实验四

1801090006 郭盈盈
2020.03.24

一、实验题目

请选择一门你熟悉的编程语言，完成如下实验内容：

1、分别基于 TCP 与 UDP 协议，编写**客户端**程序与**服务端**程序。

2、服务端程序启动后，**客户端**可以**向服务端**发送**文字信息**。

3、利用 Wireshark **分别**抓取 TCP 与 UDP 通信的数据包，要求：

（1）指出 TCP 的**三次握手**在哪里？如果有连接断开的**四次握手**，也请进行分析（如果没有抓到，则不做要求）。

（2）利用客户端向服务端发送你自己的**学号加中文姓名拼音**（如：18014JiangYe），并在 Wireshark 中**指出姓名在数据包中的位置**（注：这会作为判定是否存在抄袭的标准之一）。

（3）比较 TCP 与 UDP 数据包的**异同**。

4、提交时请提交**实验报告**、**Wireshark 抓包文件**、**全部源代码**（不需要完整的工程文件，不需要编译后的文件）。

二、相关知识

TCP 四次挥手断开

第一次挥手

客户端给服务器发送 TCP 包，用来关闭客户端到服务器的数据传送。将标志位 FIN 和 ACK 置为 1，序号为 X=1，确认序号为 Z=1。

第二次挥手

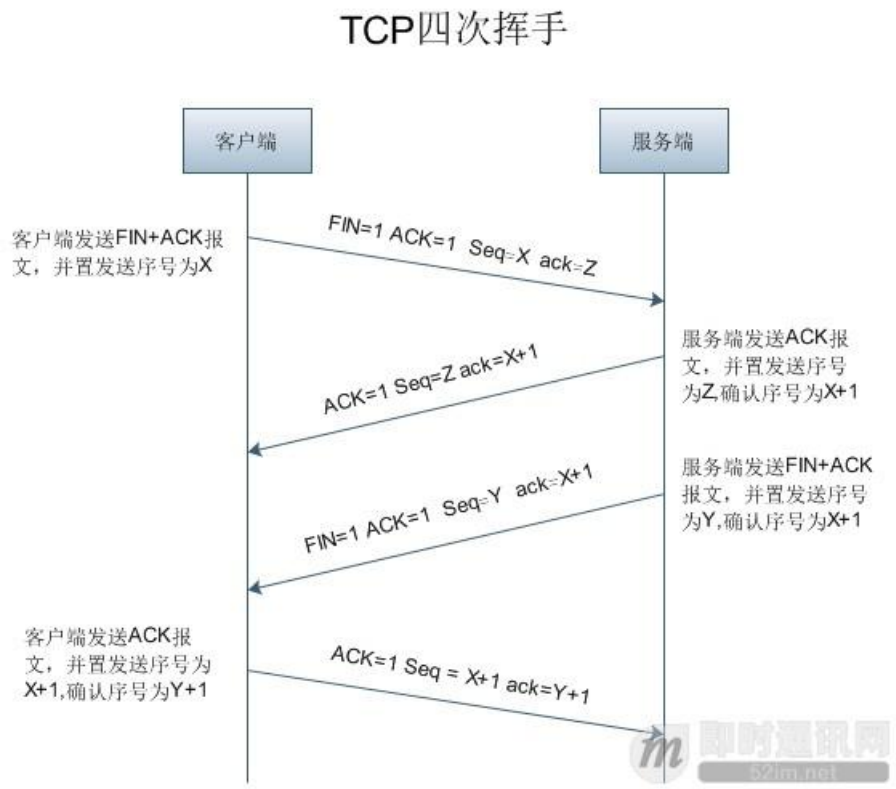
服务器收到 FIN 后，发回一个 ACK（标志位 ACK=1），确认序号为收到的序号加 1，即 X=X+1=2。序号为收到的确认序号=Z。

