华东师范大学软件工程学院实验报告

实验课程:	计算机网络	年　　级:	2022 级
实验编号:	Lab 06	实验名称:	TCP
姓　　名:	李鹏达	学　　号:	10225101460

1 实验目的

1) 学会通过 Wireshark 获取 TCP 消息
2) 掌握 TCP 数据包结构
3) 掌握 TCP 数据包各字段的含义
4) 掌握 TCP 连接建立和释放的步骤
5) 掌握 TCP 数据传输阶段的过程

2 实验内容与实验步骤

2.1 实验内容

2.1.1 捕获 TCP 报文

1. 以 http://img.zcool.cn/community/01dcd059117b12a801216a3e9c4fd5.jpg@1280w_1l_2o_100sh.jpg 为例，用 wget 确认该链接有效
2. 启动 Wireshark，在菜单栏的捕获 → 选项中进行设置，选择已连接的以太网，设置捕获过滤器为 tcp and host img.zcool.cn。我们主要观察客户端与服务器之间的 tcp 流。
3. 捕获开始后，重复第一步，重新发送请求。
4. 当 wget 命令结束后，停止 Wireshark 捕获。

2.1.2 分析 TCP 报文

要求：

1. 画出 TCP 数据报的结构图
2. 分析 TCP 数据报头各字段作用

2.1.3 TCP 连接的建立和释放

TCP 的三次握手协议主要分为以下三个步骤：

1. 客户端发送一个 SYN 包给服务器，然后等待应答。

2. 服务器端回应给客户端一个 ACK=1、SYN=1 的 TCP 数据段。
3. 客户必须再次回应服务器端一个 ACK 确认数据段。

回答问题：

1. 观察客户端与服务器的连接建立过程，画出三次握手协议的步骤图
2. 观察并分析 TCP 数据段中的 option 字段

TCP 的四次挥手主要分为以下四个步骤：

1. 客户端进程发出断开连接指令，这将导致客户端的 TCP 程序创建一个特殊的 TCP 报文段，发送到服务器。这个报文段的 FIN 字段被置为 1，表示这是一条断开连接的报文；
2. 服务器接收到客户端发来的断开连接报文，向客户端回送这个报文的确认报文（ACK 字段为 1），告诉服务器已经接收到 FIN 报文，并允许断开连接；
3. 服务器发送完确认报文后，服务器的 TCP 程序创建一条自己的断开连接报文，此报文的 FIN 字段被置为 1，然后发往客户端；
4. 客户端接收到服务器发来的 FIN 报文段，则产生一条确认报文（ACK 为 1），发送给服务器，告知服务器已经接收到了它的断开报文。服务器接收到这条 ACK 报文段后，释放 TCP 连接相关的资源（缓存和变量），而客户端等待一段时间后（半分钟、一分钟或两分钟），也释放处于客户端的缓存和变量。

回答问题：

1. 观察客户端与服务器连接的释放过程，画出释放连接的步骤图
2. 思考为什么连接的时候是三次握手，关闭的时候却是四次挥手？

2.1.4 TCP 数据传输

观察 Wireshark 生成的 IO 图表
回答问题：（以各自实验为准）

1） 实验中的下载速率是多少？Bits/s & packets/s
2） 实验中的上传速率，即 ACK 消息的发送速率是多少？Bits/s & packets/s

观察数据包传输过程

1. 观察数据传输过程中 Acknowledgment number，sequence number 和 Segment Len 之间的变化
2. 如果最近从服务器收到的 TCP 数据段的序列号是 X，那么下一个发送的 ACK 是多少？

2.1.5 问题讨论

1. Explore the congestion control and the classic AIMD behavior of TCP. To do this, you will likely want to capture a trace while you are sending (not receiving) a moderate amount of data on a TCP connection. You can then use the “TCP Stream Graph” tools as well as other analysis to observe how the congestion window changes over time.

2. Explore the reliability mechanisms of TCP more deeply. Capture a trace of a connection that includes segment loss. See what triggers the retransmissions and when. Also look at the round-trip time estimator.
3. Look at the use of options including SACK to work through the details. You should see information about ranges of received bytes during times of segment loss.
4. TCP is the transport layer underlying the web. You can see how your browser makes use of TCP by setting up concurrent connections.

