计算机网络实验2

姓名：冯朝芃 学号：2012039

实验准备：

网页编写：

编写html网页如下所示，包含个人信息文本及jpeg格式图片一张。

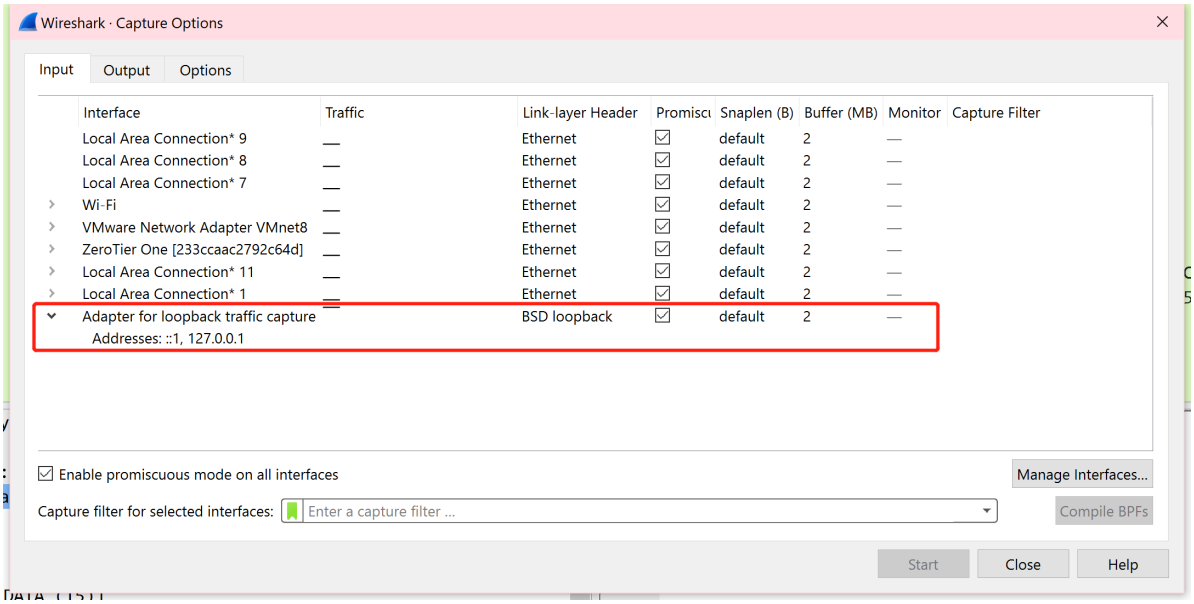
```
<!doctype html>
<html>
<head>
<meta charset='UTF-8'><meta name='viewport' content='width=device-width initial-scale=1'>
<title>我的网页</title></head>
<body><p>专业：计算机科学与技术</p>
<p>姓名：冯朝芃</p>
<p>学号：2012039</p>
<p><img src="./xiaohui.jpeg" referrerpolicy="no-referrer" alt="xiaohui"></p>
</body>
</html>
```

服务器搭建：

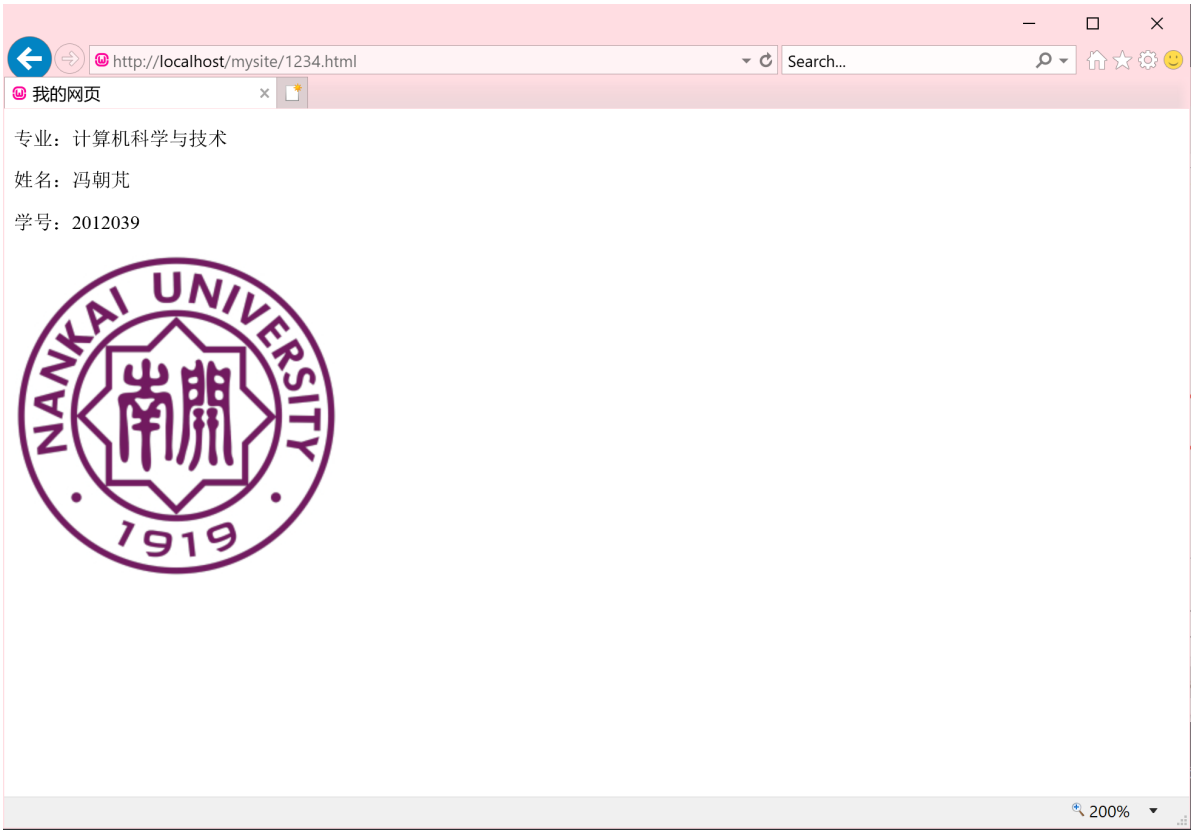
使用Wampserver软件搭建服务器环境，配置Apache2.4服务器，服务端口开放在默认的80端口上。将网页和图片放置于对应的项目文件夹中。

抓包实验：

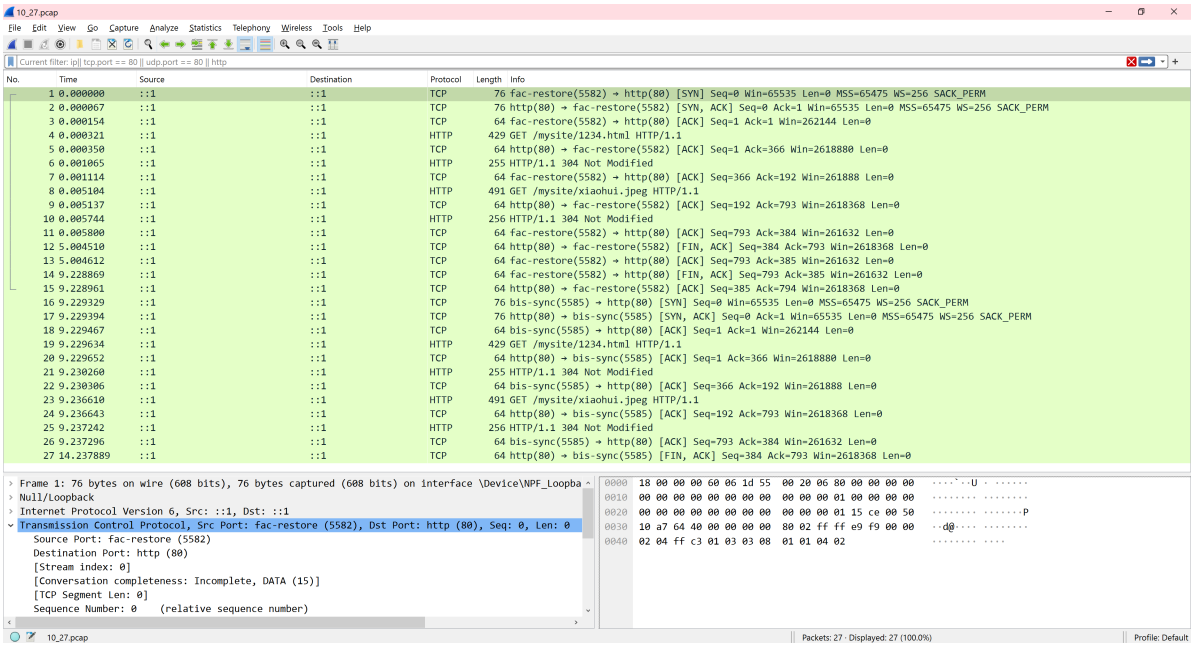
使用Wireshark4.0软件进行网络数据包抓取，选择如图所示的抓包选项，监听localhost。