第三次挥手

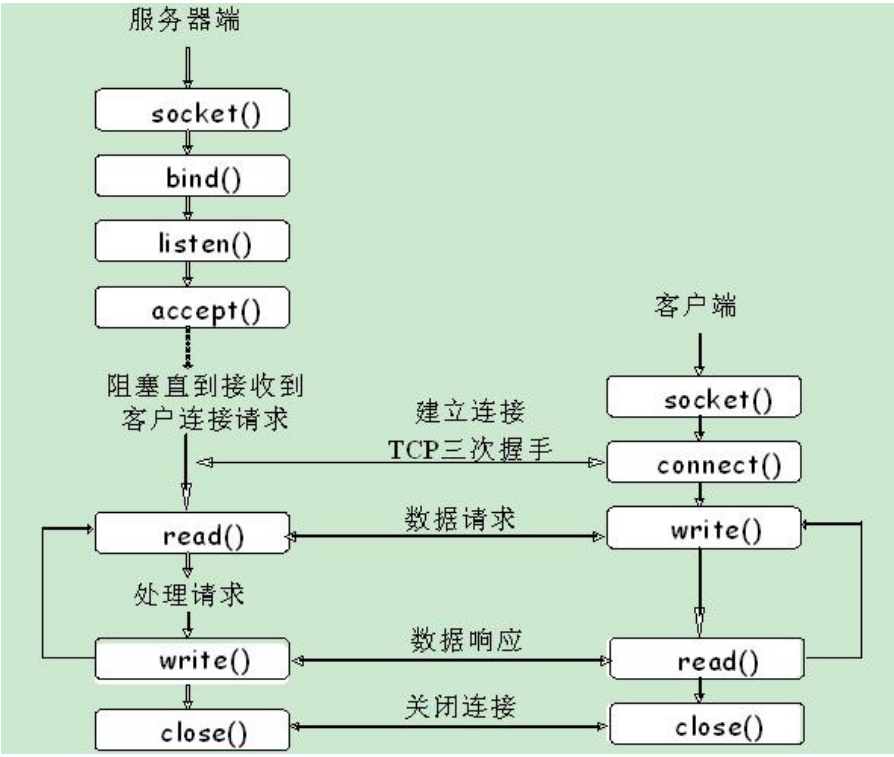
服务器关闭与客户端的连接，发送一个 FIN。标志位 FIN 和 ACK 置为 1，序号为 Y=1，确认序号为 X=2。

第四次挥手

客户端收到服务端发送的 FIN 之后，发回 ACK 确认（标志位 ACK=1），确认序号为收到的序号加 1，即 Y+1=2。序号为收到的确认序号 X=2。



TCP/UDP 服务器端和客户端程序设计



socket 函数

```
int socket(int family,int type,int protocol);
```

1. 第一个参数指明了协议簇，目前支持 5 种协议簇，最常用的有 AF_INET(IPv4 协议)和 AF_INET6(IPv6 协议)；
2. 第二个参数指明套接口类型，有三种类型可选：SOCK_STREAM(字节流套接口)、SOCK_DGRAM(数据报套接口)和 SOCK_RAW(原始套接口)；
3. 如果套接口类型不是原始套接口，那么第三个参数就为 0。

bind 函数

```
    int bind(int sockfd, const struct sockaddr * server, socklen_t addrlen);
```

1. 第一个参数是 socket 函数返回的套接口描述字；
2. 第二和第三个参数分别是一个指向特定于协议的地址结构的指针和该地址结构的长度。

listen 函数

```
    int listen(int sockfd,int backlog);
```

1. 第一个参数是 socket 函数返回的套接口描述字；
2. 第二个参数规定了内核为此套接口排队的最大连接个数。
3. 由于 listen 函数第二个参数的原因，内核要维护两个队列：以完成连接队列和未完成连接队列。未完成队列中存放的是 TCP 连接的三路握手未完成的连接，accept 函数是从已连接队列中取连接返回给进程；当以连接队列为空时，进程将进入睡眠状态。

accept 函数

```
    int accept(int listenfd, struct sockaddr *client, socklen_t * addrlen);
```

1. 第一个参数是 socket 函数返回的套接口描述字；
2. 第二个和第三个参数分别是一个指向连接方的套接口地址结构和该地址结构的长度；该函数返回的是一个全新的套接口描述字；如果对客户段的信息不感兴趣，可以将第二和第三个参数置为空。

connect 函数

```
    int connect(int sockfd, const struct sockaddr * addr, socklen_t addrlen);
```

