

**《计算机网络》课程实验报告**

**实验二**

****

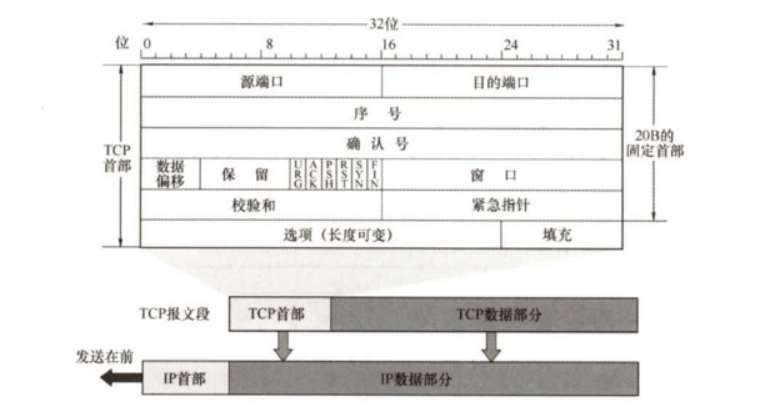
学 院 网络空间安全学院

专 业 信息安全

学 号 2112060

姓 名 孙蕗

1. **实验要求**
2. 搭建Web服务器（自由选择系统），并制作简单的Web页面，包含简单文本信息（至少包含专业、学号、姓名）、自己的LOGO、自我介绍的音频信息。页面不要太复杂，包含要求的基本信息即可。
3. 通过浏览器获取自己编写的Web页面，使用Wireshark捕获浏览器与Web服务器的交互过程，并进行简单的分析说明。
4. 使用HTTP，不要使用HTTPS。
5. 提交实验报告。
6. **TCP**
7. TCP协议



Sequence Number（seq）:用来标识从TCP发端向TCP收端发送的数据字节流，它表示在这个报文段中的的第一个数据字节在数据流中的序号;主要用来解决网络报乱序的问题。

Acknowledgment Number（ack）:32位确认序列号，包含发送确认的一端所期望收到的下一个序号。确认序号应当是上次已成功收到数据字节序号加1。只有当标志位中的ACK标志(下面介绍)为1时该确认序列号的字 段才有效。主要用来解决不丢包的问题。

ACK：此标志表示应答域有效。有两个取值：0和1，为1的时候表示应答域有效，反之为0。

SYN：表示同步序号，用来建立连接。SYN标志位和ACK标志位搭配使用，当连接请求的时候，SYN=1，ACK=0;连接被响应的时候，SYN=1，ACK=1;这个标志的数据包经常被用来进行端口扫描。扫描者发送一个只有SYN的数据包，如果对方主机响应了一个数据包回来，就表明这台主机存在这个端口。

FIN：表示发送端已经达到数据末尾，也就是说双方的数据传送完成，没有数据可以传送了，发送FIN标志位的TCP数据包后，连接将被断开。这个标志的数据包也经常被用于进行端口扫描。

1. 三次握手
2. 状态/标记：

SYN\_SENT: 客户端已发送SYN报文，等待服务器端连接

SYN\_RCVD: 服务器端接受到了SYN报文，当收到客户端的ACK报文后，进入到ESTABLISHED状态

ESTABLISHED: 连接确认

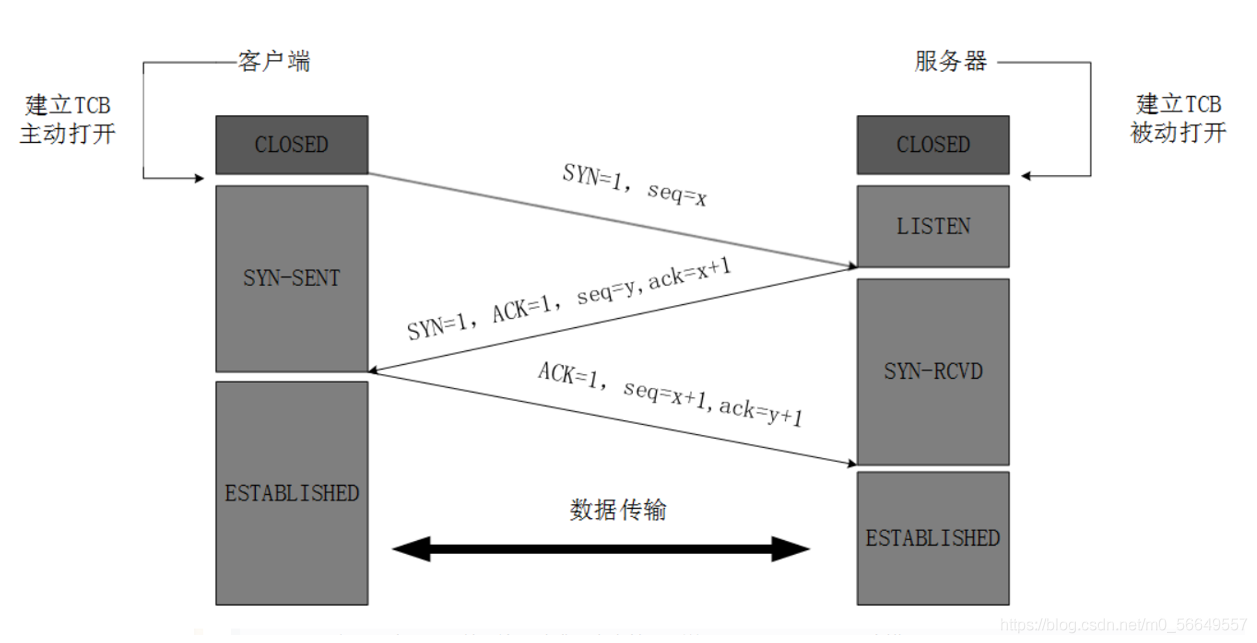
seq: 序列号，请求方将会发送的数据的第一个字节编号

ack: 确认的序号，只有ACK=1时有效。接收方收到收据后，希望对方下一次传输数据的第一个字节编号

SYN: 发起连接请求号标记位

ACK: 确认序号有效标记位

为了保证客户端和服务器端的可靠连接，TCP建立连接时要进行三次会话，也叫TCP三次握手，进行三次握手的目的是为了确认双方的接收能力和发送能力是否正常。



最开始的时候客户端和服务器都是处于CLOSED关闭状态。主动打开连接的为客户端，被动打开连接的是服务器。

TCP服务器进程先创建传输控制块TCB，时刻准备接受客户进程的连接请求，服务器进入LISTEN监听状态。

1. 第一次握手

TCP客户进程创建传输控制块TCB，然后向服务器发出连接请求报文，这是报文首部中的同部位SYN=1，同时选择一个初始序列号seq=x，此时，TCP客户端进程进入了 SYN\_SENT 同步已发送状态。

第一次握手中，客户端向服务器端发送报文，证明客户端的发送能力正常。

1. 第二次握手

TCP服务器收到请求报文后，如果同意连接，则会向客户端发出确认报文。确认报文中应该 ACK=1，SYN=1，确认号是ack=x+1，同时也要为自己初始化一个序列号seq=y，此时，TCP服务器进程进入了 SYN\_RCVD 同步收到状态。

第二次握手中，服务器端接收到报文并向客户端发送报文，证明服务器端的接收能力、发送能力正常，客户端得出客户端发送接收能力正常，但此时服务器不能确认客户端的接收能力有没有问题。

1. 第三次握手

TCP客户端收到确认后，还要向服务器给出确认。确认报文的ACK=1，ack=y+1，自己的序列号seq=x+1，此时，TCP连接建立，客户端进入ESTABLISHED已建立连接状态。

第三次握手中客户端向服务器发送报文，服务器确认客户端的接收能力正常。

1. 四次挥手
2. 状态/标记：

FIN = 1：代表要求释放连接

FIN\_WAIT\_1: 当SOCKET在ESTABLISHED状态时，它想主动关闭连接，向对方发送了FIN报文，此时该SOCKET即进入到FIN\_WAIT\_1状态。

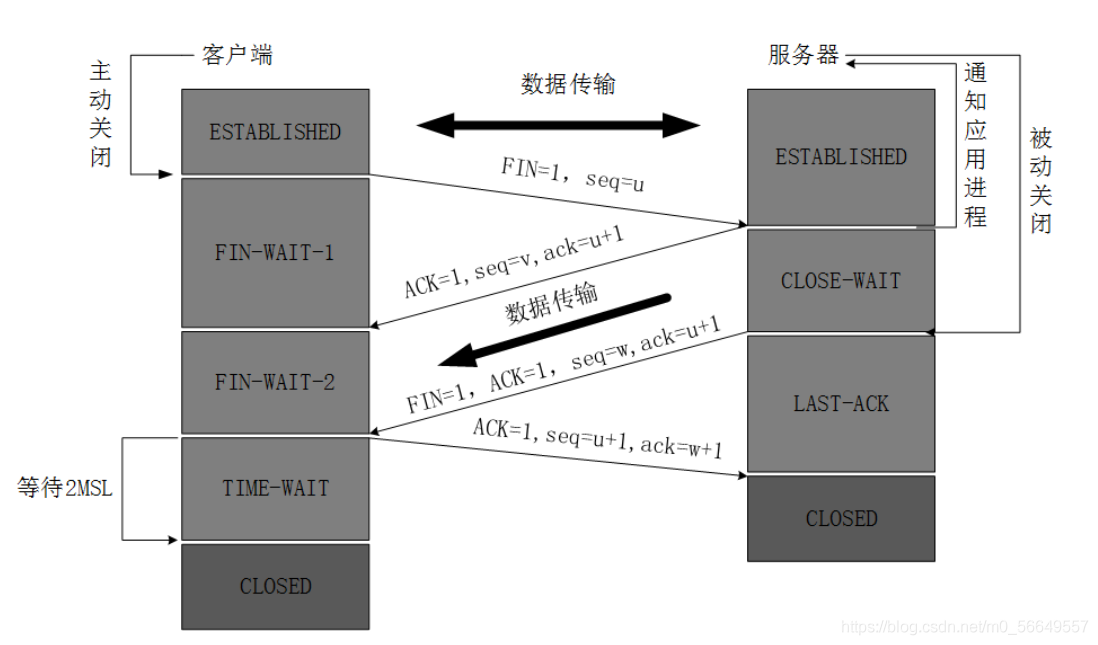
FIN\_WAIT\_2: 当对方回应ACK报文后，则进入到FIN\_WAIT\_2状态。

