厦門大學

信息学院软件工程系

《计算机网络》实验报告

题　　目 实验三　基于 PCAP 库侦听并分析网络流量

班　　级 软件工程 2020 级数媒班

姓　　名 陈鑫蕾

学　　号 22920202202877

实验时间 2022 年 11 月 02 日

2022 年 11 月 02 日

填写说明

1、本文件为 Word 模板文件，建议使用 Microsoft Word 2019 打开，在可填写的区域中如实填写；

2、填表时勿破坏排版，勿修改字体字号，打印成 PDF 文件提交；

3、文件总大小尽量控制在 1MB 以下，最大勿超过 5MB；

4、应将材料清单上传在代码托管平台上；

5、在实验课结束 14 天内，按原文件发送至课程 FTP 指定位置。

1 实验目的

通过完成实验，理解数据链路层、网络层、传输层和应用层的基本原理。掌握用 Wireshark 观察网络流量并辅助网络侦听相关的编程；掌握用 Libpcap 或 WinPcap 库侦听并处理以太网帧和 IP 报文的方法；熟悉以太网帧、IP 报文、TCP 段和 FTP 命令的格式概念，掌握 TCP 协议的基本机制；熟悉帧头部或 IP 报文头部各字段的含义。熟悉 TCP 段和 FTP 数据协议的概念，熟悉段头部各字段和 FTP 控制命令的指令和数据的含义。