1. 第一个参数是 socket 函数返回的套接口描述字；第二和第三个参数分别是一个指向套接口地址结构的指针和该结构的大小。
2. 这些地址结构的名字均已“sockaddr_”开头，并以对应每个协议族的唯一后缀结束。以 IPv4 套接口地址结构为例，它以“sockaddr_in”命名，定义在头文件<netinet/in.h>；

write 和 read 函数

```
int write(int sockfd, char *buf, int len);

int read(int sockfd, char *buf, intlen);
```

1. 参数 sockfd 是套接字描述符，对于服务器是 accept()函数返回的已连接套接字描述符，对于客户端是调用 socket()函数返回的套接字描述符；
2. 参数 buf 是指向一个用于发送信息的数据缓冲区；
3. len 指明传送数据缓冲区的大小。

TCP 与 UDP 的区别：

TCP 面向连接，可靠传输，流量控制，传输速度慢，协议开销大。

TCP 包头结构：

源端口 16 位|目标端口 16 位|序列号 32 位|回应序号 32 位|TCP 头长度 4 位|reserved 6 位|控制代码 6 位|窗口大小 16 位|偏移量 16 位|校验和 16 位|选项 32 位(可选)

这样我们得出了 TCP 包头的最小大小，就是 20 字节。

UDP 不提供可靠性，不提供流量控制，传输速度快，协议开销小。

UDP 包头结构：

源端口 16 位|目的端口 16 位|长度 16 位|校验和 16 位

UDP 的包小很多。确实如此。因为 UDP 是非可靠连接。设计初衷就是尽可能快的将数据包发送出去。所以 UDP 协议显得非常精简。

三、实验步骤与结果分析

编译基于 TCP 和 UDP 的客户端程序与服务端程序

TCP 客户端代码

tcpClient.cpp

```cpp
#include<WINSOCK2.H>
#include<STDIO.H>
#include<iostream>
#include<cstring>
using namespace std;
#pragma comment(lib, "ws2_32.lib")

int main()
{
    WORD sockVersion = MAKEWORD(2, 2);
    WSADATA data;
    if(WSAStartup(sockVersion, &data)!=0)
    {
        return 0;
    }
    while(true){
        SOCKET sclient = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if(sclient == INVALID_SOCKET)
        {
            printf("invalid socket!");
            return 0;
        }

        sockaddr_in serAddr;
        serAddr.sin_family = AF_INET;
        serAddr.sin_port = htons(8888);
        serAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
        if(connect(sclient, (sockaddr *)&serAddr, sizeof(serAddr)) == SOCKET_ERROR)
        {  //连接失败
            printf("connect error !");
            closesocket(sclient);
            return 0;
        }

        string data;
        cin>>data;
        const char * sendData;
        sendData = data.c_str();   //string转const char*
        //char * sendData = "你好，TCP服务端，我是客户端\n";
        send(sclient, sendData, strlen(sendData), 0);
        //send()用来将数据由指定的socket传给对方主机
        //int send(int s, const void * msg, int len, unsigned int flags)
        //s为已建立好连接的socket，msg指向数据内容，len则为数据长度，参数flags一般设0
        //成功则返回实际传送出去的字符数，失败返回-1，错误原因存于error

        char recData[255];
        int ret = recv(sclient, recData, 255, 0);
        if(ret>0){
            recData[ret] = 0x00;
            printf(recData);
        }
        closesocket(sclient);
    }


    WSACleanup();
    return 0;

}
```

TCP 服务器端代码

tcpClient.cpp [*] tcpSever.cpp

```cpp
#include <stdio.h>
#include <winsock2.h>
#pragma comment(lib,"ws2_32.lib")
int main(int argc, char* argv[])
{
    //初始化WSA
    WORD sockVersion = MAKEWORD(2,2);
    WSADATA wsaData;
    if(WSAStartup(sockVersion, &wsaData)!=0)
    {
        return 0;
    }

    //创建套接字
    SOCKET slisten = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(slisten == INVALID_SOCKET)
    {
        printf("socket error !");
        return 0;
    }

    //绑定IP和端口
    sockaddr_in sin;
    sin.sin_family = AF_INET;
    sin.sin_port = htons(8888);
    sin.sin_addr.S_un.S_addr = INADDR_ANY;
    if(bind(slisten, (LPSOCKADDR)&sin, sizeof(sin)) == SOCKET_ERROR)
    {
        printf("bind error !");
    }

    //开始监听
    if(listen(slisten, 5) == SOCKET_ERROR)
    {
        printf("listen error !");
        return 0;
    }

    //循环接收数据
    SOCKET sClient;
    sockaddr_in remoteAddr;
    int nAddrlen = sizeof(remoteAddr);
    char revData[255];
    while (true)
    {
        printf("等待连接...\n");
        sClient = accept(slisten, (SOCKADDR *)&remoteAddr, &nAddrlen);
        if(sClient == INVALID_SOCKET)
        {
            printf("accept error !");
            continue;
        }
        printf("接受到一个连接: %s \r\n", inet_ntoa(remoteAddr.sin_addr));

        //接收数据
        int ret = recv(sClient, revData, 255, 0);
        if(ret > 0)
        {
            revData[ret] = 0x00;
            printf(revData);
        }

        //发送数据
        const char * sendData = "你好, TCP客户端! \n";
        send(sClient, sendData, strlen(sendData), 0);
        closesocket(sClient);
    }

    closesocket(slisten);
    WSACleanup();
    return 0;
}
```