2.2 实验步骤

1. 以 `http://img.zcool.cn/community/01dcd059117b12a801216a3e9c4fd5.jpg` 为例，用 `wget` 确认该链接有效

```
PS> wget http://img.zcool.cn/community/01dcd059117b12a801216a3e9c4fd5.jpg
```

2. 启动 `Wireshark`，在菜单栏的捕获 → 选项中进行设置，选择已连接的以太网，设置捕获过滤器为 `tcp and host img.zcool.cn`。我们主要观察客户端与服务器之间的 `tcp` 流。
3. 捕获开始后，重复第一步，重新发送请求。
4. 当 `wget` 命令结束后，停止 `Wireshark` 捕获。
5. 分析 `TCP` 报文

3 实验环境

- 操作系统：`Windows 11 家庭中文版 23H2 22631.2715`
- 网络适配器：`Killer(R) Wi-Fi 6 AX1650i 160MHz Wireless Network Adapter(201NGW)`
- `Wireshark`：`Version 4.2.0 (v4.2.0-0-g54eedfc63953)`
- `wget`：`GNU Wget 1.21.4 built on mingw32`

4 实验结果与分析

4.1 捕获 `TCP` 报文

首先，我们使用 `wget` 确认链接有效。

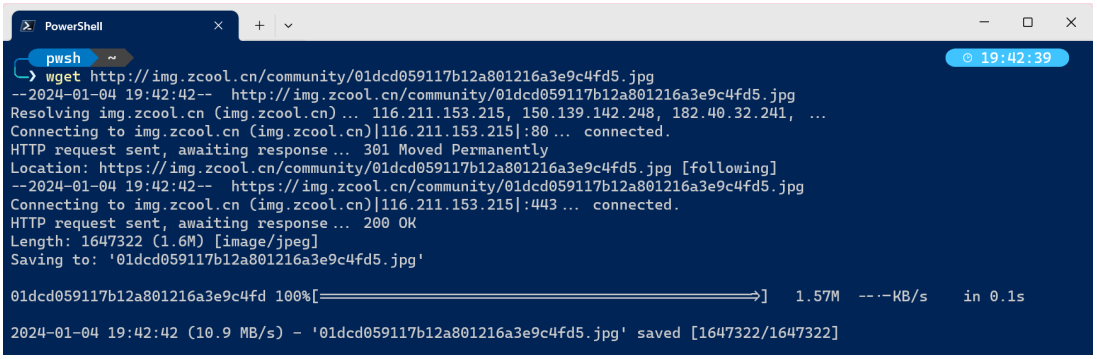

图 1: 使用 `wget` 确认链接有效

然后，我们启动 `Wireshark`，在菜单栏的捕获 → 选项中进行设置，选择已连接的以太网，设置捕获过滤器为 `tcp and host img.zcool.cn`。我们主要观察客户端与服务器之间的 `tcp` 流。

