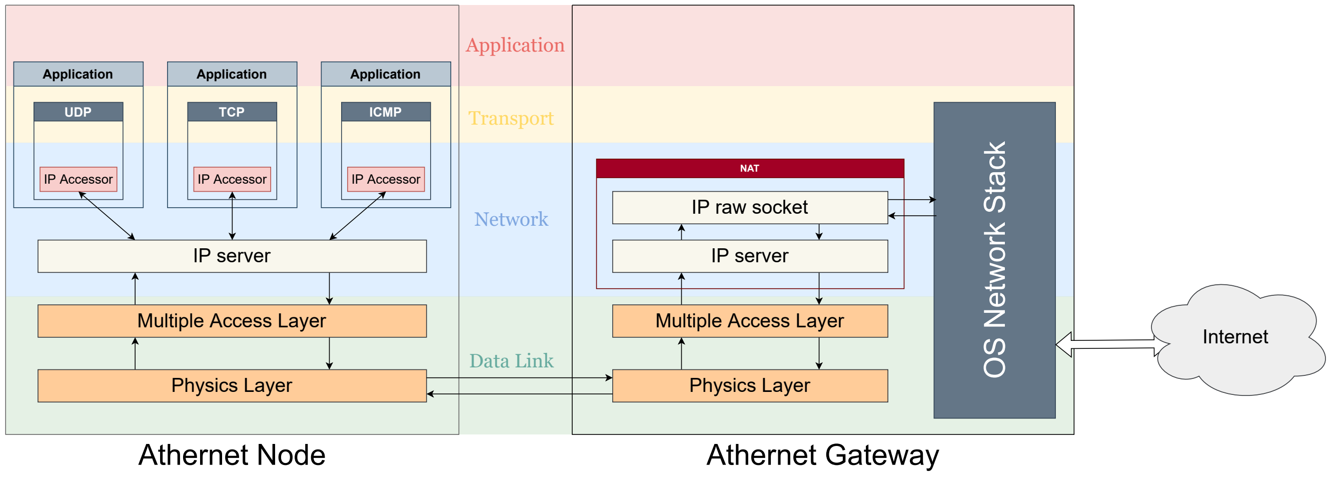

Figure 1: the overview architecture of the Athernet network stack

The network stack is composed of the following four layers where each layer is built on the previous one.

- **The link layer:** The link layer provides reliable bidirectional data transmission in local area. It is decomposed into two sublayers namely the *physics sublayer* and the *media access control sublayer*. The physical layer is responsible for the converting digital data into audio signal or the reverse process and sending/receiving data through sound waves. It supports both noisy wireless medium and audio cable media. Two different modulation schemes are implemented for the two scenarios: a passband modulation scheme that combines *BPSK* and *OFDM* for the wireless scenarios and a baseband modulation scheme using *4B5B* with *NRZI* for the cable connected scenarios. The media access layer deals with collisions that may happens when multiple nodes sends signal to the media simultaneously. *CSMA/CA* is implemented to provide reliable data transmission.

- **The network layer:** *Internet Protocol(IP)* is implemented for inter-network data transmission. An daemon process called the *IP server* regularly sends packets to peers and receives packets from peers. To allow programs in the next layer to access the network layer service, *IP accessor* which communicates with the *IP server* using unix domain socket is created. The *network address translation(NAT)* is also implemented in this layer, which enables the communication between Athernet LAN and the Internet.

- **The transport layer:** Various transport layer protocols are implemented including *transmission control protocol(TCP)*, *user datagram protocol(UDP)* and *internet control message protocol(ICMP)*. Athernet socket, a set of APIs that resembles the socket API in rust standard library, are exposed for applications to use the transport layer services.
- **The application layer:** A simple *file transfer protocol(FTP)* client supporting a set of restricted control commands and only passive transmission mode is implemented on Athernet TCP to demonstrate the capability of Athernet. For Athernet to support the pre-existing applications which are built on OS TCP/IP stack, a SOCKS5 proxy server is implemented that redirects network traffics through Athernet.