抓包开始后，使用IE11浏览器访问自建网站，等待一会使服务器与浏览器断开连接，之后再刷新一次，使浏览器与服务器主动断开连接。浏览器打开页面后效果如图所示。



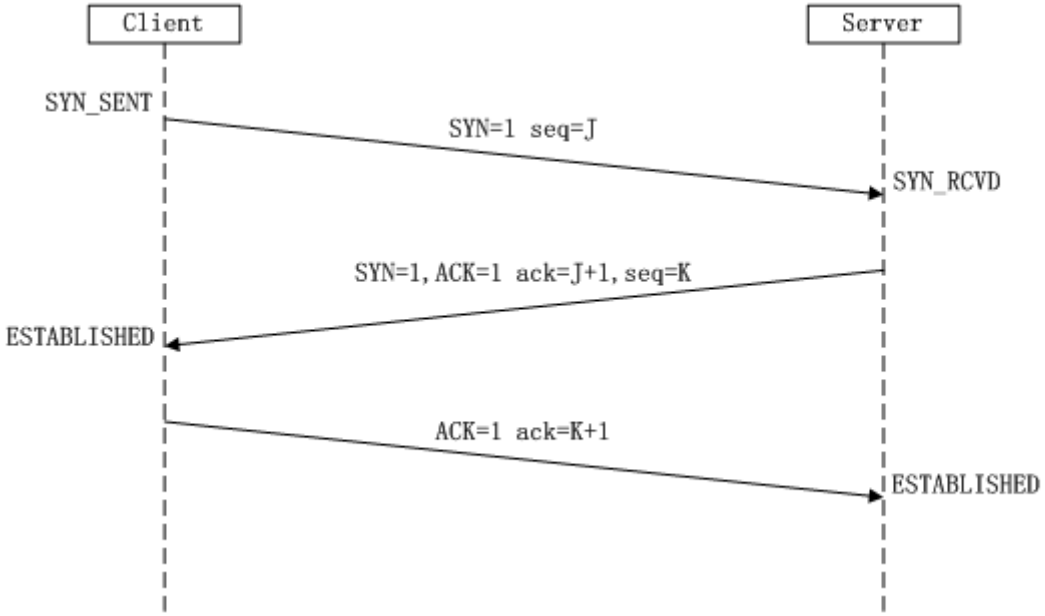
抓包完成后界面如下图所示，从时间上可以很清晰的看到：以0s开头的是三次握手和http请求的数据包，以5s开头的是服务器端超时与浏览器挥手断开连接的数据包，以9s开头的是浏览器刷新网页，与服务器挥手断连的数据包。Wireshark软件也在左侧以灰色线框标注出了一次完整的TCP连接过程。



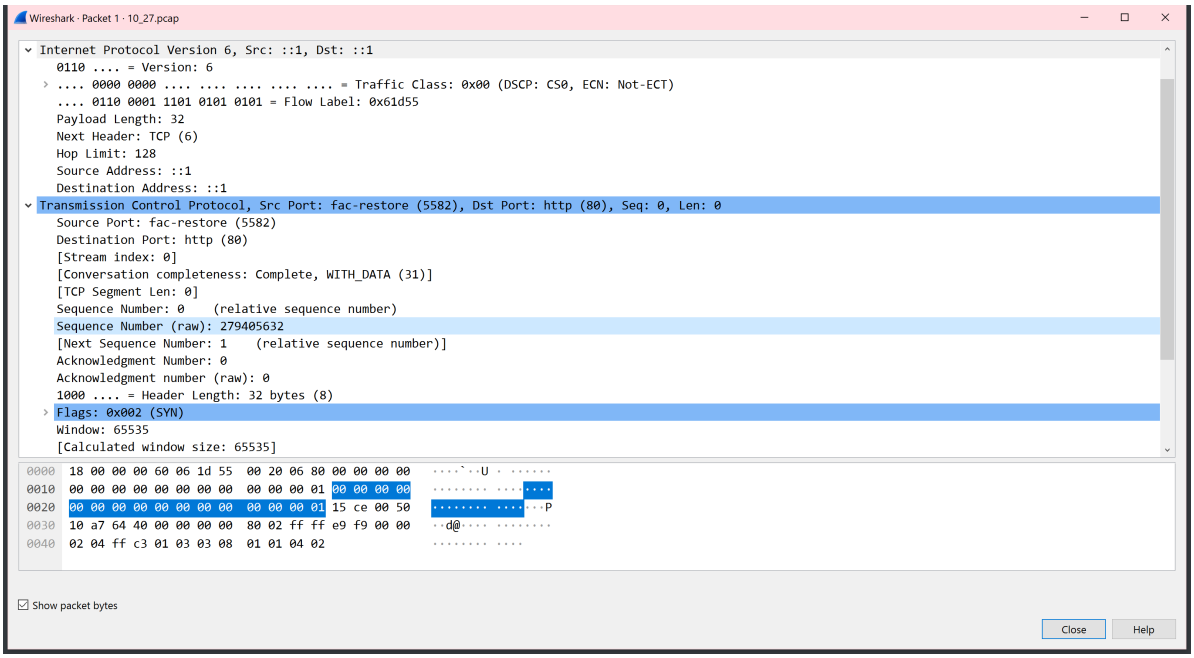
以下按时间顺序分析TCP握手、HTTP请求和响应、TCP挥手的数据包内容。

TCP握手：

TCP三次握手（Three-Way Handshake）即建立TCP连接，指建立一个TCP连接时，需要客户端和服务端总共发送3个包以确认连接的建立。图示如下。



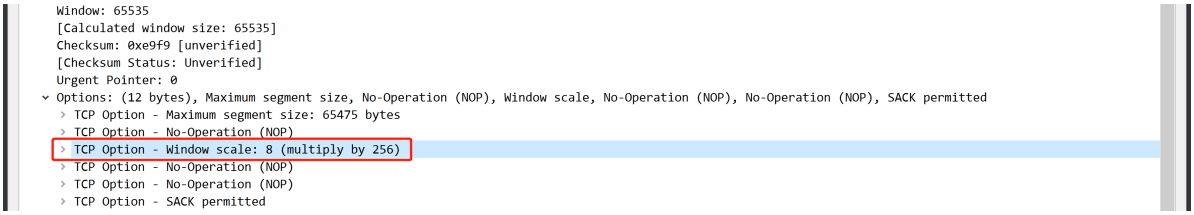
第一次握手：Client将标志位SYN置为1，随机产生一个序号seq=J，并将该数据包发送给Server，Client进入SYN_SENT状态，等待Server确认。展示数据包如下。



TCP是一种流式协议，而TCP流（TCP STREAM）可以用来形容建连-传输-挥手这个过程，Wireshark也采用了这种说法，故在之后的分析中也采用这种说法。观察图片内容可以看到，这个数据包的随机序列号是279405632，相对于整个流中客户端所发送的数据包是第0个数据包。Flags标志位只有SYN被置1，这说明它是TCP握手中的第一个数据包（第一次握手）——客户端程序向服务器发送建连请求。

