

**学生实验报告**

实验课名称：《计算机网络》课程实验

专业名称：人工智能

班级：2022240401

学号：2022905226

学生姓名：

教师姓名：

2024年 11月 1 日

[实验一 IP数据报 3](#_Toc181801462)

[1.1 实验目的 3](#_Toc181801463)

[1.2 实验要求 3](#_Toc181801464)

[1.3 实验原理 3](#_Toc181801465)

[1.4 实验方法 4](#_Toc181801466)

[1.5 实验步骤及结果分析 4](#_Toc181801467)

[1.5.1 TCP报文抓包解析 4](#_Toc181801468)

[1.5.1.1 TCP报文介绍 4](#_Toc181801469)

[1.5.1.2 实验步骤和结果 4](#_Toc181801470)

[1.5.2 UDP报文抓包解析 5](#_Toc181801471)

[1.5.2.1 UDP报文介绍 5](#_Toc181801472)

[1.5.2.2实验步骤和结果 5](#_Toc181801473)

[1.5.3 C++源程序实现抓包解包 6](#_Toc181801474)

[1.6 实验总结 9](#_Toc181801475)

[实验二 TCP三次握手和四次挥手 9](#_Toc181801476)

[2.1 实验目的 9](#_Toc181801477)

[2.2 实验要求 10](#_Toc181801478)

[2.3 实验原理 10](#_Toc181801479)

[2.4 实验方法 12](#_Toc181801480)

[2.5 实验步骤 12](#_Toc181801481)

[2.5.1 三次握手 12](#_Toc181801482)

[2.5.2 四次挥手 14](#_Toc181801483)

[2.5.3 C++程序构建客户端和抓包程序观察三次握手和四次挥手 16](#_Toc181801484)

[2.6 实验总结 19](#_Toc181801485)

# 实验一 IP数据报

## 1.1 实验目的

掌握IP协议的基本架构和组成，能够分析IP数据报内部结构，理解各个数据项的内涵。

## 1.2 实验要求

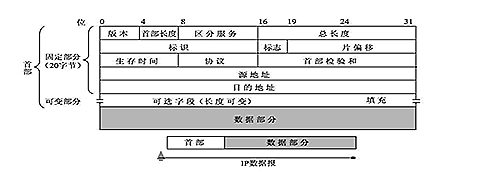
1. 理解IP数据报的概念和格式，要求记录分析2种不同类型数据内容的IP数据报：TCP、UDP。
2. 写出实验过程中出现的问题、解决方法和心得体会。
3. 使用Wireshark抓包程序抓取TCP、UDP数据报；另外编写程序抓取IP数据报。分析抓取到的数据报。

## 1.3 实验原理

在 TCP/IP 协议中，使用 IP 协议传输数据的包被称为 IP 数据包，每个数据包都包含 IP 协议规定的内容。 IP 数据报文由**首部（称为报头）**和**数据**两部分组成。 首部构成：

* 前一部分：固定长度，共20字节，称为基本首部，所有IP数据报共有。
* 可选字段：长度可变，不固定。

协议概览图：



**图1：协议概览图**

TCP/IP 协议定义了一个在网络层传输的包，称为**IP数据报**。这是一个与硬件无关的虚拟数据包，它包含了一组特定的格式化字段，这些字段定义了数据如何通过网络传输。

IP数据报具体格式：



**图2：IP数据报具体格式**

下面对各个字段进行详细介绍：

## 1.4 实验方法

使用两种方式抓取IP数据报：

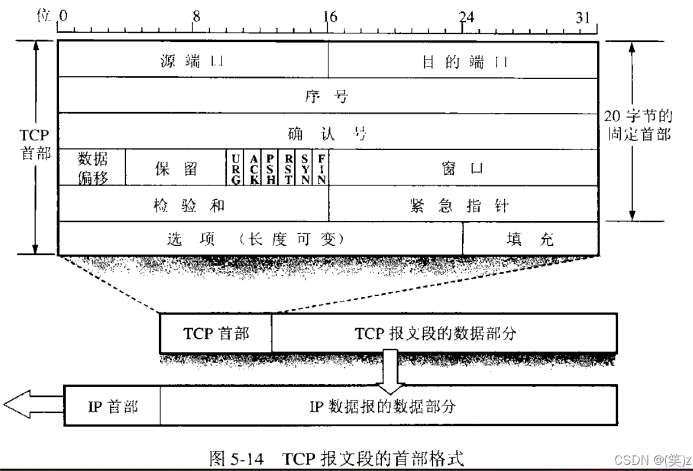
1. 使用Wireshark抓包程序抓取IP数据报
2. 编写程序抓取IP数据报