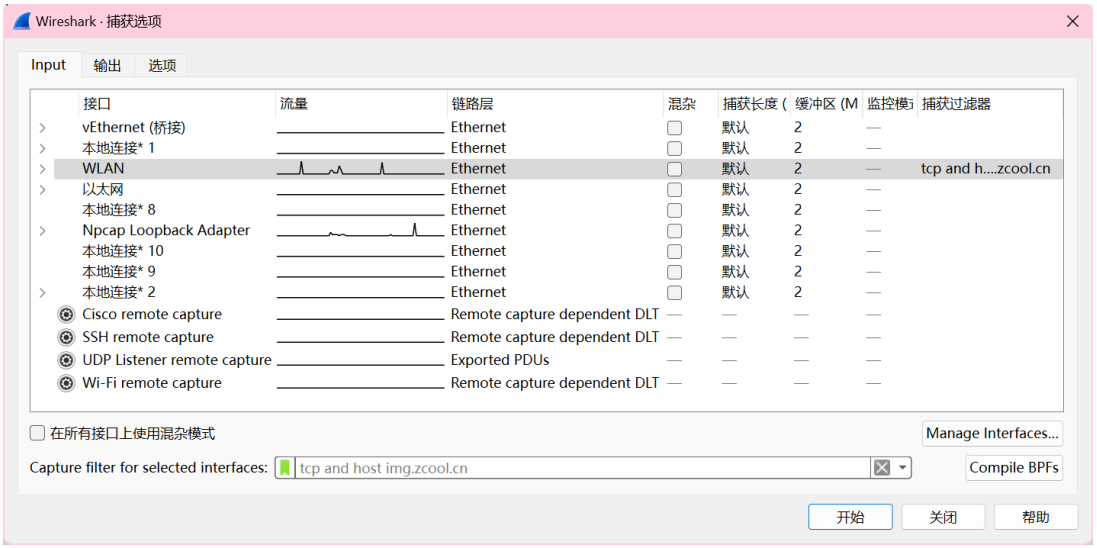

图 2: 设置捕获过滤器

捕获开始后，重复第一步，重新发送请求。

```
pwsh ~                                                                    19:42:42
❯ wget http://img.zcool.cn/community/01dcd059117b12a801216a3e9c4fd5.jpg
--2024-01-04 19:43:31--  http://img.zcool.cn/community/01dcd059117b12a801216a3e9c4fd5.jpg
Resolving img.zcool.cn (img.zcool.cn)... 116.211.153.215, 150.139.142.248, 182.40.32.241, ...
Connecting to img.zcool.cn (img.zcool.cn)|116.211.153.215|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://img.zcool.cn/community/01dcd059117b12a801216a3e9c4fd5.jpg [following]
--2024-01-04 19:43:31--  https://img.zcool.cn/community/01dcd059117b12a801216a3e9c4fd5.jpg
Connecting to img.zcool.cn (img.zcool.cn)|116.211.153.215|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1647322 (1.6M) [image/jpeg]
Saving to: '01dcd059117b12a801216a3e9c4fd5.jpg.1'

01dcd059117b12a801216a3e9c4fd5.jpg.1 100%[=================================================>]   1.57M  --.-KB/s    in 0.1s

2024-01-04 19:43:31 (11.9 MB/s) - '01dcd059117b12a801216a3e9c4fd5.jpg.1' saved [1647322/1647322]
```