数据包中还有一些其他信息。如发送端口是5582，目标端口是80，由于本机启用了ipv6，故目标和来源IP都是::1，也就是localhost。此时的确认号是0，因为这个TCP流中没有上一个数据包。该TCP首部的长度是32bytes。

可以看到TCP首部中有一个Window Size字段，它其实是接收窗口（滑动窗口）的大小。用来告知发送端自己所能接收的数据量，从而达到一部分流控的目的，与拥塞控制有关。该数据包Window大小为65535（16bits能表示最大值），在下图中我们可以看到option字段有一个缩放系数，这里是8，要左移八位，也就是说接收窗口为$65535 * 256 = 16,776,960 bytes$；缩放系数如果以后不调整就固定了。发送方的缩放系数和接收方的乘积因子可以不同，由各自决定，观察后续数据包发现这里两方恰巧是一样的。Window大小是动态调整的，后续我们可以看到很多动态调整的例子。



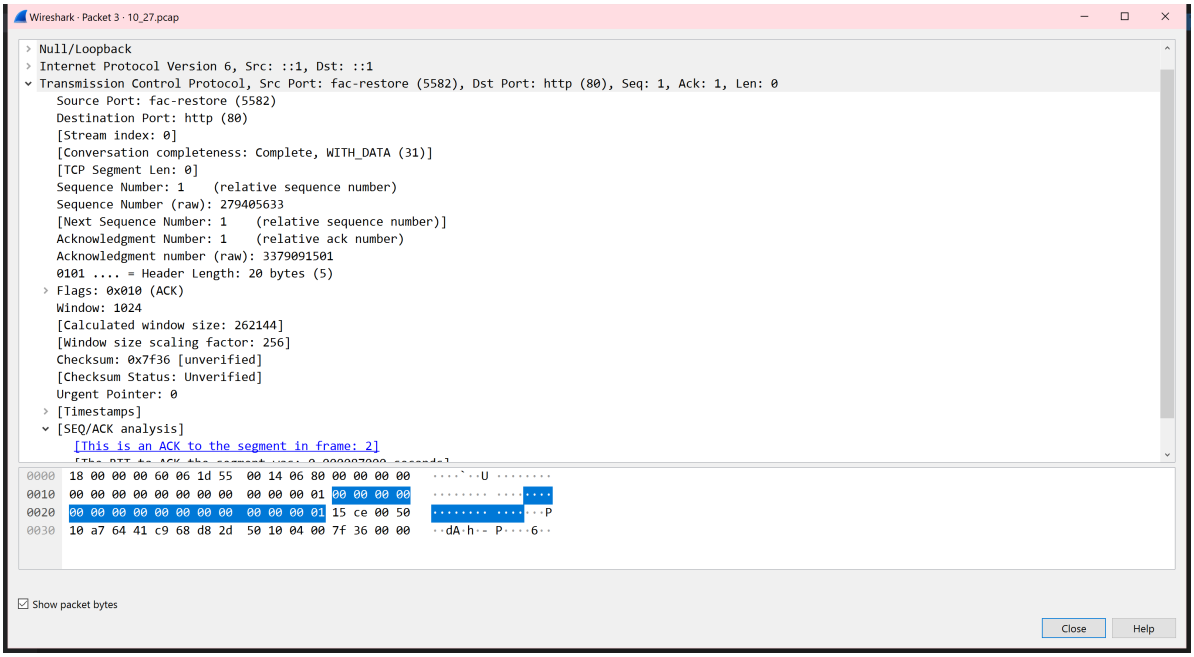
此时ACK为0，事实上TCP规定除了在最初建立连接时候的SYN包之外该位必须设置为1，该位为1时，确认应答的字段为有效。

第二次握手：Server收到数据包后由标志位SYN=1知道Client请求建立连接，Server将标志位SYN和ACK都置为1，确认号=J+1，随机产生一个序号seq=K，并将该数据包发送给Client以确认连接请求，Server进入SYN_RCVD状态。展示如下。

可以看到这个数据包从服务器端80端口发往浏览器5582端口。它的确认号原始值为279405633，是上面提到的J+1，也就是说“279405633-1=279405632被我收到了”。此时服务器生成了一个新的随机的序列号K，3379091500。要注意的，在整个TCP流中，浏览器和服务器各自作为发送端的时候，使用不同的起始序列号，分别为K和J。

此时Flags标志位的SYN和ACK都被置1，这说明是第二次握手的数据包。

第三次握手：Client收到确认后，检查确认号是否为J+1，ACK是否为1，如果正确则将标志位ACK置为1，确认号=K+1，并将该数据包发送给Server，Server检查确认号是否为K+1，ACK是否为1，如果正确则连接建立成功，Client和Server进入ESTABLISHED状态，完成三次握手，随后Client与Server之间可以开始传输数据了。截图如下。



注意到客户端发送的此数据包序列号是J+1，确认的是K+1（表示K及之前的被我收到了）。Flags标志位的ACK被置1。Window大小此时为262144bytes，也就是客户端说“我现在最多只能接受262144bytes”。