## 1.5 实验步骤及结果分析

### 1.5.1 TCP报文抓包解析

#### 1.5.1.1 TCP报文介绍

TCP、UDP报文都是传输层的。 TCP报文格式如下：



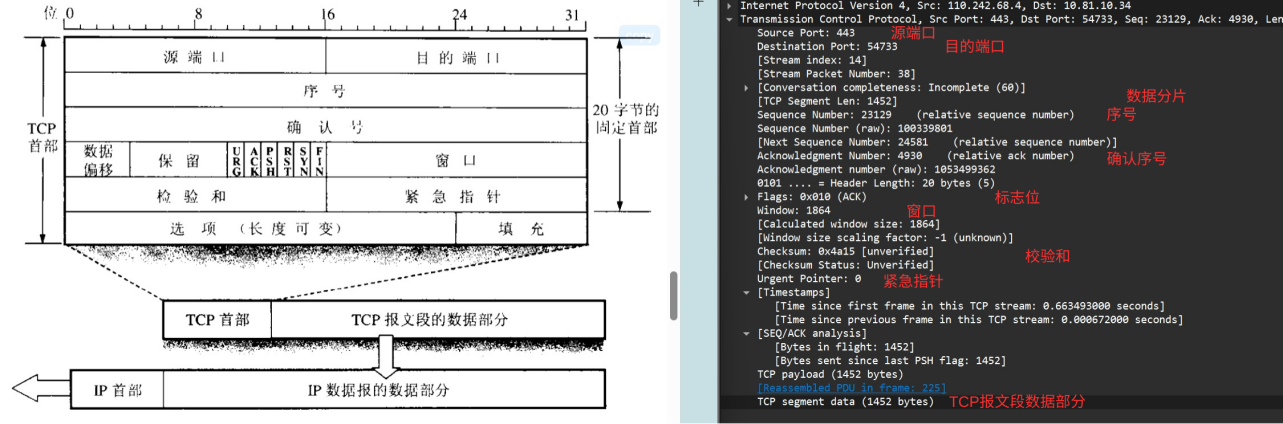
**图10：TCP报文格式**

#### 1.5.1.2 实验步骤和结果

ping www.baidu.com得其IP地址为110.242.68.3。接着在wireshark中设置过滤条件：

ip.addr == 110.242.68.3

捕捉到baidu网站的TCP数据报文（分片）如下：

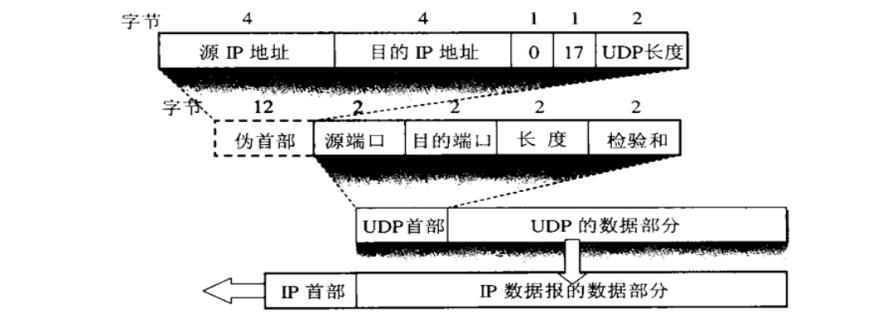


**图11：TCP报文数据分析**

### 1.5.2 UDP报文抓包解析

#### 1.5.2.1 UDP报文介绍

UDP报文格式如下：



**图12：UDP报文格式**

UDP伪首部（Pseudo Header）是一个虚拟的首部，**不是实际UDP报文的一部分**，仅用于**校验和的计算**。它的主要目的是为了验证数据是否正确到达目标主机的目标协议。伪首部只用于校验和的计算，不会随数据包一起发送。

UDP首部固定长度为8字节。是无连接的传输协议，不保证可靠传输，无流量控制，相比TCP更轻量级，开销更小。

#### 1.5.2.2实验步骤和结果

抓取UDP包需要以下几个步骤：

采用DNS查询（端口53）。

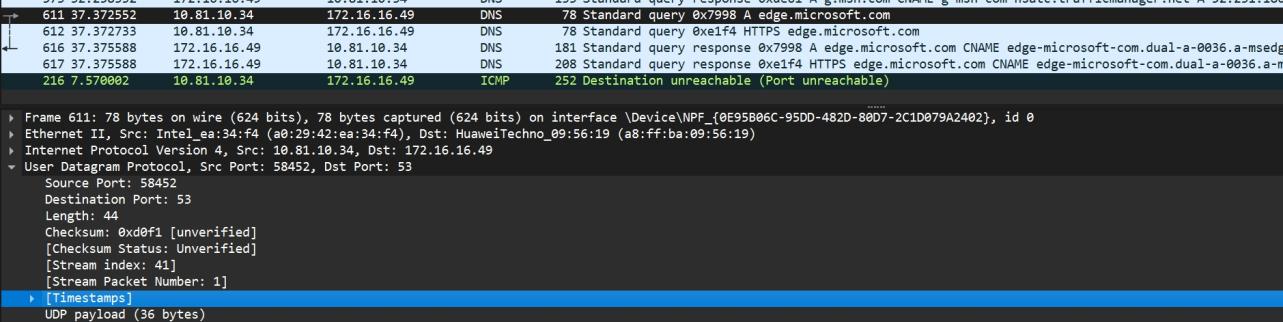
设置Wireshark过滤条件：

* udp // 只显示UDP协议的数据包  
  udp.port == 53 // 只显示DNS相关的UDP包  
  udp.length > 100 // 过滤特定长度的UDP包