CLOSE\_WAIT：等待关闭连接。

LAST\_ACK: 被动关闭一方在发送FIN报文后，最后等待对方的ACK报文。当收到ACK报文后，可以进入到CLOSED可用状态。

CLOSED: 表示连接中断。

2MSL：MSL全称为Maximum Segment Lifetime，是一个TCP报文片段在网络中最大的存活时间。2MSL对应于一次消息的来回（一个发送和一个回复）所需的最大时间。如果直到2MSL，主动断开方都没有再一次收到对方的报文（如FIN报文），则可以推断ACK已经被对方成功接收，此时，主动断开方将最终结束自己的TCP连接。所以，TCP的TIME\_WAIT状态也称为2MSL等待状态。



数据传输完毕后，双方都可释放连接。最开始的时候，客户端和服务器都是处于ESTABLISHED状态，然后客户端主动关闭，服务器被动关闭。

1. 第一次挥手

客户端发出连接释放报文，并且停止发送数据。释放数据报文首部，FIN=1，其序列号为seq=u（前面已经传送过来的数据的最后一个字节的序号加1）。此时，客户端进入FIN\_WAIT\_1（终止等待1）状态。

1. 第二次挥手

服务器端接收到连接释放请求报文后，发出确认报文，ACK=1，ack=u+1，并且带上自己的序列号seq=v，此时，服务端就进入了CLOSE\_WAIT 关闭等待状态。若服务器端还有数据要发送给客户端，客户端还会接受，服务器端会持续一段时间。客户端接收到服务器端的确认请求ACK=1后，客户端就会进入FIN\_WAIT\_2（终止等待2）状态，等待服务器发送连接释放报文。

1. 第三次挥手

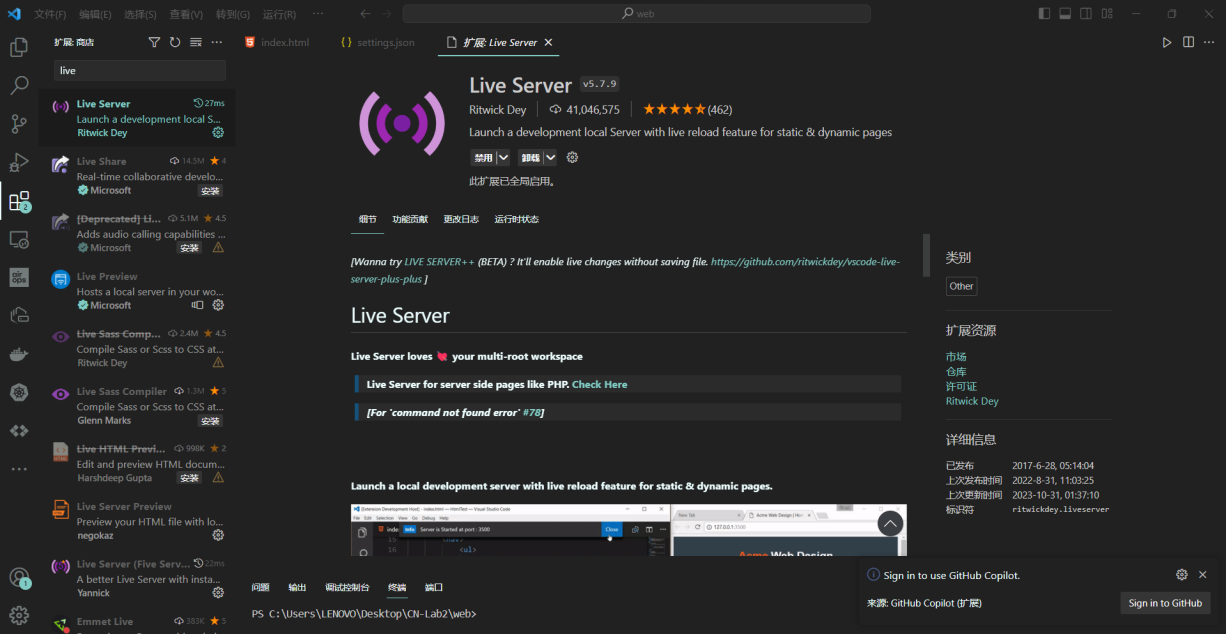
服务器将最后的数据发送完毕后，就向客户端发送连接释放报文FIN=1，服务器就进入了LAST\_ACK（最后确认）状态，等待客户端的确认,seq=w,ack=u+1。

1. 第四次挥手

客户端收到服务器的连接释放报文后，发出确认，ACK=1，ack=w+1，而自己的序列号是seq=u+1，此时，客户端就进入了TIME\_WAIT（时间等待）状态，但此时TCP连接还未终止，经过2MSL后（最长报文寿命），当客户端撤销相应的TCB后，客户端才会进入CLOSED关闭状态，服务器端接收到确认报文后，会立即进入CLOSED关闭状态。

1. **实验过程**
2. **搭建Web服务器**

使用VSCode的Live Server扩展搭建本地代理服务器。



Live Server：一个具有实时加载功能的小型服务器。可以在项目中实时用live-server作为一个实时服务器实时查看开发的网页或项目效果。

对liveserver的settings.json配置如下：

{

    "workbench.colorTheme": "Community Material Theme Darker",*// 设置VSCode的颜色主题为"Community Material Theme Darker"。*

    "workbench.iconTheme": "vscode-icons",*// 设置VSCode的图标主题为"vscode-icons"。*

    "editor.unicodeHighlight.nonBasicASCII": false,*// 在代码编辑器中禁用对非基本ASCII字符的高亮显示*

    "php.debug.executablePath": "", *// 指定PHP调试的可执行文件路径，这里为空。*

    "open-php-html-js-in-browser.documentRootFolder": "/", *// 设置"Open PHP/HTML/JS in Browser"扩展的文档根文件夹为根目录。*

    "open-php-html-js-in-browser.selectedBrowser": "Edge",*// 设置"Open PHP/HTML/JS in Browser"扩展的默认浏览器为Microsoft Edge。*

    "code-runner.runInTerminal":true,*// 配置"Code Runner"扩展在终端中运行代码。*

    "open-in-browser.default":"",*// "Open in Browser"扩展的空默认配置。*

    "vsicons.dontShowNewVersionMessage": true,*// 禁止"VSCode Icons"扩展显示新版本更新消息*

    "liveServer.settings.useWebExt": true,*// 启用"Live Server"扩展的Web扩展功能*

    "liveServer.settings.NoBrowser": false,*// 禁用"Live Server"启动时不打开浏览器的选项。*

    "liveServer.settings.donotShowInfoMsg": true,*// 阻止"Live Server"扩展显示信息消息。*

    "liveServer.settings.donotVerifyTags": true,*// 禁用"Live Server"扩展的标签验证。*

    "liveServer.settings.AdvanceCustomBrowserCmdLine": "",*// 为"Live Server"扩展指定浏览器的自定义命令行选项，这里为空。*

    "liveServer.settings.root":"/",*//设置根目录，也就是打开的文件会在该目录下找*

    "liveServer.settings.port": 9999,*//设置本地服务的端口号*

    "liveServer.settings.host": "localhost",*// 指定本地服务器的主机名。*

    "liveServer.settings.proxy": {*// 配置"Live Server"扩展的代理设置，但目前已禁用（"enable": false）。*

        "enable": false,

        "baseUri": "/",

        "proxyUri": "http://127.0.0.1:80"

    },

    "liveServer.settings.ignoreFiles": [*//设置忽略的文件*

        ".vscode/\*\*",

        "\*\*/\*.scss",

        "\*\*/\*.sass",

        "\*\*/\*.ts"

    ],

    "liveServer.settings.CustomBrowser":null,*//设置默认打开的浏览器*

    "liveServer.settings.useLocalIp": true,*// 启用在"Live Server"扩展中使用本地IP地址。*

    "liveServer.settings.ChromeDebuggingAttachment": false,*// 禁用"Live Server"扩展中的Chrome调试附件。*

    "liveServer.settings.fullReload": false*// 配置"Live Server"扩展在检测到更改时是否执行完全页面重载，但当前设置为false。*