UDP 客户端代码

tcpClient.cpp | [*] tcpSever.cpp | udpClient.cpp | [*] udpSever.cpp

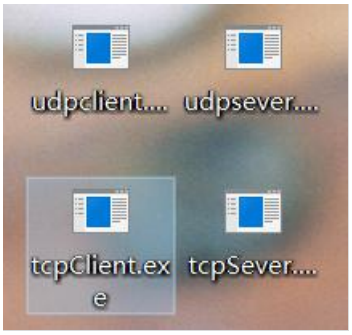
```cpp
#include <winsock2.h>
#include<STDIO.H>
#include<iostream>
#include<cstring>
using namespace std;
#pragma comment(lib,"ws2_32.lib")

int main(int argc, char* argv[])
{
    WORD socketVersion = MAKEWORD(2,2);
    WSADATA wsaData;
    if(WSAStartup(socketVersion, &wsaData) != 0)
    {
        return 0;
    }
    SOCKET sclient = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

    sockaddr_in sin;
    sin.sin_family = AF_INET;
    sin.sin_port = htons(8888);
    sin.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
    int len = sizeof(sin);

    string data;
    cin>>data;
    const char * sendData;
    sendData = data.c_str();   //string转const char*
    sendto(sclient, sendData, strlen(sendData), 0, (sockaddr *)&sin,len);

    char recvData[255];
    int ret = recvfrom(sclient, recvData, 255, 0, (sockaddr *)&sin,&len);
    if(ret > 0)
    {
        recvData[ret] = 0x00;
        printf(recvData);
    }

    closesocket(sclient);
    WSACleanup();
    return 0;
}
```

UDP 服务器端代码

tcpClient.cpp | [*] tcpSever.cpp | udpClient.cpp | [*] udpSever.cpp

```cpp
#include <stdio.h>
#include <winsock2.h>
#pragma comment(lib,"ws2_32.lib")
int main(int argc, char* argv[])
{
    WSADATA wsaData;
    WORD sockVersion = MAKEWORD(2,2);
    if(WSAStartup(sockVersion, &wsaData) != 0)
    {
        return 0;
    }

    SOCKET serSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if(serSocket == INVALID_SOCKET)
    {
        printf("socket error !");
        return 0;
    }

    sockaddr_in serAddr;
    serAddr.sin_family = AF_INET;
    serAddr.sin_port = htons(8888);
    serAddr.sin_addr.S_un.S_addr = INADDR_ANY;
    if(bind(serSocket, (sockaddr *)&serAddr, sizeof(serAddr)) ==SOCKET_ERROR)
    {
        printf("bind error !");
        closesocket(serSocket);
        return 0;
    }
    sockaddr_in remoteAddr;
    int nAddrLen = sizeof(remoteAddr);
    while (true)
    {
        char recvData[255];
        int ret = recvfrom(serSocket, recvData, 255, 0, (sockaddr*)&remoteAddr, &nAddrLen);
        if (ret > 0)
        {
            recvData[ret] = 0x00;
            printf("接受到一个连接: %s \r\n",inet_ntoa(remoteAddr.sin_addr));
            printf(recvData);
        }

        const char * sendData = "一个来自服务端的UDP数据包\n";
        sendto(serSocket, sendData,strlen(sendData), 0, (sockaddr *)&remoteAddr, nAddrLen);

    }
    closesocket(serSocket);
    WSACleanup();
    return 0;
}
```