2 实验环境

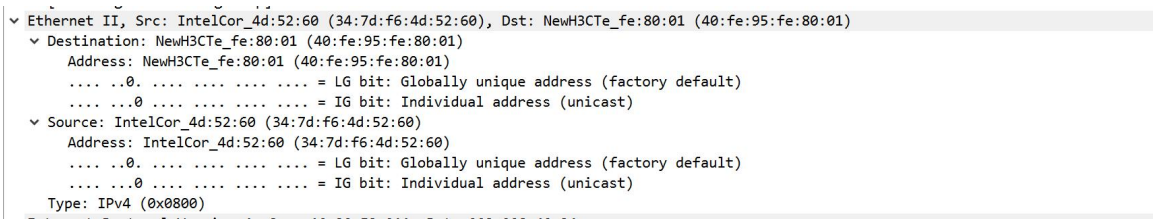
操作系统：Windows

编程语言：C++

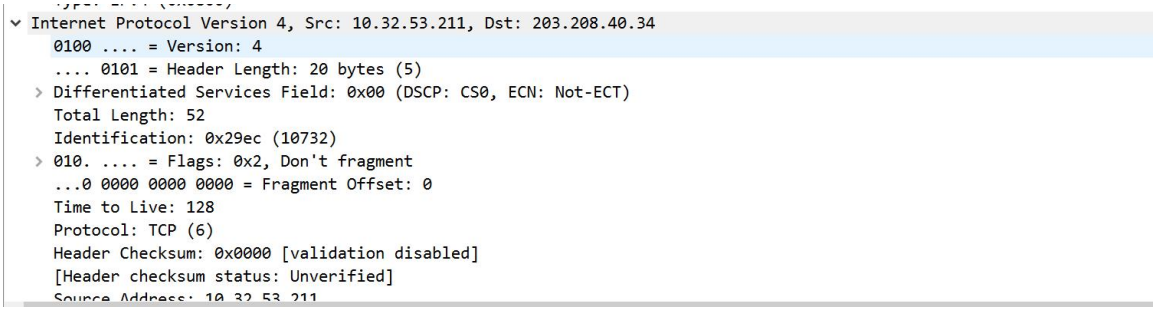
3 实验结果

1. 用侦听解析软件观察数据格式：

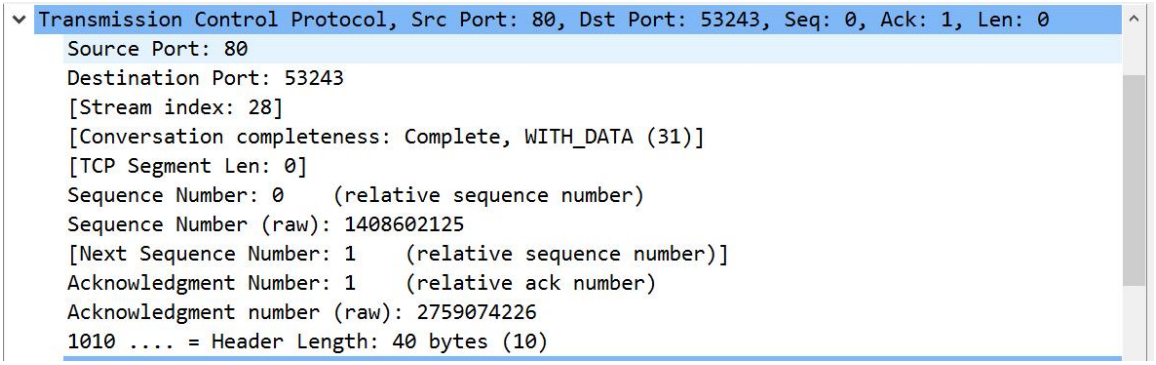
帧头部信息：



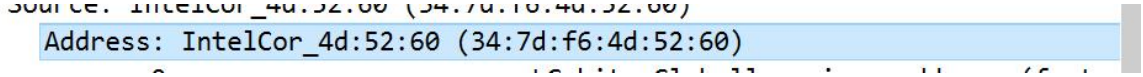
IP 报包头部信息：



TCP 数据段：

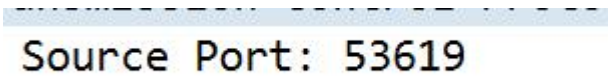


Mac 地址:



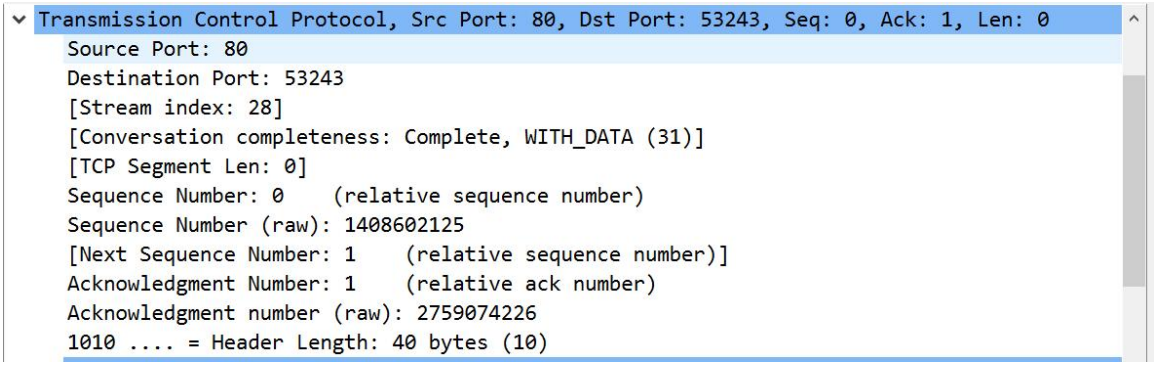
IP 地址：

TCP 端口：

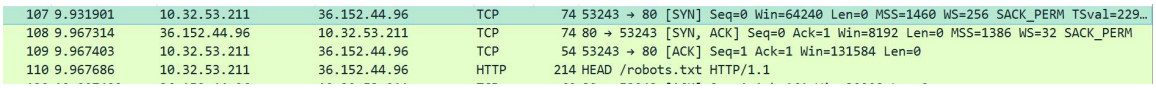


2. 用侦听解析软件观察 TCP 机制

TCP 数据段：



TCP 三次握手：



TCP 四次挥手：

142	10.009310	36.152.44.96	10.32.53.211	HTTP	151 HTTP/1.1 200 OK
143	10.009731	36.152.44.96	10.32.53.211	TCP	60 80 → 53243 [FIN, ACK] Seq=98 Ack=161 Win=30208 Len=0
144	10.009861	10.32.53.211	36.152.44.96	TCP	54 53243 → 80 [ACK] Seq=161 Ack=99 Win=131328 Len=0
150	10.011358	10.32.53.211	36.152.44.96	TCP	54 53243 → 80 [FIN, ACK] Seq=161 Ack=99 Win=131328 Len=0
215	10.050951	36.152.44.96	10.32.53.211	TCP	60 80 → 53243 [ACK] Seq=99 Ack=162 Win=30208 Len=0