Figure 1 shows the overall architecture of the Athernet. The Athernet nodes are interconnected and can connect to the Internet through the Athernet gateway, which runs a NAT.

The paper is devided into five parts. Through chapter two to five, we describes the design of the four layers of our network stack. In the last chapter, we analyze the problems in our design and make a technical summary.

2 Data Link Layer

The foundation of the entire network stack is the data link layer which provides reliable bidirectional packet transmission between two nodes in a shared noisy acoustic medium. The data link layer consists of the physics sublayer and the medium access control sublayer.

2.1 Physics Sublayer (PHY)

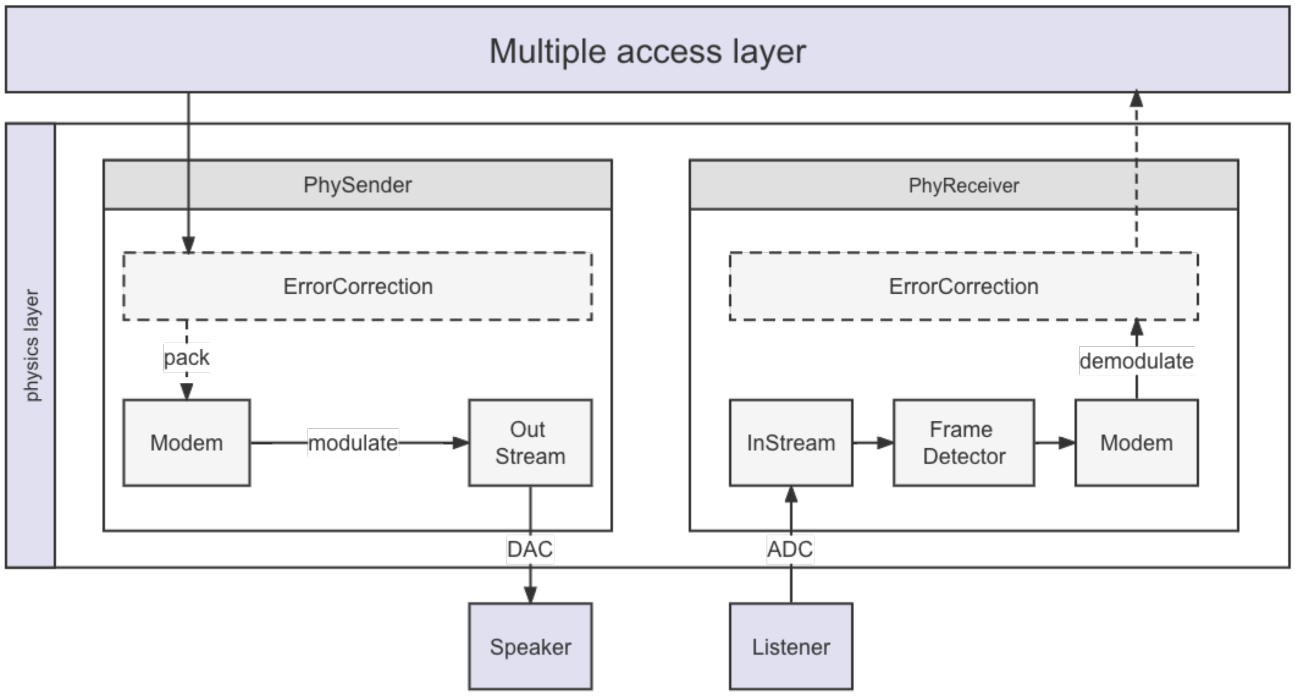

Figure 2: the physics layer

Bulit on the audio I/O library, PHY enables basic data transmission with no delivery or integrity guarantee. A PHY frame is a sequence of PCM audio signal samples that represents a chunk of bytes. It is the transmission unit on this layer. Each frame contains a fixed preamble signal at the beginning for detection as well as synchronization and a payload signal that encodes 0/1 bits.

