

计算机通信网络大作业项目文档

陈子璇 518030910078

2020 年 12 月 17 日

1 组员及分工

1.1 组员信息

1. 陈子璇（518030910078）
2. 闫宁（518030910093）

1.2 分工

1. 基本功能实现：通过指定需要侦听的网卡，侦听进出本主机的数据包（ETHERNET, ARP, IPv4, IPv6, ICMP, IGMP, TCP, UDP, HTTP）并解析数据包的内容——陈子璇
2. IP 分片重组——闫宁
3. 包过滤：侦听指定源、目的 IP 或 MAC 地址，或指定类型的数据包——陈子璇
4. 数据包查询：能够通过匹配输入关键字段对数据包数据部分内容进行查找，显示符合条件的数据包——陈子璇
5. 数据包保存：保存选中的数据包到名为 ” < savetime > _packet.txt” 的文件，保存内容具备可读性——陈子璇
6. 文档撰写——陈子璇

2 概述

winSniffer 是一个运行于 Windows 系统下，基于 Winpcap4.1.2 的抓包软件。使用了 visual studio 2019 提供的 MFC 框架进行构建。

2.1 运行环境

1. Windows 10
2. Winpcap 4.1.2
3. IDE: Visual Studio 2019

2.2 编译工具

Visual Studio 2019 自带的编译工具 MSVC++ 14.22 _MSC_VER == 1922

2.3 程序文件列表

```
.
├── README.md
├── res
│   ├── winSniffer.ico
│   └── winSniffer.rc2
├── pcap/
│   └── winSniffer.pcap
├── capture/
│   └── *_packet.txt
├── # 项目文件
├── winSniffer.sln
├── winSniffer.rc
├── winSniffer.vcxproj
├── winSniffer.vcxproj.user
├── winSniffer.vcxproj.filters
└── winSniffer.aps
```

```
|
|_ # 头文件
|_ pch.h
|_ framework.h
|_ resource.h
|_ stdafx.h
|_ targetver.h
|_ threadParam.h
|_ protocolHeader.h
|_ packetHeader.h
|_ packetPool.h
|_ packetCatcher.h
|_ packetDumper.h
|_ winSniffer.h
|_ winSnifferDlg.h
|_ utils.h
|_ #cpp 文件
|_ pch.cpp
|_ winSniffer.cpp
|_ winSnifferDlg.cpp
|_ threadParam.cpp
|_ packetHeader.cpp
|_ packetPool.cpp
|_ packetCatcher.cpp
|_ packetDumper.cpp
|_ utils.cpp
```

3 主要算法

抓包过程实现主要算法：

1. 调用 `pcap_findalldevs_ex()` 函数查询本机上所有网卡
2. 调用 `pcap_open()` 打开指定网卡并进行监听
3. 调用 `pcap_dump_open()` 将所监听的网卡对应的句柄备份到指定路径

下

4. startCapture 函数创建新的线程根据备份的句柄进行抓包, 将抓到的包存入数据包池 *m_pool* 中
5. stopCapture 函数结束抓包过程, 返回

4 主要数据结构

4.1 MAC 地址

```
1 typedef struct MAC_Address {  
2     u_char bytes[6];  
3 }MAC_Address;
```

4.2 IPv4 地址

```
1 typedef struct IP_Address {  
2     u_char bytes[4];  
3 }IP_Address;
```

4.3 IPv6 地址

```
1 typedef struct IPv6_Address {  
2     u_char bytes[16];  
3 }IPv6_Address;
```

4.4 数据包 Packet

```
1 class packet {  
2     public:  
3     /* 数据包头部 */
```

```

4      Ethernet_Header *eth_header;
5      IP_Header       *ip_header;
6      IPv6_Header     *ipv6_header;
7      ARP_Header      *arp_header;
8      ICMP_Header     *icmp_header;
9      IGMP_Header     *igmp_header;
10     TCP_Header      *tcp_header;
11     UDP_Header      *udp_header;
12
13     /* 数据包所携带信息 */
14     CString          message;
15     /* HTTP数据包信息 */
16     u_char           *http_msg;
17     /* 数据包数据 */
18     u_char           *packet_data;
19     /* 捕捉数据包长度及数据包到达时间 */
20     struct pcap_pkthdr *header;
21     /* 数据包编号 */
22     u_short          num;
23     /* 协议 */
24     CString          protocol;
25 }

```

4.5 数据链路层帧 Ethernet

```

1  /* 数据链路层帧 Ethernet */
2  typedef struct Ethernet_Header {
3      MAC_Address dst;           // 目的MAC地址
4      MAC_Address src;          // 源MAC地址
5      u_short eth_type;         // 类型
6  } Ethernet_Header;