}

1. **Web页面**
2. html代码

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

    <meta *Cache-Control*="no-store">

    <title>Lab2</title>

    <link *rel*="stylesheet" *href*="static/css/test.css">

</head>

<body>

<h1>实验2: 配置Web服务器与分析交互过程</h1>

<img *class*="logo" *src*="static/image/logo.png" *alt*="">

<div>

<p>姓名 : 孙蕗</p>

<p>学号 : 2112060</p>

<p>年级 : 2021级</p>

<p>专业 : 信息安全</p>

</div>

<div *style*="text-align: center;">

    <audio *controls*>

        <source *src*="static/audio/introduction.mp3" *type*="audio/mpeg">

    </audio>

</div>

</body>

</html>

1. Web页面

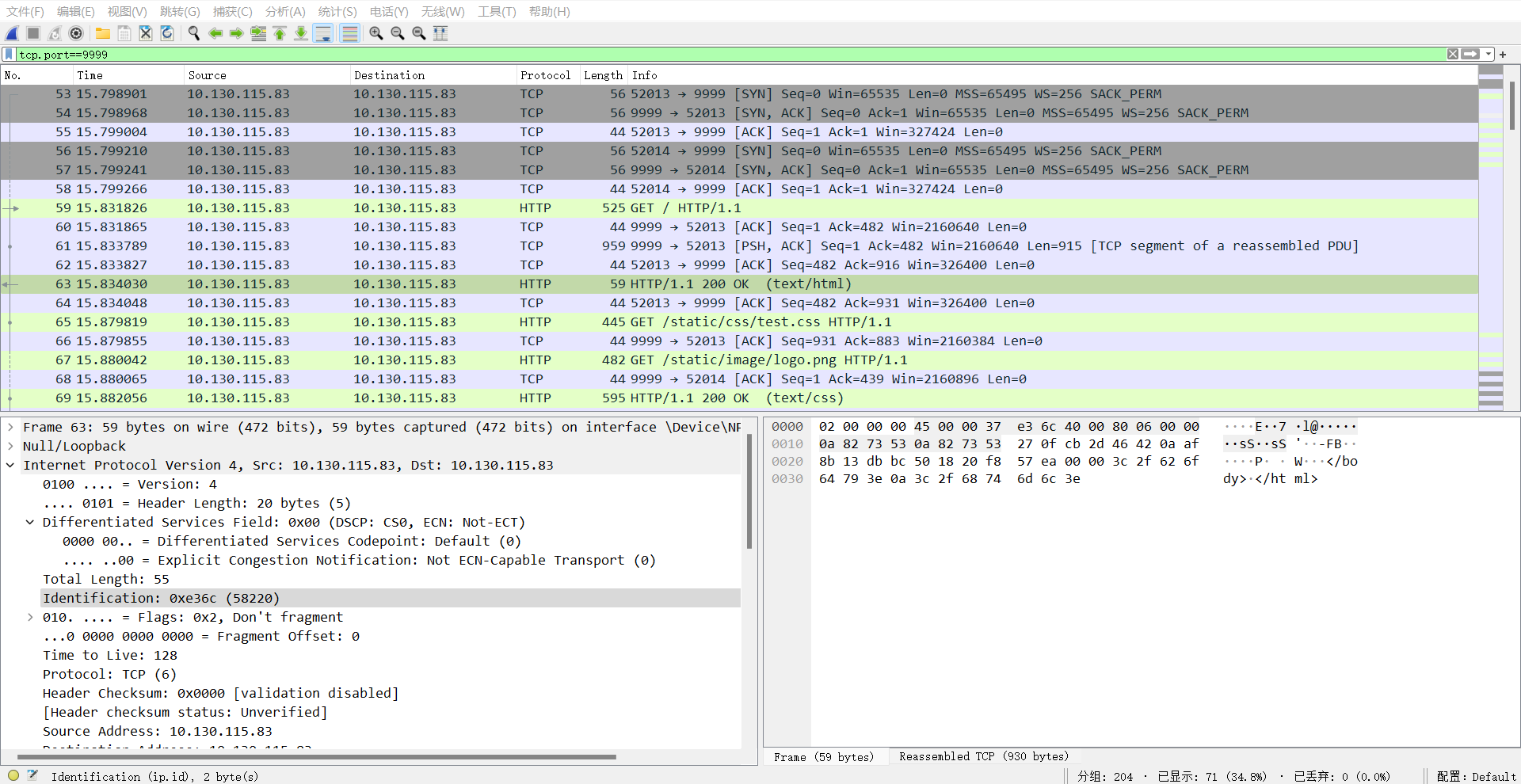


1. **WireShark捕获浏览器与Web服务器的交互过程**

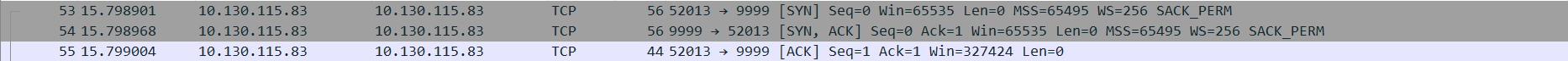
安装wireshark的同时安装Npcap。打开WireShark后，看到Adapter for loopback traffic capture。本次实验中本机既作为客户端又作为服务器端，WireShark可通过此接口抓取本地回环包。



根据Live Server.settings的设置，在过滤器中输入tcp.port == 9999，对本实验的Web服务器抓包。

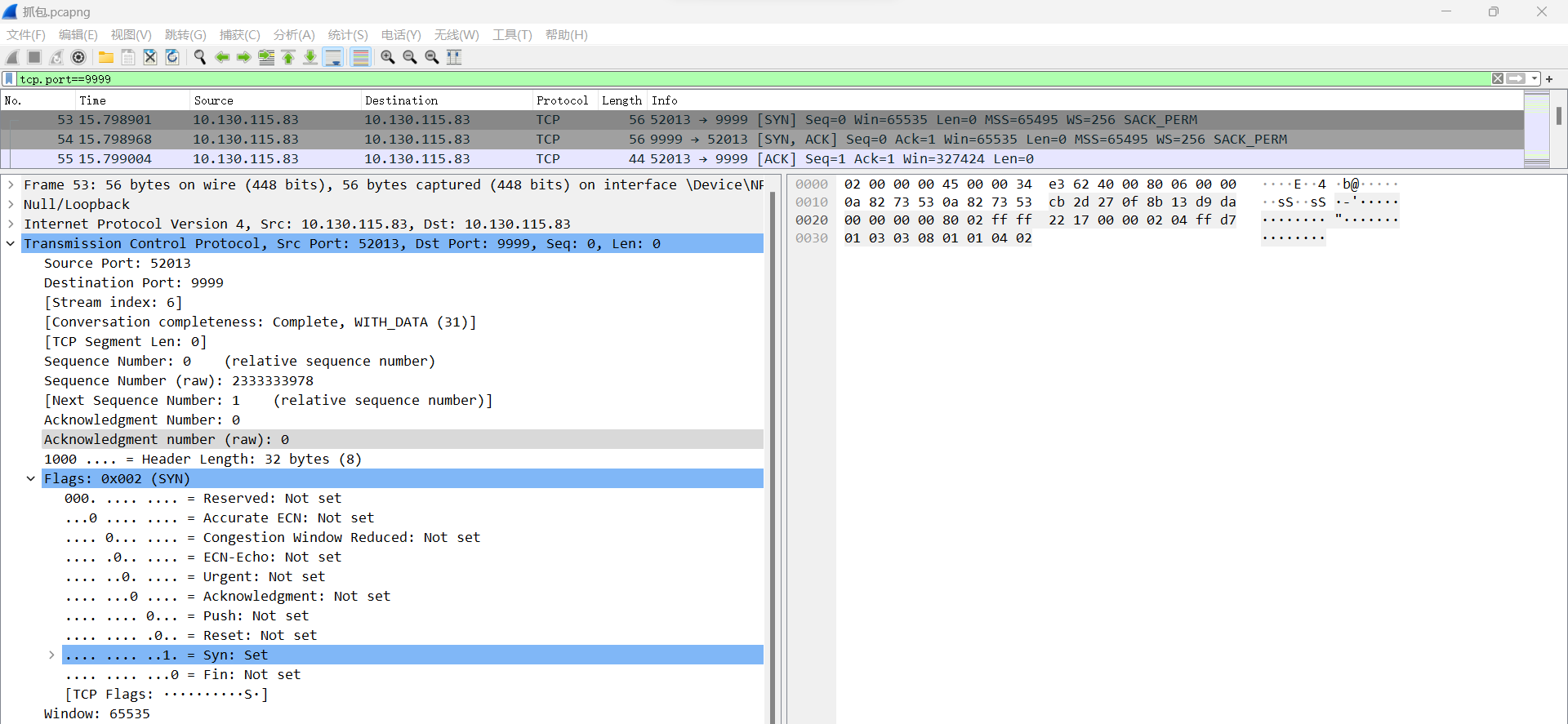


1. **分析浏览器与Web服务器的交互过程**
2. TCP三次握手连接建立过程



1. 第一次握手

TCP客户进程向服务器发出连接请求报文，SYN=1。同时选择一个初始序列号seq=x，代表客户端请求建立连接。此时，TCP客户端进程进入了 SYN\_SENT 同步已发送状态。



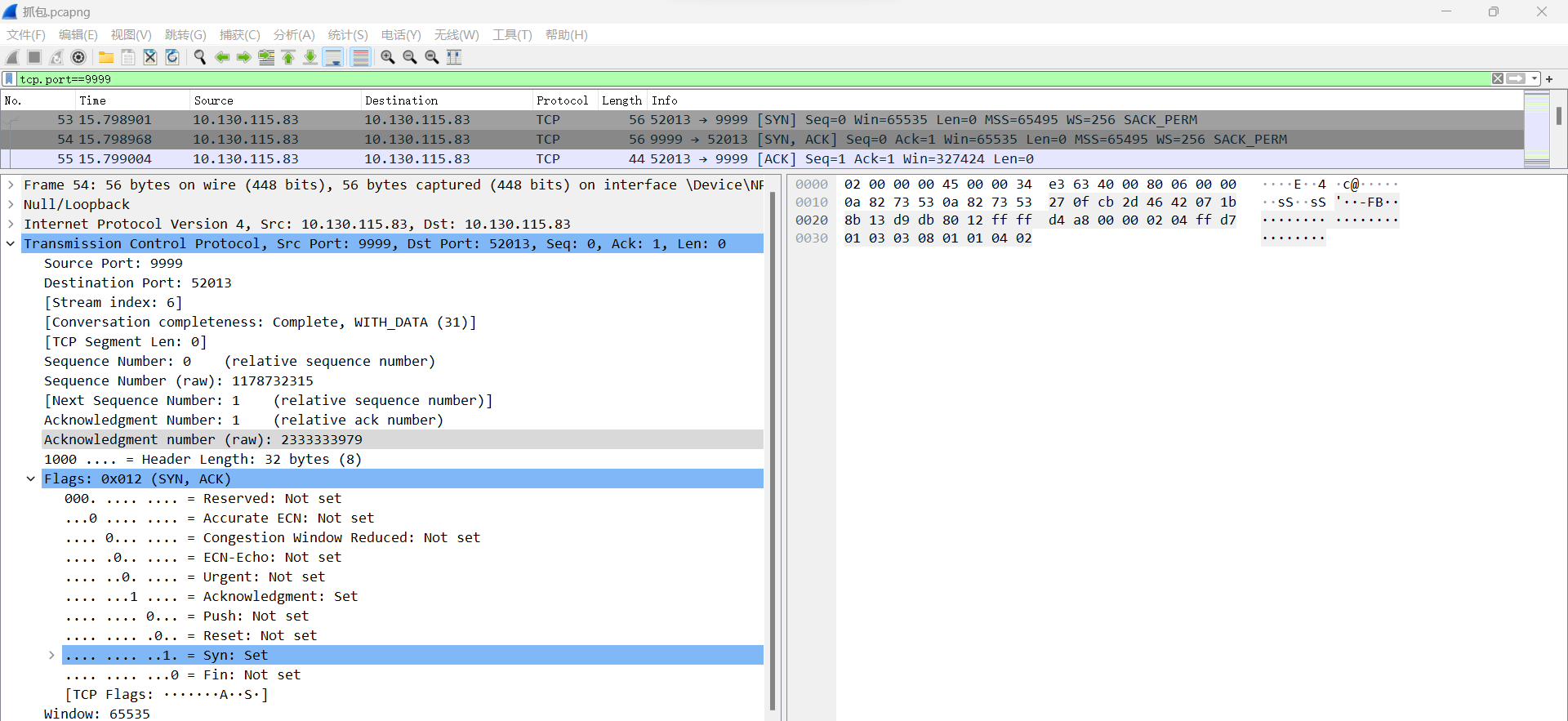
SYN = 1：标志位，表示请求建立连接；

ACK = 0：表示当前没有接收到数据。

发送的具体数据第一个字节编号记为x，赋值seq，此处seq = 0，ack = 0

1. 第二次握手

TCP服务器收到连接请求报文段后，如果同意连接，则会向客户端发出确认报文。确认报文中 ACK=1，SYN=1，确认号是ack=x+1，同时也要为自己初始化一个序列号seq=y，此时，TCP服务器进程进入了 SYN\_RCVD 同步收到状态。