For data transmission, PHY layer constructs a frame from a chunk of bytes and send it through the medium. For data receiving, PHY layer repeatedly pulls audio samples from the medium and find the beginning of PHY frames. When a frame is identified, the payload signal is extracted and decoded.

2.1.1 Preamble Signal Design

A chirp is a signal whose instaneous frequency increases or decreases with time. $x_p(t)$ is a linear chirp signal whose lowest and highest instaneous frequency are f_a and f_b respectively.

$$x_p(t) = \begin{cases} \pi \frac{f_b - f_a}{T} t^2 + 2\pi f_a t & t \in [0, T] \\ \pi \frac{f_a - f_a}{T} (t - T)^2 + 2\pi f_b (t - T) & t \in [T, 2T] \end{cases}$$

Chirp signal has a distinct frequency-domain feature so it won't interfere background nosie and the carrier wave. Auto-correlation energy peak detection is used to find the starting position of the preamble signal which is followed by the payload signal.

2.1.2 Modulation and Demodulation

In order to transmit and receive digital signals, a modulation scheme has to be implemented. Air and audio cabel are media of different characteristic, so they need different modulation schemes.

For wireless scenario, background noisy is strong from 0Hz to 6000Hz, so a passband modulation scheme combining binary phase shift keying (BPSK) and orthogonal frequency-division multiplexing (OFDM) is used. BPSK encodes bits with phase changes:

$$\begin{aligned} x_0(t) &= \sin(2\pi f t) \\ x_1(t) &= \sin(2\pi f t + \pi) = -\sin(2\pi f) \end{aligned}$$

are used to represent 0 bit and one 1 respectively. To recover a bit from audio signal, consider the dot products $\langle x_0(t), x_0(t) \rangle$ and $\langle x_0(t), x_1(t) \rangle$:

$$\begin{aligned} \int_a^b (x_0(t) \cdot x_0(t))\, \mathrm{d}t &= \int_a^b x_0^2(t) \mathrm{d}t > 0 \\ \int_a^b (x_0(t) \cdot x_1(t))\, \mathrm{d}t &= -\int_a^b x_0^2(t) \mathrm{d}t < 0 \end{aligned}$$

A zero bit is produced if the dot of the received samples and samples of $x_0(t)$ is positive, otherwise a one bit is produced.

To increase the utilization rate of the spectrum, OFDM of $N = 64$ is introduced. However, unlike radio medium, transmitting real part and imaginary part of a complex number simultaneously is not possible, so only half of the frequency channels can be used to encode bits. Also, several channels of extremely low and high frequency are not usable as background noise is too strong.

As for cabel-connected scenario, baseband transmission is feasible as low frequency background noise no longer exists. 4B5B + non-return to zero inverted(NRZI) enables higher bandwidth and lower error rate.

2.2 Medium Access Control sublayer

The medium access control layer regulates access to the shared medium and implements error detection and retransmission mechanism. A MAC frame is placed in the payload field of a PHY frame so that it can be sent/received through PHY layer.

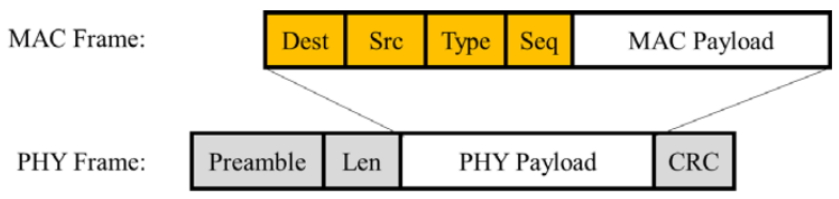

Figure 3: the MAC frame

2.2.1 CSMA/CA

We implemented *Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)*. Each device will listen to the channel before transmitting. If the channel is busy, the device will wait for a random time and then try to resend the data. The time is chosen by the exponential backoff algorithm. The state transition diagram is shown in Figure 4