分别编译运行生成.exe 可执行文件

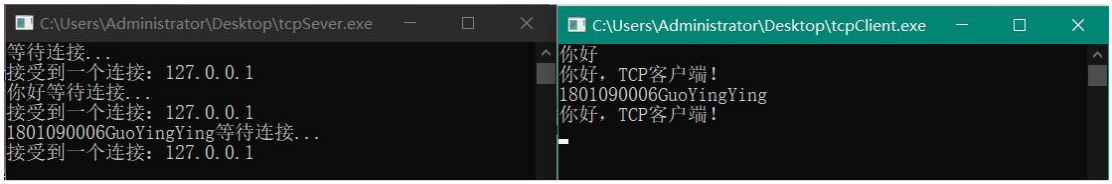


利用 Wireshark 分别抓取 TCP 和 UDP 通信的数据包

以管理员身份打开 Wireshark,选取点击 Npcap Loopback Adapter 开始抓包

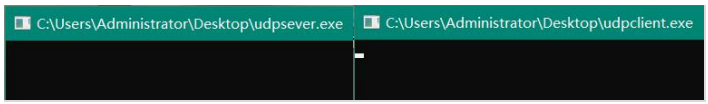
先运行 TCP 服务器端程序，后运行 TCP 客户端程序

利用客户端向服务端发送你自己的学号加中文姓名拼音

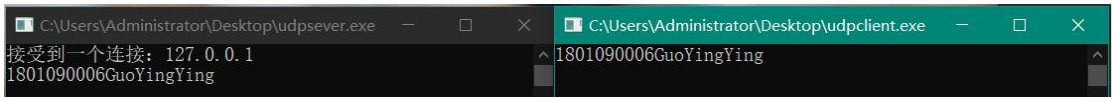


结束并保存抓包文件

先运行 UDP 服务器端程序，后运行 UDP 客户端程序



利用客户端向服务端发送你自己的学号加中文姓名拼音

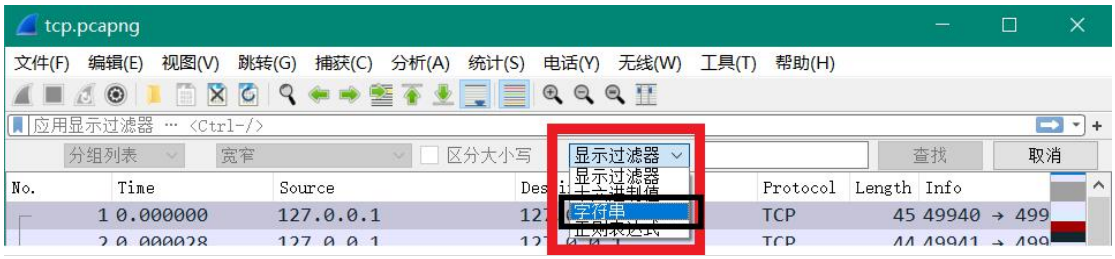