生成UDP流量，使用nslookup命令进行DNS查询：

* nslookup www.baidu.com

DNS查询分析：



**图13：通过DNS查询捕获UDP包分析**

从抓包结果可以看到：

源端口：随机分配的本地端口。 目的端口：53（DNS服务端口）。

长度：包含8字节UDP首部和DNS查询数据 。校验和：确保数据完整性 。

数据部分：包含DNS查询的具体内容。

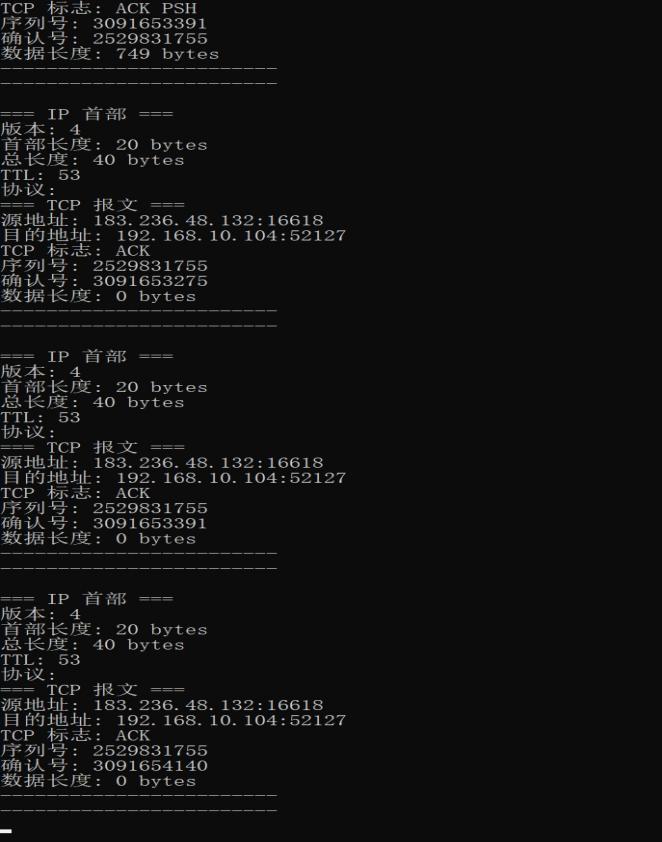
### 1.5.3 C++源程序实现抓包解包

数据报文格式定义：

// IP 头部结构  
typedef struct ip\_hdr  
{  
 unsigned char ip\_header\_len : 4; // 首部长度  
 unsigned char ip\_version : 4; // 版本  
 unsigned char ip\_tos; // 服务类型  
 unsigned short ip\_total\_length; // 总长度  
 unsigned short ip\_id; // 标识  
 unsigned short ip\_frag\_offset; // 标志(3 bits) + 片偏移(13 bits)  
 unsigned char ip\_ttl; // 生存时间  
 unsigned char ip\_protocol; // 协议类型  
 unsigned short ip\_checksum; // 首部校验和  
 unsigned int ip\_srcaddr; // 源IP地址  
 unsigned int ip\_destaddr; // 目的IP地址  
} IP\_HDR, \*PIP\_HDR;  
// TCP 头部结构  
typedef struct tcp\_hdr  
{  
 unsigned short source\_port; // 源端口  
 unsigned short dest\_port; // 目的端口  
 unsigned int sequence; // 序号  
 unsigned int acknowledge; // 确认号  
 unsigned char ns : 1; // ECN-nonce标志  
 unsigned char reserved\_part1 : 3; // 保留  
 unsigned char data\_offset : 4; // 数据偏移  
 unsigned char fin : 1; // FIN标志  
 unsigned char syn : 1; // SYN标志  
 unsigned char rst : 1; // RST标志  
 unsigned char psh : 1; // PSH标志  
 unsigned char ack : 1; // ACK标志  
 unsigned char urg : 1; // URG标志  
 unsigned char ecn : 1; // ECN标志  
 unsigned char cwr : 1; // CWR标志  
 unsigned short window; // 窗口大小  
 unsigned short checksum; // 校验和  
 unsigned short urgent\_pointer; // 紧急指针  
} TCP\_HDR, \*PTCP\_HDR;  
// UDP 头部结构  
typedef struct udp\_hdr  
{  
 unsigned short source\_port; // 源端口  
 unsigned short dest\_port; // 目的端口  
 unsigned short length; // UDP长度  
 unsigned short checksum; // 校验和  
} UDP\_HDR, \*PUDP\_HDR;

数据报文处理：   
 // 获取源IP和目的IP  
 struct in\_addr source, dest;  
 source.s\_addr = iphdr->ip\_srcaddr;  
 dest.s\_addr = iphdr->ip\_destaddr;  
 // 打印IP头部信息  
 printf("\n=== IP Header ===\n");  
 printf("Version: %d\n", iphdr->ip\_version);  
 printf("Header Length: %d bytes\n", iphdr->ip\_header\_len \* 4);  
 printf("Total Length: %d bytes\n", ntohs(iphdr->ip\_total\_length));  
 printf("TTL: %d\n", iphdr->ip\_ttl);  
 printf("Protocol: ");  
 // 根据协议类型处理不同的数据包  
 switch (iphdr->ip\_protocol)  
 {  
 case IPPROTO\_TCP:  
 {  
 printf("\n=== TCP Packet ===\n");  
 TCP\_HDR \*tcphdr = (TCP\_HDR \*)(buffer + iphdr->ip\_header\_len \* 4);  
 unsigned short src\_port = ntohs(tcphdr->source\_port);  
 unsigned short dst\_port = ntohs(tcphdr->dest\_port);  
 // 打印源地址和目标地址  
 printf("Source: %s:%d\n", inet\_ntoa(source), src\_port);  
 printf("Destination: %s:%d\n", inet\_ntoa(dest), dst\_port);

抓取和解析效果：



**图14：C++抓取解包TCP报文**



**图15：C++抓取解包UDP报文**



**图16：C++抓取解包ICMP报文**

该程序没有设置过滤条件，所以会抓取所有TCP、UDP、ICMP格式的数据报文，需要设置过滤条件时，在ProcessIPPacket函数中更改代码设置即可。

## 1.6 实验总结

通过本次实验，我掌握了IP数据报的基本架构和组成，能够分析IP数据报内部结构，理解各个数据项的内涵。还了解了TCP、UDP报文的基本格式和特点，以及它们在网络中的应用。

实验过程中出现的问题和解决方法：

1. 在抓取TCP数据报文时，由于数据报文长度较长，导致Wireshark抓包程序无法完整抓取数据报文。我通过设置过滤条件，只抓取特定IP地址的数据报文，解决了这个问题。
2. 抓取UDP数据报文时，数据报文长度较短，Wireshark抓包程序也无法完整抓取数据报文。通过设置过滤条件，只抓取特定IP地址的数据报文，解决问题。

实验心得体会：

1. 通过本次实验，我更加深入地理解了IP数据报的基本架构和组成，以及TCP、UDP报文的基本格式和特点。
2. 通过实验过程中出现的问题和解决方法，我学会了如何设置过滤条件，只抓取特定IP地址的数据报文。
3. 理解了wireshark的基本工作原理，以及如何更高效地抓取报文。

# 实验二 TCP三次握手和四次挥手

## 2.1 实验目的

1. 掌握TCP三次握手在网络中的基本实现过程和实现原理。
2. 理解TCP协议基本架构、分析TCP报文结构、了解TCP的确认机制。

## 2.2 实验要求

1. 根据实验要求仿真实现TCP三次握手过程
2. 捕获并分析TCP连接建立过程中发送和接收的数据报文
3. 写出实验过程中出现的问题、解决方法和心得体会

## 2.3 实验原理

所谓的“TCP三次握手”： 为了对每次发送的数据量进行跟踪与协商，确保数据段的发送和接收同步，根据所接收到的数据量而确认数据发送、接收完毕后何时撤消联系，并建立虚连接。过程如下：

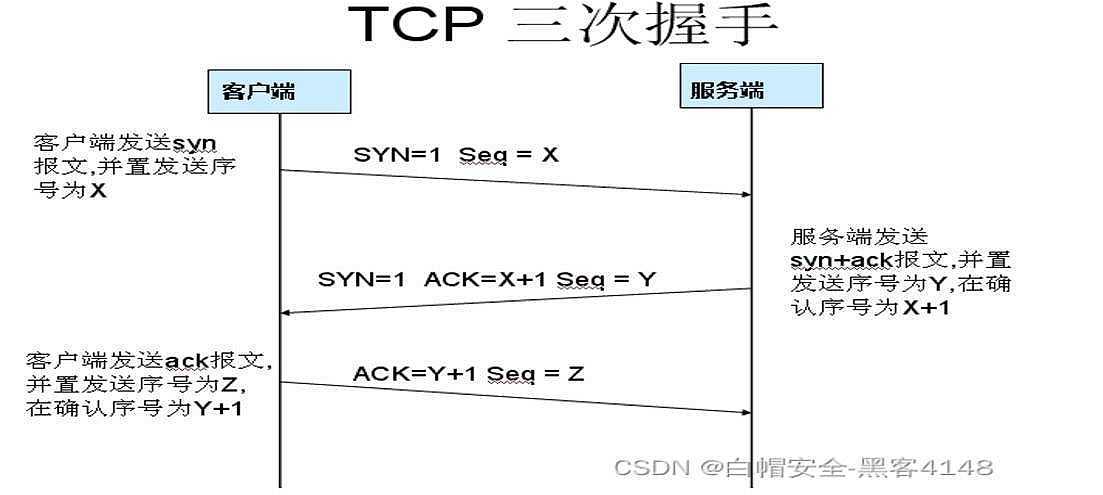
* 第一次握手：建立连接时，客户端发送syn包（seq=j）到服务器，并进入SYN\_SENT状态，等待服务器确认；SYN：同步序列编号（Synchronize Sequence Numbers）。
* 第二次握手：服务器收到syn包，必须确认客户端的SYN（ack=j+1），同时自己也发送一个SYN包（seq=k），即SYN+ACK包，此时服务器进入SYN\_RECV状态。
* 第三次握手：客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK(ack=k+1)，此包发送完毕，客户端和服务器进入ESTABLISHED（TCP连接成功）状态，完成三次握手。

