计算机网络课程实验报告

姓名：汤志文

学号：2111441

专业：物联网工程

实验 2：配置 Web 服务器

一、实验内容：

（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、实验准备：

学习实验所需的相关理论知识：

html 语言，css 样式,javascript 脚本语言。

学习如何搭建 web 服务器。

学习 wireshark 抓包过程。

三、实验过程：

（1）web 服务器启动验证。

编写简单的 html 文件和 js 文件尝试启动 web 服务器，使用 Express 框架创建一个简单的 Web 服务器是一种常见的做法。

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
  res.sendFile(__dirname + '/public/index.html');
});


app.listen(port, () => {
  console.log(`Server is running on http://localhost:${port}`);
});
```

```html
<!DOCTYPE html>
<html>
<head>
  <title>My Web Page</title>
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
  <h1>Welcome to My Web Page</h1>
  <p>This is a simple web page.</p>
  <script src="script.js"></script>
</body>
</html>
```

（2）html 编写

编写符合实验要求的 html 文件：

根据微信的页面，编写了一个类似的 web 页面，采用分文件编写，其中包含 5 个 html 页面。

代码框架及部分分析：

1. html 部分代码：

编写页面，并使用 css 对页面的样式进行编写，这里仅展示了一部分代码，剩余代码在 public 文件夹下：

```html
<li>
    <div class="img">
        <img src="img/马云.jpg"/>
    </div>
    <div class="txt">
        <p><b>马云</b></p>
        <p>年轻人，我看好你，来阿里巴巴工作吗</p>
    </div>
    <div class="date">
        13:10
    </div>
</li>
<li>
    <div class="img">
        <img src="img/头像.jpg"/>
    </div>
    <div class="txt">
        <p><b>汤志文 2111441 物联网工程</b></p>
        <audio controls>
            <source src="1.mp3" type="audio/mpeg">
            Your browser does not support the audio element.
        </audio>
```

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <style>
            .wrap{
                display:flex;
                flex-direction:column;
                height:100vh;
            }
            *{
                padding:0;
                margin:0;
            }
            .body{
                font-size:20px;
            }
            .wrap>.top>p{
                font-size:2em;
                height:100%;
                line-height:150px;
                font-weight: bold;
                padding-left:450px;
            }
            .wrap>.top{
                height:150px;
                background:#f2f2f2;

            }
            .wrap>.content{
                flex:1;
            }
```

2. server.js 服务器文件：

```javascript
// 设置端口号
const port = 3000;

// 获取主机IPv4地址
function getIPv4Address() {
    const networkInterfaces = os.networkInterfaces();
    let ipAddress;

    for (const interfaceName of Object.keys(networkInterfaces)) {
        for (const networkInterface of networkInterfaces[interfaceName]) {
            if (
                !networkInterface.internal &&
                networkInterface.family === 'IPv4' &&
                (interfaceName.includes('WLAN') || interfaceName.includes('ETH'))
            ) {
                ipAddress = networkInterface.address;
                return ipAddress;
            }
        }
    }
}

// 设置静态文件目录
app.use(express.static(__dirname + '/public'));

// 定义路由处理器
app.get('/', (req, res) => {
    res.sendFile(__dirname + '/public/wechat.html');
});
```

实验效果演示：

运行 server.js 启动 web 服务器

```
E:\11大三上课程\计网\work2\my-web-server>node server.js
Server is running on http://localhost:3000
http://10.130.3.36:3000
```

