

计算机网络实验报告

Lab2 配置Web服务器，编写简单页面，分析交互过程

网络空间安全学院 物联网工程

2110951 梁晓储

代码已发布到github: https://github.com/WangshuXC/Computer_network

一、实验要求

- (1) 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。
- (2) 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。
- (3) 使用HTTP，不要使用HTTPS。
- (4) 提交实验报告。

二、功能实现与代码分析

web服务器使用流程

1. 安装[nodejs](#)
2. 在web文件目录下执行 `npm install` 命令，安装所需依赖项
3. 运行 `node app.js` 通过app.js来启动web服务器

Web服务器启动文件：app.js

```
const express = require('express');
const app = express();
const path = require('path');
const os = require('os');

app.use(express.static(path.join(__dirname, './')));

const port = 6200;

app.listen(port, '0.0.0.0', () => {
  const networkInterfaces = os.networkInterfaces();
  let ipAddress;

  // 遍历网络接口，找到无线网和有线网口的IPv4地址
  Object.keys(networkInterfaces).forEach((interfaceName) => {
    networkInterfaces[interfaceName].forEach((networkInterface) => {
      if (
        !networkInterface.internal &&
        (networkInterface.family === 'IPv4') &&
        (interfaceName.includes('WLAN') || interfaceName.includes('ETH'))
      ) {
        ipAddress = networkInterface.address;
      }
    });
  });
});
```

```

        //排除掉虚拟机的Ipv4地址
    ) {
        ipAddress = networkInterface.address;
        console.log(`Device Name: ${interfaceName}`);
        console.log(`IP Address: ${ipAddress}`);
    }
    });
});

//方便直接打开网址
console.log(`Available on:`);
console.log(`http://localhost:${port}`);
console.log(`http://${ipAddress}:${port}`);
});

```

在文件目录下使用 `node app.js` 即可启动web服务器，同时会在控制台以 `ip地址+端口号` 的形式输出web地址，方便直接打开web

除此之外，我在实验设备安装WSL之前只有WLAN一个网络接口，固不需要进行ip地址的筛选即是正确的WLAN校园网ipv4，但是存在虚拟机只后可能会获取到虚拟机的ipv4地址，所以添加了 `(interfaceName.includes('WLAN') || interfaceName.includes('ETH'))` 来进行网络接口的筛选，排除掉虚拟机的ipv4地址

HTML部分代码

```

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>2110951自我介绍</title>
  <style>
    /* css代码已省略 */
  </style>
</head>

<body>
  <div id="app">
    <div class="container">
      <div class="logo">
        
      </div>
      <div class="info">
        <p class="title">自我介绍</p>
        <p>梁晓储</p>
        <p>2110951</p>
        <p>物联网工程</p>
      </div>
      <a class="github" href="https://www.github.com/wangshuxc"
target="_blank">
        访问我的Github

```

```

    </a>
    <div class="audio">
      <p>音频介绍</p>
      <audio id="audioElement" src="audio.mp3"></audio>
      <button id="playButton" onclick="togglePlay()"></button>
      <div class="progressBar">
        <div class="progressBarFill" :style="{ width: progress + '%'"
      }"></div>
      </div>
    </div>
  </div>
</div>

<script>
  var audioElement = document.getElementById('audioElement');
  audioElement.addEventListener('timeupdate', updateProgressBar);
  audioElement.addEventListener('ended', resetAudio);

  var isPlaying = false;
  var playButton = document.getElementById('playButton');
  playButton.textContent = isPlaying ? '⏸' : '▶';
  var progress = 0;

  function togglePlay() {
    if (isPlaying) {
      audioElement.pause();
      playButton.textContent = '▶';
    } else {
      audioElement.play();
      playButton.textContent = '⏸';
    }
    isPlaying = !isPlaying;
  }



  function updateProgressBar() {
    var progressBarFill = document.querySelector('.progressBarFill');
    progress = (audioElement.currentTime / audioElement.duration) * 100;
    progressBarFill.style.width = progress + '%';
  }

  function resetAudio() {
    audioElement.currentTime = 0;
    progress = 0;
    isPlaying = false;
    playButton.textContent = '▶';
  }

  var audioElement = new Audio('audio.mp3');
  audioElement.addEventListener('timeupdate', updateProgressBar);
  audioElement.addEventListener('ended', resetAudio);
  var isPlaying = false;
  var progress = 0;
</script>
</body>

</html>