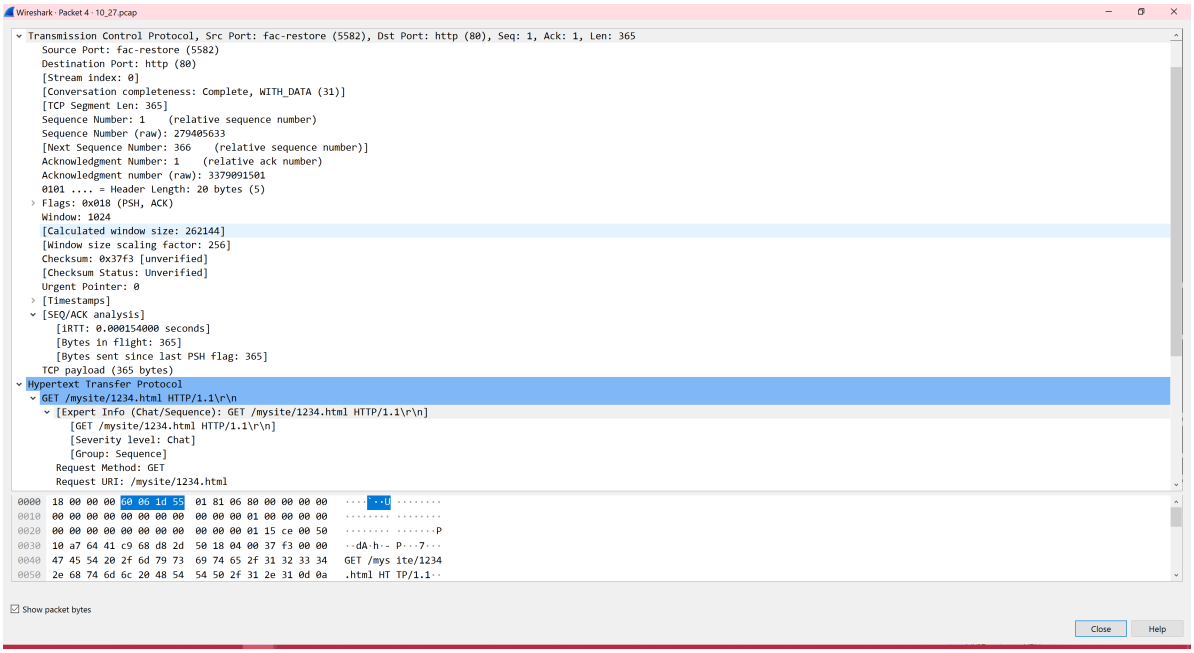
以上三个数据包中的数据段都是没有的。

值得注意一点，只是发送ACK并不占用序列号，而序列号一般是跟着要发送的字节走的，可以留意一下，后面的数据包浏览器将从J+1开始发送，服务器将从K+1开始发送。

HTTP请求和响应：

1

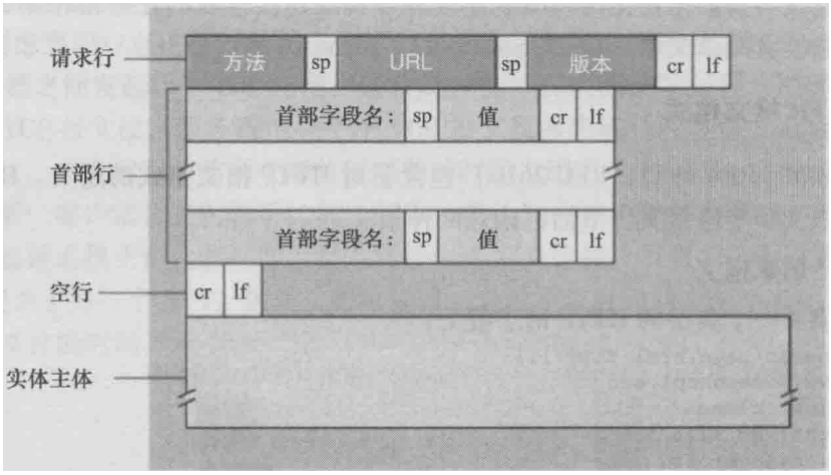
建立连接后，浏览器发送的第一个数据包包含一个HTTP GET请求。截图如下。



TCP首部部分的序列号为J+1，确认序列号为K+1。Flags字段给出了ACK和PSH，PSH表示接收方应该尽快将这个报文交给应用层。

TCP层将HTTP报文放在了TCP数据部分，长度365bytes。

HTTP请求报文格式如下：



请求报文由三部分组成：请求行(request line)、首部行(header line)、实体体(entity body)。

请求行有三个字段：方法、URL、HTTP版本

（1）方法：可以取不同的值，包括GET、POST、HEAD、PUT和DELETE等。该报文使用GET方法拉取HTML网页。

（2）URL：请求对象的标识。这里请求对象标识就是：/mysite/1234.html

（3）HTTP版本：这里HTTP版本为1.1。

首部行由多组键值对（首部字段名：首部字段值）组成。这里包括：

User-Agent：浏览器类型。

Host：初始URL中的主机和端口。

Accept：浏览器可接受的MIME类型，也就是数据类型信息，这里是text/html, application/xhtml+xml, image/jxr, */*。

Accept-Encoding：浏览器能够进行解码的数据编码方式，比如gzip。Servlet能够向支持gzip的浏览器返回经gzip编码的HTML页面。许多情形下这可以减少5到10倍的下载时间。

Accept-Language：浏览器所希望的语言种类。

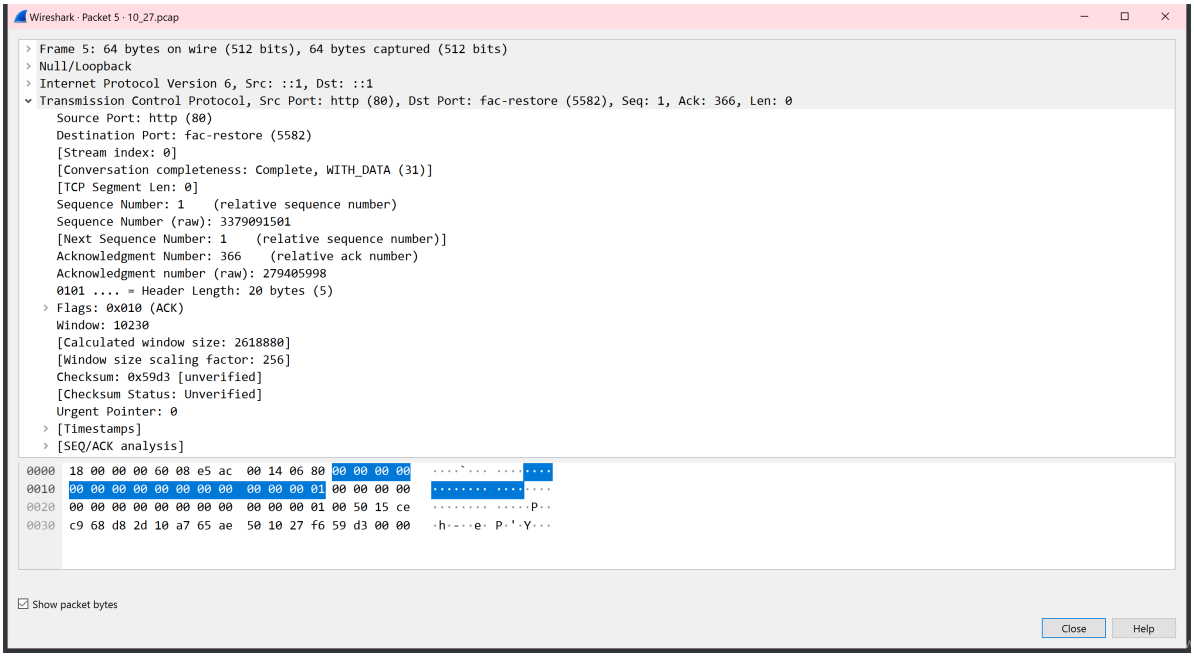
Connection：表示是否需要持久连接，这里是HTTP1.1默认就是持久连接。如果Servlet看到这里的值为“Keep-Alive”，或者看到请求使用的是HTTP 1.1（HTTP 1.1默认进行持久连接），它就可以利用持久连接的优点，当页面包含多个元素时（例如Applet，图片），显著地减少下载所需要的时间。要实现这一点，Servlet需要在应答中发送一个Content-Length头，最简单的实现方法是：先把内容写入ByteArrayOutputStream，然后在正式写出内容之前计算它的大小。

If-Modified-Since: Wed, 26 Oct 2022 16:44:38 GMT\r\n 这是浏览器缓存中保存的该文件的最后修改时间。浏览器发送HTTP请求时，把If-Modified-Since一起发到服务器去，服务器会把这个时间与服务器上实际文件的最后修改时间进行比较。如果时间一致，那么返回HTTP状态码304（Not Modified），客户端接到之后，直接把本地缓存文件显示到浏览器中。如果时间不一致，就返回HTTP状态码200和新的文件内容，客户端接到之后，会丢弃旧文件，把新文件缓存起来，并显示到浏览器中。

If-None-Match: "158-5ebf2bd92f45d"\r\n 这里记录的是首次发起这个这个请求时服务器返回的Etag，ETag实体标签一般为资源实体的哈希值是服务器生成的一个标记，用来标识返回值是否有变化，Etag的优先级高于Last-Modified。在下一次发起与之前相同的请求时，客户端会同时发送一个If-None-Match，而它的值就是Etag的值。然后，服务器会比对这个客服端发送过来的Etag是否与服务器的相同，如果相同，就将If-None-Match的值设为false，返回状态为304，客户端继续使用本地缓存，不解析服务器返回的数据；如果不相同，就将If-None-Match的值设为true，返回状态为200，客户端重新解析服务器返回的数据。