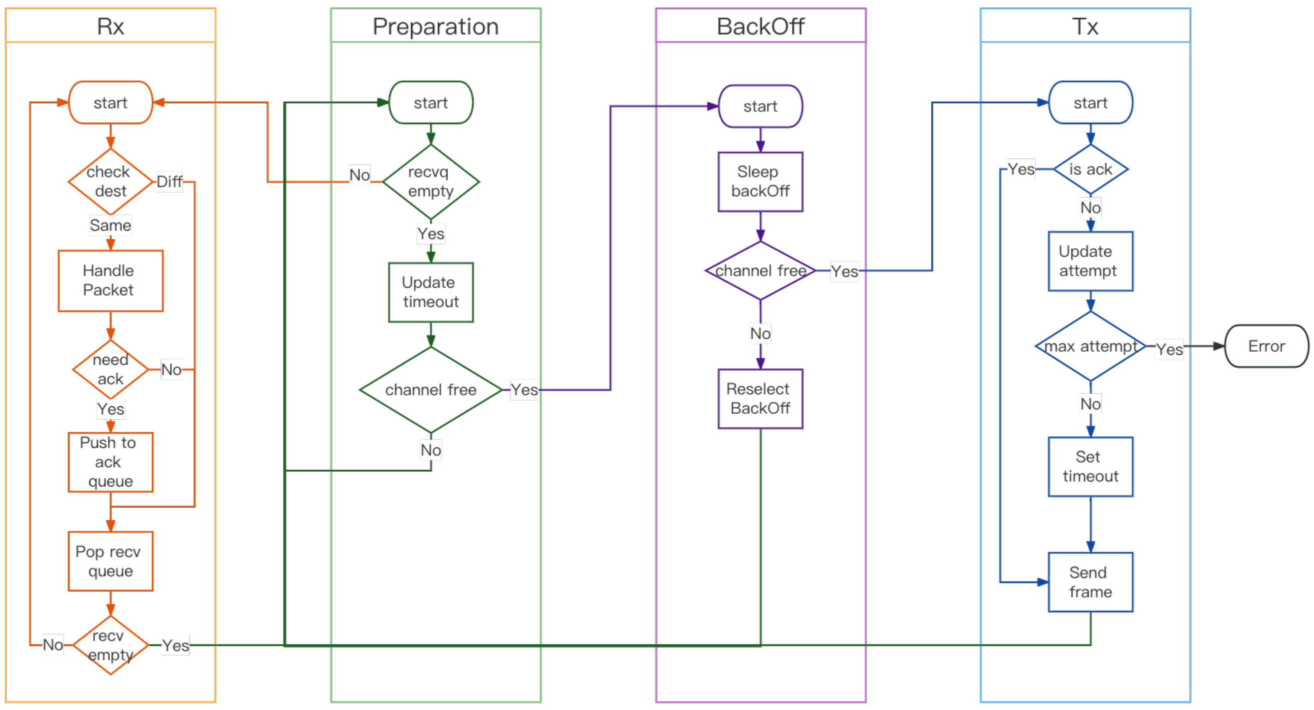

Figure 4: the CSMA/CA procdcure

2.2.2 Time-division multiplexing MAC

Due to the latency of the audio driver, the original CSMA/CA suffers from the remote collision. We also implemented Time-division multiplexing MAC., which allocates a fixed time slot for the device to send and receive the packets. We can use the pre-existing Network Time Protocol to synchronize the time of local nodes.

2.2.3 Sequence Number and Error Detection

The sequence number and CRC checksum fields are used to drop out-of-order packets and corrupted packets. A sliding window transmission control is implemented to provide reliable in-order data transmission.

3 Network Layer

3.1 Internet Protocol

On each Athernet node, a daemon process call the IP server constantly receives IP packets targetted to the node and forwards IP packets to peers. Processes running on Athernet nodes uses IP accessor which communicates with IP server through datagram oriented unix domain socket to access the network layer services.