```

4.6 IP 数据包头部 IP Header

```
1  /* IP数据包头部 */
2  typedef struct IP_Header {
3      /* 版本号(4 bits) + 首部长度(4 bits) */
4      u_char          ver_headerLen;
5      /* 服务类型 */
6      u_char          tos;
7      /* 总长度 */
8      u_short         total_len;
9      /* 标识 */
10     u_short         identifier;
11     /* 标志(3 bits) + 片偏移(13 bits) */
12     u_short         flags_offset;
13     /* 生存时间 */
14     u_char          ttl;
15     /* 协议 */
16     u_char          protocol;
17     /* 首部校验和 */
18     u_short         checksum;
19     /* 源IP地址 */
20     IP_Address      src;
21     /* 目的IP地址 */
22     IP_Address      dst;
23     /* 选项和填充 */
24     u_int           option_padding;
25 }IP_Header;
```

4.7 IPv6 数据包头部 IPv6 Header

```
1  typedef struct IPv6_Header {
2      /* 版本号(4 bits) */
3      u_char          version;
```

```

4      /* 优先级(8 bits) */
5      u_char          traffic;
6      /* 流标识(20 bits) */
7      u_short         label;
8      /* 报文长度 (16 bit) */
9      u_char          length[2];
10     /* 下一头部 (8 bit) */
11     u_char          next_header;
12     /* 跳数限制 (8 bit) */
13     u_char          limits;
14     /* 源IPv6地址 (128 bit) */
15     IPv6_Address    src;
16     /* 目的IPv6地址 (128 bit) */
17     IPv6_Address    dst;
18 } ipv6_header;

```

4.8 ARP 数据包头部 ARP Header

```

1  /* ARP数据包头部 */
2  typedef struct ARP_Header {
3      /* 16位硬件类型 */
4      u_short         hw_type;
5      /* 16位协议类型 */
6      u_short         protocol_type;
7      /* 8位硬件长度 */
8      u_char          hw_len;
9      /* 8位协议长度 */
10     u_char          protocol_len;
11     /* 16位操作码 */
12     u_short         opcode;
13     /* 源MAC地址 */
14     MAC_Address     src_mac;
15     /* 源IP地址 */

```

```

16     IP_Address          src_ip;
17     /* 目的MAC地址 */
18     MAC_Address         dst_mac;
19     /* 目的IP地址 */
20     IP_Address          dst_ip;
21     /* 填充 */
22     u_char               padding[18];
23 }ARP_Header;

```

4.9 ICMP 数据包头部 ICMP Header

```

1  /* ICMP数据包头部 */
2  typedef struct ICMP_Header {
3      /* 类型 */
4      u_short          icmp_type;
5      /* 代码 */
6      u_char           icmp_code;
7      /* 校验和 */
8      u_short          icmp_checksum;
9      /* 标识 */
10     u_short          icmp_id;
11     /* 序列号 */
12     u_short          icmp_seq;
13     /* 时间戳 */
14     u_short          icmp_timestamp;
15 }ICMP_Header;

```

4.10 IGMP 数据包头部 IGMP Header

```

1  /* IGMP数据包头部 */
2  typedef struct IGMP_Header {
3      /* 类型 */

```



```

4      u_short          igmp_type;
5      /* 最大响应时延 */
6      u_short          max_resp;
7      /* 校验和 */
8      u_short          igmp_checksum;
9      /* 组地址 */
10     IP_Address        group_addr;
11 }IGMP_Header;

```

4.11 TCP 数据包头部 TCP Header

```

1  /* TCP数据包头部 */
2  typedef struct TCP_Header {
3      /* 16位源端口 */
4      u_short          src;
5      /* 16位目的端口 */
6      u_short          dst;
7      /* 32位序列号 */
8      u_int             seq;
9      /* 32位确认号 */
10     u_int             ack;
11     /* 首部长(4bits) + 保留位(6bits) +
12     URG(1 bit) + ACK(1 bit) + PSH(1 bit)
13     + RST(1 bit) + SYN(1 bit) + FIN(1 bit) */
14     u_short          headerLen_rsv_flags;
15     /* 16位窗口大小 */
16     u_short          win_size;
17     /* 16位校验和 */
18     u_short          checksum;
19     /* 16位紧急指针 */
20     u_short          urg_ptr;
21     /* 选项 */
22     u_int             option;