GET方法实体体部分缺省。

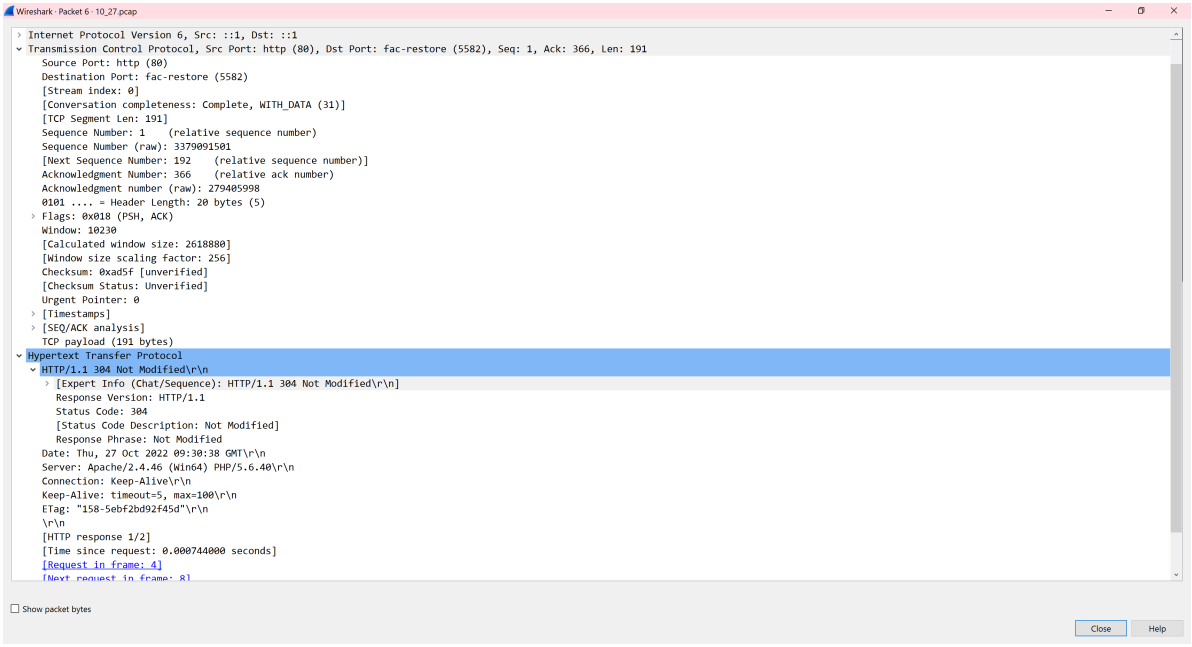
下一个数据包是服务器对该数据包的响应，其Flags中仅ACK被置一，如下图所示。注意到其序列号为K+1，其确认号为279405998，用这个数字减去J+1的话，正好是要发送的HTTP请求报文的长度，这充分说明序列号衡量的是要发送的信息的长度（而跟TCP首部基本无关）。这里的Window大小再次发生变化，扩大了十倍到10230。



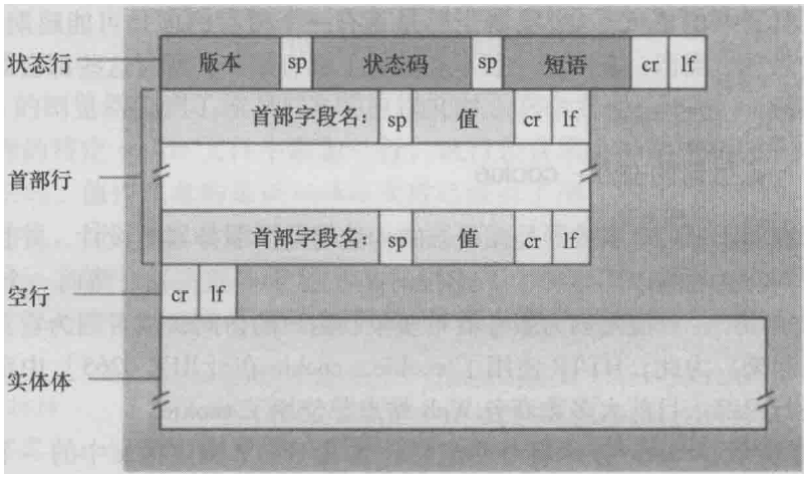
2

服务器对上一个请求的响应。6号数据包是服务器对上一个GET请求的响应。截图如下。可以看到序列号还是K+1，跟上一个ACK无关。

Flags字段增加了PSH。



HTTP响应报文格式如下：



响应报文同样由三部分组成：状态行(status line)、首部行(header line)、实体体(entity body)。

状态行有三个字段：HTTP版本、状态码、状态信息

下面列出常见状态码及状态信息

- 200 OK：请求成功，信息在返回的响应报文中。
- 301 Moved Permanently：请求对象被永久转移了，新的URL定义在响应报文的首部行Location中。客户端自动获取新的URL。
- 404 Not Found：被请求的文档不在服务器上。
- 505 HTTP Version Not Supported：服务器不支持请求报文使用的HTTP协议版本。

  这里我们返回的是304，因为之前已经访问过这个网页，而304意味着本地的浏览器缓存是有效的，浏览器可以直接使用缓存，不需要服务器向其传送数据。

同样的，首部行由多组键值对（首部字段名：首部字段值）组成。

Connection：服务器通知客户，发送完报文后是否持续该TCP连接。这里为HTTP1.1默认保持连接。

Date：服务器产生并发送该响应报文的日期时间。

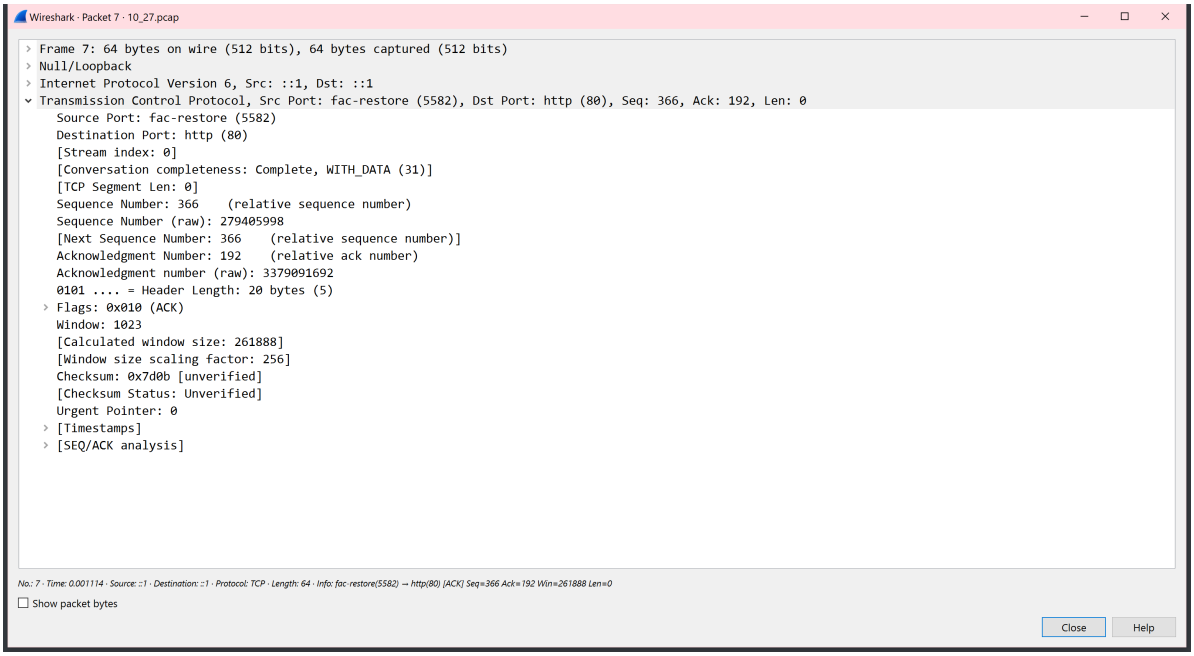
Server：指明产生响应报文的服务器类型，类似于请求报文首部行中User-agent字段。这里为Apache Web服务器。