IP server maintains a send queue of IP packets and a reassemble queue of MAC frames. It periodically takes one packet out of the queue, split it pieces, wrap them in MAC frames and send them to peers via MAC layer service. When a MAC frame is received from peer, IP server append it to the reassemble queue and try to combine the received pieces into an complete IP packet.

IP server maintains a set of bind rules where a rule is a 3-tuple of a transport layer protocol, a port number and a unix domain socket path. When an IP packet is received, IP server run the following procedure:

1. Inspect the IP destination field. If the IP destination address is not equal to the IP address of this node, push it to the send queue. Otherwise, it is a packet targeting the current Athernet node.

2. For a packet targeting at current ndoe, extract the next level protocol field and parse the next-level destination port number.

3. Try to find a rule that matches the protocol and port number. If not such rule exists, discard the IP packet. Otherwise, extract the socket path field and send the entire IP packet to the IP accessor which is bind to the socket path.

An IP accessor can add or remove a bind rule for the IP server by sending bind or unbind request messages through unix domain socket. A process can uses IP accessor to send a send packet request messages through unix domain socket to command the IP server to push a packet to the send queue. A process will receive IP packets from the IP server that matches the bind rule.

3.2 Network address translation

A gateway node that is connected to the Internet uses NAT to link Athernet with the Internet. It uses IP raw socket to capture all incoming IP packets from the OS TCP/IP stack.

NAT translate a 3-tuple of Athernet IP address, transport protocol and port number to another port number. If Athernet internal node sends an IP packet targeting at an Internet host to the gateway, the gateway node changes the IP source address to gateway IP address and the next level port number to NAT translated port number. After recalculating the checksum, the gateway node forwards the packet out with IP raw socket. When the gateway receives an IP packet, it goes through the reverse direction and forwards the translated IP packet to Athernet peers.

By doing so, Athernet internal nodes can communicate with any Internet host.

4 Transport Layer

4.1 Internet Control Message Protocol and User Datagram Protocol

Both ICMP and UDP are simple and stateless; therefore, they are implemented in the same manner. For the sending side, pack the bytes into the corresponding ICMP/UDP packet, and send them through an IP accessor. They will then be packed into the IP packet and forwarded to the network layer. For the receiving side, they wait until the next packet is sent from the IP accessor, unpack them according to the protocol and read out the bytes.