SYN = 1，ACK = 1：确认报文

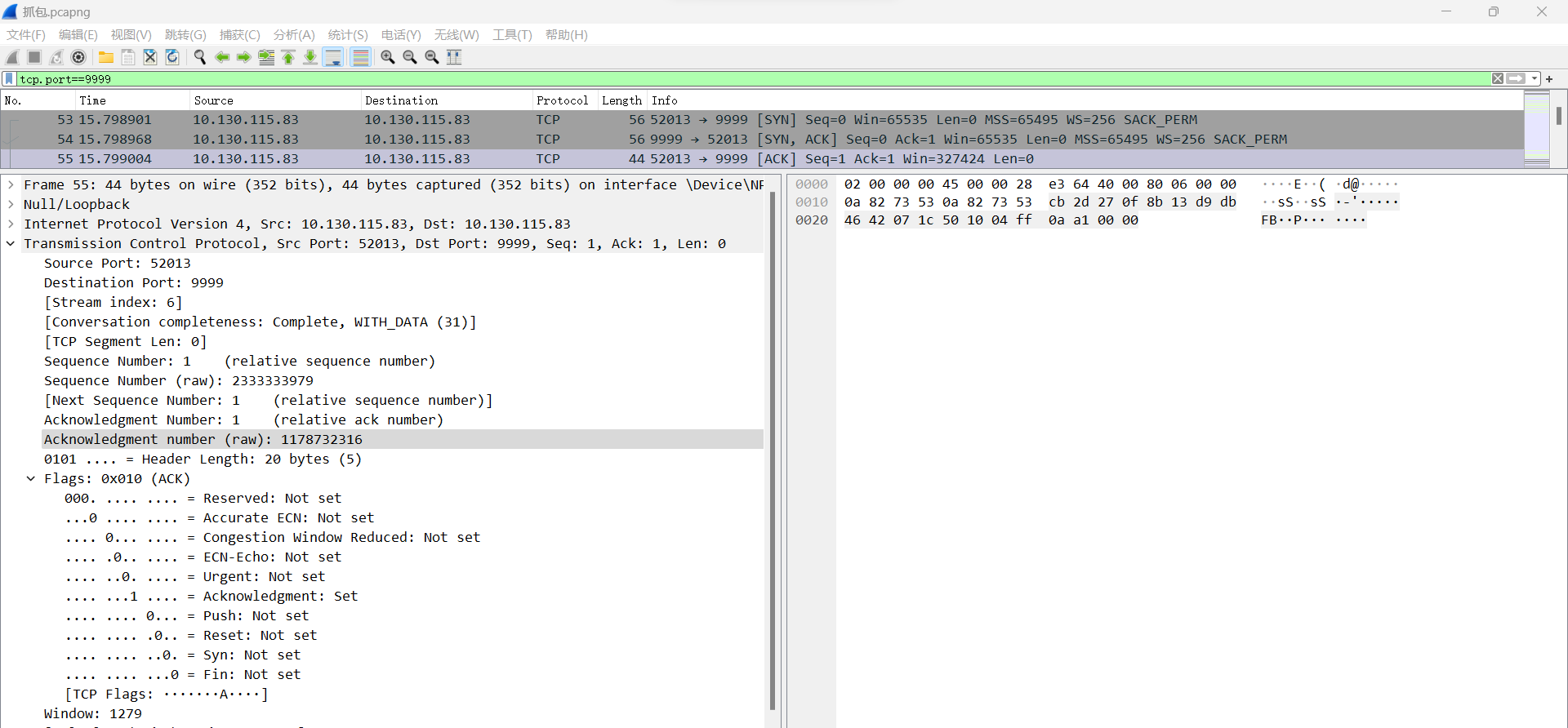
发送的具体数据第一个字节编号seq= 0 = y

服务器端希望客户端下一次返回的编号ack = 1 = x+1。

1. 第三次握手

TCP 客户进程收到 TCP 连接请求确认报文段后，还要向 TCP 服务器进程发送一个普通的 TCP 确认报文段并进入连接已建立状态。源IP向目标IP发送一个确认报文，ACK=1，SYN=0，三次握手结束。

此时对服务端进行连接确认，所以SYN=0,只需要返回ACK = 1代表确认。发送的具体数据第一个字节编号记为seq = x+1，希望服务端下次传输的数据第一个字节编号记为ack = y + 1，此时，TCP连接建立，客户端进入ESTABLISHED已建立连接状态。

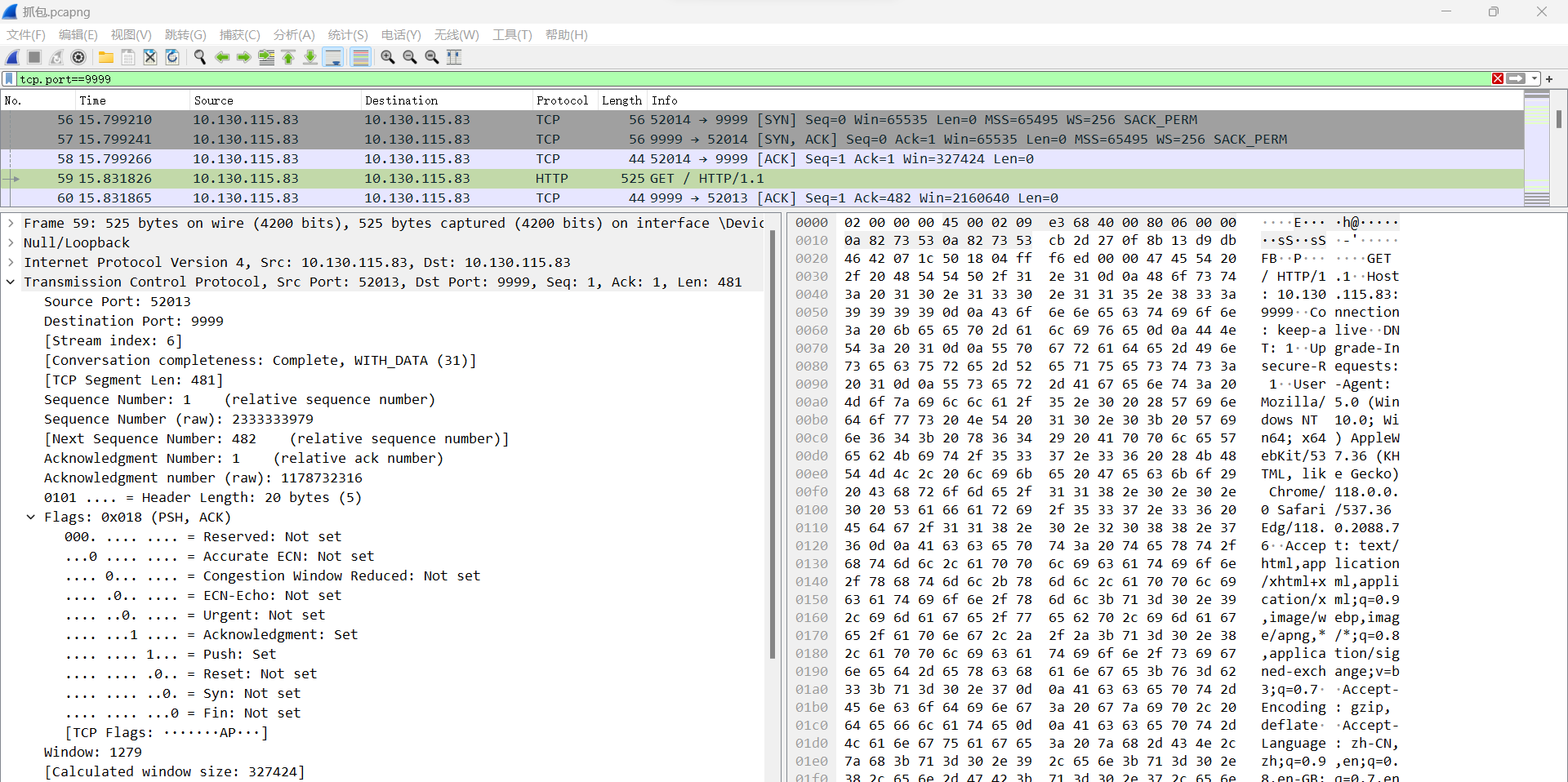


ACK = 1：确认报文

发送的具体数据第一个字节编号seq = 1 = x+1

服务器端希望客户端下一次返回的编号ack = 1 = y+1。

1. 发送与接收数据
2. 获取网页资源
3. 客户端向服务器发送一个 SYN=0，ACK=1，acknowledgement number=1，sequence number=1 的数据包，请求服务器数据。