TCP 窗口机制：

```
TCP      66 53619 → 443 [ACK] Seq=783 Ack=4986 Win=131072 Len=0 TSval=2697991 TSecr=43…
TCP      66 443 → 53619 [ACK] Seq=4986 Ack=783 Win=67840 Len=0 TSval=432870239 TSecr=2…
TCP      66 443 → 53618 [ACK] Seq=4339 Ack=582 Win=66816 Len=0 TSval=3588201545 TSecr=…
TCP      66 443 → 53618 [ACK] Seq=4339 Ack=769 Win=67840 Len=0 TSval=3588201545 TSecr=…
```

```
Window: 512
[Calculated window size: 131072]
[Window size scaling factor: 256]
```

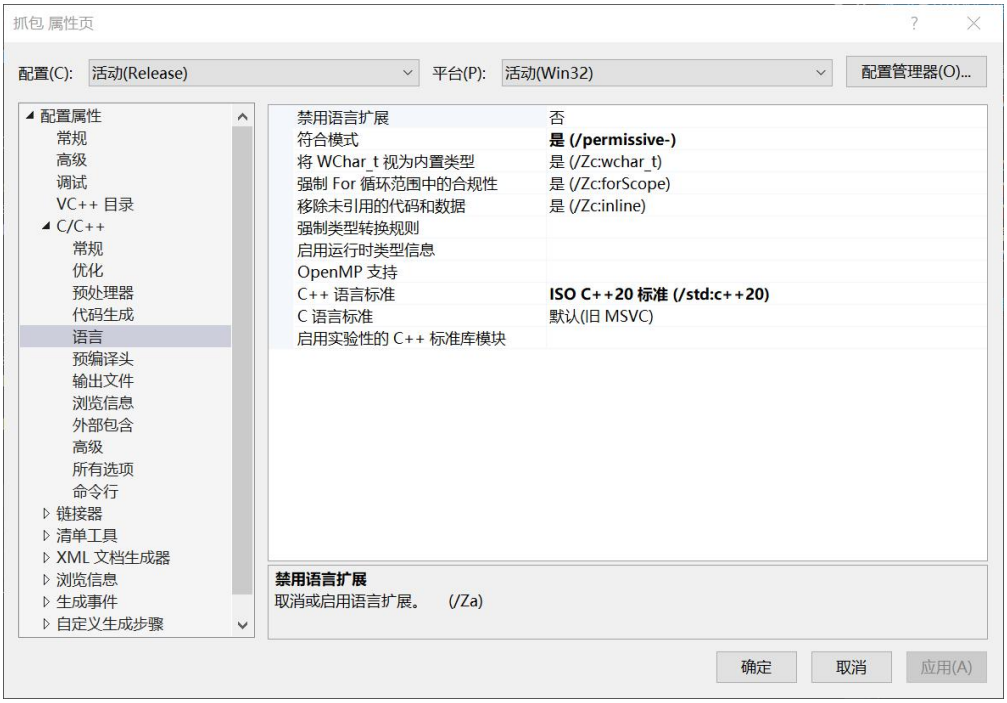
TCP 拥塞控制机制：

当接收窗口所设定的速率无法使服务器正常处理数据时即出现拥塞时，能够调整接收窗口大小来减小数据传输速率。通过减小返回给发送端的 ACK 报文的 TCP 头窗口大小值来实现。在发生分组丢失的极端情况下，TCP 会暂时将窗口大小减为当前值的一半。