Keep-Alive: timeout=5, max=100\r\n　这里设置保活时间是5秒，max是最多一百次请求，强制断掉连接

ETag: "158-5ebf2bd92f45d"\r\n　这里服务器再次返回了请求资源的哈希。

实体体部分这里仍然缺省，因为服务器认为浏览器可以使用缓存。

7号数据包是浏览器对服务器返回的ACK，截图如下。



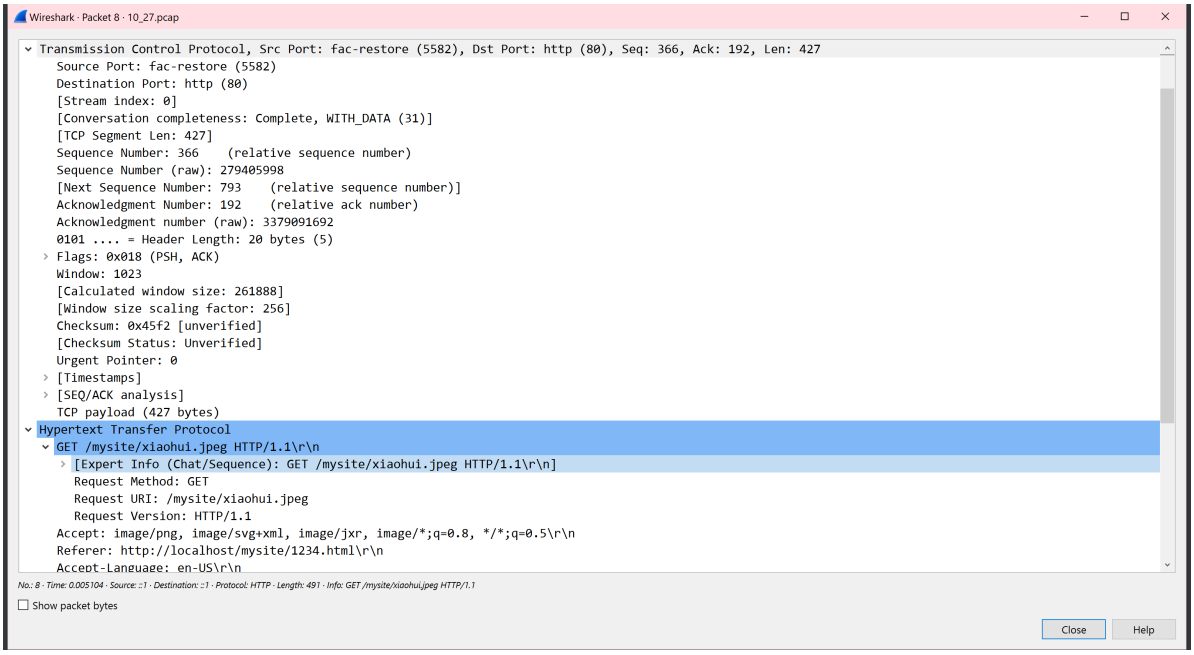
可以看到，其序列号为279405998，确认号为3379091692，减去K+1正好是上一个报文中HTTP响应报文的长度。

3

8-11号数据包是浏览器使用GET请求html文档中的图片对象、服务器确认、服务器响应、浏览器确认的过程，与前述类似。

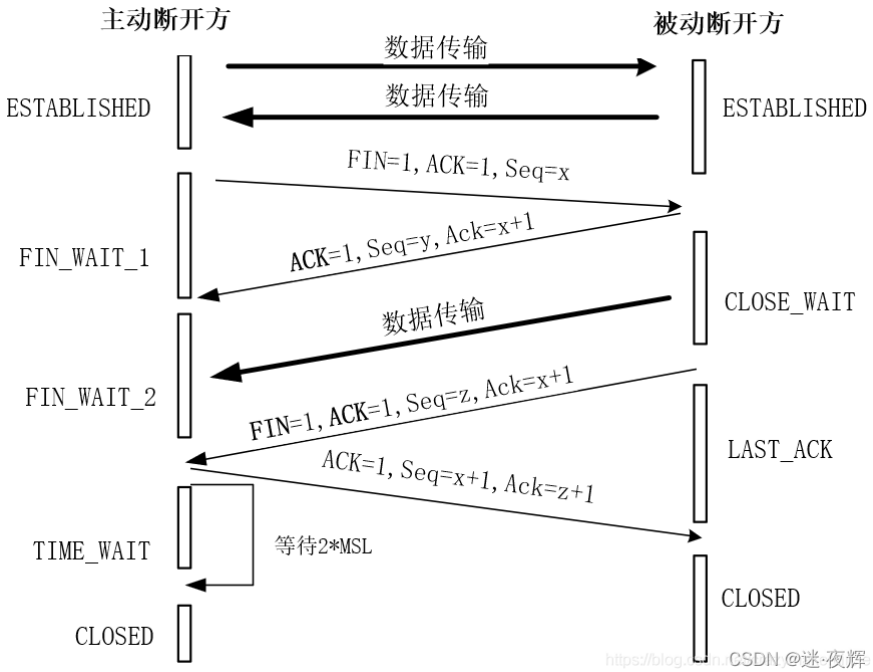
此时浏览器在代码中发现一个<img>标签引用了一张图片，向服务器发出请求。此时浏览器不会等到图片下载完，而是继续渲染后面的代码。

观察下图中的8号数据包，发现其请求的文件已经变为图片文件。



TCP挥手：

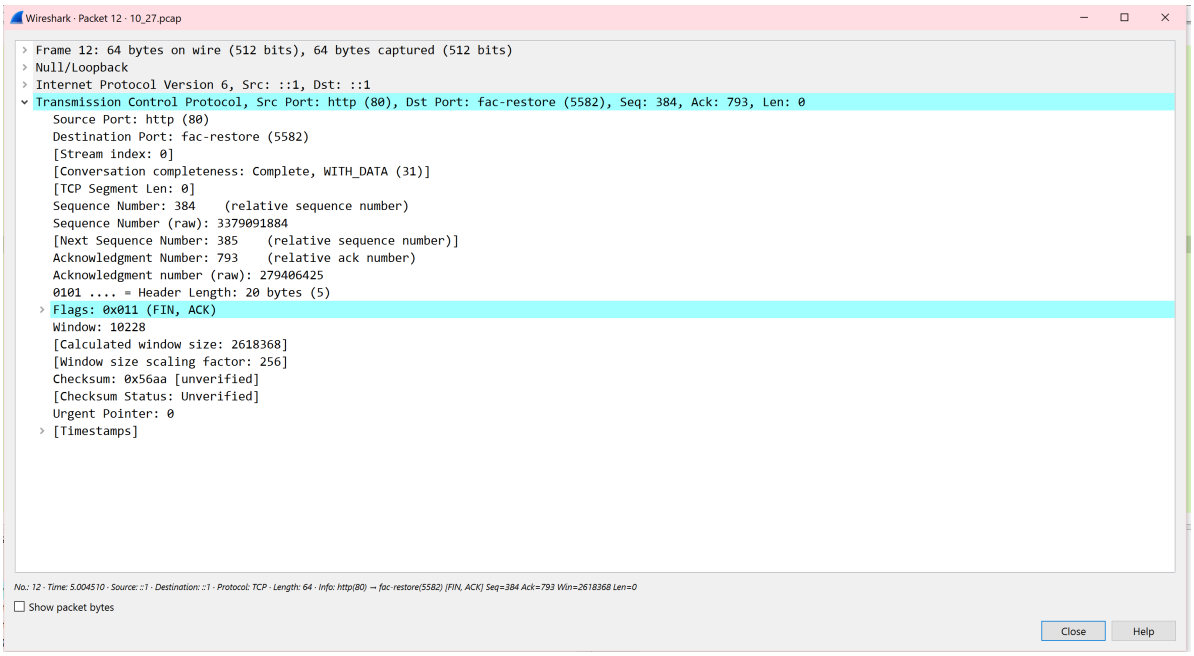
TCP四次挥手示意图：



第一次挥手：

主动断开方（客户端或服务端）向对方发送一个FIN结束请求报文，并设置序列号和确认号，随后主动断开方进入FIN_WAIT1状态，这表示主动断开方已经没有业务数据要发给对方了，准备关闭SOCKET连接了。

分析第12个数据包，首先关注其来源和时间，来源是服务器而时间刚好是5s左右，也就是保活时间超时时。即服务器和浏览器没有进一步通信的内容，这导致服务器主动断开连接，便向浏览器主动发送第一次挥手数据包。截图如下。

