**Frank Yao’s project**

**Little Package Catcher**

1. Background

I have course Computer Networks this term, learning about what the structure of the network is, how every layer runs, etc. So I choose to focus on this topic as my course project. In the beginning, I download the Wireshark and learned the usage of it, did some source analysis and statistical analysis, then I start to think about if I can make a similar program myself, so I did this work.

1. Introduction

This program is based on JnetPcap, refers to famous package capture software Wireshark. It is an implementation of the capture of packages, which are passing through chosen Network Interface Card, and sending of any package input. I choose a dozen of most useful features such as Ethernet MAC, IP4 address, ports, and show them directly in the brief and pretty UI which is coded by Javafx. What's more, a sending deed through NIC with anything we input can be executed. This project also takes advantage of multiple threads to make sure everything runs smoothly and simultaneously.

1. Theory

Nowadays, the computer network employs a set of protocols, usually Tcp/Ip protocol stack to implement data transmission. Protocol in essence is a publicly acknowledged set of rules of the meaning of 01 streams, called different headers for every protocol. For communication, the headers cannot be encrypted, so that we can parse the 01stream and resolve the real information included. However, the real load inside is usually encrypted, and the deciphering belongs to cryptography, we do not include it here.

The protocols the program resolve is showed below, with corresponding layers.

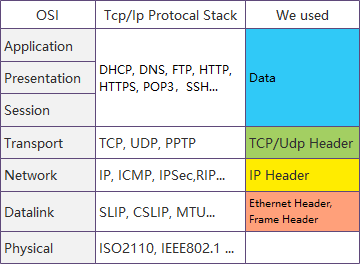


Fig1. Protocol and layer

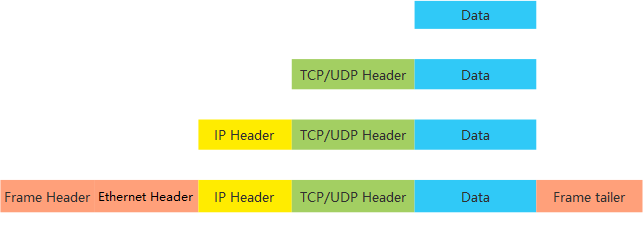


Fig2. Protocol and layer

1. Configuring Environment
   1. WinPcap

For many years, WinPcap has been recognized as the industry-standard tool for link-layer network access in Windows environments, allowing applications to capture and transmit network packets bypassing the protocol stack, and including kernel-level packet filtering, a network statistics engine, and support for remote packet capture. WinPcap consists of a driver that extends the operating system to provide low-level network access and a library that is used to easily access low-level network layers. It is the base of many applications, including Wireshark.

To put WinPcap into the computer, just download the installer and run.

* 1. JnetPcap

The access of lower levels is not available by Java Platform, so we need special SDKs to help.

JnetPcap is a widely used open-source SDK made by Sly Technologies. It uses Java Native Interface to encapsulate Libpcap, WinPcap, and something else. So that, through JnetPcap, we can process the whole package from frame head to actual data.

To install JnetPcap, we need to: First, put JnetPcap.dll under the Java.library.path in the system environment. Then import jar into our project dependence.

What we have to notice is that the version of Jnetpcap, both dll, and jar, must correspond with the java version:32 bits or 64 bits, or we can't use Jnetpcap correctly.

1. Project
   1. Purpose Function:

Scan NIC For usable device

Catch Package with choice of Device and if use Promiscuous Mode

Show the structured record as a table in user interface real-time

Jump to another scene dealing with sending our package.

Send our package we input in the text field

* 1. Project Structure

I divide the whole project into three levels: Data, Service, View. This fits the design pattern. With my layers divided, everything becomes more lucid.

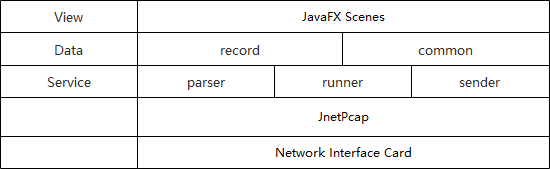
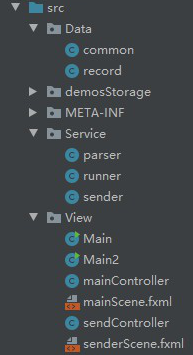


Fig3. project structure

* + 1. Service Layer:

This layer deals with our main actions.