所谓的“TCP四次挥手”：

* 数据传输完毕后，数据传输首先结束的一端（比如客户端）向另一端（比如服务器）发送一个FIN包。Flags位的FIN=1，表示这是一个结束请求。
* 另一端收到FIN包后，发送一个ACK包，Flags位的ACK=1，表示确认收到结束请求。
* 服务端也结束数据传输后，向客户端发送一个FIN包，Flags位的FIN=1，表示这是一个结束请求。
* 客户端收到FIN包后，发送一个ACK包，Flags位的ACK=1，表示确认收到结束请求。

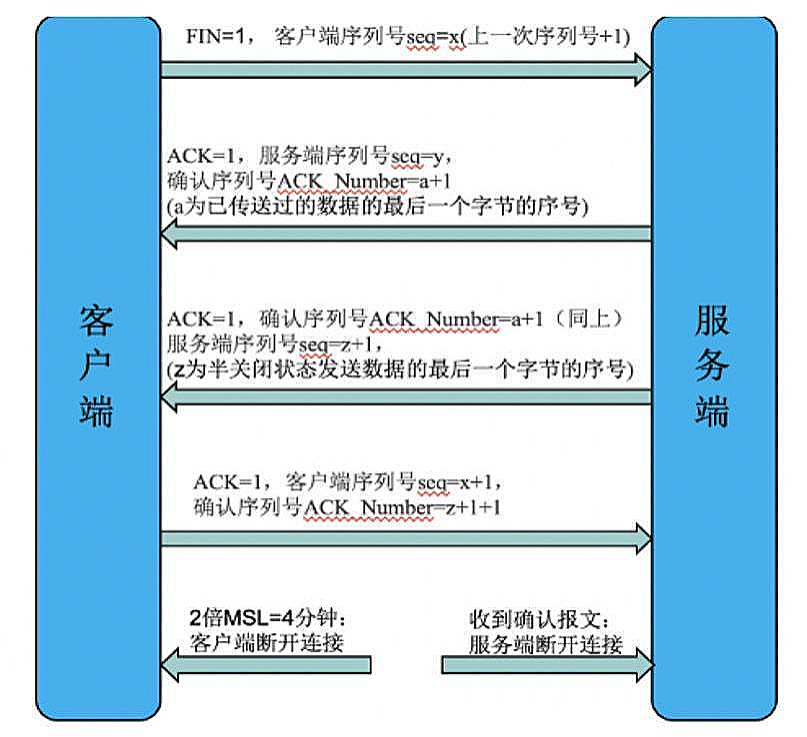
过程示意图：

三次握手：



**图17：TCP三次握手过程**

四次挥手：



**图18：TCP四次挥手过程**

三次握手步骤：

1. 客户端发送一个SYN包。
   * **SYN = 1，ACK = 0，seq = x；** seq是客户端的初始序列号，ACK = 0表示不需要确认；SYN = 1表示这是一个同步请求。
2. 服务器收到SYN包后，发送一个SYN+ACK包。
   * **SYN = 1，ACK = 1，seq = y，ack = x + 1；** seq是服务器的初始序列号，ACK = 1表示需要确认；ack = x + 1表示确认收到客户端的SYN包。
3. 客户端收到SYN+ACK包后，发送一个ACK包。
   * **ACK = 1，seq = x + 1，ack = y + 1；** ACK = 1表示确认收到服务器的SYN+ACK包。

三次握手完成，客户端和服务器进入ESTABLISHED状态，连接建立。

四次挥手：

1. 数据传输完毕后，数据传输首先结束的一端（比如客户端）向另一端（比如服务器）**发送一个FIN包**。Flags位的FIN=1，表示这是一个结束请求。
2. 另一端收到FIN包后，**发送一个ACK包**，Flags位的ACK=1，表示确认收到结束请求。
3. 服务端也结束数据传输后，**向客户端发送一个FIN包**，Flags位的FIN=1，表示这是一个结束请求。
4. 客户端收到FIN包后，**发送一个ACK包**，Flags位的ACK=1，表示确认收到结束请求。