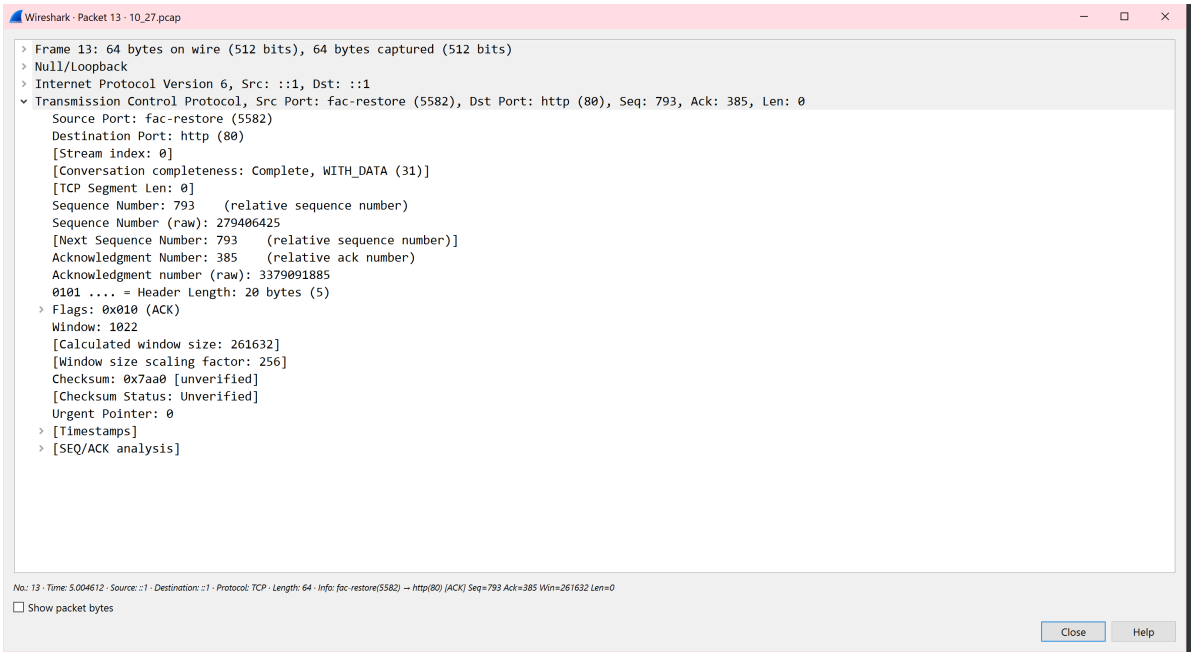


可见Flags字段，FIN（释放一个连接）和ACK被置1。

第二次挥手：

被动断开方收到FIN断开请求后会发送一个ACK响应报文，表明同意断开请求。随后被动断开方就进入CLOSE-WAIT状态（等待关闭状态），此时若被动断开方还有数据要发送给主动方，主动方还会接受。主动方收到ACK报文后，由FIN_WAIT_1转换成FIN_WAIT_2状态。

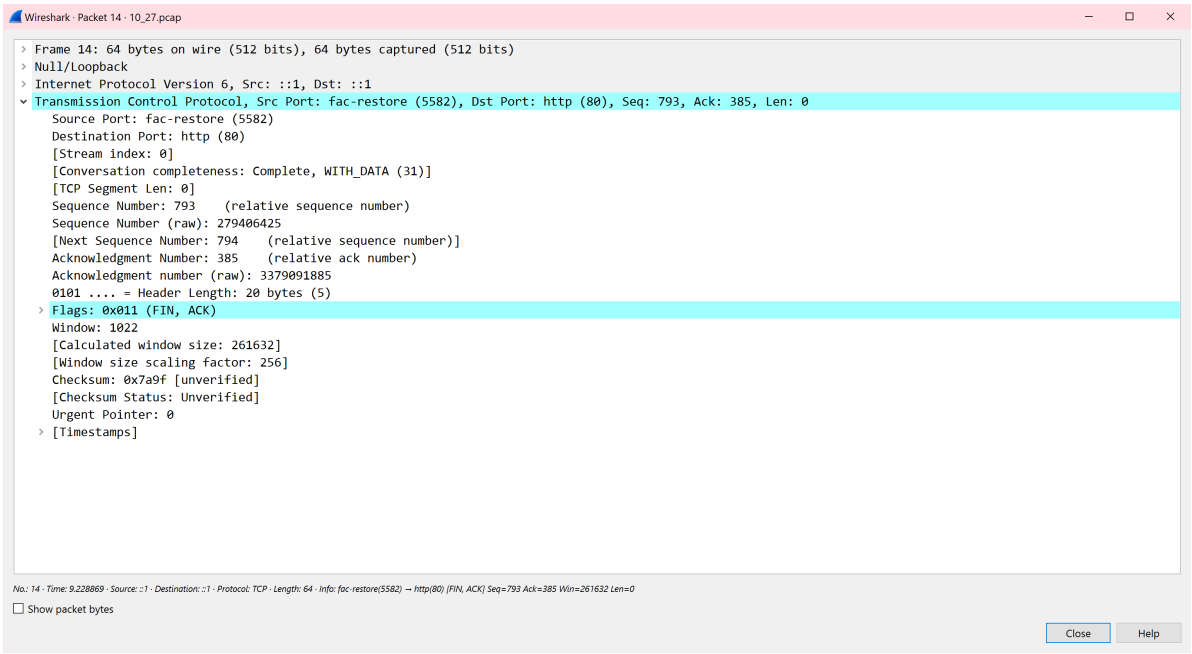
对数据包截图如下，可以看到是被动断开的浏览器对上一个服务器发来的数据包的确认，Flags只有ACK被置1。



第三次挥手：

被动断开方的CLOSE-WAIT（等待关闭）结束后，被动方会向主动方发送一个FIN+ACK报文。表示被动方的数据都发完了。然后被动方进入LAST_ACK状态。

CLOSE-WAIT实际上是被动断开方用于等待上层应用退出的，上层应用不关闭TCP连接，FIN包就不会发出。这里我们经过一段时间的等待后刷新网页以关闭之前的连接，因而这个数据包发出的时间是9s左右。截图如下



可以看到是被动断开的浏览器发送数据包，Flags中FIN和ACK被置1。

第四次挥手：

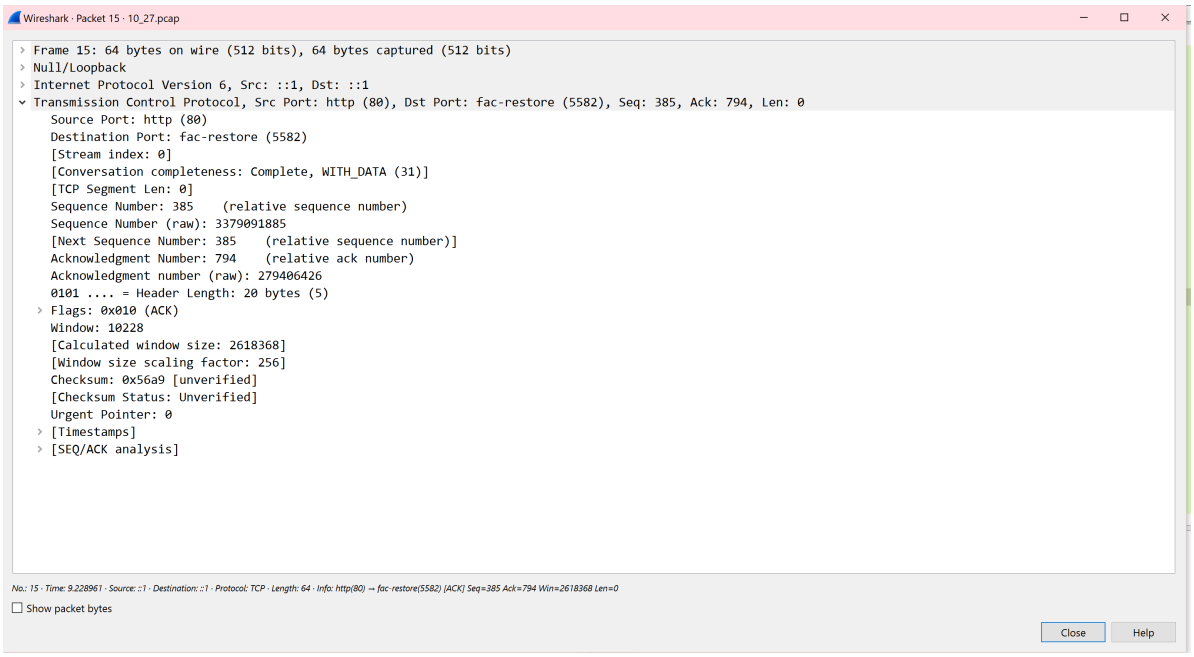
主动断开方收到FIN+ACK断开响应报文后，还需进行最后确认，向被动方发送一个ACK确认报文，然后主动方进入TIME_WAIT状态，在等待完成2MSL时间后，如果期间没有收到被动方的报文，则证明对方已正常关闭，主动断开方的连接最终关闭。被动方在收到主动方第四次挥手发来的ACK报文后，就关闭了连接。

