ATNet: A computer network based on acoustic channel

Zhe Ye
ShanghaiTech University
yezhe@shanghaitech.edu.cn

Shuyue Wang
ShanghaiTech University
wangshy3@shanghaitech.edu.cn

1 INTRODUCTION

In this paper, we will give a detailed discussion about what and how we have achieved in our 4 course projects of CS120 Computer Network.

During 4 projects, we aimed to build a computer network based on acoustic channel from bottom-up. In project 1 and 2, we used Python to build the entire system, called pATNet. However, the latency of the underlying acoustic layer (*python-sounddevice*) in Python is unbearable. We have proposed several workarounds, but the results were not satisfying. Therefore, we switched to Java in project 3 and 4. The Java version is called jATNet. It was proven to provide a reliable link with low latencies. In the latter discussion, ATNet mainly refers to jATNet.

The core infrastructure of ATNet can be divided into 3 layers: Physical Layer, MAC Layer and Network Layer. Each layer runs a specific protocol and exchanges certain format of data. We also implemented some services and applications built upon these layers, e.g. Ping utilities of different layers, NAT services and FTP applications.

2 PROTOCOLS AND DATA FORMATS

The figure 1 describes the overall data format.

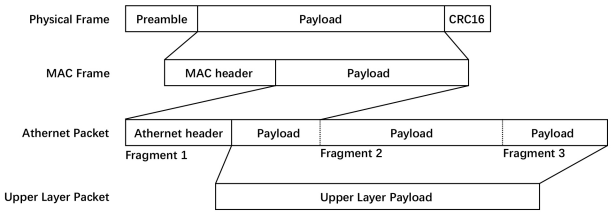

Figure 1: Data Format Explained

2.1 Physical Layer

In the physical layer, ATNet runs Athernet Physical Protocol (APP) and data are exchanged with **Physical Frame**.

2.1.1 *Physical Frame Format.*

The physical frame consists of 3 part: preamble, payload and CRC16.

The preamble is a predefined acoustic wave used for positioning the start of the physical frame.

The payload carries the data to transmit, and CRC16 is calculated on the payload and is used to detect data corruption during transmission. The payload and CRC16 are modulated into acoustic waves before transmission, and demodulated by receiver (see 3.1 for details).

The payload is implemented as fixed length. The sender and receiver should be configured with the same length of payload before communications. This may cause problems like a waste of resource when we are sending data much smaller than the fixed payload length, e.g. ACKs in MAC layer. We have also implemented prototypes of variable length payload support. One prototype is long-frame short-frame design, the first bit comes directly after the preamble indicates whether the incoming physical frame is a long frame or a short one. Another prototype is that the first byte comes directly after the preamble indicates the length of the incoming frame. But in consideration of simplicity and performance of the system, we decided to use fixed length frame. In practice, the payload length is typically small, e.g. 64 bytes.

In the optional part, we also includes Reed-Solomon codes in the physical frame to correct corrupted frame.

2.1.2 *Athernet Physical Protocol.*

The protocol is straight-forward. We run CRC checks on the received frame. If the frame is corrupted, we simply drop it; otherwise we extract out the payload and hand it over to the MAC layer.

2.2 MAC Layer

In the MAC layer, ATNet runs Athernet MAC Protocol (AMAC) and data are exchanged with **MAC Frame**.

2.2.1 *MAC Frame Format.*

The MAC frame consists of a MAC frame header and payload.

The MAC frame header occupies 2 bytes. The first byte consists of a 4 bits destination MAC address and a 4 bits source MAC address. The second byte consists of a 4 bits type identifer and a 4 bits flag field.

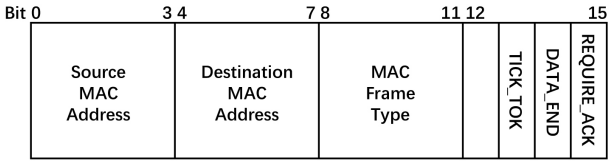

Figure 2: MAC Frame Header Format

The 4 bits type identifer is a enumerated type:

- DATA(0x0), indicates that the frame payload includes data;
- ACK(0x1), indicates that the frame is the ACK of the last received DATA frame.