3. 用 WinPcap 侦听网络数据

环境配置：

首先先将 npcap 文件夹下的 include 和 lib 连接到项目。记得选择 C++语言标准为 17 版本及以上，否则将会报错。（因为后面采用了一点新标准的语法）



创建以太网报头、IP 报头、TCP 报头格式

```
    }
};

struct IPHeader {
    u_char ver_ihl;
    u_char tos;
    u_short tlen;
    u_short identification;
    u_short flags_fo;
    u_char ttll;
    u_char proto;
    u_short crc;
    IPV4 saddr;
    IPV4 daddr;
    //u_int op_pad;
};

struct TCPHeader {
    u_short sport;
    u_short dport;
    union {
        u_short _seq[2];
        u_int sequence;
    };
    union {
        u_short _conf[2];
        u_int confirm;
    };
    u_short len;
    u_short wind;
    u_short crc;
    u_short ur_point;
};
```

创建计数器类用于统计数据，将 MAC 地址与 IP 地址转换为字符串作为 map 的 key 用于统计一段时间内来自某一 IP 的包总长度。分别统计接收方与发送方，接着每五秒将统计信息打印出来。

```cpp
class Counter {
private:
    std::map<std::string, u_int> _send, _recv;

    // 将MAC地址与IP地址转换为字符串
    // e.g. AA:BB:CC:DD:EE:FF 192.168.  1.  1
    std::string _convertToString(const u_char mac[6], const u_char ip[4]) {
        char buffer[64];
        sprintf_s(buffer, "%02X-%02X-%02X-%02X-%02X-%02X %3d.%3d.%3d.%3d",
            mac[0], mac[1], mac[2], mac[3], mac[4], mac[5],
            ip[0], ip[1], ip[2], ip[3]);
        return std::string(buffer);
    }

    int _interval;
public:
    Counter(int interval) : _interval(interval) {
    }
    // 添加发送包统计
    // mac: 发送方mac地址
    // ip: 发送方ip地址
    // bytes: 包字节数
    void AddSend(const u_char mac[6], const u_char ip[4], u_int bytes) {
        std::string s = _convertToString(mac, ip);
        if (!_send.count(s))
            _send[s] = 0;
        _send[s] += bytes;
    }
    // 添加接收方统计
    // 参数同发送包统计
    void AddRecv(const u_char mac[6], const u_char ip[4], u_int bytes) {
        std::string s = _convertToString(mac, ip);
        if (!_recv.count(s))
            _recv[s] = 0;
        _recv[s] += bytes;
    }
    // 清空计数器
    void Clear() {
        _send.clear();
        _recv.clear();
    }
    // 打印统计信息
    // f: 目标文件流，默认为保准输出流
    void Print(FILE* f = stdout) {
        fprintf(f, "\nStatistics in last %d seconds:\n\nBytes sended:\n", _interval);
        for (auto& s : _send) {
            fprintf(f, "%s: %10u\n", s.first.c_str(), s.second);
        }
        fprintf(f, "\n\nBytes recieved:\n");
        for (auto& s : _recv) {
            fprintf(f, "%s: %10u\n", s.first.c_str(), s.second);
        }
    }
};

int main() {
```