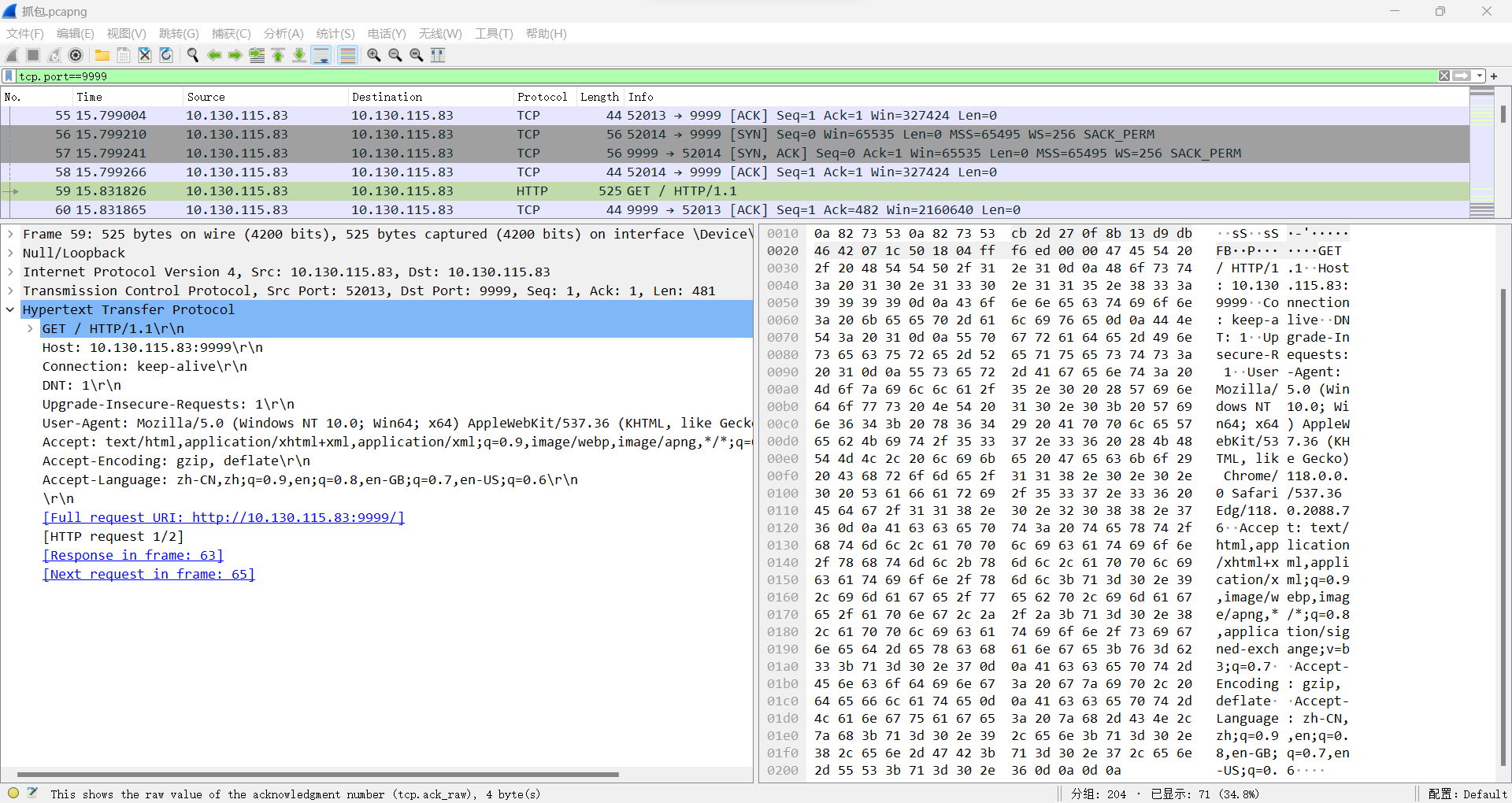


ACK = 1：客户端回应服务器发来的请求

acknowledgement number = 1：表明服务器只发送过长度为 1 的请求连接数据；

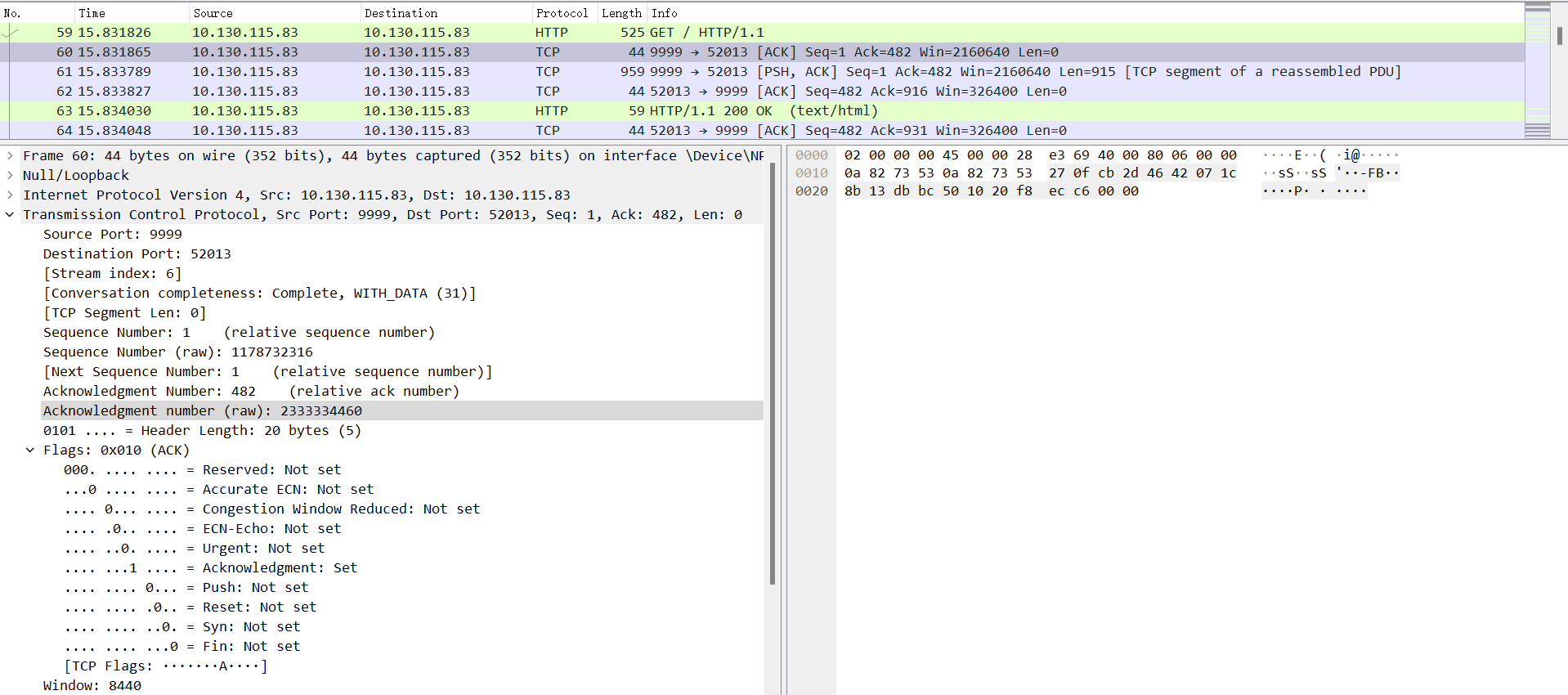
第三次握手时，客户端只发送过长度为 1 的请求连接数据,所以 seq=1。

数据段长度为481字节，因此下一个seq的值就是482。

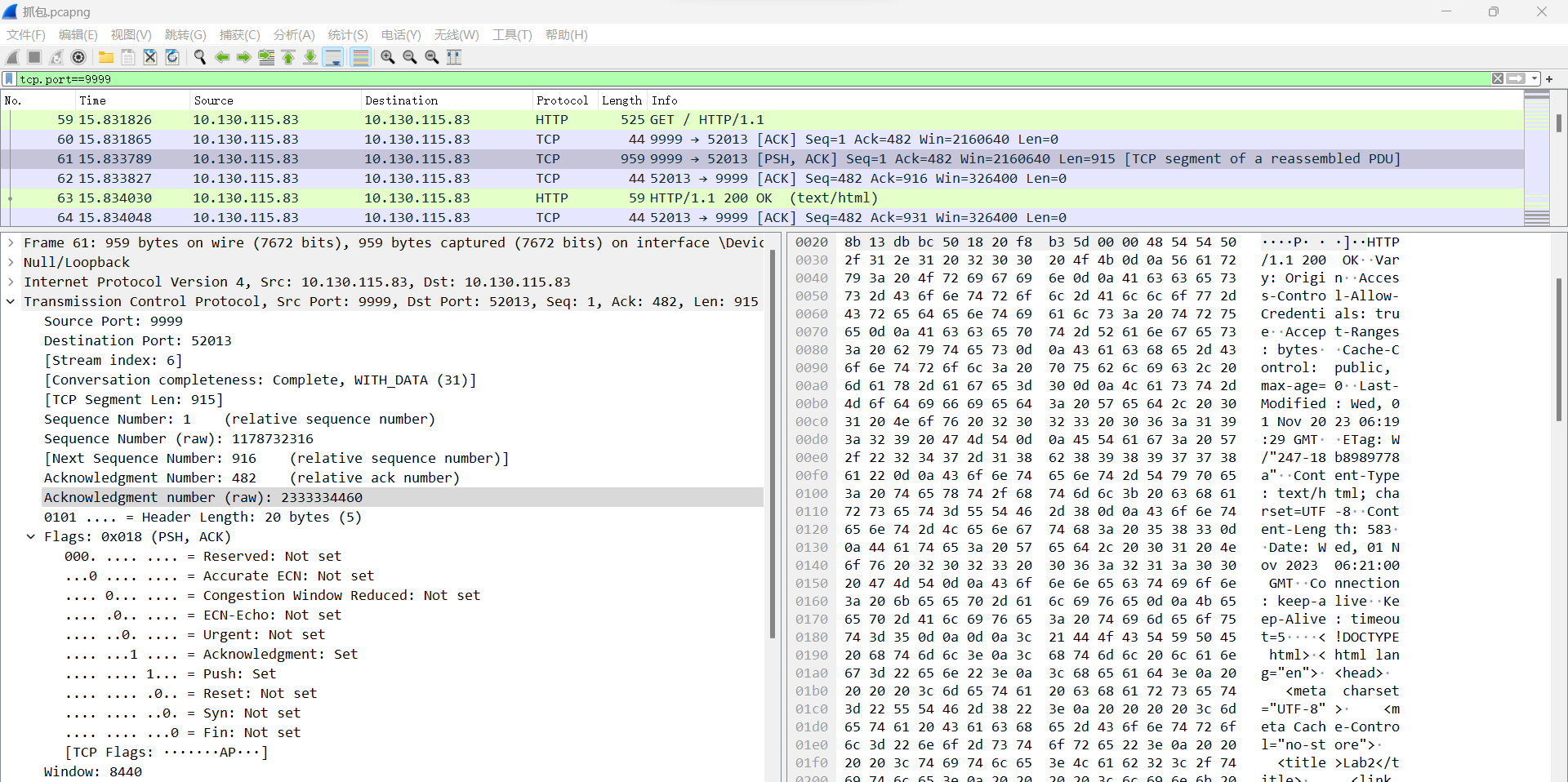


1. GET 协议方法
2. Host 请求的主机名，也就是服务器的地址
3. Connection 连接方式， keep-alive 表示长连接的方式。
4. User-Agent 表示的客户端代理，也就是浏览器的类型
5. Accept 表示客户端可以识别的响应内容。
6. Accept-Encoding 表示客户端接受的编码格式
7. Accept-Language 表示客户端接受的语言集