```

在html中将logo、info、audio存放在一个名为 `container` 的div标签中，方便使用css对它进行美化
同时使用JavaScript代码添加了点击后会在  和  的按钮（当播放音频完毕后按钮会重置为初始状态），添加了展示音频播放进度的进度条

因为本人对于网页的外观有一定的需求，固使用css对上述页面进行美化

```
<style>
  body {
    display: flex;
    justify-content: center;
    align-items: center;
  }

  .container {
    display: flex;
    flex: 1;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    width: 50vw;
    height: auto;
    padding: 40px;
    background-color: #edf4fb;
    border-radius: 10px;
    margin-top: 5vh;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
  }

  .logo {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    width: auto;
    height: 15vh;
    background-color: white;
    color: rgb(0, 0, 0);
    border-radius: 10px;
    padding: 10px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
  }

  .logo img {
    width: 100%;
    height: 100%;
    border-radius: 10px;
  }

  .info {
```

```

display: flex;
flex-direction: column;
align-items: center;
justify-content: center;
width: 30%;
height: 10%;
background-color: white;
color: rgb(0, 0, 0);
border-radius: 10px;
padding: 20px;
margin: 20px;
box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
overflow: hidden;
}

.info p {
font-size: inherit;
margin: 5px;
padding: 0;
white-space: nowrap;
overflow: hidden;
text-overflow: ellipsis;
}

.title {
font-size: calc(1em + 3px);
font-weight: bold;
margin-top: 10px;
}

.github {
display: flex;
flex-direction: column;
align-items: center;
justify-content: center;
width: auto;
background-color: white;
border-radius: 10px;
border: none;
outline: none;
padding: 10px;
margin-bottom: 20px;
box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

cursor: pointer;
}

.github:focus,
.github:hover {
background-color: rgb(231, 231, 231);
}

a.github {
text-decoration: none;
color: black;
}

```

```
}

.audio {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  width: 40%;
  height: 5%;
  background-color: white;
  color: rgb(0, 0, 0);
  border-radius: 10px;
  padding: 20px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.audio p {
  font-size: larger;
  padding-bottom: 20px;
  padding: 0;
}

.audio button {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  margin-bottom: 20px;
  background-color: white;
  border-radius: 10px;
  padding: 10px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

  font-size: larger;
  outline: none;
  border: none;
}

.progressBar {
  width: 80%;
  height: 10px;
  background-color: #ccc;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
  margin-top: 10px;
  border-radius: 10px;
}

.progressBarFill {
  height: 100%;
  width: 0%;
  background-color: #c0daf7;
  border-radius: 10px;
}
</style>
```

三、抓包过程与分析

前期准备

为了方便进行抓包，我使用计算机A中部署web服务器并且在计算机B访问网页并且进行抓包

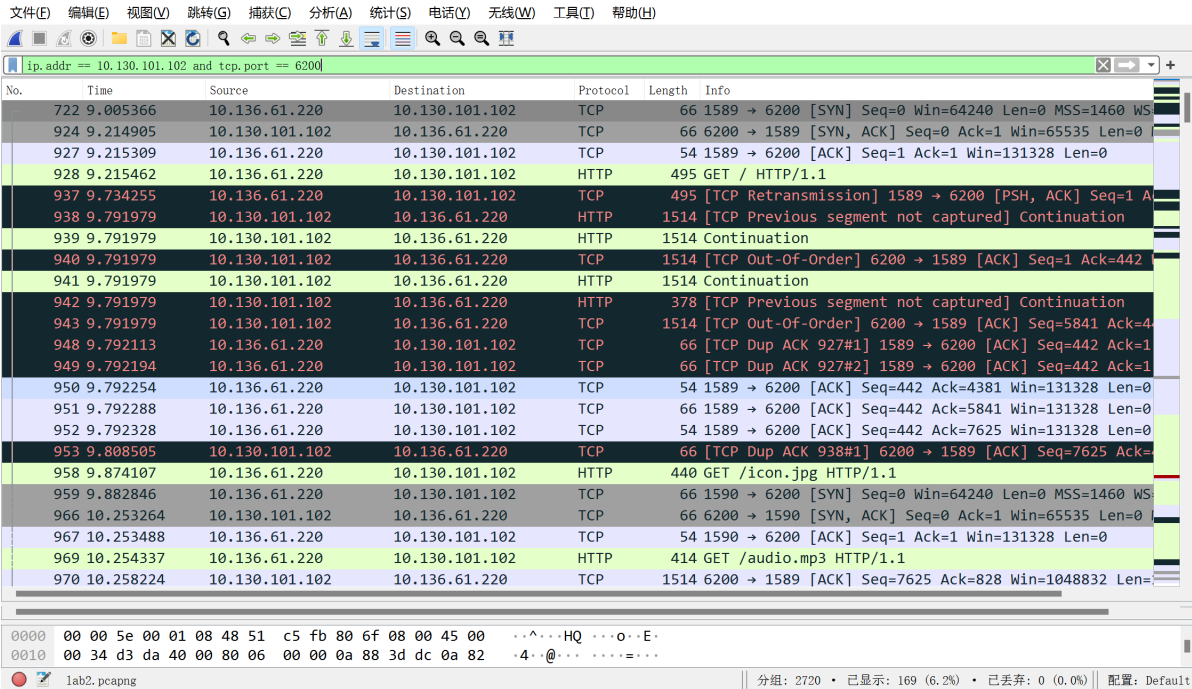
计算机A的ipv4地址为 10.130.101.102，开放端口 6200 用于部署web服务器