```

```
23 }TCP_Header;
```

4.12 UDP 数据包头部 UDP Header

```
1  /* UDP数据包头部 */
2  typedef struct UDP_Header {
3      /* 源端口 */
4      u_short          src;
5      /* 目的端口 */
6      u_short          dst;
7      /* 长度 */
8      u_short          len;
9      /* 校验和 */
10     u_short          checksum;
11 }UDP_Header;
```

5 程序测试截图及说明

5.1 抓包并查看捕获数据包信息

1. 选择指定网卡
2. 点击 Start 按钮开始抓包
3. 点击 End 按钮结束抓包

5.2 数据包过滤

1. 指定数据包类型为 TCP
2. 指定源 IP 地址为本机 IP 地址
3. 点击 Filter 按钮

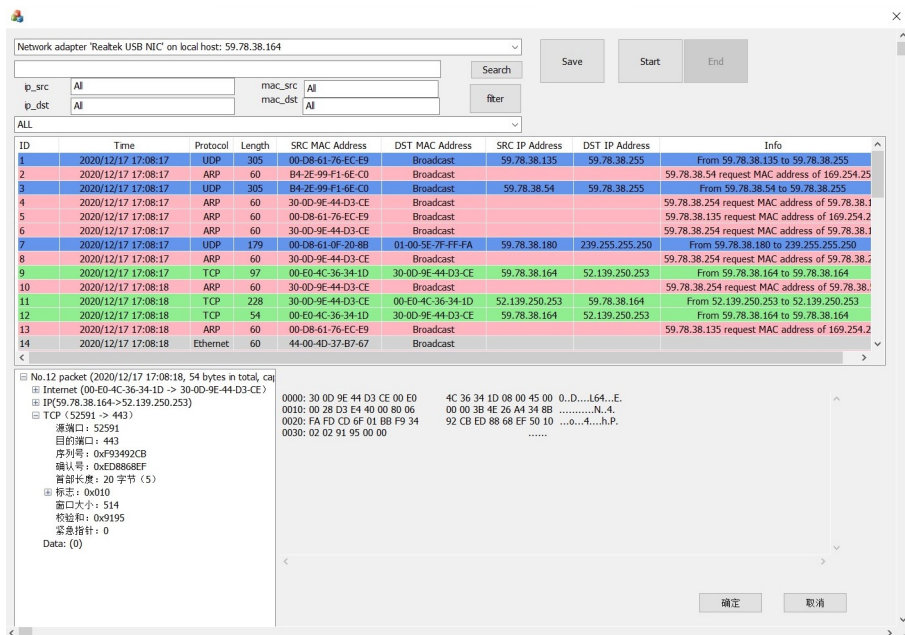


图 1: 抓包并查看捕获数据包信息

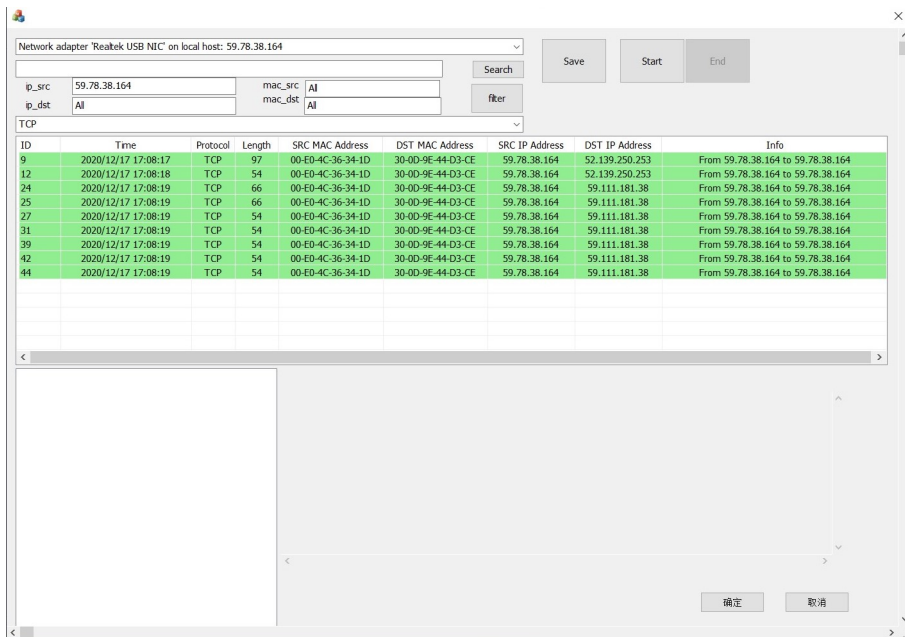


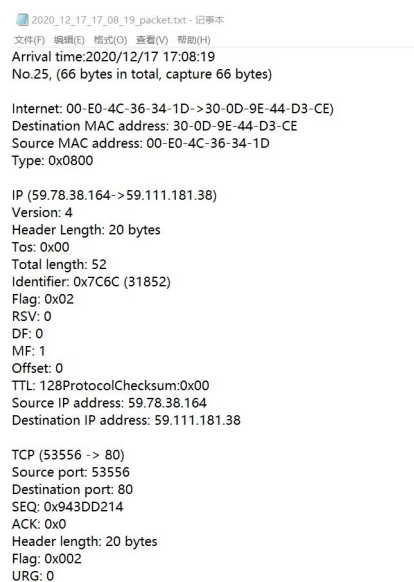
图 2: 包过滤

5.3 数据包查询

1. 在搜索框中输入关键字
2. 点击 Search 按钮

5.4 数据包保存

1. 选中要保存的数据包
2. 点击 Save 按钮



```
2020_12_17_17_08_19_packet.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
Arrival time:2020/12/17 17:08:19
No.25, (66 bytes in total, capture 66 bytes)

Internet: 00-E0-4C-36-34-1D->30-0D-9E-44-D3-CE)
Destination MAC address: 30-0D-9E-44-D3-CE
Source MAC address: 00-E0-4C-36-34-1D
Type: 0x0800

IP (59.78.38.164->59.111.181.38)
Version: 4
Header Length: 20 bytes
Tos: 0x00
Total length: 52
Identifier: 0x7C6C (31852)
Flag: 0x02
RSV: 0
DF: 0
MF: 1
Offset: 0
TTL: 128ProtocolChecksum:0x00
Source IP address: 59.78.38.164
Destination IP address: 59.111.181.38