在浏览器中输入 http://10.130.3.36:3000 访问 web 服务器：

验证播放音频功能：

wireshark 抓包分析：

启动服务器并访问，使用 wireshark 进行抓包并使用筛选条件（ip.addr==10.130.3.36 and tcp.port==3000）：

No.	Time	Source	Destination	Protocol	Length	Info
8887	44.890785	10.130.27.249	10.130.3.36	TCP	66	64283 → 3000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
8888	44.890853	10.130.3.36	10.130.27.249	TCP	66	3000 → 64283 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
8889	44.891750	10.130.27.249	10.130.3.36	TCP	66	64284 → 3000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
8890	44.891793	10.130.3.36	10.130.27.249	TCP	66	3000 → 64284 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
8891	44.891839	10.130.3.36	10.130.27.249	HTTP/X…	1115	HTTP/1.1 200 OK
8892	44.892583	10.130.3.36	10.130.27.249	TCP	1514	3000 → 64280 [ACK] Seq=3718 Ack=1235 Win=130048 Len=1460 [TCP segment of a reassembled PDU]
8893	44.892583	10.130.3.36	10.130.27.249	HTTP/X…	1111	HTTP/1.1 200 OK
8894	44.894209	10.130.27.249	10.130.3.36	TCP	66	64285 → 3000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
8895	44.894209	10.130.27.249	10.130.3.36	TCP	66	64286 → 3000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
8896	44.894245	10.130.3.36	10.130.27.249	TCP	66	3000 → 64285 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
8897	44.894274	10.130.3.36	10.130.27.249	TCP	66	3000 → 64286 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
8898	44.894875	10.130.27.249	10.130.3.36	TCP	54	64283 → 3000 [ACK] Seq=1 Ack=1 Win=131328 Len=0
8899	44.894875	10.130.27.249	10.130.3.36	HTTP	470	GET /img/%E5%BE%AE%E4%BF%A1%E4%BF%A1%E6%81%AF.svg HTTP/1.1
8900	44.896019	10.130.3.36	10.130.27.249	TCP	1514	3000 → 64283 [ACK] Seq=1 Ack=417 Win=130816 Len=1460 [TCP segment of a reassembled PDU]
8901	44.896019	10.130.3.36	10.130.27.249	HTTP/X…	323	HTTP/1.1 200 OK
8902	44.896376	10.130.27.249	10.130.3.36	TCP	54	64284 → 3000 [ACK] Seq=1 Ack=1 Win=131328 Len=0
8903	44.896511	10.130.27.249	10.130.3.36	HTTP	461	GET /img/%E9%80%9A%E8%AE%AF%E5%BD%95.svg HTTP/1.1
8904	44.896511	10.130.27.249	10.130.3.36	HTTP	452	GET /img/%E5%8F%91%E7%8E%B0.svg HTTP/1.1
8905	44.897332	10.130.3.36	10.130.27.249	TCP	1514	3000 → 64284 [ACK] Seq=1 Ack=408 Win=130816 Len=1460 [TCP segment of a reassembled PDU]
8906	44.897332	10.130.3.36	10.130.27.249	HTTP/X…	1045	HTTP/1.1 200 OK
8907	44.897489	10.130.27.249	10.130.3.36	TCP	54	64280 → 3000 [ACK] Seq=1235 Ack=6235 Win=131328 Len=0