4.2 Transmission Control Protocol

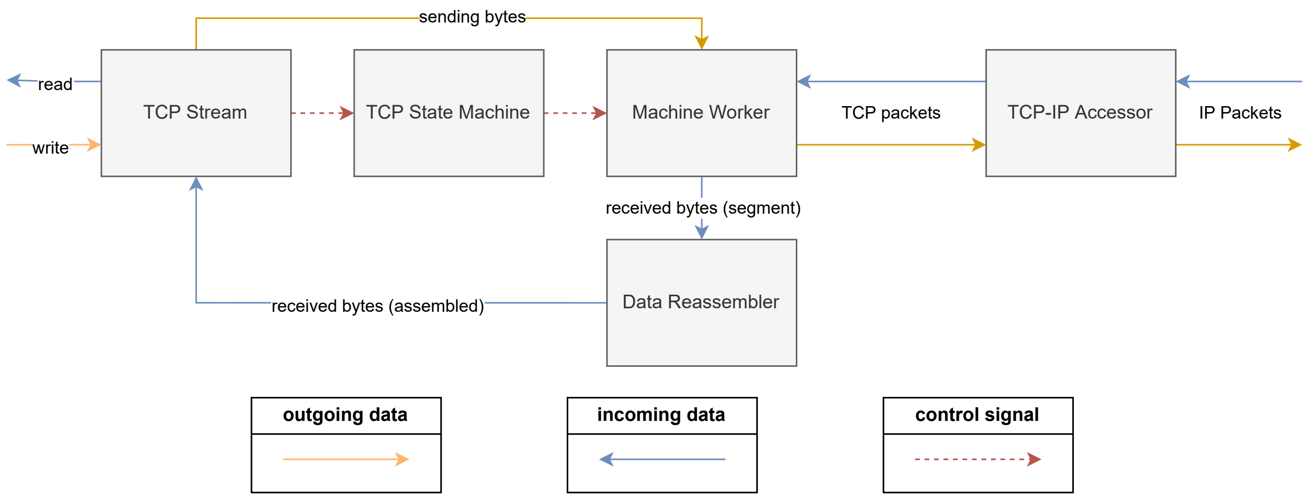

Figure 5: the structure of TCP Stream

TCP has been divided into two parts: *TCP Stream* and *TCP Listener*. TCP Stream represents a single TCP connection between the local and remote socket. After creating a TCP stream by either connecting to a remote host or accepting a connection on a TCP Listener, data can be transmitted by reading and writing to it. A TCP Listener is a TCP server listening on a specified port for incoming connections. After accepting an incoming connection, it will produce a TCP Stream.

4.2.1 TCP Stream

As depicted in Figure 5, TCP Stream is composed of five parts.

- **TCP Stream:** The host interacts directly with the TCP Stream. TCP Stream will send the data to the Machine Worker and receive data from Data Reassembler.
- **TCP State Machine** will handle the control signal (such as connecting to the remote or closing the connection) from TCP Stream and send them to the Machine Worker. TCP State Machine is also responsible for releasing the resources after the host drops TCP Stream.
- **Machine Worker** handles the state transition of the TCP. It will also pack the data from the TCP stream, send them to the TCP-IP Accessor, unpack the incoming TCP packets, and send the unpacked bytes to Data Reassmebler.
- **TCP-IP Accessor** is the bridge between the network layer and the transport layer. It packs the TCP packets into IP Packets, periodically receives IP packets, and sends the unpacked TCP packets to the Machine Worker.