结束并保存抓包文件

筛选数据包得到所需信息（以 TCP 数据包为例）

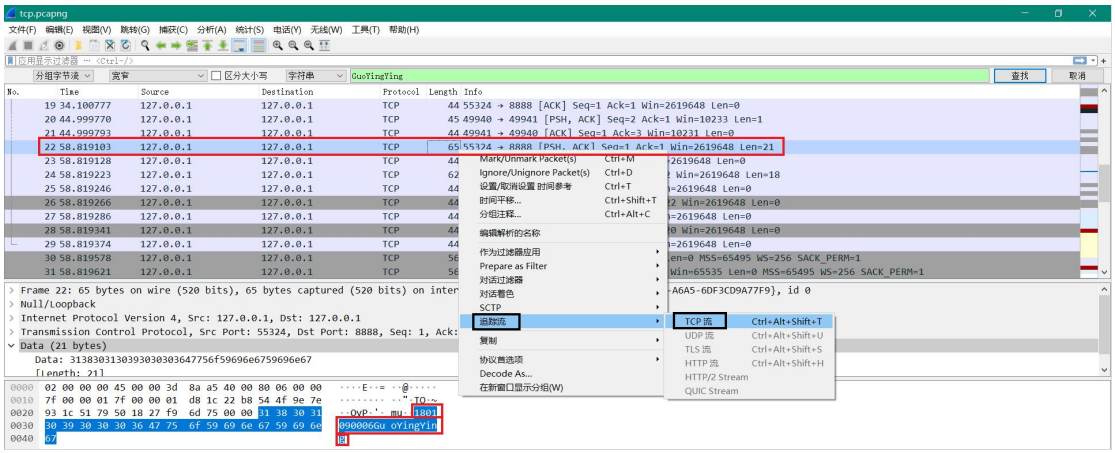
在工具栏找到“查找一个分组”的选项，点击

点击显示过滤器，选择字符串

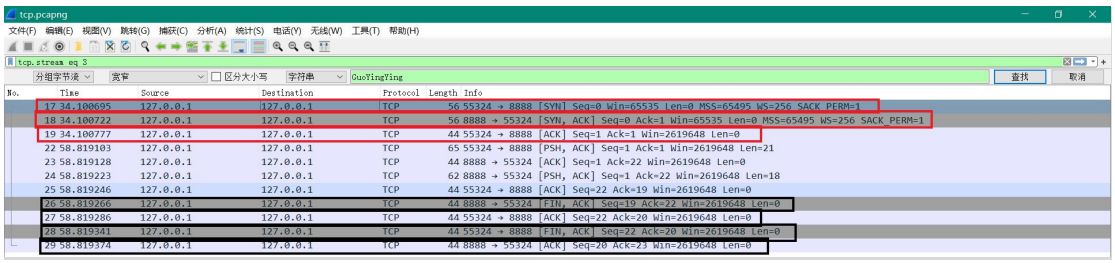


在字符串后的框内写入筛选字符串（如:GuoYingYing）；点击分组列表，选择分组字节流

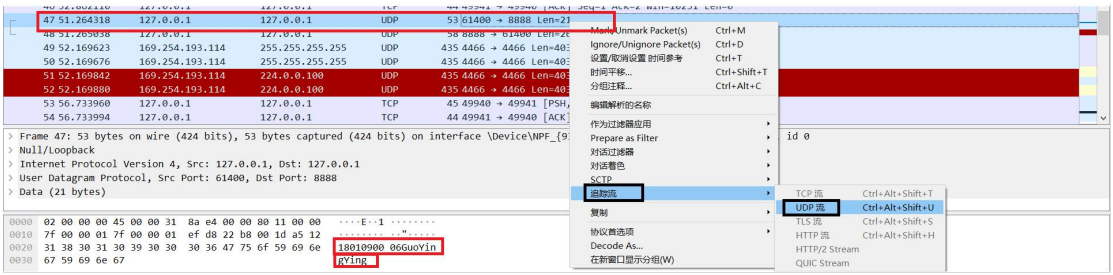
找到有筛选信息的包，右键点击追踪流，点击 TCP 流



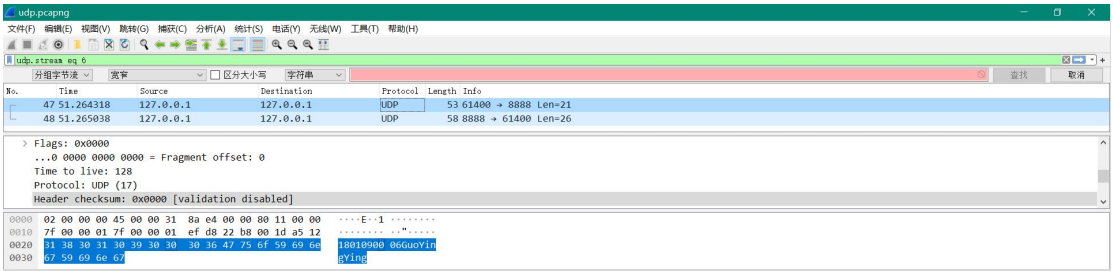
得到筛选后的 TCP 数据包（红色为三次握手，黑色为四次挥手）



同理，找到有筛选信息的包，右键点击追踪流，点击 UDP 流