四次挥手完成，客户端和服务器进入CLOSED状态，连接关闭。

## 2.4 实验方法

利用网络协议分析工具，实现TCP协议在网络中的三次握手。分析并理解TCP报文结构。利用C++程序捕获TCP三次握手和四次挥手过程，并分析数据报。

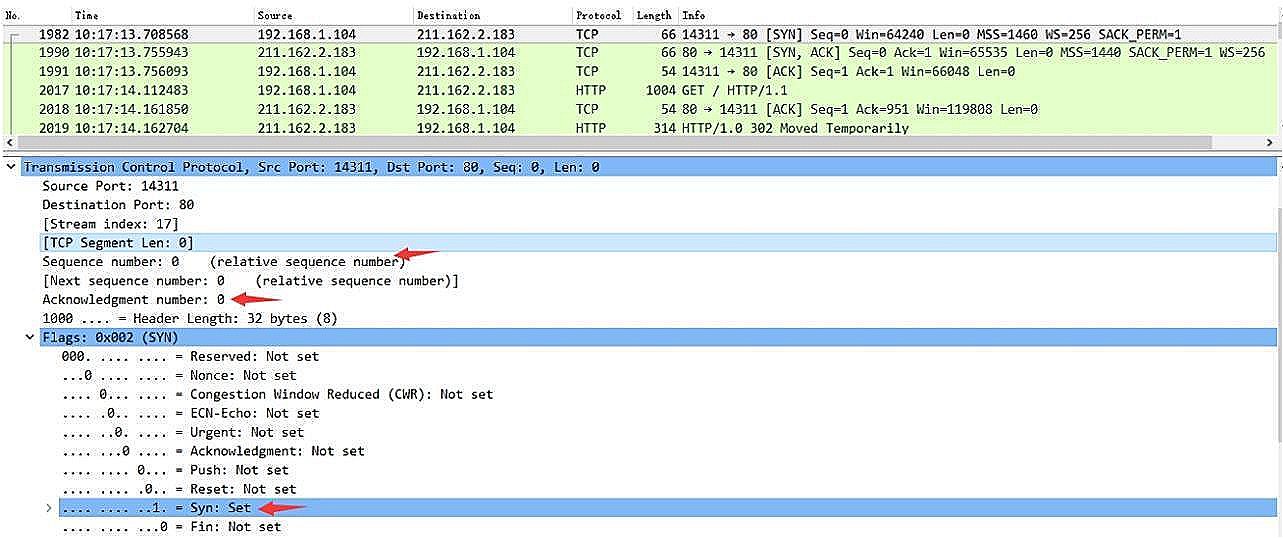
## 2.5 实验步骤

### 2.5.1 三次握手

从百度抓包，并用wireshark分析。

抓包过程就是先获取百度服务器IP地址（ping），然后设置过滤条件为百度服务器IP地址，并设置捕获时间，开始抓包。在一开始建立连接时，会进行三次握手。

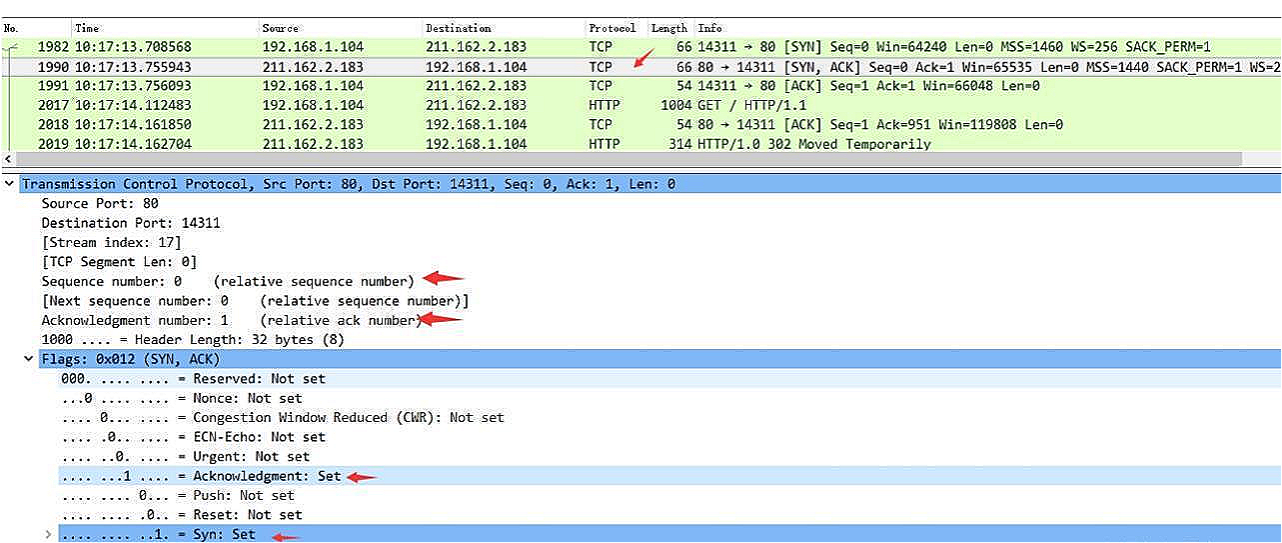
第一次握手：



**图19：第一次握手**

第一次握手，客户端向服务器发送SYN包，seq=x，此时seq = 0，表示客户端的初始序列号为0。SYN = 1，表示这是一个同步请求。ACK = 0，表示不需要确认。

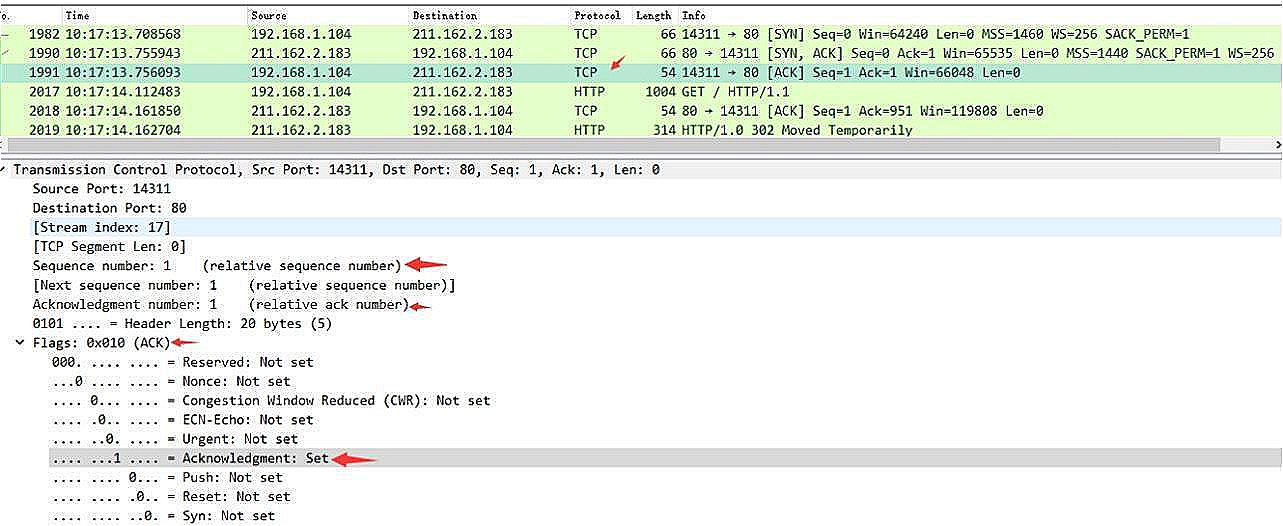
第二次握手：



**图20：第二次握手**

第二次握手，服务器收到SYN包，发送SYN+ACK包，seq=y，ack=x+1，此时seq = 0，表示服务器的初始序列号为0。ack = x + 1 = 1，表示确认收到客户端的SYN包。SYN = 1，表示这是一个同步请求。ACK = 1，表示需要确认。

第三次握手：



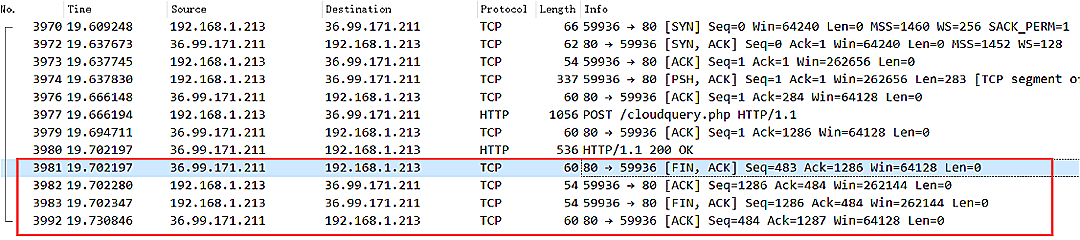
**图21：第三次握手**

第三次握手，客户端收到SYN+ACK包，发送ACK包，seq = x + 1 = 1，ack=y+1 = 1，此时seq = 1，ack = 1，表示客户端的初始序列号为1，确认收到服务器的SYN+ACK包。ACK = 1，表示需要确认。