打开wireshark后在计算机B访问 http://10.130.101.102:6200,等待一段时间后关闭抓包并且对已经抓好的包进行分析

抓包过程

首先在wireshark中对抓好的包进行筛选，除去其他请求和接收的干扰

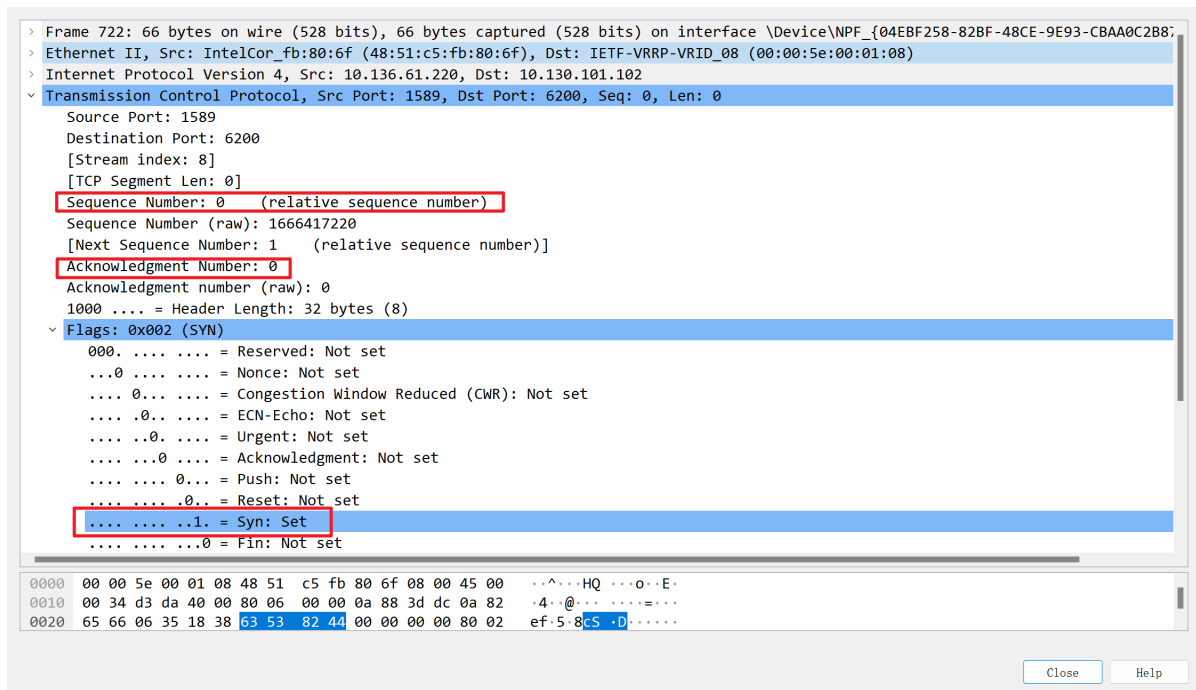
使用 ip.addr == 10.130.101.102 and tcp.port == 6200，只对网页进行抓包