得到筛选后的 UDP 数据包

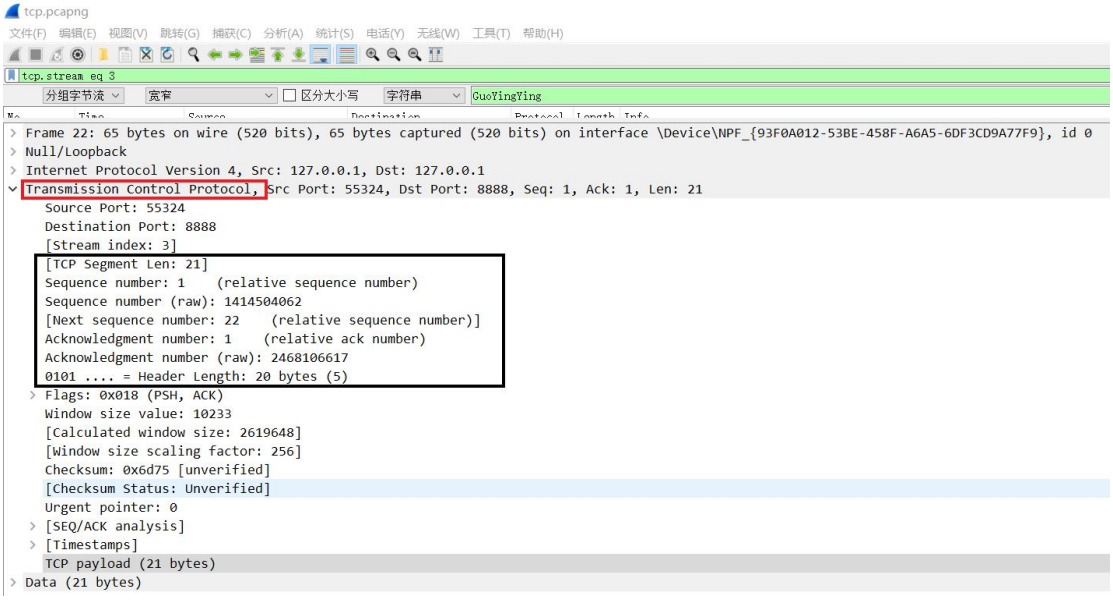


比较 TCP 与 UDP 数据包的异同

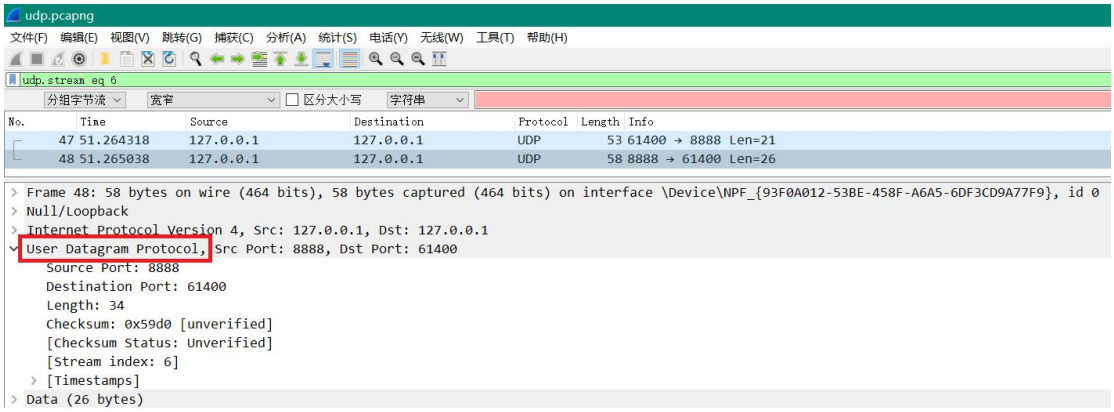
1. 在数据包数量上，TCP 明显比 UDP 多，且 TCP 中的三次握手连接和四次挥手断开，UDP 中都没有

2. IP 数据包格式，

TCP 包头的最小大小，就是 **20 字节**。



UDP 的包小很多。



四、实验收获与总结

通过对 TCP 四次挥手的学习，进一步了解 TCP 的服务器端和客户端的传送信息原理。比较 TCP 和 UDP 的数据包，UDP 比 TCP 简单，但 UDP 不可靠，丢包率高，适用于 IP 电话、实施电话会议这种交互同信；而 TCP 实现双向传输确认，更可靠。