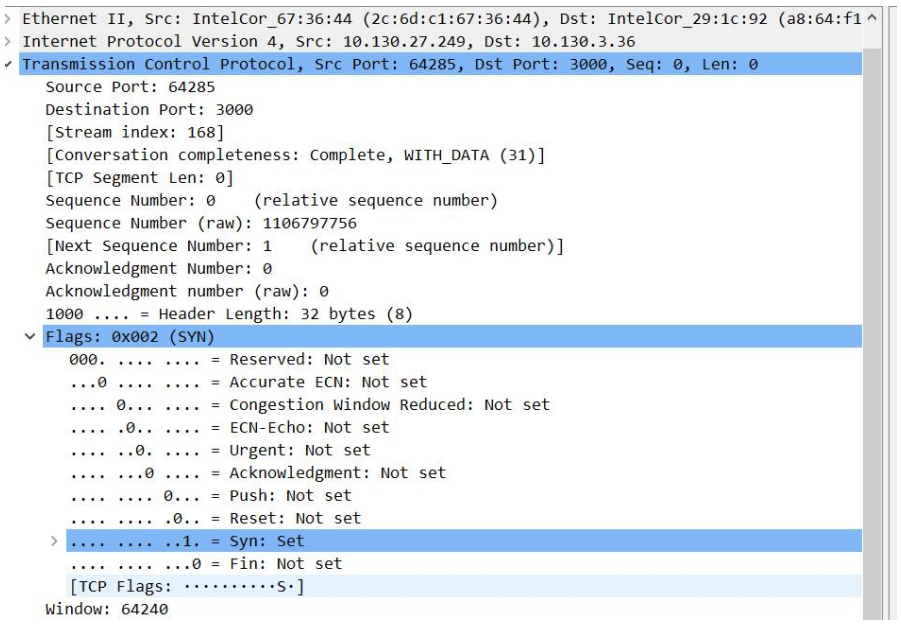
进一步提取信息进行分析

三次握手

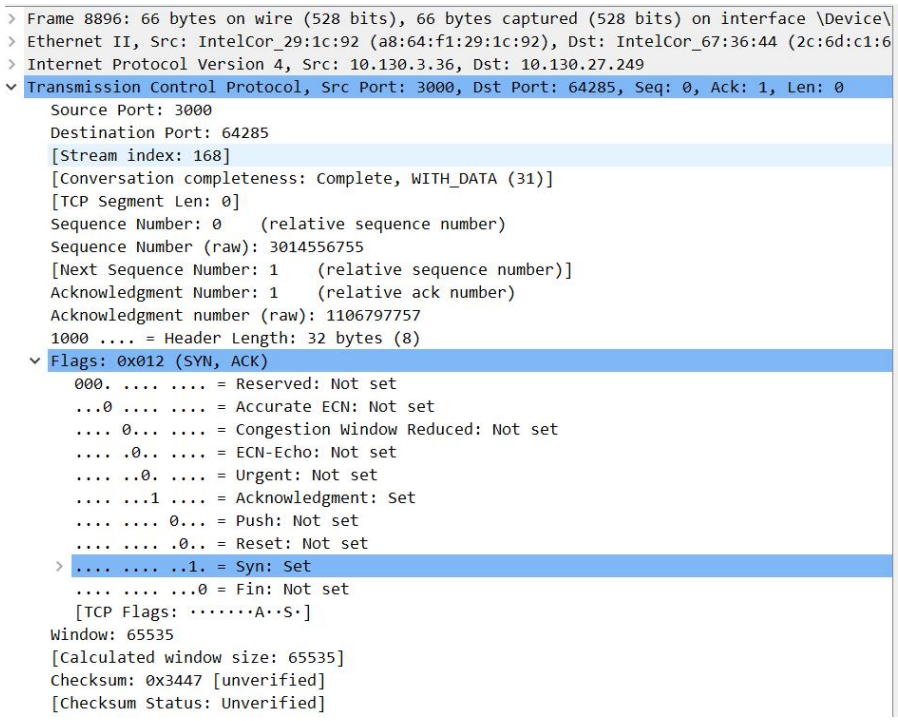
8894	44.894209	10.130.27.249	10.130.3.36	TCP	66	64285 → 3000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
8895	44.894209	10.130.27.249	10.130.3.36	TCP	66	64286 → 3000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
8896	44.894245	10.130.3.36	10.130.27.249	TCP	66	3000 → 64285 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
8897	44.894274	10.130.3.36	10.130.27.249	TCP	66	3000 → 64286 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM

8912	44.901202	10.130.27.249	10.130.3.36	TCP	54	64285 → 3000 [ACK] Seq=1 Ack=1 Win=131328 Len=0
8913	44.902249	10.130.27.249	10.130.3.36	TCP	54	64286 → 3000 [ACK] Seq=1 Ack=1 Win=131328 Len=0

第一次连接，客户端主动连接服务端，seq=0,ack=0



第二次是服务器向客户端发送的数据，代表一个确认连接，SYN=1,ACK=1



第三次连接是客户端向服务端发送的确认，这一步确认连接后，三次握手结束，连接成功建立。

```
> Frame 8912: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Devic
> Ethernet II, Src: IntelCor_67:36:44 (2c:6d:c1:67:36:44), Dst: IntelCor_29:1c:92 (a8:64:f1
> Internet Protocol Version 4, Src: 10.130.27.249, Dst: 10.130.3.36
v Transmission Control Protocol, Src Port: 64285, Dst Port: 3000, Seq: 1, Ack: 1, Len: 0
    Source Port: 64285
    Destination Port: 3000
    [Stream index: 168]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 1106797757
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 3014556756
    0101 .... = Header Length: 20 bytes (5)
  v Flags: 0x010 (ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Accurate ECN: Not set
      .... 0... .... = Congestion Window Reduced: 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]
```

