Athernet: A Toy Network For Network Courses

***Abstract*—Athernet is the course project in ShanghaiTech CS120. It uses audio signal to transmit data. This report is an overview of our implementation of Athernet.**

I. Introduction

We chose C++ as our language for the project and JUCE as our framework. We also used the ASIO driver for windows audio to get the best latency performance. We refactored the whole project many times to build a well-designed software. At the language level, we exploited the RAII idiom for resource management, so there is no more *new* or *delete*. The network is all about the movement of data, so there are many data movements in the Athernet. We utilized the C++ move semantics and perfect forwarding to reach better performance. Our final implementation has multiple layers, and they are carefully designed to achieve the 'low in coupling and high in cohesion' principle. The layers are:

1) **Physical layer**: In the physical layer, we modulate the bits into samples, which are the amplitudes in time by using *PSK* and *OFDM*. The physical layer can send the data through both acoustic wave (needs much more samples per bit) or the electronic wave in the audio cable.
2) **MAC layer**: In the MAC layer, we implemented one of the Medium Access Control Protocol, *CSMA/CA*. When the sender receives an ACK packet, it guarantees that a packet arrives to the recipient. If it didn't receive any packet after some time, the packet is resent. For collision avoidance, the backoff algorithm is used. If the device detected someone is sending (by listening to the channel power), it will not send any more frames and sleep for a backoff time.
3) **IP/UDP layer**: We used a single layer to perform the duty of original IP and UDP layer. An IP header with IPv4 address and a UDP header with UDP port and payload length are implemented. Then we implemented the *network address translation* (NAT). A gateway is designed to forward packets between the Athernet and the Internet, allowing Athernet traffic to be routed through the Internet and vice versa. If the gateway received a packet in certain port, the gateway will forward the payload to another device in the Internet. Also, we implemented the Ping utilities, we can ping from the Athernet node to any other Internet position, and we can forward external ping with certain payload to the Athernet node.
4) **Application layer**: With the Athernet to Internet gateway, we are able to communicate with the Internet. We implemented the *File Transfer Protocol* (FTP) in the application layer. It supports login, change directory, list directory and passive mode file receiving operations.

II. Project 1: Transmitting bits

A. Modulating The Data

In order to transfer data in the medium, we first need to modulate it. In my design, the AudioFrame is responsible for modulating data to audio samples. When an AudioFrame is constructed, the input data will be modulated using the Modulator.

For the Modulator, we used 2 orthogonal carrier wave with 8000 and 16000 Hz frequency and applied Phase Shifting Keying to modulate data. To demodulate them, we just dot product the received samples with the original modulate audio and compare each of them and select the biggest result's index as our output. The one that matches the received sample will produce a biggest dot product value.

$$cos(2\pi ft)cos(2\pi ft) = \frac{1}{2}(cos(2\pi 2ft) + 1) > 0$$
$$cos(2\pi ft + \pi)cos(2\pi ft) = \frac{1}{2}(cos(2\pi 2ft) - 1) < 0$$

B. Detecting The Frame

The biggest challenge on receiving the physical frames are detecting the preamble, i.e. the physical header. We used the same header with the provided matlab program. The preamble is a piece of audio with frequency from 6000Hz to 16000Hz and then to 6000Hz. To detect it, we make a dot product with the received samples and search for the local maximum.

We carefully considered the header detect algorithm's needs and eventually came up with a solid design that can detect and demodulate the frames on the fly. The RingBuffer is the most important data structure in the design, and it supports write, read multiple elements with memcpy and a peek method that accepts a user-defined function which can be a dot product or power calculation subroutine. The peek method does not read out the data since the header detector needs to know more samples than the header itself to determine whether a dot product result is a local maximum.

C. Error Correction

For error correction, we tried to use the Reed-Solomon forward error correction algorithm. We used an open-source library called Schifra Reed Solomon to encode our data when constructing the Frame instances, the downside is the algorithm only supports fixed-length encoding and it can only fix errors on the byte level, so our final implementation did not use it.

D. Performance Result

We were able to achieve transmission speeds of 800bps with a high probability of 100% accurate rate and 1.76kbps with a greater than 99% correct rate across the air using PSK and two carrier wave OFDM in project 1.

III. Project 2: Medium Access Control

Project2 moves from wireless link to wired link, which reduces the difficulty in transmitting data, so our Athernet can reach higher bandwith after project2.