The 4 bits flag field consists of:

- REQUIRE_ACK(0x1), indicates that the DATA frame requires an ACK;
- DATA_END(0x2), indicates that the DATA frame is the end of the fragmentation (controlled by upper layer);
- TICK_TOK(0x4), used to tell difference between a retransmitted frame and a new frame.

2.2.2 *Athernet MAC Protocol.*

We used CSMA as our Athernet MAC Protocol. When transmitter wants to transmit data, it must sense the acoustic channel. If the channel is busy now, it would defer its transmission. If the receiver received a frame with REQUIRE_ACK flag, it should send an ACK frame. There would be a configured timeout, if the DATA frame or ACK frame is lost, the transmitter would retransmit or abort. When transmitter wants to transmit a new DATA frame, it will flip the TICK_TOK field; when retransmitting, it will not do so. Therefore, the receiver can distinguish between a new DATA frame and a retransmitted frame. Sine the protocol is Stop-and-Wait, the TICK_TOK flag only needs 1 bit.

To avoid overheads, we decided to do fragmentation in both MAC layer and network layer. By limitations of physical frame size, a Athernet Packet would be fragmented into several MAC frames. Unlike the Internet Protocol, the Athernet packet header would only appear in the first of these MAC frames. To indicate which frame is the end of the fragmentation, DATA_END flag would be set.

2.3 Network Layer

In the network layer, ATNet runs Athernet Protocol (AP) and data are exchanged with **Athernet Packet**.

2.3.1 *Athernet Packet Format.*

The Athernet packet consists of a Athernet packet header and payload.

The Athernet packet header occupies 10 bytes. The 1 to 4 bytes is the IPv4 address of the source. The 5 to 8 bytes is the IPv4 address of the destination. The 9 to 10 bytes is the length of the packet (including header).

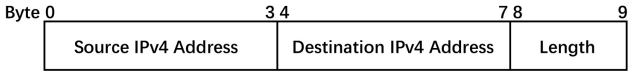

Figure 3: Athernet Packet Header Format

It is a simplified Internet Protocol. The reason we do not include a upper layer protocol identifer like IP packet header does is we always use only one upper layer protocol during one project check.

2.3.2 *Athernet Protocol.*

The protocol is straight-forward. Athernet protocol handles fragmentation to let the data fragments fit in the acoustic channel. It makes upper layer opaque to fragmentation.

The fragmentation method is discussed in 2.2.2

2.4 Upper Layer

2.4.1 *Athernet User Datagram Protocol.*

It is a simplification of UDP. It extends AP header with 2 bytes source port and 2 bytes destination port.

2.4.2 *Athernet Control Message Protocol.*

It is a simplification of ICMP. It extends AP header with 1 byte type field (ECHO or ECHO_REPLY), 2 bytes identification and 2 bytes sequence number.

It is used in Ping utilities.

2.4.3 *Transmission Control Protocol.*

We have implemented a TCP client in ATNet. It can do active 3-way handshake and active/passive 4-way teardown. It uses the header exactly like the TCP header.

However, due to the synchronization issue in our implementation, sometimes it will send out-of-order ACKs (although it does not affect data transmission and receive). In consider of this, and also because of lack of time, we do not use it in practice.

3 ARCHITECTURE

The figure 4 describes the overall architecture of ATNet.