http

8872	44.853756	10.130.27.249	10.130.3.36	HTTP	492	GET / HTTP/1.1
8874	44.863640	10.130.3.36	10.130.27.249	HTTP	1129	HTTP/1.1 200 OK (text/html)
8876	44.871368	10.130.27.249	10.130.3.36	HTTP	452	GET /img/%E4%BF%A1%E5%8F%B7.svg HTTP/1.1
8881	44.875968	10.130.27.249	10.130.3.36	HTTP	452	GET /img/%E7%BD%91%E7%BB%9C.svg HTTP/1.1
8885	44.887357	10.130.27.249	10.130.3.36	HTTP	452	GET /img/%E7%94%B5%E6%B1%A0.svg HTTP/1.1
8886	44.888793	10.130.27.249	10.130.3.36	HTTP	452	GET /img/%E9%97%B9%E9%93%83.svg HTTP/1.1
8899	44.894875	10.130.27.249	10.130.3.36	HTTP	470	GET /img/%E5%BE%AE%E4%BF%A1%E4%BF%A1%E6%81%AF.svg HTTP/1.1
8903	44.896511	10.130.27.249	10.130.3.36	HTTP	461	GET /img/%E9%80%9A%E8%AE%AF%E5%BD%95.svg HTTP/1.1
8904	44.896511	10.130.27.249	10.130.3.36	HTTP	452	GET /img/%E5%8F%91%E7%8E%B0.svg HTTP/1.1
8909	44.898599	10.130.27.249	10.130.3.36	HTTP	443	GET /img/%E6%88%91.svg HTTP/1.1
8914	44.902249	10.130.27.249	10.130.3.36	HTTP	550	GET /%E4%BF%A1%E6%81%AF.html HTTP/1.1
8918	44.902997	10.130.3.36	10.130.27.249	HTTP	1236	HTTP/1.1 200 OK (text/html)
8921	44.915010	10.130.27.249	10.130.3.36	HTTP	475	GET /img/%E6%B7%BB%E5%8A%A0.svg HTTP/1.1
8922	44.915341	10.130.27.249	10.130.3.36	HTTP	475	GET /img/%E6%90%9C%E7%B4%A2.svg HTTP/1.1
8924	44.916033	10.130.27.249	10.130.3.36	HTTP	475	GET /img/%E9%A9%AC%E4%BA%91.jpg HTTP/1.1
8926	44.917098	10.130.27.249	10.130.3.36	HTTP	475	GET /img/%E5%A4%B4%E5%83%8F.jpg HTTP/1.1
8934	44.917145	10.130.3.36	10.130.27.249	HTTP	1091	HTTP/1.1 200 OK (JPEG JFIF image)
8946	44.918503	10.130.27.249	10.130.3.36	HTTP	427	GET /1.mp3 HTTP/1.1
9012	44.931198	10.130.3.36	10.130.27.249	HTTP	204	HTTP/1.1 200 OK (JPEG JFIF image)
9128	44.948372	10.130.3.36	10.130.27.249	HTTP	313	HTTP/1.1 206 Partial Content (audio/mpeg)
9134	44.966020	10.130.27.249	10.130.3.36	HTTP	437	GET /favicon.ico HTTP/1.1

对于第一个 get 请求，建立 http 连接，传输 html 内容。

一系列 get 将页面所需的图片及音频进行传输，以支持 web 页面的正常功能。

四次挥手

9213	49.928775	10.130.3.36	10.130.27.249	TCP	54	3000 → 64284 [FIN, ACK] Seq=63922 Ack=829 Win=130560 Len=0
9217	49.932372	10.130.27.249	10.130.3.36	TCP	54	64284 → 3000 [ACK] Seq=829 Ack=63923 Win=131328 Len=0
9218	49.974145	10.130.3.36	10.130.27.249	TCP	54	3000 → 64283 [FIN, ACK] Seq=196591 Ack=1173 Win=130048 Len=0
9219	49.975923	10.130.27.249	10.130.3.36	TCP	54	64283 → 3000 [ACK] Seq=1173 Ack=196592 Win=130816 Len=0

在第一次挥手中，客户端发送了一个带有[FIN,ACK]标志位的包给服务器，表示客户端已经完成数据的发送，并要求关闭连接。

服务器接收到客户端的第一次挥手后，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已经准备好关闭连接。

客户端接收到服务器的确认后，发送一个带有[FIN,ACK] 标志位的包给服务器，表示客户端也准备关闭连接

服务器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

四、实验感悟：

网络通信基础：通过编写 Web 服务器，我深入了解了 HTTP 协议和 TCP/IP 协议栈。我更加理解了客户端和服务器之间的通信方式，包括握手、请求和响应的过程。

Web 服务器的工作原理：实现一个简单的 Web 服务器让我了解了服务器如何接受 HTTP 请求、解析请求头、处理请求并发送响应。

抓包工具的使用：Wireshark 是一款功能强大的网络分析工具，可以捕获和分析网络数据包。通过使用 Wireshark，我观察到了 HTTP 请求和响应的详细信息，包括请求头、响应头和传输的数据。

协议的细节：在编写 Web 服务器和使用 Wireshark 抓包时，我注意到了 HTTP 请求和响应的各种字段，如请求方法、状态码、标头字段等。