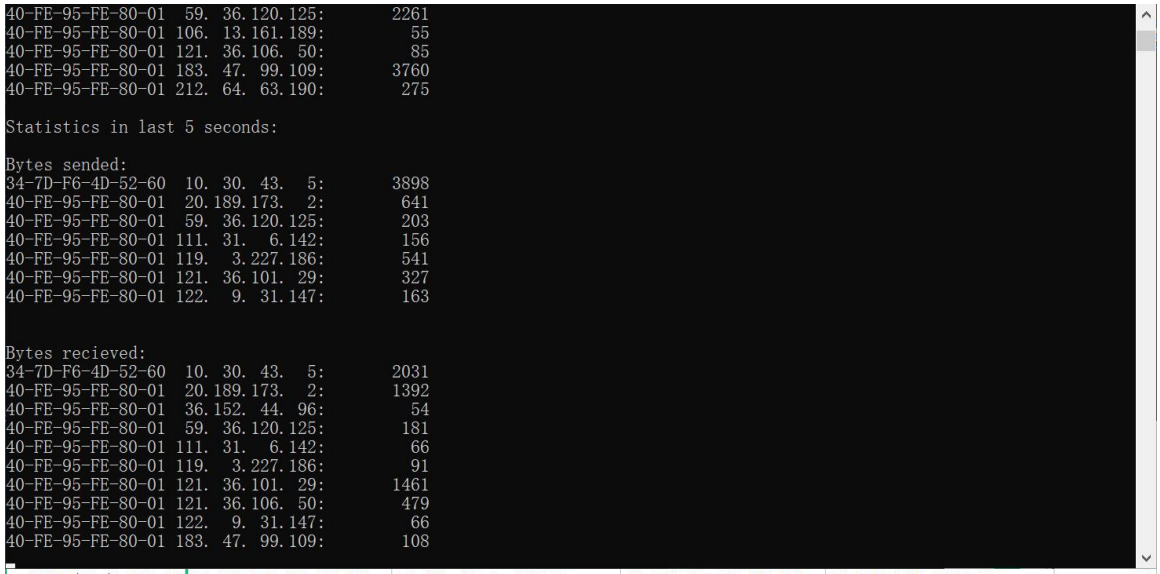
接着先获取 adapters，选择合适的 adapters，并设置过滤器为“ip and tcp”，

```
    // filtering
    bpf_program fcode;
    if (pcap_compile(adhandle, &fcode, "ip and tcp", 1, netmask) < 0) {
        fprintf(stderr, "\nUnable to compile the packet filter. Check the syntax.\n");
        pcap_freealldevs(all_devs);
        return 1;
    }

    if (pcap_setfilter(adhandle, &fcode) < 0) {
        fprintf(stderr, "\nError setting the filter.\n");
        pcap_freealldevs(all_devs);
        return 1;
    }
```

处理捕获到的 TCP 包，先解析包头时间，头部，端口。接着将解析的数据写入 csv，同时更新 counter。添加超时处理。

```
for (res = pcap_next_ex(adhandle, &header, &pkt_data); res >= 0; res = pcap_next_ex(adhandle, &header, &pkt_data)) {
    if (res == 0)
        continue;

    tm ltime;
    char timestr[64];
    time_t local_tv_sec;

    // 解析包头时间
    local_tv_sec = header->ts.tv_sec;
    localtime_s(&ltime, &local_tv_sec);
    strftime(timestr, sizeof(timestr), "%Y-%m-%d %H:%M:%S", &ltime);

    if (startTime == 0)
        startTime = local_tv_sec;

    // 解析头部
    EthernetHeader* mh = (EthernetHeader*)pkt_data;
    IPHeader* ih = (IPHeader*)(pkt_data + 14);
    int ip_len = (ih->ver_ihl & 0xf) * 4;
    TCPHeader* th = (TCPHeader*)((u_char*)ih + ip_len);

    // 解析端口
    u_short sport, dport;
    sport = ntohs(th->sport);
    dport = ntohs(th->dport);

    // 写入csv
    Log(timestr, mh, ih, sport, dport, header->len, log);
    // 更新counter
    c.AddSend(mh->smac, ih->saddr.byte, header->len);
    c.AddRecv(mh->dmac, ih->daddr.byte, header->len);

    // 超时后打印统计信息
    if (local_tv_sec - startTime > interval) {
        c.Print();
        c.Clear();
        startTime = 0;
    }
}

fclose(log);

if (res == -1) {
    printf("Error reading the packets: %s\n", pcap_geterr(adhandle));
    return -1;
}
```