TCP (53556 -> 80)
Source port: 53556
Destination port: 80
SEQ: 0x943DD214
ACK: 0x0
Header length: 20 bytes
Flag: 0x002
URG: 0
```

图 3: 数据包保存格式

6 遇到的问题及解决方法

本次大作业遇到的一个主要的问题是 vs2019 配置 winpcap 开发环境。由于 winpcap 库属于外部依赖, 需要手动配置 pcap.lib 路径。首先要在 vs2019 的包含目录和库目录中添加 winpcap 的 lib/x64 和 include 目录路径, 之后在依赖项中添加 ws2_32.lib、wpcap.lib、Packet.lib 三个库文件。这样在引用 `#include"pcap.h"` 的时候就不会报错了。

由于之前没有接触过 MFC，所以 MFC 框架使用对我来说也是一个比较巨大的挑战。主要难点在于对 MFC 组件的了解比较少，想要应用任何一个组件的时候都要查阅文档了解其各种属性以及对应的方法。另外 MFC 的另一个比较坑的地方是需要使用 CString 作为输出对象，刚开始我并不清楚这一点，所以使用 string 等类型进行输出的时候常常会遇到报错，再通过仔细阅读文档之后我才明白需要使用 CString 类型。在查阅了 CString 的相关用法之后我才解决了这个问题。

7 体会及建议

7.1 体会

通过本次大作业，我更加深刻地了解了课上所学习的关于网络互联与各层数据报相关的内容。在实现各类数据包头对应的数据结构的过程中查阅课程 PPT 和网上资料都进一步加深了我对于相关内容的理解与认识。另外，在解析数据包内容的实现过程中，对于计算机网络的分层结构，我有了更加清晰的认识，同时也学习了如何对不同分层的数据包进行解析的方法。

本次大作业也使得我学会了如何使用 MFC 框架搭建 GUI，虽然最开始入门 MFC 有一定的难度，也踩过很多坑，但通过阅读相关资料和文档，我逐渐掌握了基本的 MFC 用法。这段学习经历不仅仅让我掌握了 MFC 程序的构建方法，也一定程度上提升了我阅读文档的能力。

7.2 建议

参考文献

- [1] WinPcap 文档 (https://www.winpcap.org/docs/docs_412/html/main.html)
- [2] vs2019 配置 winpcap 开发环境 (<https://www.cnblogs.com/taos/p/12553925.html>)
- [3] 一步步开发 sniffer(Winpcap + MFC) (<https://blog.csdn.net/litingli/article/details/5950962>)

[4] MFC 文档 (<https://docs.microsoft.com/zh-cn/cpp/mfc/mfc-desktop-applications?view=msvc-160>)