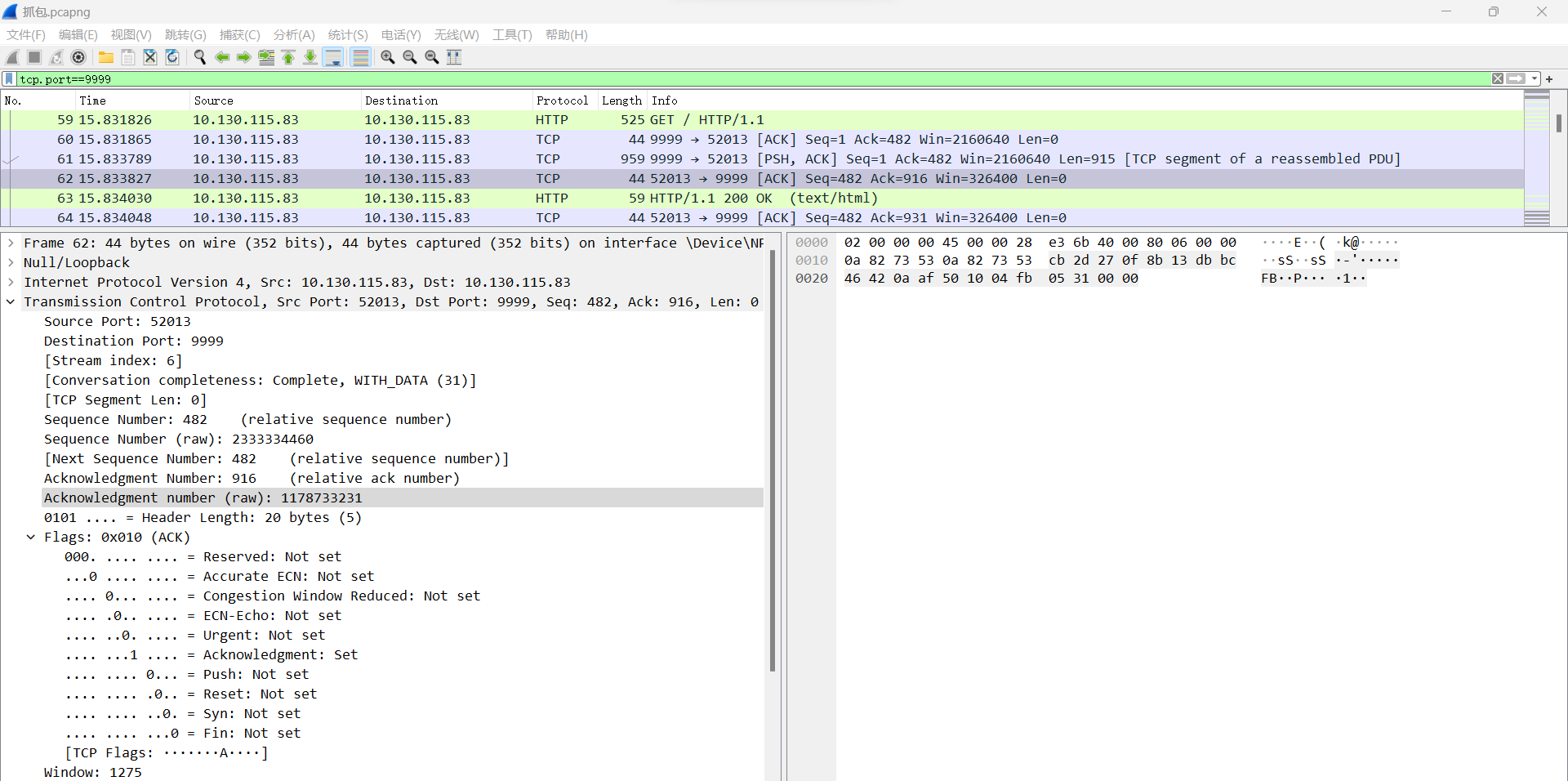
② 服务器在接收到了客户端发送的GET请求之后，会通过TCP回应一条确认报文（ACK=1），ack=482。