运行效果：



2022/11/8 20:49	34-7D-F6-4D-52-60	10.30.43.5:49869	40-FE-95-FE-80-01	59.36.120.125:80	54
2022/11/8 20:49	40-FE-95-FE-80-01	59.36.120.125:80	34-7D-F6-4D-52-60	10.30.43.5:49869	143
2022/11/8 20:49	34-7D-F6-4D-52-60	10.30.43.5:49869	40-FE-95-FE-80-01	59.36.120.125:80	54
2022/11/8 20:49	34-7D-F6-4D-52-60	10.30.43.5:53290	40-FE-95-FE-80-01	121.36.101.29:443	85
2022/11/8 20:49	34-7D-F6-4D-52-60	10.30.43.5:53428	40-FE-95-FE-80-01	121.36.101.29:443	85
2022/11/8 20:49	34-7D-F6-4D-52-60	10.30.43.5:53383	40-FE-95-FE-80-01	123.60.221.20:443	85
2022/11/8 20:49	34-7D-F6-4D-52-60	10.30.43.5:53421	40-FE-95-FE-80-01	121.36.101.29:443	85
2022/11/8 20:49	34-7D-F6-4D-52-60	10.30.43.5:53429	40-FE-95-FE-80-01	121.36.101.29:443	85
2022/11/8 20:49	34-7D-F6-4D-52-60	10.30.43.5:53507	40-FE-95-FE-80-01	36.152.44.95:80	54
2022/11/8 20:49	34-7D-F6-4D-52-60	10.30.43.5:53502	40-FE-95-FE-80-01	36.152.44.96:80	54
2022/11/8 20:49	40-FE-95-FE-80-01	59.36.120.125:80	34-7D-F6-4D-52-60	10.30.43.5:49869	143
2022/11/8 20:49	34-7D-F6-4D-52-60	10.30.43.5:53432	40-FE-95-FE-80-01	121.36.101.29:443	85
2022/11/8 20:49	34-7D-F6-4D-52-60	10.30.43.5:49869	40-FE-95-FE-80-01	59.36.120.125:80	54
2022/11/8 20:49	40-FE-95-FE-80-01	52.182.141.63:443	34-7D-F6-4D-52-60	10.30.43.5:53508	472
2022/11/8 20:49	34-7D-F6-4D-52-60	10.30.43.5:53508	40-FE-95-FE-80-01	52.182.141.63:443	254
2022/11/8 20:49	40-FE-95-FE-80-01	119.3.227.186:11113	34-7D-F6-4D-52-60	10.30.43.5:49764	475
2022/11/8 20:49	34-7D-F6-4D-52-60	10.30.43.5:49764	40-FE-95-FE-80-01	119.3.227.186:11113	91
2022/11/8 20:49	40-FE-95-FE-80-01	52.182.141.63:443	34-7D-F6-4D-52-60	10.30.43.5:53508	108
2022/11/8 20:49	34-7D-F6-4D-52-60	10.30.43.5:53508	40-FE-95-FE-80-01	52.182.141.63:443	1440
2022/11/8 20:49	34-7D-F6-4D-52-60	10.30.43.5:53508	40-FE-95-FE-80-01	52.182.141.63:443	882
2022/11/8 20:49	40-FE-95-FE-80-01	119.3.227.186:11113	34-7D-F6-4D-52-60	10.30.43.5:49764	66
2022/11/8 20:49	34-7D-F6-4D-52-60	10.30.43.5:53517	40-FE-95-FE-80-01	36.152.44.95:80	74
2022/11/8 20:49	40-FE-95-FE-80-01	20.195.65.207:443	34-7D-F6-4D-52-60	10.30.43.5:53269	100
2022/11/8 20:49	40-FE-95-FE-80-01	36.152.44.95:80	34-7D-F6-4D-52-60	10.30.43.5:53517	74
2022/11/8 20:49	34-7D-F6-4D-52-60	10.30.43.5:53517	40-FE-95-FE-80-01	36.152.44.95:80	54