* + - 1. Parser

Parser split the raw package into the parts we need. I mainly call the interface in JnetPcap. As time is limited, I choose the most used protocols: Frame+Ethernet+IP4/6+TCP/UDP+Data(+Http). Details of these protocols are also parsed like showed below:

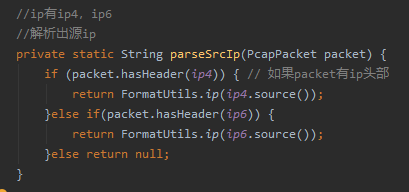


Fig. 5 Parser Fig.4 Project Structure

The split and formatted data is stored in the parser itself temporally waiting for use.

* + - 1. Runner

Runner has a function that scans the NIC ports and gets device information, and store the information into common place, which I will explain later.

When put into running, Runner initially set port and if using promiscuous mode and timeout, which is a max time limit for a wait of a new package. Then it opens a port live for use. We have an inner class jpacketHandler, overriding nextPacket function for loop use, the loop is a function inside JnetPcap, the jpacketHandler, as a parameter, is passed when we use the unchangeable loop function every time.

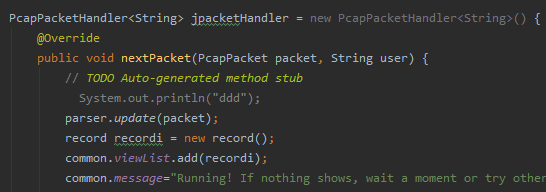


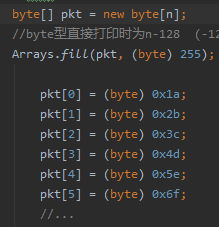
Fig.6 Runner

Inner jpackethandler, the overrode function update the static parser and create a new record object according to the fresh parser, then this record is appended in the list which is used in refreshing our ui scene.

Then we get into the loop finally, I define a new loop including the old loop, which is set only one time once used. The reason for doing this is that we have to wait a moment instead of keeping refreshing our data, even 5millisecond is enough, or it will fall into false caused by the asymmetric write and read.

* + - 1. Sender

IIn the sender, I open another port, not causing errors when running with the runner simultaneously if dealing with the same device.

First, create a default frame header, then create an array in byte type, which is our data above frame, according to the protocols, we can make any packet we want by setting by bit like showed right. The length of this array should be identical to the parameter used in creating a frame head which indicates the length of the data part.

After we combine head and data, the whole made package should be put into sending queue and send or add more packages to send. This time, if the sender and runner are at the same NIC, the runner can see the sent package.

Fig.7 part of sender

* + 1. Data Layer
       1. Common

The common class is static as a parser which means we have only one public of them. The common class maintains the overall data, like the NIC range, message, the list for showing in the scene, etc. The parser and runner refresh the records in the list and the controller of the javaFX scene maintains the input of users, for example, device number for use.

What more important is that it keeps the IDs of threads, which runs parallel. And we have a bunch of methods inside to control the threads, making sure they run and end correctly.

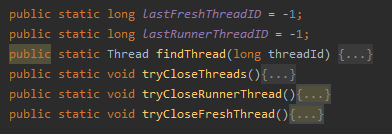


Fig.8 Common class, thread control part

* + - 1. Record

Inside record, the data structure totally corresponds with that in the table showed directly in the scene, and it provides the final data that showed in the table, so it modifies some data when initialized.

* + 1. View Layer
       1. Main scene

The controller of the main scene updates the NIC information when initialized and keep listening to events. When the start button is clicked, a new thread is created to run the runner, keep catching packages. Simultaneously, another thread that keeps refreshing the table showed to users is also created. We can start and stop at any time.

When clicked another button, a new stage of the sender will be created, it is also independent when running, which means users can catch and send at the same time.