A. MAC Frame

MAC Frame is the basic unit in the MAC layer's transmission. Every MAC frame has a header, indicating the MAC source, destination and type of the frame, length of the frame, and a CRC16 checksum.

```
struct MACHeader {
    uint8_t dest;
    uint8_t src;
    uint8_t type;
    uint8_t id;
    uint16_t len;
    uint16_t crc16;
};
```

Since it only consists of the raw data types, I used struct instead of class to define it. The MAC frame is more complex than the header. When it is constructed by input data (data frames) or a type (ACK frames), the constructor will assign the frame a unique id (temporarily), and calculate its CRC value. When it is constructed by a physical frame, it will check if the physical frame is valid by checking the data length, the source and destination address and the CRC value. The result will be stored in m_isGoodMACFrame.

B. CSMA+ACK State Machine

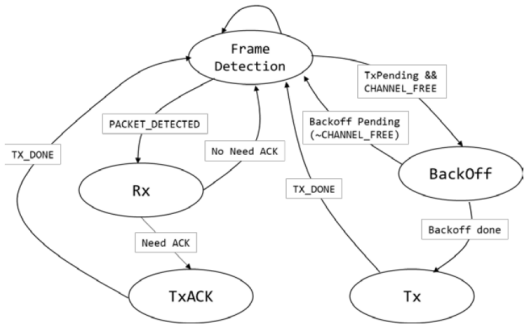

Fig. 1. The CSMA State Machine

We used a total of 4 threads to implement the state machine. Additionally, we implemented a BlockingQueue by inheriting the std::list. A thread will go to sleep when it is requiring an element from an empty blocking queue. The MAC transmitter thread (Tx) will send a packet using the CSMASenderQueue, and wait for a reply from the receiver. If the reply successfully received in time, it will send a new one, otherwise, it will try to resend the packet. The MAC receiver thread (Rx) will check every packet it receives. If the packet is an ACK, it will notify the Tx thread, and if the packet is a DATA frame, it will ask the Tx thread to send an ACK to the other side. The CSMASenderQueue will communicate with the audio device and check its state to determine when to send the frames. The frame detector is in the audioDeviceIOCallback, which is called at every IO device events, when a frame is detected and demodulated, the Rx thread will be notified.

C. Performance Result

With the stop-and-wait protocol, our wired link can achieve a throughput of 6.1kbps with the half duplex connection. When two nodes are using macperf to measure the throughput at the same time, both nodes can achieve >2.0kbps throughput. The network's round trip time is about 80ms measured by macping utility. We implemented the sliding window protocol in MAC layer at last, and the best throughput is 10kbps with 5 sliding window size and 60 bytes per frame.

IV. Project 3: Gateway

The goal of project3 is to build a gateway for the Athernet to connect to the Internet. We implemented the UDP and ICMP protocol forwarding in project3. For UDP protocol, the idea is easy that an port in the gateway corresponding to one port in the Athernet. For the ICMP protocol, we used PcapPlusPlus to capture the ICMP ping request and analyse it to determine whether we should forward it to the Athernet.

For simplicity, every packet in this layer is UDP packet with source and destination IP addresses and ports. We implemented three main classes for this layer:

1) ANetClient: The client provides many 'send' methods, from sending an array of data to sending an integer. Some methods can be used in both Athernet and Internet. If you want to let the gateway know that you will ping to Internet, send a StartPingReq first, and to stop it, send a StopPingReq. If you want the gateway to forward the external pings, send a StartRecvReq.
2) ANetServer: The server provides methods corresponding to the send methods in the client, and it can also reply the ICMP pings from the Athernet.
3) ANetGateway: The gateway can forward UDP connections and the ICMP ping request from the Athernet to the Internet and vice versa. Some utilities need to be notified by the client by sending StartPingReq or StartRecvReq at first.

V. Project 4: File Transfer Protocol

Project4 is to build a FTP client and gateway for Athernet. Since we haven't had any experience about building an FTP client in C++, I searched the existing implementations in the Internet and got one called cppftpstack that works. I cloned his git repository and compiled the library, but I found that the layer of abstraction is higher than what I want. Therefore, I modified his code to support the raw control commands for FTP.

Our design is to send the strings from Athernet to the gateway, and the gateway will establish the TCP connections to the FTP servers. When the gateway receives a packet, it extracts the server's response and sends it to the Athernet.

VI. Conclusion

We used C++ as our programming language since it can provide high performance. Although Athernet is a toy network and the bottleneck is the audio cable, we still consider software structure and performance as an important factor. This project teaches us how to build a network from scratch and we learned a lot from it, from network concepts to the design of interfaces to the C++ programming language. The physical layer is a little bit hard to debug and test, and it takes time to make it work. If conditions permit, I prefer to program on a real Network interface controller.

Acknowledgment

To the authors of the libraries and code I used in my project:

- Schifra Reed Solomon
- CRCPP
- Steinberg ASIO SDK
- The JUCE framework
- PcapPlusPlus
- cppftpstack