4. 解析侦听到的网络数据

前面步骤与 3 同

创建 FTPlogin 类，用于记录完整的一次 FTP 登录，包括使用的用户名，密码以及结果。最后将登录信息输出到文件流 f。

```cpp
// 记录完整的一次FTP登录，包括使用的用户名、密码以及结果
class FTPLogin {
private:
    std::string info;
    std::string user, pwd;
    int flag;
public:
    FTPLogin(std::string info) : info(info) {
        flag = -1;
    }
    void SetUser(std::string user) {
        this->user = user;
    }
    void SetPwd(std::string pwd) {
        this->pwd = pwd;
    }
    void SetFlag(bool flag) {
        this->flag = flag;
    }
    // FTP登录信息是否统计完整
    bool IsIntergral() {
        return !user.empty() && !pwd.empty() && flag != -1;
    }
    // 将登录信息输出到文件流f
    void Print(FILE* f = stdout) {
        assert(flag != -1);
        fprintf(f, "%s,%s,%s,%s\n", info.c_str(), user.c_str(), pwd.c_str(), flag == 1 ? "SUCCEED" : "FAILED");
    }
};
```

创建 FTPSet 类记录所有不完整 FTP 登陆请求的存储集合

首先将 FTP 登录额外信息（时间、源地址、目标地址、源端口、目标端口）转换为字符串，swap 参数用于控制是否反转发送/接收方

IPCS 函数将发送方 IP 与接收方 IP 转换为字符串，作为区分 FTP 登录请求的 key

ModifyFTP 函数更新指定 key 对应的 FTP 登录请求

```cpp
// 所有不完整FTP登录请求的存储集合
class FTPSet {
public:
    // 将FTP登录额外信息（时间、源地址、目标地址、源端口、目标端口）转换为字符串，swap参数用于控制是否反转发送/接收方
    static std::string ConvertToString(const char* timestr, const EthernetHeader* mh, const IPHeader* ih, int sport, int dport, bool swap = false) {
        char buffer[128];
        if (!swap) {
            sprintf_s(buffer, "%s,%02X-%02X-%02X-%02X-%02X-%02X,%d.%d.%d.%d:%d,%02X-%02X-%02X-%02X-%02X-%02X,%d.%d.%d.%d:%d",
                timestr,
                mh->smac[0], mh->smac[1], mh->smac[2], mh->smac[3], mh->smac[4], mh->smac[5],
                ih->saddr.byte[0], ih->saddr.byte[1], ih->saddr.byte[2], ih->saddr.byte[3], sport,
                mh->dmac[0], mh->dmac[1], mh->dmac[2], mh->dmac[3], mh->dmac[4], mh->dmac[5],
                ih->daddr.byte[0], ih->daddr.byte[1], ih->daddr.byte[2], ih->daddr.byte[3], dport
            );
        }
        else {
            sprintf_s(buffer, "%s,%02X-%02X-%02X-%02X-%02X-%02X,%d.%d.%d.%d:%d,%02X-%02X-%02X-%02X-%02X-%02X,%d.%d.%d.%d:%d",
                timestr,
                mh->dmac[0], mh->dmac[1], mh->dmac[2], mh->dmac[3], mh->dmac[4], mh->dmac[5],
                ih->daddr.byte[0], ih->daddr.byte[1], ih->daddr.byte[2], ih->daddr.byte[3], dport,
                mh->smac[0], mh->smac[1], mh->smac[2], mh->smac[3], mh->smac[4], mh->smac[5],
                ih->saddr.byte[0], ih->saddr.byte[1], ih->saddr.byte[2], ih->saddr.byte[3], sport
            );
        }
        return std::string(buffer);
    }

    // 将发送方IP与接收方IP转换为字符串，作为区分FTP登录请求的key
    static std::string IPCS(const IPV4& client, const IPV4& server) {
        char buffer[128];
        sprintf_s(buffer, "%d.%d.%d.%d %d.%d.%d.%d",
            client.byte[0], client.byte[1], client.byte[2], client.byte[3],
            server.byte[0], server.byte[1], server.byte[2], server.byte[3]
        );
        return std::string(buffer);
    }

    // 更新指定key对应的FTP登录请求
    // key: 使用IPCS函数生成的key
    // s: FTP包额外信息
    // action: 以key对应的FTPLogin*作为参数的回调函数，用于更新FTPLogin
    // f: FTP登录请求收集完整后结果写入的文件流
    void ModifyFTP(const std::string& key, const std::string& s, std::function<void(FTPLogin*)> action, FILE* f = stdout) {
        if (_map.find(key) == _map.end())
            _map[key] = new FTPLogin(s);
        auto temp = _map[key];
        action(temp);
        if (temp->IsIntergral()) {
            temp->Print(f);
            _map.erase(key);
            delete temp;
        }
    }
private:
    std::map<std::string, FTPLogin*> _map;
};

int main() {
    pcap_if_t* all_devs;
```