### 2.5.2 四次挥手

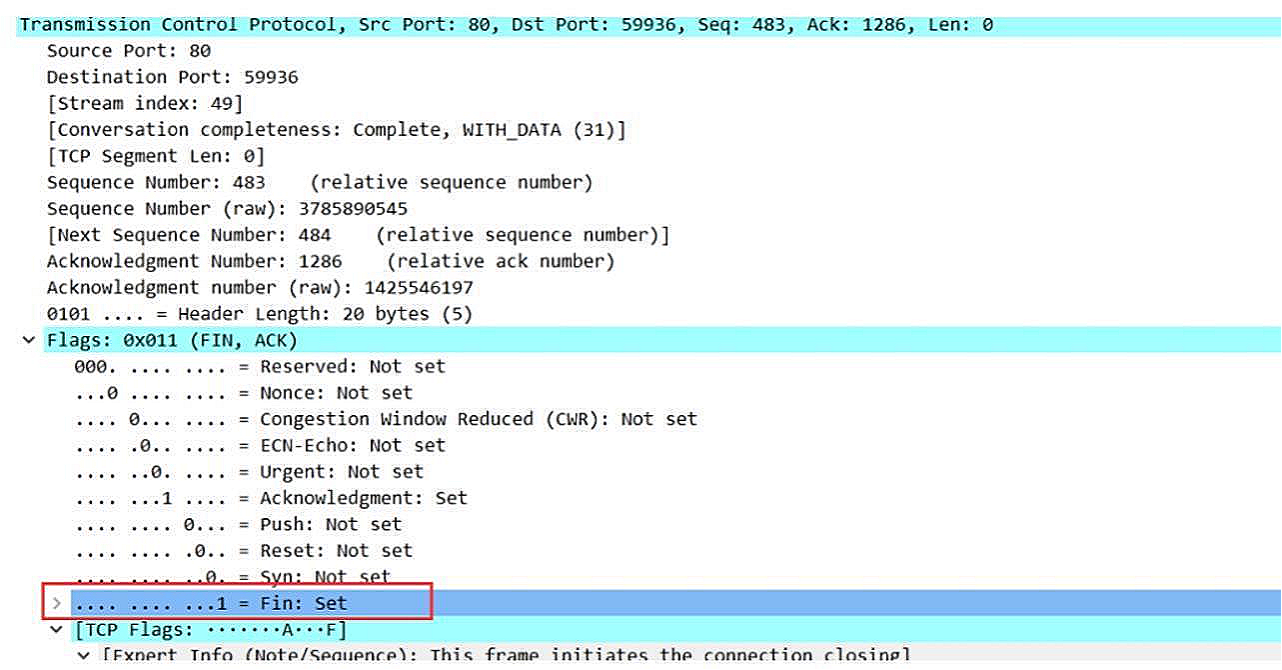
当数据传输结束了，客户端和服务器之间就开始断开连接了。断开连接需要经历四次挥手。

抓包四次挥手数据如下：



**图22：四次挥手数据包**

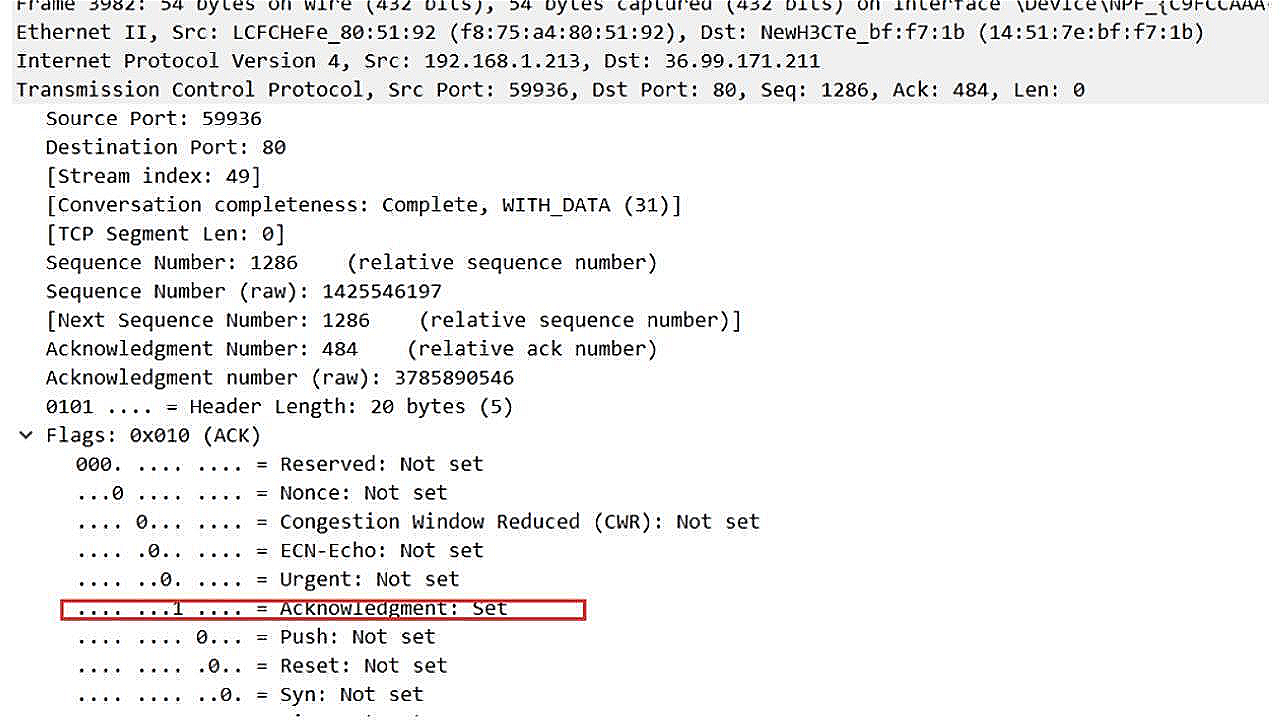
第一次挥手：



**图23：第一次挥手**

可以看到，第一次挥手是客户端向服务器发送FIN包，seq = x，此时seq = 1，表示客户端的初始序列号为1。FIN = 1，表示这是一个结束请求。ACK = 0，表示不需要确认。