* + - 1. Sender Scene

This part will control the send deed, which is designed as objects and threads. We can send many things in the same time, the later ones will stay and wait in the system.

I also set a demo with a 10000 times loop for demonstrating.

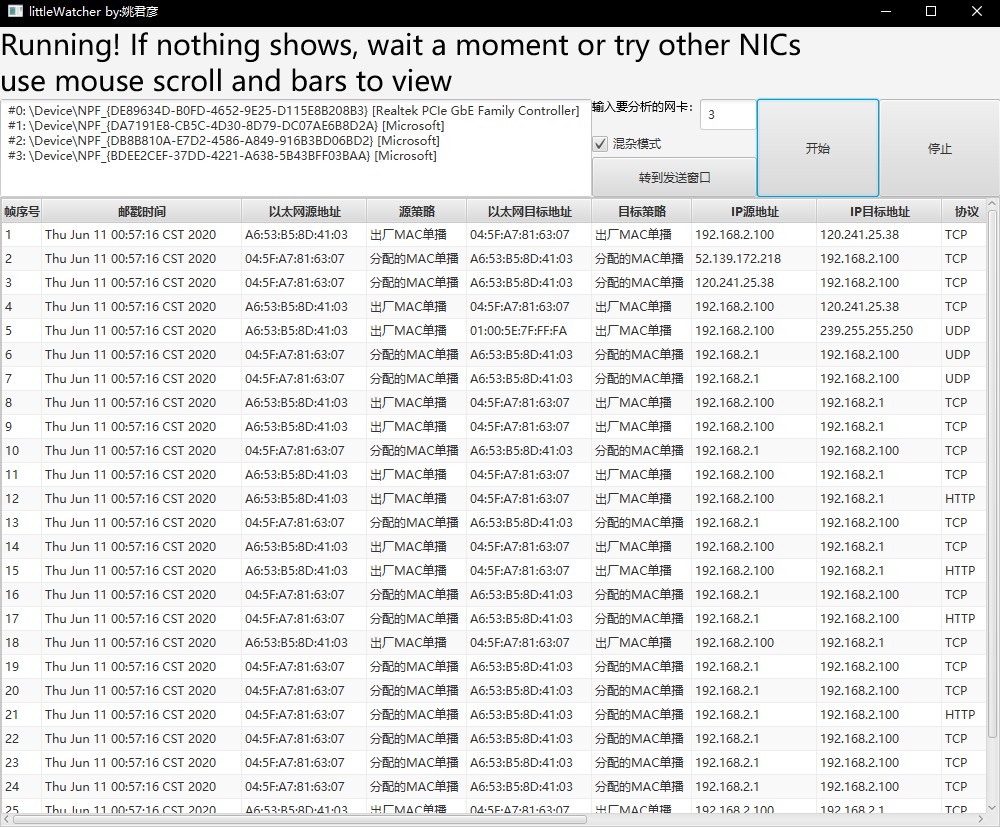
* 1. Project Packaging

In IDEA IDE, I set artifact with JnetPacp jar included then build it into a jar file.

In windows, write a .bat file to run the jar package through the command line automatically. However, .bat is just a shortcut to run the program, on other machines we still need to run jar directly.

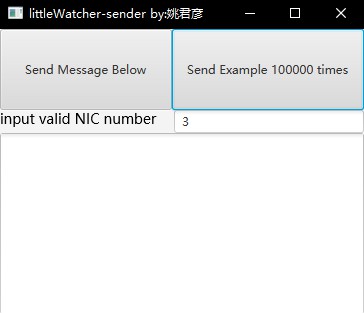
1. Function & Usage & Analysis

As we can see clearly in the screenshot, each package arrived and passed are caught and show in a table form clearly. The main information is parsed and processed.





Fig,9,10 running program window

Then, click "switch to sending window", a new window jumps out. When click "send Examples 100000 times", our example package is sent tautologically. At the same time, we can see what we sniffed in the origin window as showed in the snapshot.

As a consequence of wide use of exception dealing, with any input, the program will not be interrupted. By multiple threads, the software can run smoothly. This software can deal with sniffing and sending perfectly.