图 3: 重新发送请求

当 wget 命令结束后，停止 Wireshark 捕获。捕获结果如下：

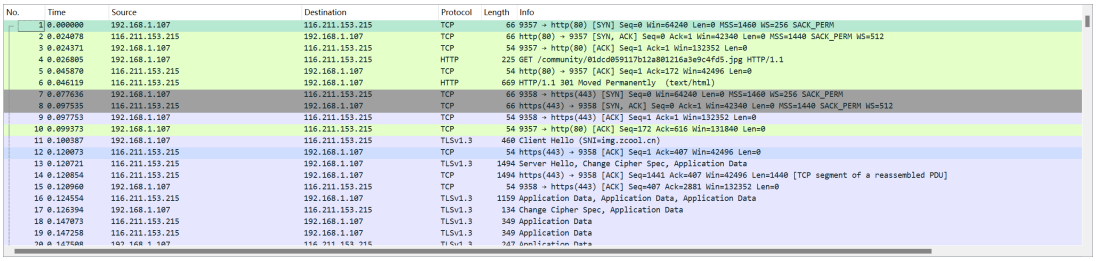

图 4: 捕获结果

4.2 分析 TCP 报文

选择一个 TCP 数据包，如下所示：

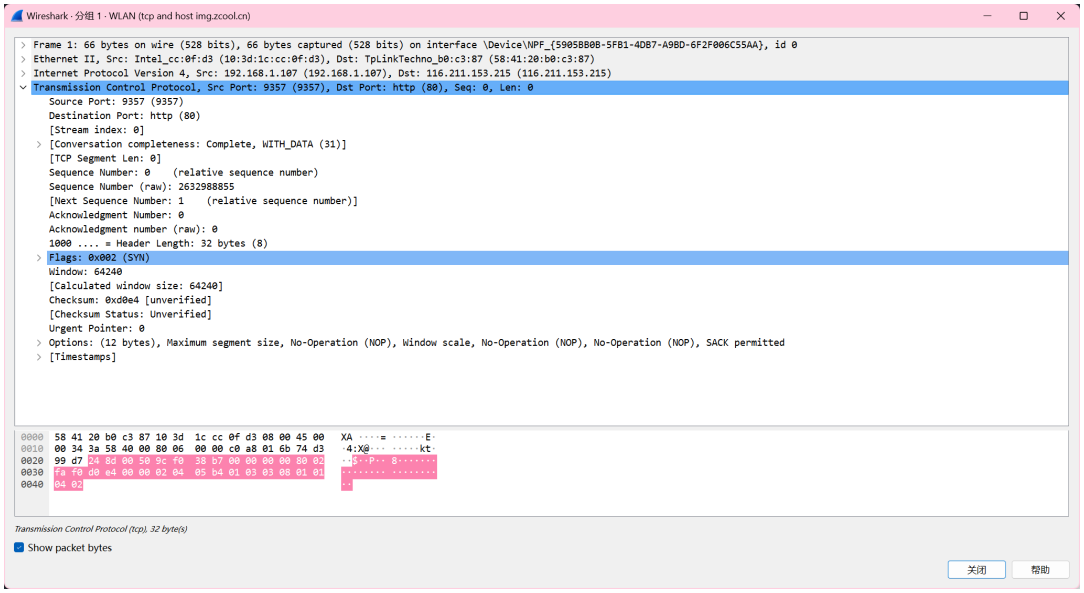

图 5: 一个 TCP 数据包

可以画出 TCP 包的结构如下：

<table>
<tr><td colspan="2">源端口 Source Port

2 bytes</td><td colspan="2">目的端口 Destination Port

2 bytes</td></tr>
<tr><td colspan="4">序列号 Sequence Number

4 bytes</td></tr>
<tr><td colspan="4">确认号 Acknowledgment Number

4 bytes</td></tr>
<tr><td colspan="2">头部长度 Header Length

共 2 bytes</td><td>标志 Flags</td><td>窗口大小 Window

2 bytes</td></tr>
<tr><td colspan="2">校验和 Checksum

2 bytes</td><td colspan="2">紧急指针 Urgent Pointer

2 bytes</td></tr>
<tr><td colspan="4">可选项 Options

0-40 bytes</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="4">负载 Payload

n bytes</td></tr>
</table>

表 1: UDP 包结构

其中 Flags 字段包括 Reserved, Accurate ECN, Congestion Window Reduced, ECN-Echo, Urgent, Acknowledgment, Push, Reset, Syn, Fin。

TCP 数据包头部各字段的含义如下：

- 源端口：源端口号，占 2 字节，用来标识源主机的应用程序进程。
- 目的端口：目的端口号，占 2 字节，用来标识目的主机的应用程序进程。
- 序列号：占 4 字节，用来标识从 TCP 源端向目的端发送的字节流，发起方发送数据时对此进行标记。
- 确认号：占 4 字节，只有 ACK 标志位为 1 时，确认号字段才有效，确认号等于上次接收到的字节序号加 1。

- 头部长度：占 1 字节，指示 TCP 头部长度。
- 标志：与头部长度合占 2 字节。各标志位的含义如下：
  - Reserved：保留位。
  - Accurate ECN：显式拥塞通知。
  - Congestion Window Reduced：拥塞窗口减小。
  - ECN-Echo：显式拥塞通知回显。
  - URG：紧急指针（urgent pointer）有效。
  - ACK：确认序号有效。
  - PSH：接收方应该尽快将这个报文交给应用层。
  - RST：重置连接。
  - SYN：发起一个新连接。
  - FIN：释放一个连接。
- 窗口大小：占 2 字节，窗口大小是指发送方在收到确认前允许发送的字节数。
- 校验和：占 2 字节，检验 TCP 首部和 TCP 数据的正确性。
- 紧急指针：占 2 字节，仅当 URG 标志为 1 时，紧急指针才有效。紧急指针告诉系统此报文段中有紧急数据，紧急数据的字节数由紧急指针指出。
- 可选项：占 0-40 字节，可选项可以有 0 个或多个，用于一些可选的设置。