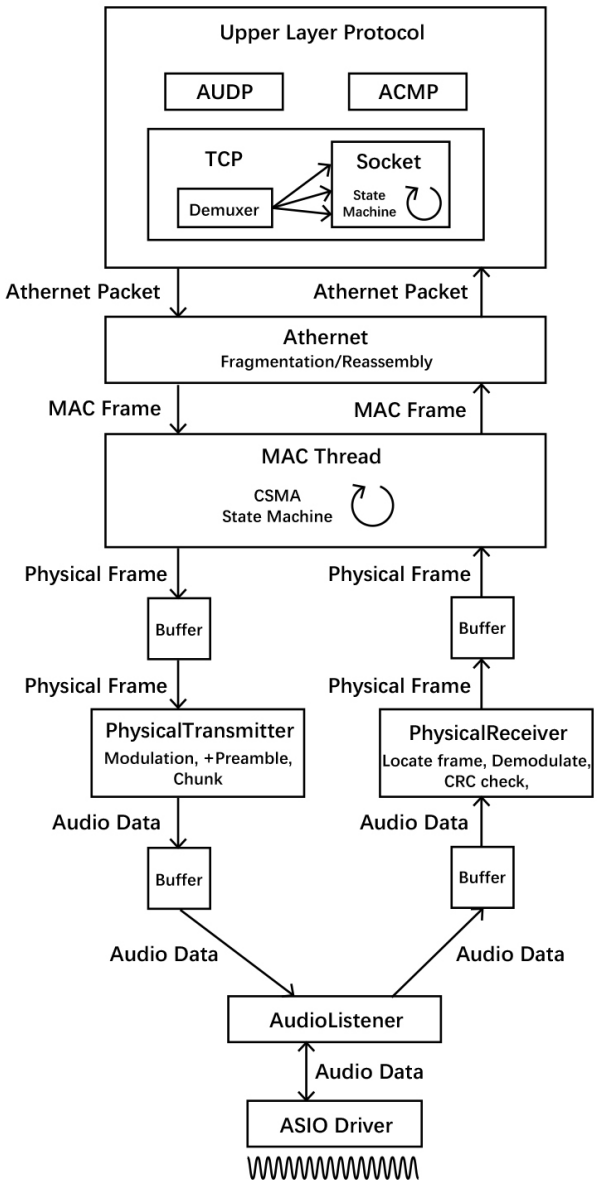

Figure 4: Architecture

3.1 Physical Layer

The physical layer has a AudioLisener componenet for handling ASIO driver and audio operations. It also provide interface for MAC layer to do carrier sense.

The PhysicalTransmitter thread is in charge of modulate payload, add preamble, split audio data into chunks with size determined by ASIO driver.

The PhysicalReceiver thread is in in charge of locate physical frames by preamble, demodulate audio data and check CRC status.

We use Phase Shift Keying for modulation. The detailed process is very similar to the sample code.

3.2 MAC Layer

The MAC layer wraps the physical layer, and has a MACThread for handling MAC frames. It will poll physical layer for available received MAC frames. It will also poll transmission frame buffer (where frames to send is bufferred) and send these frames, and wait for ACK it requires one. It will also do CSMA.

3.3 Network Layer

The network layer simply wraps the MAC layer and handles fragmentation.

3.4 TCP implementation

The TCP layer wraps the network layer and runs a TCPThread. The TCPThead handles registration of TCPSockets, and demultiplexing Athernet packets to corresponding sockets.

Each TCPSocket has its identifer: port. Also it has a TCP state machine thread.

4 APPLICATIONS

4.1 NAT service

We have implemented various NAT services for different protocols and projects (AUDP, ACMP, TCP, FTP).

An NAT service in ATNet is a bridge between ATNet and Internet. So it does not only do network address translation, but also does media transform.

4.2 FTP service

We have implemented a FTP client according to Project 4 specification. The FTP client we implemented supports these commands: `CONN`, `USER`, `PASS`, `PWD`, `CWD`, `PASV`, `LIST` and `RETR`. We do not use out TCP implementation due to lack of time, but start TCP connections directly in the NAT node.

5 CONCLUSION

Personally, we quite like these project. We gained deeper understanding of computer network concepts and implementations. The mandatory/optional design also gives a progressive experience.

However, there are still some aspects we dislike. First, the implementation of physical is strongly depended on parameter tuning. We have such a feeling especially in project 1 and 2. Second, different programming language can cause very different experience, which is indeed out of our expectations.

For the future, we hope that instructors can provide an unified experiment platform for inter-connecting, or even a project skeleton. Therefore, we could spend less time on parameter tuning and have more time exploring more contents about computer networks.