MSL是Maximum Segment Lifetime的英文缩写，可译为“最长报文段寿命”，它是任何报文在网络上存在的最长的最长时间，超过这个时间报文将被丢弃。等待2MSL时间呢的主要原因有两点：

1.为了保证主动断开方发送的最后一个ACK报文段能够到达被动方。这个ACK报文段有可能丢失，被动方收不到这个ACK就会重传已发送的FIN+ACK报文段。

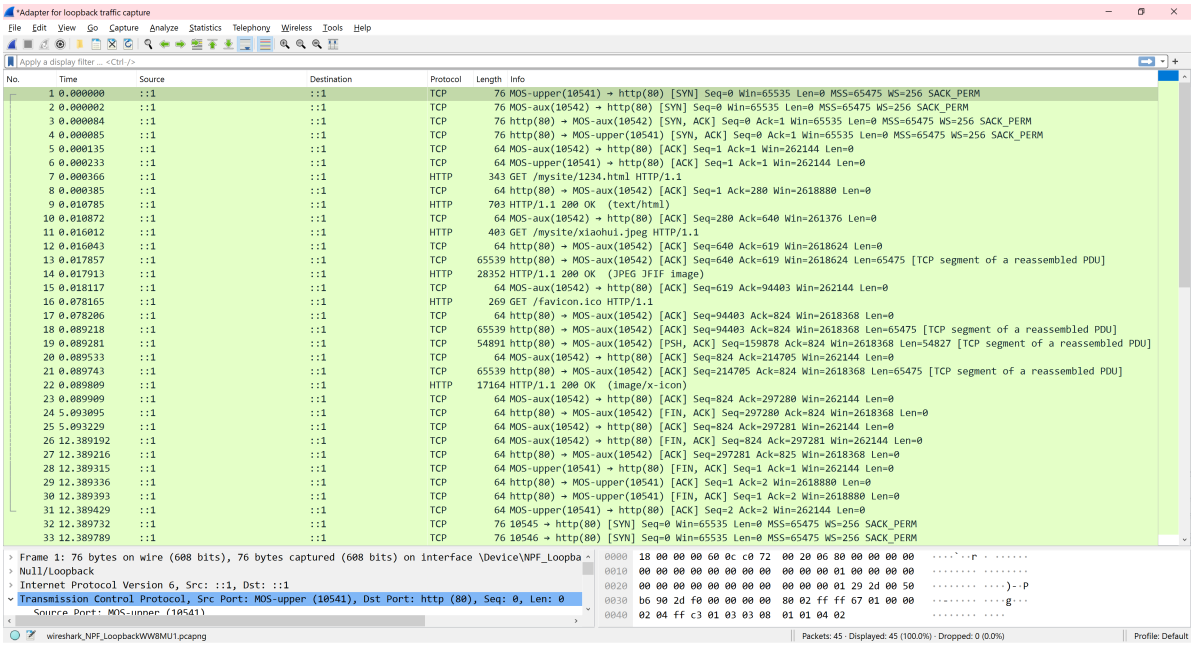
2.经过时间2MSL，就可以使本链接持续时间内所产生的所有报文段都从网络中消失，这样就可以使下一个新的连接中不会出现旧的连接请求报文段。

截取数据包如下：是主动方（此为服务器）发送的ACK。



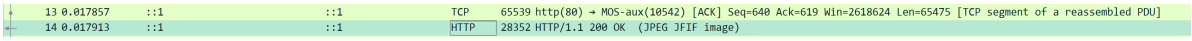
实验完成后，保存Wireshark PCAP文件到本地。

对比：首次打开该网页的情况：



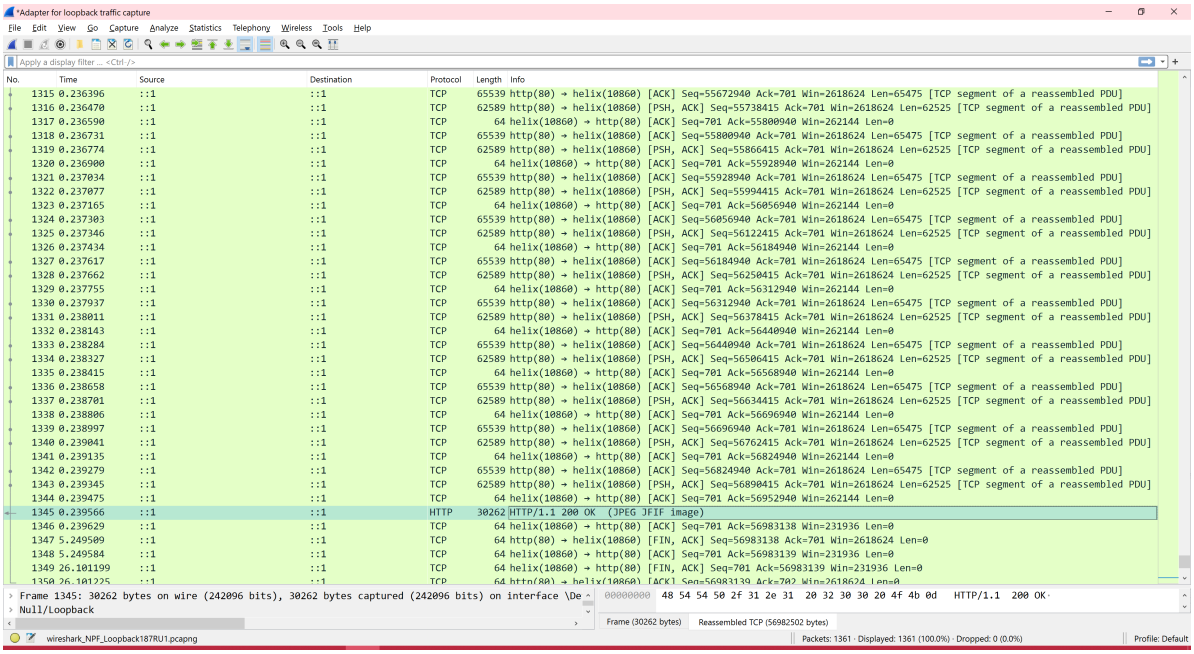
将浏览器缓存清除后重新打开网页，并抓包记录该过程。发现浏览器和服务器建立了两个连接，这使得传输数据包增多。由于没有任何缓存，所有文件要重新获取，发现HTTP响应报文的状态码为200，数据段为对应的文件或图片，且请求了网站图标。

这里还要注意一点，TCP数据包并不能保证一次能回传整个请求的文件或者说任何应用层抽象出来的信息的“分段”。比如这里用到的校徽图片为92KB，注意如下的两个数据包



前一个数据包包含HTTP响应的一部分，两个数据包拼合，其HTTP数据段才是完整的图片信息。Wireshark会将除最后一个分段外之前的不完整分段标记为“TCP segment of a reassembled PDU”。

我们用一个体积更大的图片进行测试（55MB）



可以发现这个文件被拆分成了上千个包（因为每个包最大是64KB）。

参考资料：

Wireshark抓包实验*yuminglc的博客-CSDN博客*wireshark抓包实验

简述TCP的三次握手过程 - 98剑南春 - 博客园 (cnblogs.com)

HTTP请求和响应详解 - 木木-林 - 博客园 (cnblogs.com)

TCP四次挥手*迷·夜挥的博客-CSDN博客*tcp4次挥手