Fig.11 sending window

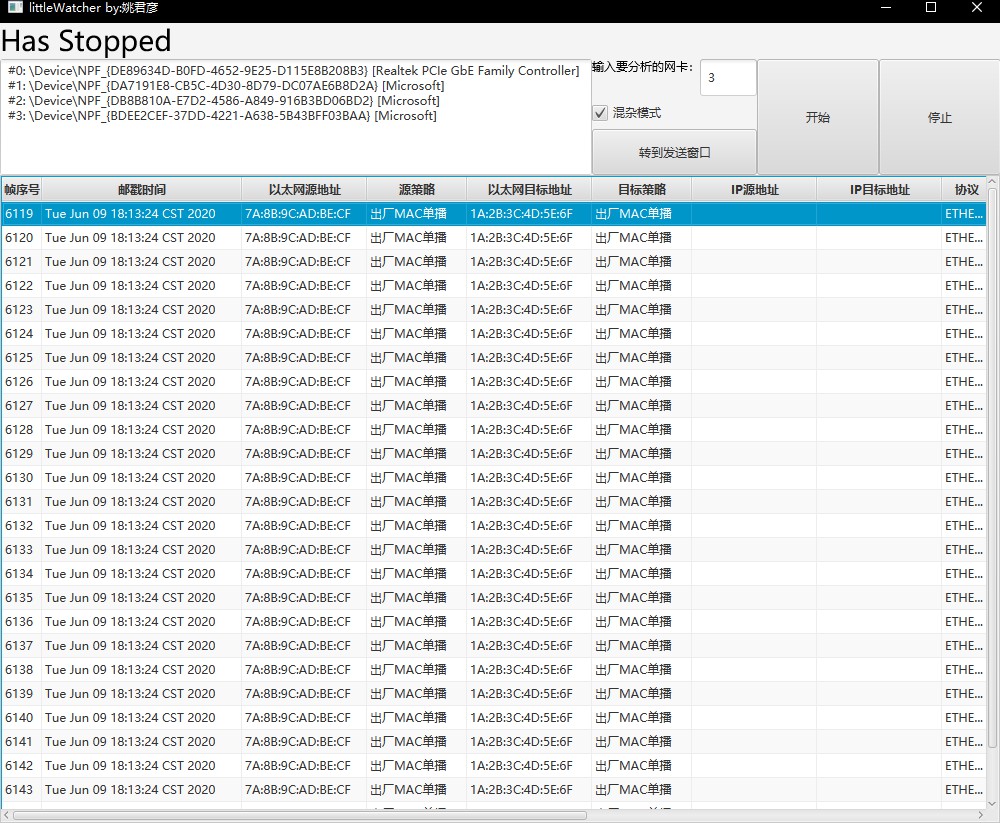


Fig.12 part of the capture window after clicking “send example 100000 times”

1. Future Work
   1. More Protocols

Many protocols like ARP, ICMP, and many higher levels of the network can be parsed and analyzed.

* 1. Complex functions

Many higher levels of application like a statistic, analysis can be implemented.

* 1. Professional usage

As we can make and send any package, any usage and attack can be made. The easiest use can be fake UDP packages attack. Once we set the destination IP, and send a fake, random package to it, it will be collapsed and the victim cannot find out our address or any useful information.

1. Summary

Through the implementation of these functions, I have a much deeper understanding of computer networks. It's my first solo program that implements from the base to the user interface so that it's an important experience to me. The basic knowledge of theory is important, which could directly decide the understanding and the practice of theory to the implementation. Only with deep understanding can a person make effective innovation, otherwise he is just coding labor. The basic design of the project is also significant because a good frame makes the development efficient. I adjust the structure while developing, and I believe this method spends much more time than having a good frame at first. I spent a bunch of time to deal with the exhausting bugs, if I could be experienced and have more practical basic ideas, the time wasted can be declined. Finally, any great project needs teamwork nowadays, even with such a simple-look project, I spend about a week. The cooperation will make it much easier. After my presentation, I will post my work on GitHub for more people to learn and have my project extended by others, if possible.