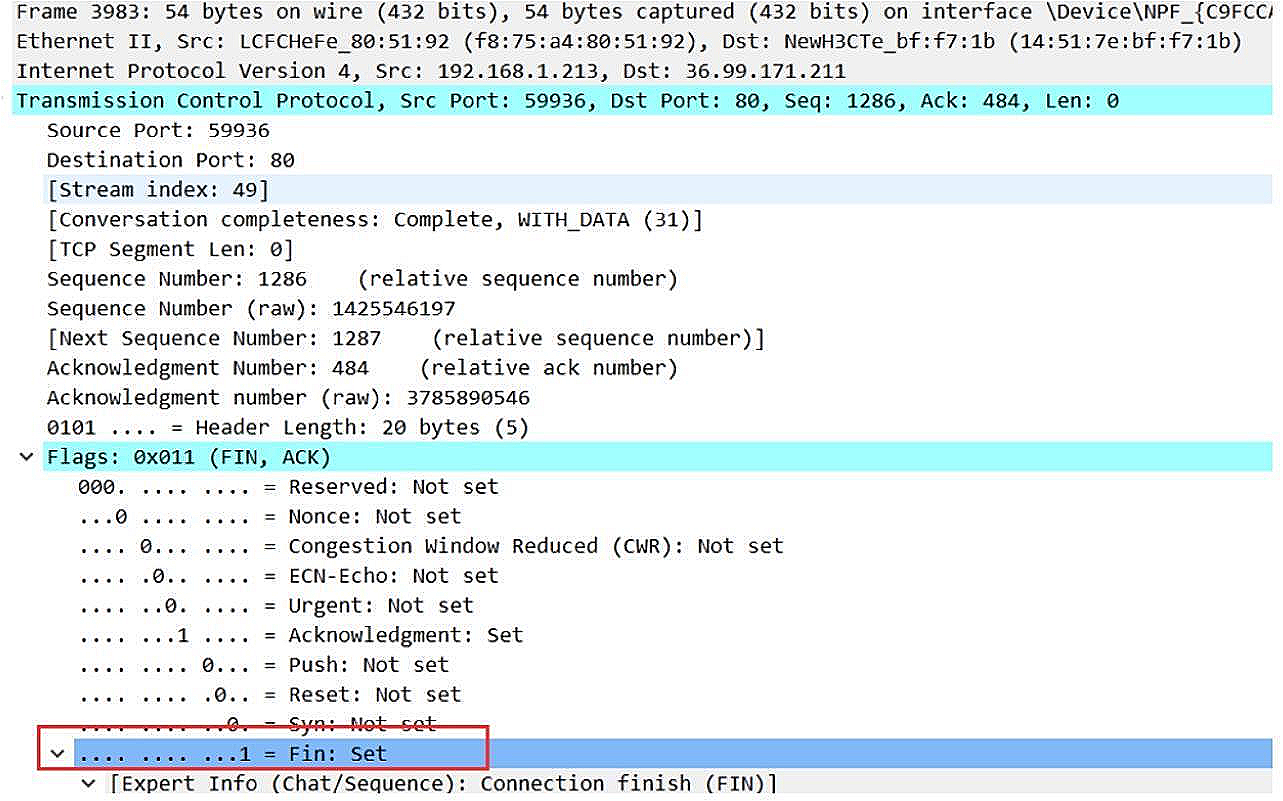
第二次挥手：



**图24：第二次挥手**

第二次挥手是服务器收到FIN包，发送ACK包，seq = y，ack = x + 1，此时seq = 0，ack = 2，表示服务器的初始序列号为0，确认收到客户端的FIN包。FIN = 1，表示这是一个结束请求。ACK = 1，表示需要确认。

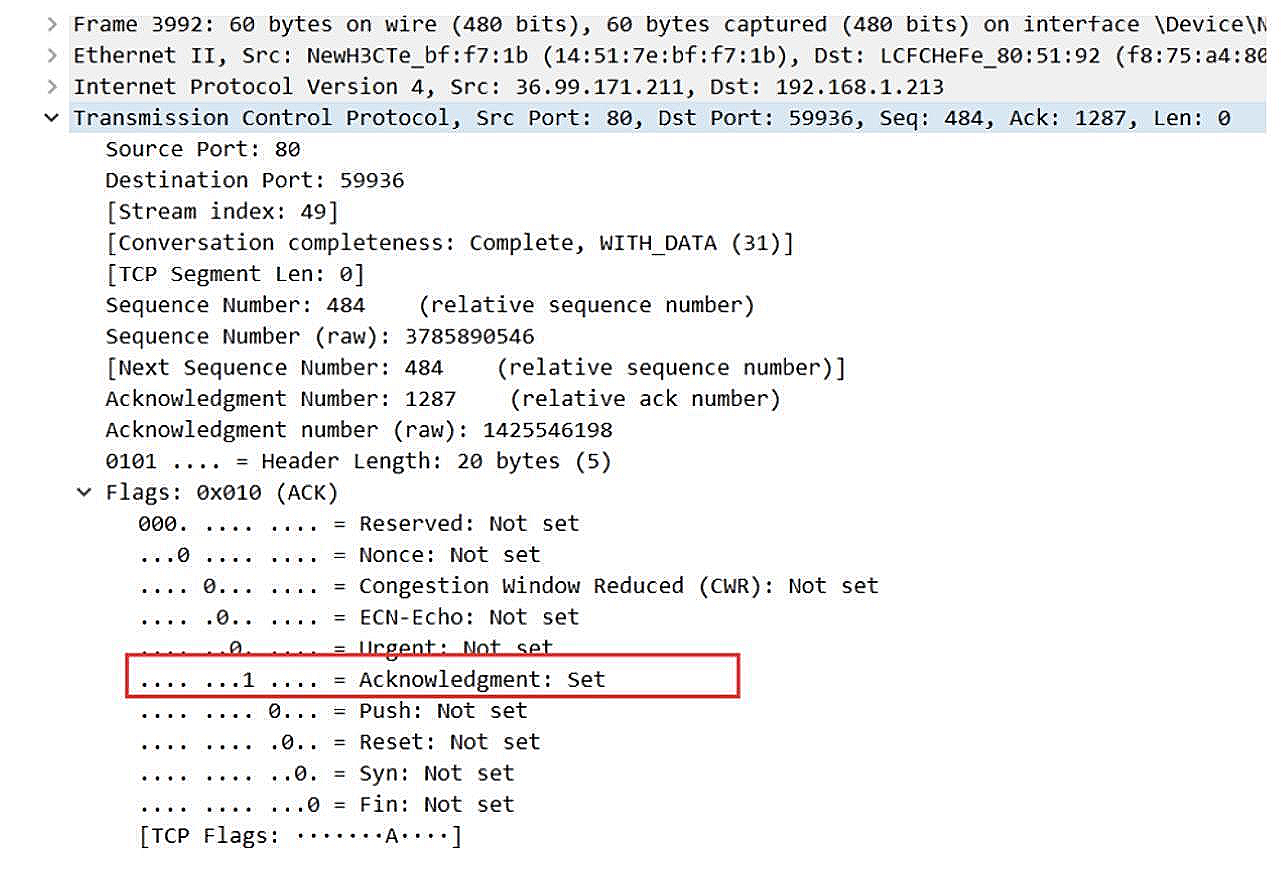
第三次挥手：



**图25：第三次挥手**

第三次挥手是服务端也结束数据传输后，向客户端发送FIN包，seq = y + 1，此时seq = 1，表示服务器的初始序列号为1。FIN = 1，表示这是一个结束请求。ACK = 0，表示不需要确认。