③报文中包含了HTTP的响应信息。因此数据段长度不再为0，长度为915。这个响应信息中包含了HTTP的响应头中的数据。



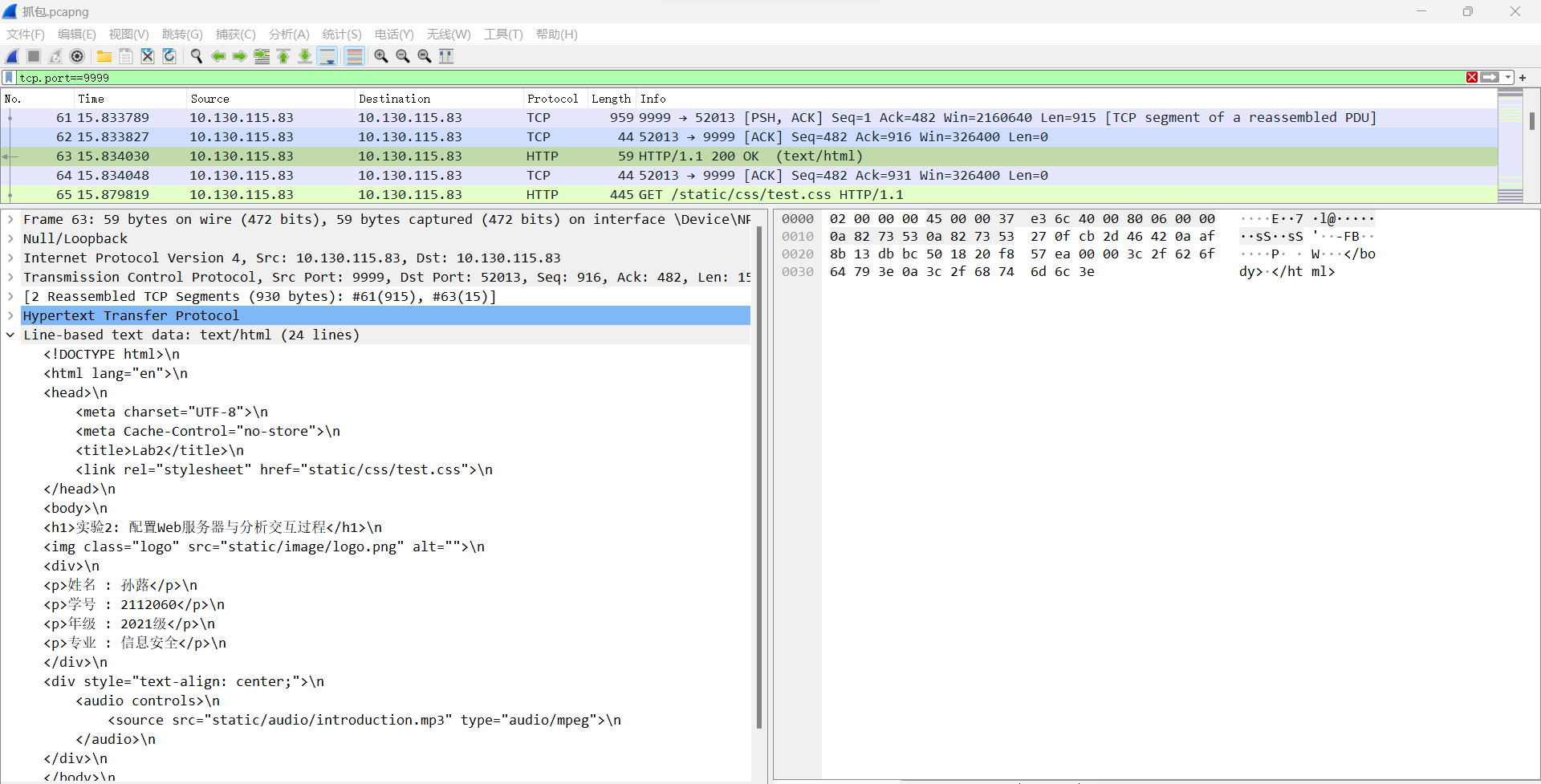
④ 客户端在收到服务器的响应之后，同样会给服务器端发送确认报文。



seq = 482,ack = 916,ACK = 1

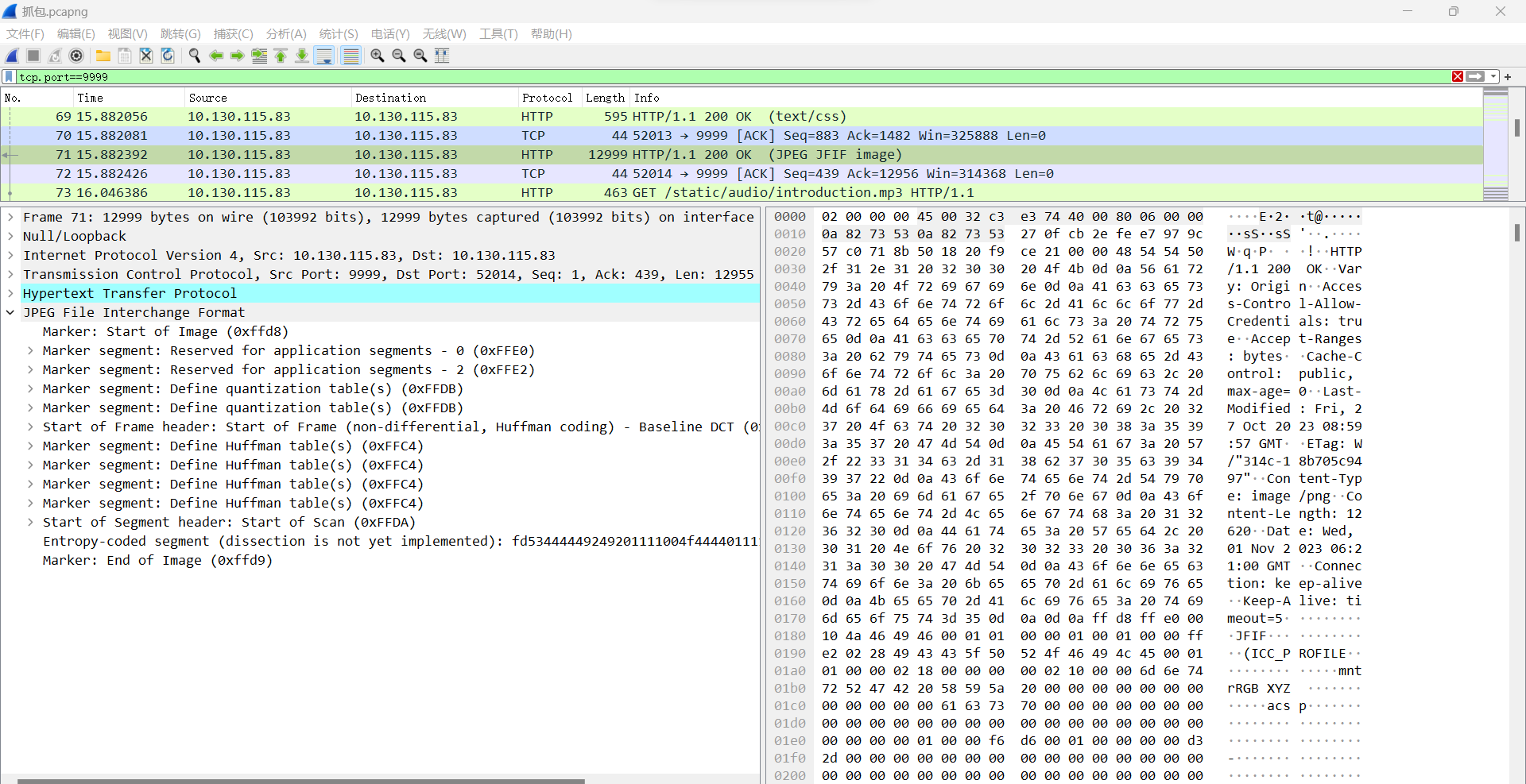
1. 获取服务器文字

在Line-based text data里能看到html代码文件数据。

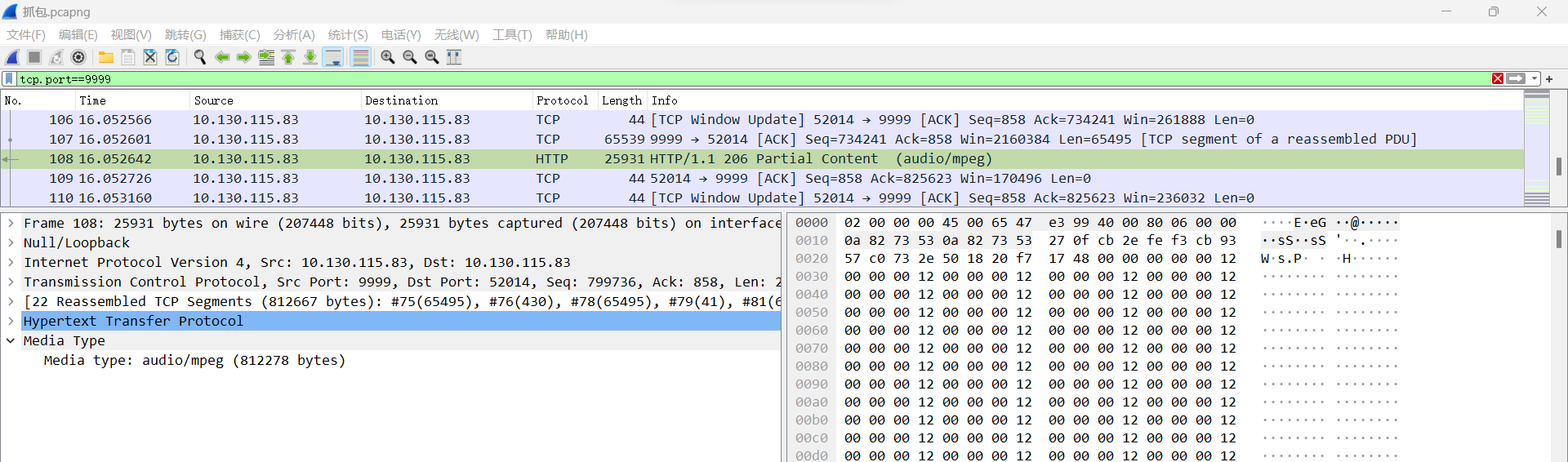


1. 获取服务器图片

在JPEG File Interchange Format能看到图片的十六进制编码

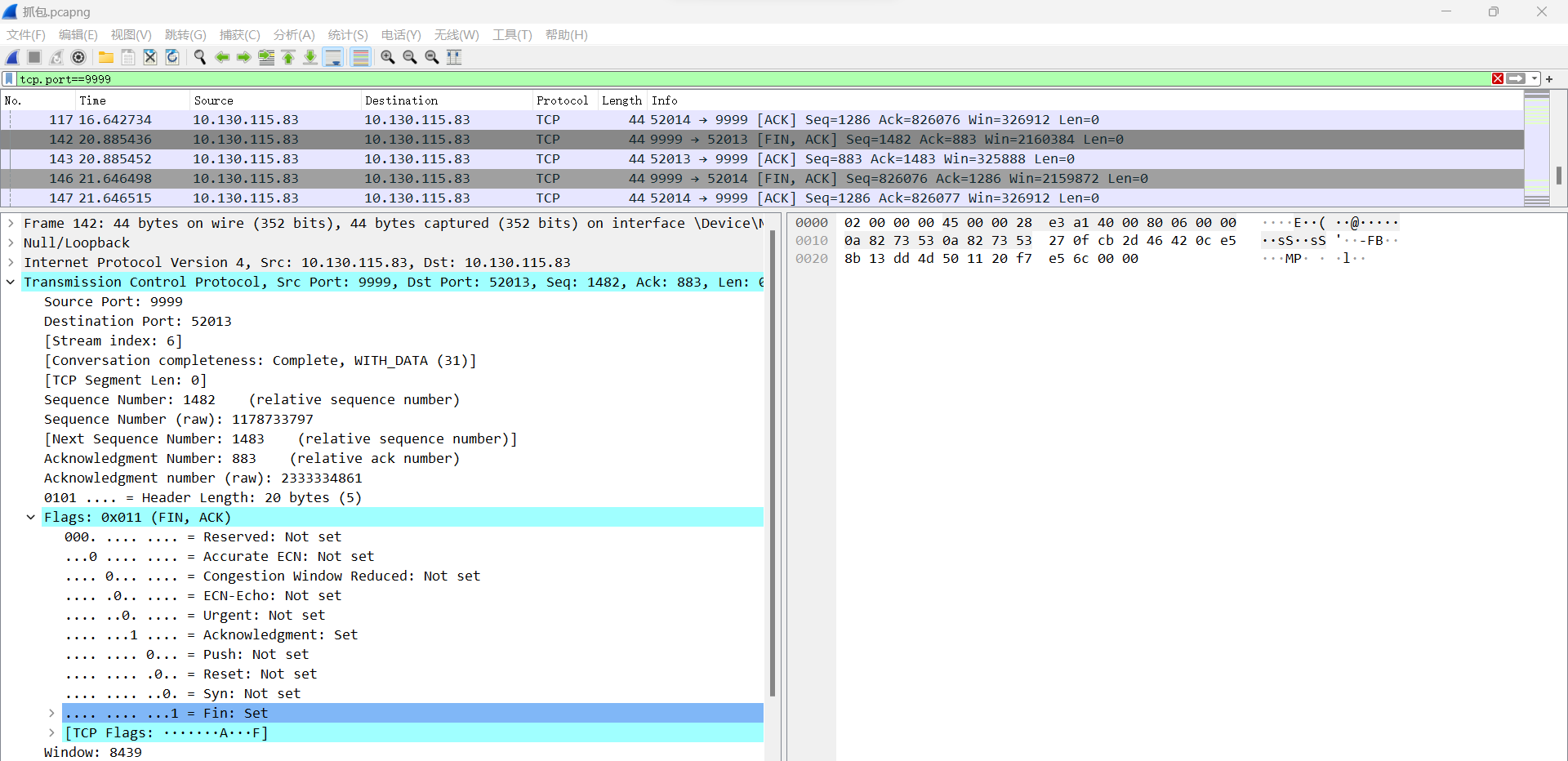


1. 获取服务器音频



1. 四次挥手
2. 第一次挥手

TCP 客户进程会发送 TCP 连接释放报文段，并进入终止等待 1(FIN\_WAIT\_1)状态。客户端发出连接释放报文，并且停止发送数据。释放数据报文首部，FIN=1，其序列号为seq=u（前面已经传送过来的数据的最后一个字节的序号加1）。



FIN = 1：表明这是一个 TCP 连接释放报文段；

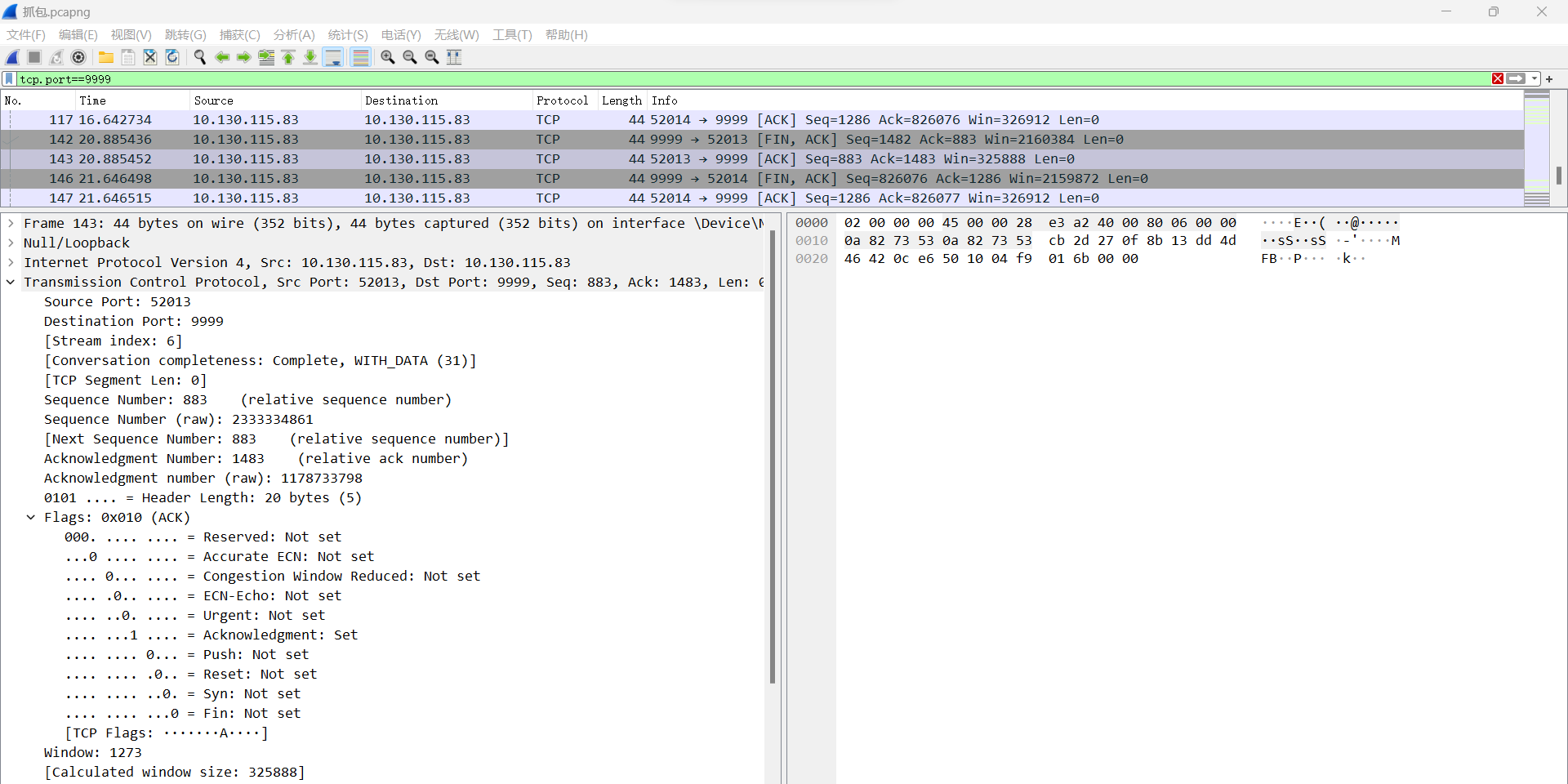
ACK = 1：表明对之前收到的报文段进行确认；

发送的具体数据第一个字节编号seq = u = 1482,

服务器端希望客户端下一次返回的编号ack = v = 883

1. 第二次挥手

服务器端接收到连接释放请求报文后，发出确认报文，ACK=1，ack=u+1，并且带上自己的序列号seq=v，此时，服务端就进入了CLOSE\_WAIT 关闭等待状态。若服务器端还有数据要发送给客户端，客户端还会接受，服务器端会持续一段时间。客户端接收到服务器端的确认请求ACK=1后，客户端就会进入FIN\_WAIT\_2（终止等待2）状态，等待服务器发送连接释放报文。