其他步骤与 TCP 类似，只是需要更改过滤器为“ip and tcp and port ftp”

```cpp
    // filtering
    bpf_program fcode;
    if (pcap_compile(adhandle, &fcode, "ip and tcp and port ftp", 1, netmask) < 0) {
        fprintf(stderr, "\nUnable to compile the packet filter. Check the syntax.\n");
        pcap_freealldevs(all_devs);
        return 1;
    }

    if (pcap_setfilter(adhandle, &fcode) < 0) {
        fprintf(stderr, "\nError setting the filter.\n");
        pcap_freealldevs(all_devs);
        return 1;
    }
```

利用包长度、以太报头长度、IP 报头长度、TCP 报头长度来计算载荷长度

```cpp
int payload_len = header->len - 14 - ip_len - tcp_len;
```

最后依次解析 USER，PASS 命令以及服务器响应。此处 if 语句使用了 C++17 新语法。

```
// 解析USER命令
if (char cmdArg[64]; sscanf_s(payload, "USER %s", cmdArg, 64) == 1) {
    ftp.ModifyFTP(ftp.IPCS(ih->saddr, ih->daddr), ftp.ConvertToString(timestr, mh, ih, sport, dport), [&cmdArg](FTPLogin* fl) {
        fl->SetUser(cmdArg);
        }, f);
}
// 解析PASS命令
else if (char cmdArg[64]; sscanf_s(payload, "PASS %s", cmdArg, 64) == 1) {
    ftp.ModifyFTP(ftp.IPCS(ih->saddr, ih->daddr), ftp.ConvertToString(timestr, mh, ih, sport, dport), [&cmdArg](FTPLogin* fl) {
        fl->SetPwd(cmdArg);
        }, f);
}
// 解析服务器响应，230:成功
else if (std::string temp(payload); temp.find("230") != std::string::npos) {
    ftp.ModifyFTP(ftp.IPCS(ih->daddr, ih->saddr), ftp.ConvertToString(timestr, mh, ih, sport, dport, true), [](FTPLogin* fl) {
        fl->SetFlag(1);
        }, f);
}
// 解析服务器响应，530:失败
else if (std::string temp(payload); temp.find("530") != std::string::npos) {
    ftp.ModifyFTP(ftp.IPCS(ih->daddr, ih->saddr), ftp.ConvertToString(timestr, mh, ih, sport, dport, true), [](FTPLogin* fl) {
        fl->SetFlag(0);
        }, f);
}
fflush(f);
}
```

运行结果：

A	B	C	D	E	F	G	H
2022/11/8 20:47	34-7D-F6-4D-52-60	10.30.43.5:53256	40-FE-95-FE-80-01	121.192.180.66:21	student	software	FAILED
2022/11/8 20:47	34-7D-F6-4D-52-60	10.30.43.5:53258	40-FE-95-FE-80-01	121.192.180.66:21	student	software	SUCCEED

4 实验代码

本次实验的代码已上传于以下代码仓库：

https://gitee.com/Cutie_Chen/computer-network/tree/master/3

5 实验总结

通过捕获并分析以太网帧，分析常见数据包的帧格式，熟悉以太网中常用协议，数据包在网络中的发送方式、各报头的作用；

学会了用 Wireshark 侦听网络数据。