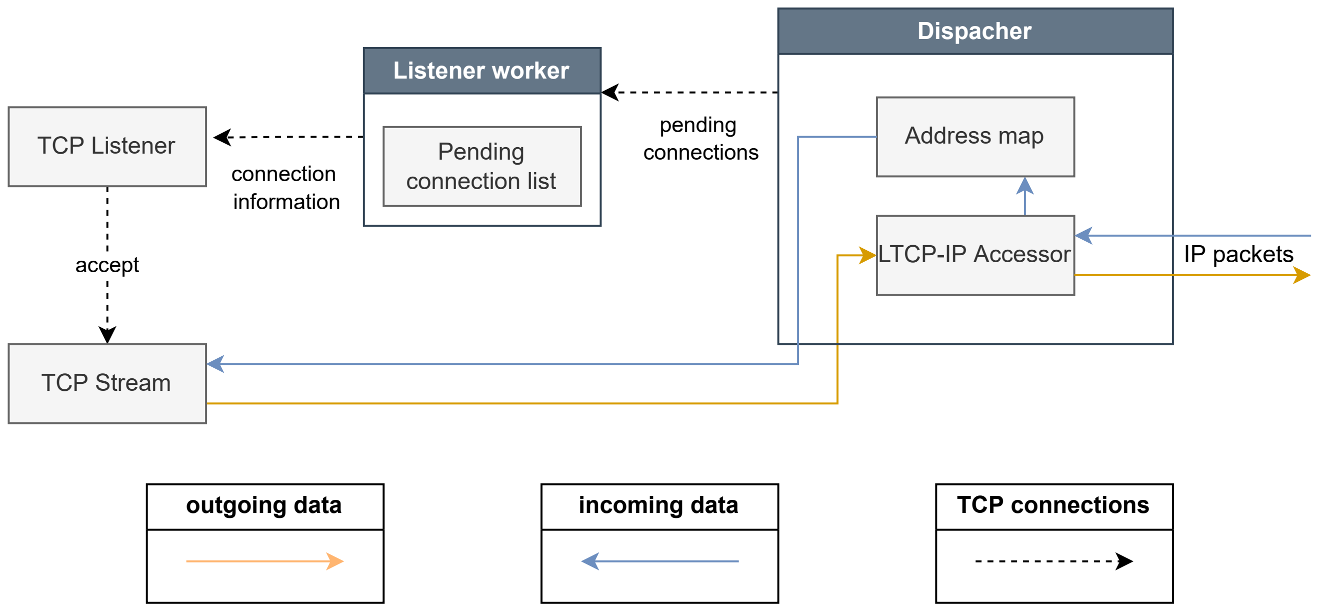

Figure 6: the structure of TCP Listener

- **Data Reassembler** receives the byte segment from the Machine Worker and reassembles them according to the TCP sequence number. The reassembled bytes will then be sent to the TCP Stream for the host to read.

4.2.2 TCP Listener

As shown in Figure 6, TCP Listener is composed of three parts:

- **TCP Listener** responds to the host's request. It will ask the Listener Worker for information on the incoming connection if it tries to accept a connection. The created TCP Stream will change its TCP-IP accessor to the Dispatcher.
- **Listener Woker** receives the connection from the Dispatcher and temporarily stores it for the TCP Listener.
- **Dispacher** owns an LTCP-IP Accessor to communicate with the IP server. On receiving the IP packet, it will dispatch the inner TCP packet according to its Address map. If the TCP is an SYN packet, it will create a new entry in the Address Map and send the information to the Listener Worker. On receiving TCP packets from the TCP Stream, it will pack them into IP packets and forward them to the IP server.

The TCP Listener is capable of handling multiple connections at the same time.

5 Application Layer

The network stack implemented previously enabled us to write some Internet applications for the Athernet. We implemented a simple FTP client to retrieve files from the remote servers according to RFC959 and a proxy allowing other network applications to be used inside the Athernet.

6 Conclusion

We implemented Athernet using rust programing language, making it easy to be moved to other platforms and has significant efficiency. The Athernet has been tested on GNU/Linux and macOS, and it is effortless to deploy on both systems. Overall, our work powerfully demonstrates the feasibility of acoustic networks. Athernet is a promising alternative data communication method that could have a wide range of applications. Further research could be carried out to improve the system's performance and to explore other potential applications.

Acknowledgement

We would like to express our sincere gratitude to our course teacher Zhice Yang, for providing the knowledge and guidance to complete this project. We would also like to thank our colleagues, Wenchao Li and Jiajun Cheng. Their valuable feedback, suggestions, and support have been instrumental in completing this project.

We would also like to thank the Rust community for providing us with the resources, libraries and platform to develop this project. In completing this project, we used the following libraries (listed in alphabetical order):

- az: Casts and checked casts.
- bitvec: Addresses memory by bits, for packed collections and bitfields
- clap: A simple to use, efficient, and full-featured Command Line Argument Parser.
- crc: Rust implementation of CRC(16,32,64) with support for various standards.
- crossbeam: Tools for concurrent programming.
- crossbeam-channel: Multi-producer multi-consumer channels for message passing.
- cordic: Special functions for fixed-point numbers using the CORDIC method.
- console: A terminal and console abstraction for Rust.
- cpal: A low-level cross-platform audio I/O library.
- fixed: Fixed-point numbers.
- hound: A wave encoding and decoding library.
- log: A lightweight logging facade for Rust.
- env_logger: A logging implementation for log which is configured via an environment variable.
- parking_lot: More compact and efficient implementations of the standard synchronization primitives
- pnet: Cross-platform, low level networking using the Rust programming language.
- postcard: A no_std + serde compatible message library for Rust.
- rand: Random number generators and other randomness functionality.
- reed-solomon-erasure: Rust implementation of Reed-Solomon erasure coding.
- rustfft: High-performance FFT library written in pure Rust.
- serde: A generic serialization/deserialization framework
- socket2:Utilities for handling networking sockets with a maximal amount of configuration possible intended.

We have great respect for their developers and maintainers.