FIN = 0：表明在等待所有数据传输完成才能中断连接；

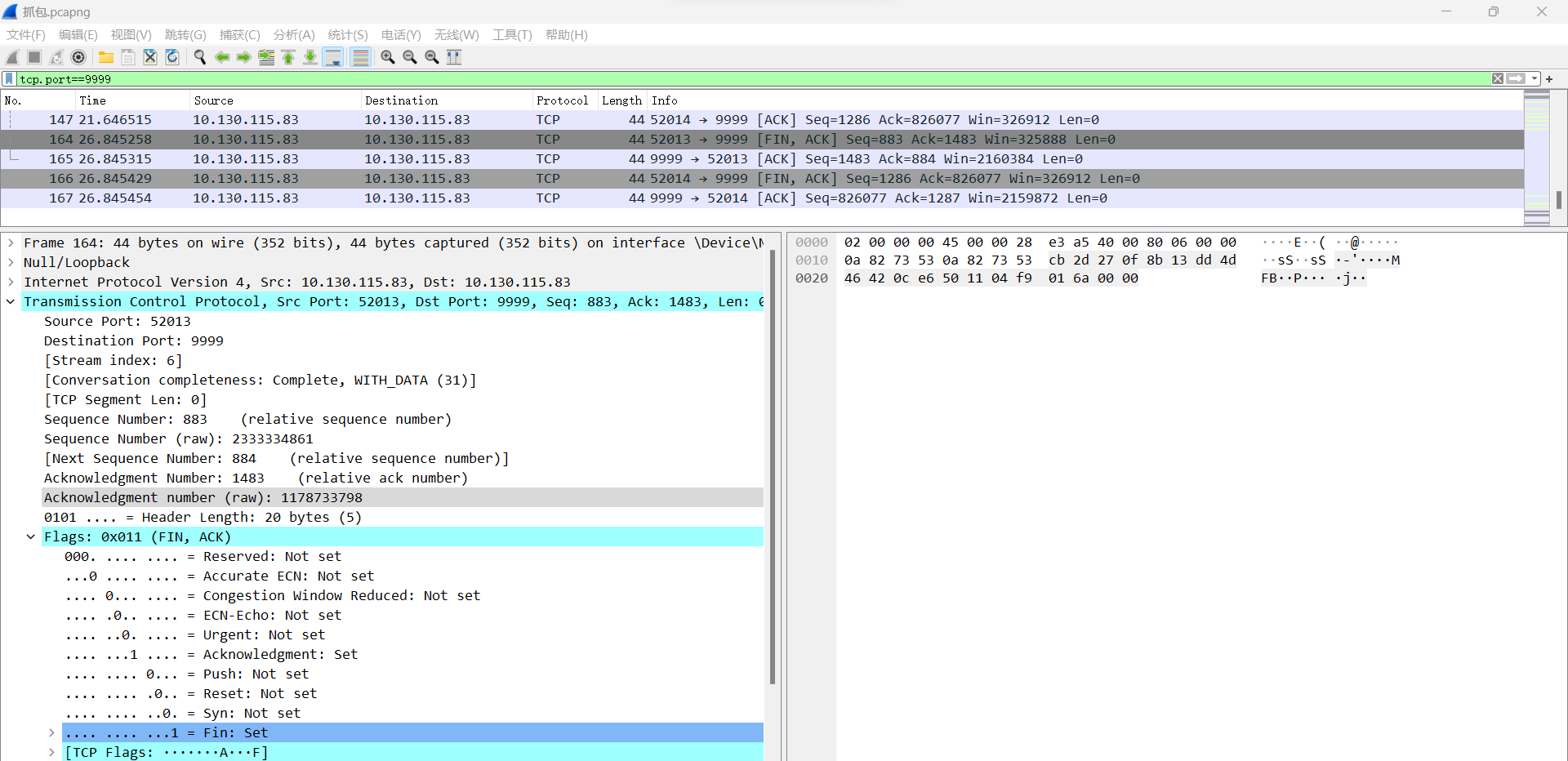
ACK = 1：表明回应客户端发来的请求；

发送的具体数据第一个字节编号seq = 883 = v,

服务器端希望客户端下一次返回的编号ack = 1483 = u+1

1. 第三次挥手

服务器将最后的数据发送完毕后，就向客户端发送释放报文FIN=1，服务器就进入了LAST\_ACK（最后确认）状态，等待客户端的确认,seq=w,ack=u+1。



FIN = 1：表明服务器认为可以中断连接；

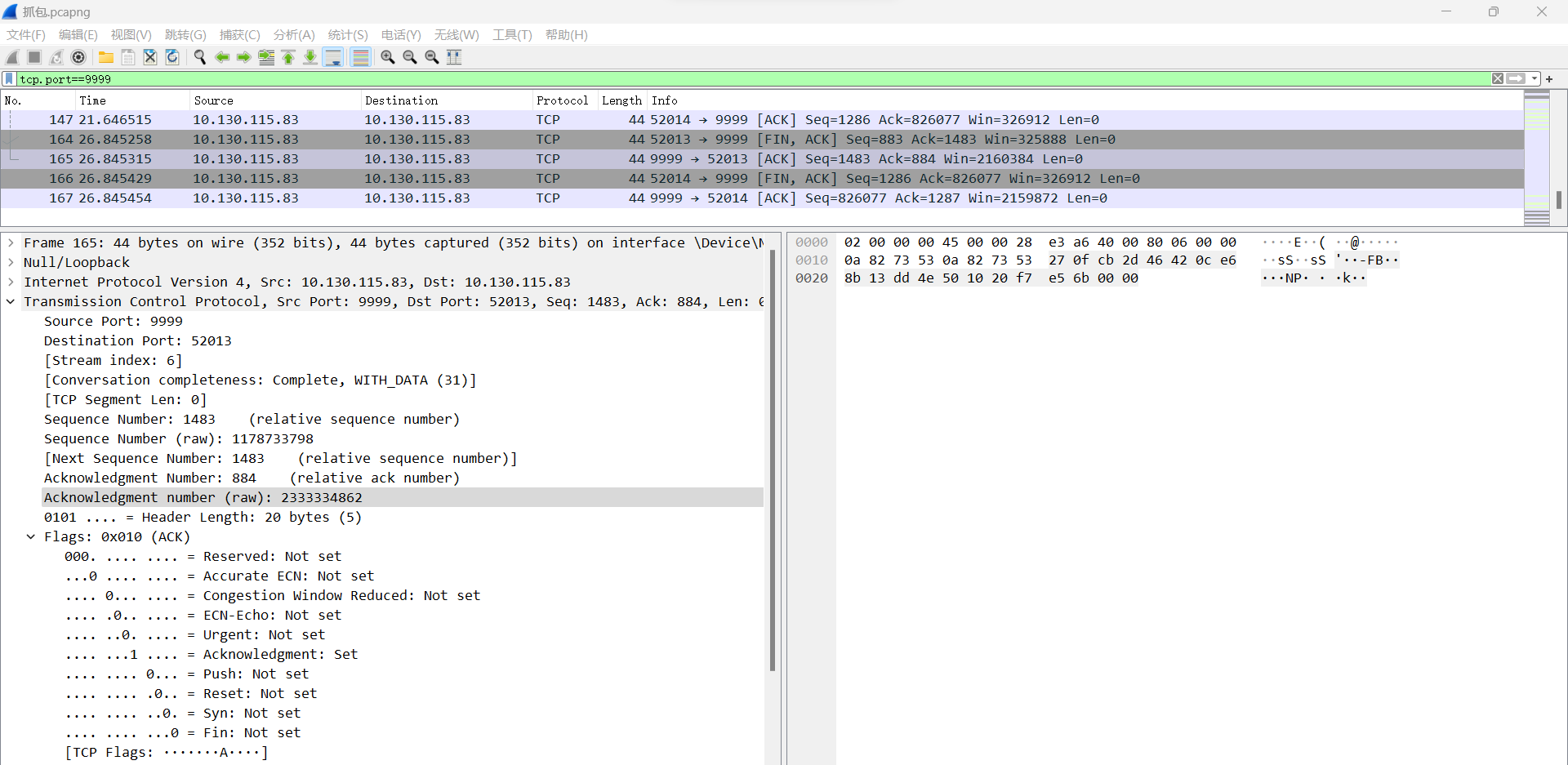
ACK = 1：表明回应客户端发来的请求；

发送的具体数据第一个字节编号seq = 883 = w ,

服务器端希望客户端下一次返回的编号ack = 1483 = u+1

1. 第四次挥手

客户端收到服务器的连接释放报文后，发出确认，ACK=1，ack=w+1，而自己的序列号是seq=u+1，此时，客户端就进入了TIME\_WAIT（时间等待）状态，但此时TCP连接还未终止，经过2MSL后（最长报文寿命），当客户端撤销相应的TCB后，客户端才会进入CLOSED关闭状态，服务器端接收到确认报文后，会立即进入CLOSED关闭状态。



FIN = 0：表明不需要再次终止连接；

ACK = 1：表明回应服务器发来的请求；

发送的具体数据第一个字节编号seq = 1483 = u+1 ,

服务器端希望客户端下一次返回的编号ack = 884 = w+1 ;