1. 三次握手分析

10.136.61.220	10.130.101.102	TCP	66	1589 → 6200 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS
10.130.101.102	10.136.61.220	TCP	66	6200 → 1589 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
10.136.61.220	10.130.101.102	TCP	54	1589 → 6200 [ACK] Seq=1 Ack=1 Win=131328 Len=0

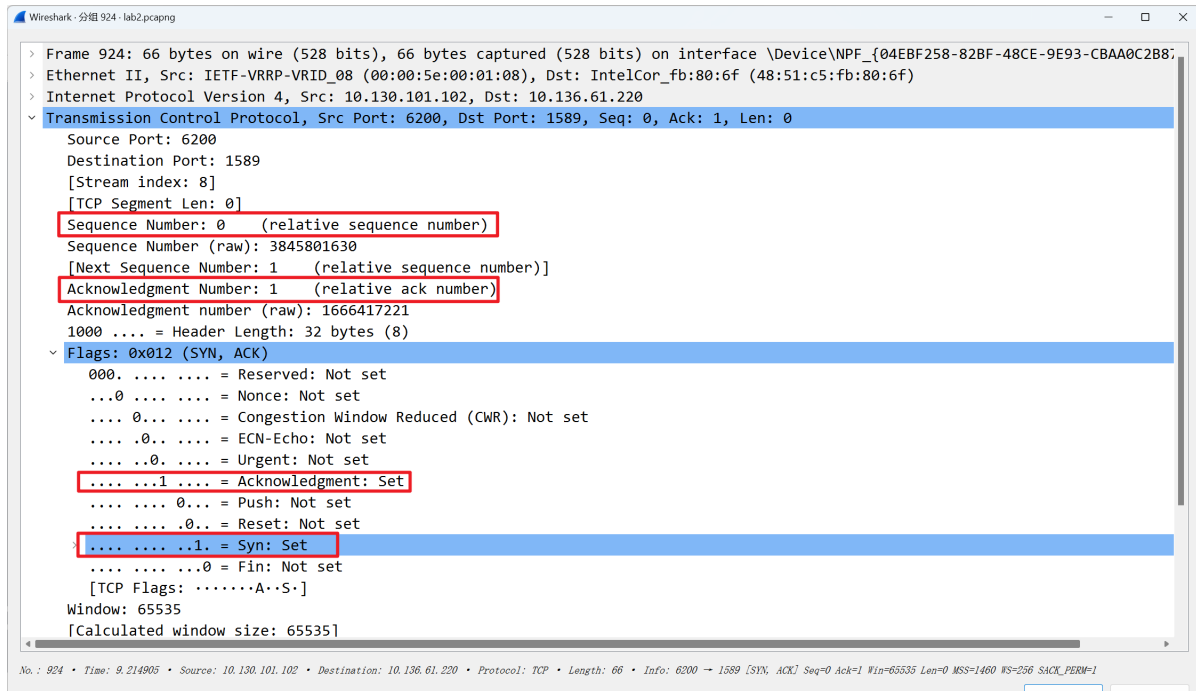
对于第一次连接：



第一次连接是客户端主动要连接服务端的，可以看到传输控制协议里Seq=0(Seq是序列号)，代表初次连接，Ack=0(确认码)，初次连接为0

除了此之外，还要给标志位，也就是flags=初次连接需要给SYN=1的标志位表示请求建立连接。

对于第二次连接：



第二次握手是服务端的回馈6200端口给1589的端口数据，初次连接所以Seq=0，Ack=上一次客户端的序列号+1；

标志位是SYN=1和ACK=1，代表这是一个确认的回馈连接

对于第三次连接：


```
> Frame 927: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_{04EBF258-82BF-48CE-9E93-CBAA0C2B87}
> Ethernet II, Src: IntelCor_fb:80:6f (48:51:c5:fb:80:6f), Dst: IETF-VRRP-VRID_08 (00:00:5e:00:01:08)
> Internet Protocol Version 4, Src: 10.136.61.220, Dst: 10.130.101.102
> Transmission Control Protocol, Src Port: 1589, Dst Port: 6200, Seq: 1, Ack: 1, Len: 0
  Source Port: 1589
  Destination Port: 6200
  [Stream index: 8]
  [TCP Segment Len: 0]
  Sequence Number: 1 (relative sequence number)
  Sequence Number (raw): 1666417221
  [Next Sequence Number: 1 (relative sequence number)]
  Acknowledgment Number: 1 (relative ack number)
  Acknowledgment number (raw): 3845801631
  0101 .... = Header Length: 20 bytes (5)
  Flags: 0x010 (ACK)
    000. .... = Reserved: Not set
    ...0 .... = Nonce: Not set
    .... 0... = Congestion Window Reduced (CWR): Not set
    .... .0.. = ECN-Echo: Not set
    .... ..0. = Urgent: Not set
    .... ...1 = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
    .... .... ..0. = Syn: Not set
    .... .... ...0 = Fin: Not set
    [TCP Flags: .....A....]
  Window: 513
  [calculated window size: 131328]
```