第四次挥手：



**图26：第四次挥手**

第四次挥手是客户端收到FIN包，发送ACK包，seq = x + 1 = 2，ack = y + 1 = 2，此时seq = 2，ack = 2，表示客户端的初始序列号为2，确认收到服务器的FIN包。ACK = 1，表示需要确认。

### 2.5.3 C++程序构建客户端和抓包程序观察三次握手和四次挥手

客户与服务端部分代码：

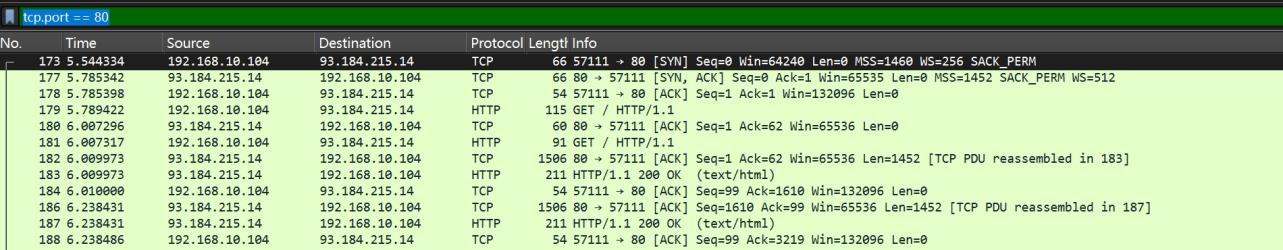
// 设置控制台代码页  
 system("chcp 65001");  
 // 初始化 WinSock  
 WSADATA wsaData;  
 if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)  
 {  
 printf("WSAStartup 失败\n");  
 return 1;  
 }  
 // 创建套接字  
 SOCKET sock = socket(AF\_INET, SOCK\_STREAM, 0);  
 if (sock == INVALID\_SOCKET)  
 {  
 printf("创建套接字失败\n");  
 WSACleanup();  
 return 1;  
 }  
 // 解析域名  
 struct hostent \*host = gethostbyname("example.com");  
 if (host == nullptr)  
 {  
 printf("解析主机名失败\n");  
 closesocket(sock);  
 WSACleanup();  
 return 1;  
 }  
 // 设置服务器地址  
 sockaddr\_in addr;  
 addr.sin\_family = AF\_INET;  
 addr.sin\_port = htons(80); // HTTP端口  
 addr.sin\_addr.s\_addr = \*(unsigned long \*)host->h\_addr;  
   
 printf("按回车键开始连接...\n");  
 getchar();  
 // 连接服务器  
 printf("正在连接服务器...\n");  
 if (connect(sock, (sockaddr \*)&addr, sizeof(addr)) == SOCKET\_ERROR)  
 {  
 printf("连接失败\n");  
 closesocket(sock);  
 WSACleanup();  
 return 1;  
 }

客户端的服务器端口为80

运行客户端之后，用wireshark设置过滤条件：

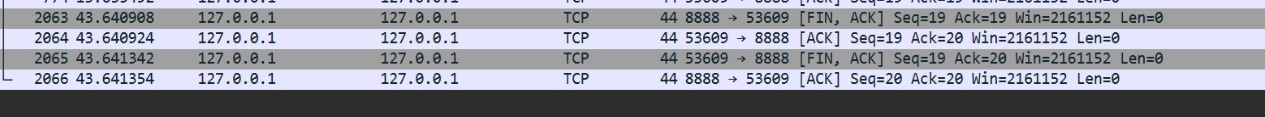
tcp.port == 80

可以看到三次握手过程：



**图27：C++客户端、服务端三次握手**

WinSock抓包程序设置过滤条件也是tcp.port == 80。



**图28：C++客户端、服务端四次挥手**

代码实现结果如下：



**图29：C++抓包抓取三次握手**

可以看到三次握手过程。

文本

中度可信度描述已自动生成

**图30：C++抓包抓取四次挥手**

## 2.6 实验总结

通过本次实验，我掌握了TCP三次握手和四次挥手的基本过程和实现原理，能够分析TCP报文结构，了解TCP的确认机制。

实验过程中出现的问题和解决方法：

1. 在抓取TCP连接过程时，一开始没有看到三次握手的过程。这是因为没有设置正确的端口过滤条件，后来通过设置tcp.port == 80成功捕获到了握手过程。
2. 客户端程序连接速度太快，不容易观察到完整的握手过程。通过在程序中添加用户交互（按回车继续）和延时，使得握手过程更容易观察。
3. 一开始使用HTTPS网站（443端口）测试时抓不到包，改用HTTP网站（80端口）后成功捕获到了TCP连接过程。

实验心得体会：

1. 通过实际抓包分析，深入理解了TCP三次握手的必要性：
   * 第一次握手确认客户端的发送情况
   * 第二次握手确认服务器的接收和发送情况
   * 第三次握手确认客户端的接收情况
2. 理解了TCP四次挥手的过程：
   * 为什么需要四次而不是三次

TCP四次挥手的目的是确保双方都能够正确地关闭连接,并确保数据的可靠传输。如果只进行三次挥手,可能会导致:

- 如果客户端发送的FIN包在网络中丢失,服务器无法接收到关闭请求,连接无法关闭。

- 如果服务器发送的FIN包在网络中丢失,客户端无法接收到服务器的关闭请求,连接无法关闭。

通过四次挥手,可以解决以上问题,保证双方都能够正确地关闭连接。

* + FIN和ACK标志位的作用，FIN标志开始发送结束报文，ACK标志是否收到确认。
  + TIME\_WAIT状态的重要性

1. 掌握了TCP报文结构的关键字段：
   * 序列号(SEQ)和确认号(ACK)的作用
   * 各种标志位(SYN, FIN, ACK等)的含义
   * 窗口大小的动态调整

这次实验加深了对TCP协议的理解，特别是连接的建立和断开过程，对今后进行网络编程和协议分析都很有帮助。