客户端1589给6200服务端的反馈，Seq=1,因为这是客户端的第二次交互了，Ack=上一次服务端连接的序列号+1

标志位为：ACK，表示确认收到了连接回复，三次握手就建立连接完毕

2. HTTP分析

10.136.61.220	10.130.101.102	HTTP	495 GET / HTTP/1.1
10.136.61.220	10.130.101.102	TCP	495 [TCP Retransmission] 1589 → 6200 [PSH, ACK] Seq=1 A
10.130.101.102	10.136.61.220	HTTP	1514 [TCP Previous segment not captured] Continuation
10.130.101.102	10.136.61.220	HTTP	1514 Continuation
10.130.101.102	10.136.61.220	TCP	1514 [TCP Out-Of-Order] 6200 → 1589 [ACK] Seq=1 Ack=442
10.130.101.102	10.136.61.220	HTTP	1514 Continuation
10.130.101.102	10.136.61.220	HTTP	378 [TCP Previous segment not captured] Continuation
10.130.101.102	10.136.61.220	TCP	1514 [TCP Out-Of-Order] 6200 → 1589 [ACK] Seq=5841 Ack=4
10.136.61.220	10.130.101.102	TCP	66 [TCP Dup ACK 927#1] 1589 → 6200 [ACK] Seq=442 Ack=1
10.136.61.220	10.130.101.102	TCP	66 [TCP Dup ACK 927#2] 1589 → 6200 [ACK] Seq=442 Ack=1
10.136.61.220	10.130.101.102	TCP	54 1589 → 6200 [ACK] Seq=442 Ack=4381 Win=131328 Len=0
10.136.61.220	10.130.101.102	TCP	66 1589 → 6200 [ACK] Seq=442 Ack=5841 Win=131328 Len=0
10.136.61.220	10.130.101.102	TCP	54 1589 → 6200 [ACK] Seq=442 Ack=7625 Win=131328 Len=0
10.130.101.102	10.136.61.220	TCP	66 [TCP Dup ACK 938#1] 6200 → 1589 [ACK] Seq=7625 Ack=

- 对于第一个HTTP的GET请求，是建立HTTP连接，传输HTML页面内容
 - 补充：如果非初次连接，服务端可能会向客户端发送 304 Not Modified，意味着客户端缓存的资源仍然是最新的，并且与服务器上的资源相同，服务器将返回 HTTP 304 Not Modified 状态码。这表示服务器并没有返回请求的资源，而是返回一个空包和 304 状态码告诉客户端它可以继续使用它们本地缓存的内容。这对于减少网络流量和提高性能非常有用，因为此时客户端可以避免不必要的数据传输。
- "Continuation"是HTTP/1.1协议的一种扩展机制，用于支持分块传输编码（chunked transfer encoding）。分块传输编码是一种将数据分为若干块进行逐步传输的机制，可以在数据还未完全传输完成时就开始处理部分数据。当使用分块传输编码时，每个块都包含一个大小字段和块数据，而"continuation"则表示后续的块数据。

当显示"Continuation"时，意味着当前数据包仅包含了实体数据的一部分，后续的数据会在接下来的数据包中传输。这样可以使得数据的传输和处理更灵活，并提高传输效率。

- 黑红色的抓包数据代表了一些协议错误或异常情况，和实验分析无关，固按下不表

- 对于三个紫色的1589向6200发送的ACK请求，表明客户端正确收到了服务端发送的数据，并且向服务端进行说明

10.136.61.220	10.130.101.102	HTTP	440 GET /icon.jpg HTTP/1.1
10.136.61.220	10.130.101.102	TCP	66 1590 → 6200 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS
10.130.101.102	10.136.61.220	TCP	66 6200 → 1590 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
10.136.61.220	10.130.101.102	TCP	54 1590 → 6200 [ACK] Seq=1 Ack=1 Win=131328 Len=0
10.136.61.220	10.130.101.102	HTTP	414 GET /audio.mp3 HTTP/1.1
10.130.101.102	10.136.61.220	TCP	1514 6200 → 1589 [ACK] Seq=7625 Ack=828 Win=1048832 Len=
10.130.101.102	10.136.61.220	TCP	1514 6200 → 1589 [ACK] Seq=9085 Ack=828 Win=1048832 Len=
10.130.101.102	10.136.61.220	TCP	1514 6200 → 1589 [ACK] Seq=10545 Ack=828 Win=1048832 Len
10.130.101.102	10.136.61.220	TCP	1514 6200 → 1589 [ACK] Seq=12005 Ack=828 Win=1048832 Len
10.130.101.102	10.136.61.220	TCP	1514 6200 → 1589 [ACK] Seq=13465 Ack=828 Win=1048832 Len
10.130.101.102	10.136.61.220	TCP	1514 6200 → 1589 [ACK] Seq=14925 Ack=828 Win=1048832 Len
10.130.101.102	10.136.61.220	TCP	1514 6200 → 1589 [ACK] Seq=16385 Ack=828 Win=1048832 Len
10.130.101.102	10.136.61.220	TCP	1514 6200 → 1589 [ACK] Seq=17845 Ack=828 Win=1048832 Len
10.130.101.102	10.136.61.220	TCP	1514 6200 → 1589 [ACK] Seq=19305 Ack=828 Win=1048832 Len
10.130.101.102	10.136.61.220	TCP	1514 6200 → 1589 [ACK] Seq=20765 Ack=828 Win=1048832 Len
10.130.101.102	10.136.61.220	TCP	1514 6200 → 1589 [ACK] Seq=22225 Ack=828 Win=1048832 Len
10.130.101.102	10.136.61.220	TCP	1514 6200 → 1589 [ACK] Seq=23685 Ack=828 Win=1048832 Len
10.130.101.102	10.136.61.220	TCP	1514 6200 → 1589 [ACK] Seq=25145 Ack=828 Win=1048832 Len
10.130.101.102	10.136.61.220	TCP	1514 6200 → 1589 [ACK] Seq=26605 Ack=828 Win=1048832 Len
10.130.101.102	10.136.61.220	TCP	1514 6200 → 1589 [ACK] Seq=28065 Ack=828 Win=1048832 Len
10.136.61.220	10.130.101.102	TCP	54 1589 → 6200 [ACK] Seq=828 Ack=29525 Win=131328 Len=

- 两个GET请求分别是请求logo图片和自我介绍音频
- 对于又出现一次三次握手，是客户端又开了一个线程进行数据接收
- 对于多条6200->1589的ACK请求，是服务端将一个数据文件进行分块传输，而最后一条1589->6200则是客户端向服务端发送数据成功接收的信息

3. 四次挥手分析

10.130.101.102	10.136.61.220	TCP	56 6200 → 1590 [FIN, ACK] Seq=104084 Ack=361 Win=10493
10.136.61.220	10.130.101.102	TCP	54 1590 → 6200 [ACK] Seq=361 Ack=104085 Win=131328 Len=
10.136.61.220	10.130.101.102	TCP	54 1590 → 6200 [FIN, ACK] Seq=361 Ack=104085 Win=13132
10.130.101.102	10.136.61.220	TCP	56 6200 → 1590 [ACK] Seq=104085 Ack=362 Win=1049344 Lei

- 在第一次挥手中，服务端发送了一个带有 [FIN, ACK] 标志位的包给客户端，表示服务端已经完成数据的发送，并要求关闭连接。同时，服务端还确认了客户端发送的序列号为1487的数据。
- 客户端接收到服务端的第一次挥手后，发送一个带有 [ACK] 标志位的确认包给服务端，确认服务端的请求，并告知服务端客户端已经准备好关闭连接。
- 客户端接收到服务端的确认后，发送一个带有 [FIN, ACK] 标志位的包给服务端，表示客户端也准备关闭连接。
- 服务端接收到客户端的第三次挥手后，发送一个带有 [ACK] 标志位的确认包给客户端，确认客户端的请求，并告知客户端服务器端也准备好关闭连接。

通过以上四次挥手，双方成功地关闭了TCP连接。

四、Web运行结果展示