4.3 TCP 连接的建立和释放

4.3.1 三次握手

我们知道，三次握手的过程如下：

1. 客户端发送一个 `SYN` 包给服务器，然后等待应答。
2. 服务器端回应给客户端一个 `ACK=1`、`SYN=1` 的 `TCP` 数据段。
3. 客户必须再次回应服务器端一个 `ACK` 确认数据段。

在 `Wireshark` 中，我们可以看到三次握手的过程如下：

1	0.000000	192.168.1.107	116.211.153.215	TCP	66 9357 → http(80) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
2	0.024078	116.211.153.215	192.168.1.107	TCP	66 http(80) → 9357 [SYN, ACK] Seq=0 Ack=1 Win=42340 Len=0 MSS=1440 SACK_PERM WS=512
3	0.024371	192.168.1.107	116.211.153.215	TCP	54 9357 → http(80) [ACK] Seq=1 Ack=1 Win=132352 Len=0

图 6: 三次握手

画出三次握手协议的步骤图如下：

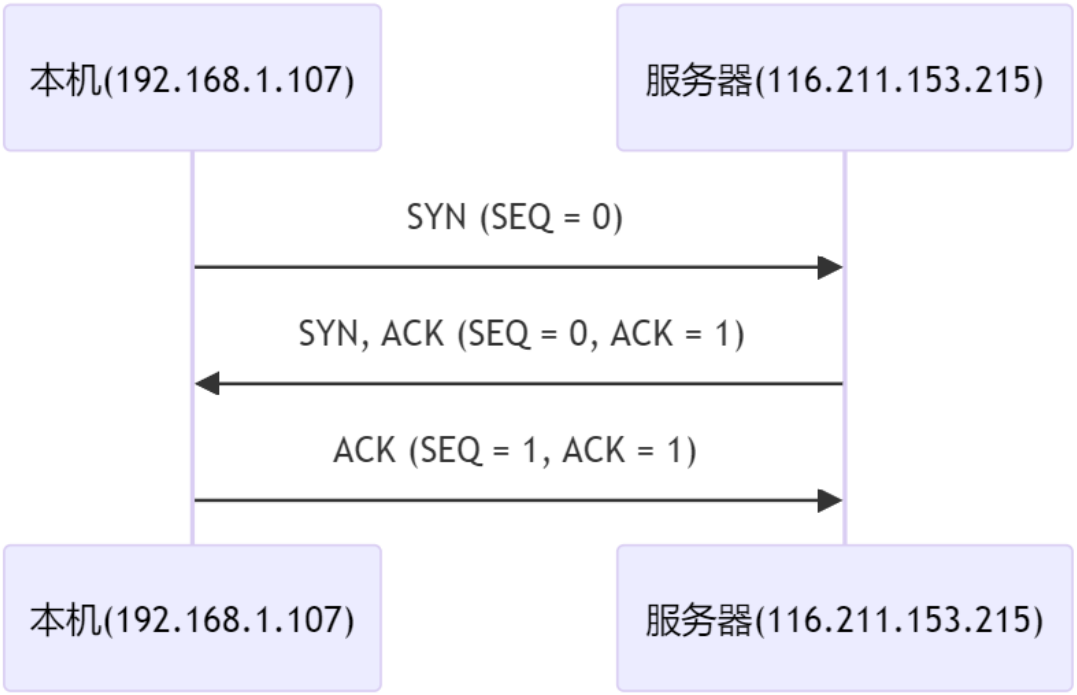

图 7: 三次握手协议的步骤图

4.3.2 四次挥手

TCP 的四次挥手主要分为以下四个步骤：

1. 客户端进程发出断开连接指令，这将导致客户端的 TCP 程序创建一个特殊的 TCP 报文段，发送到服务器。这个报文段的 FIN 字段被置为 1，表示这是一条断开连接的报文；
2. 服务器接收到客户端发来的断开连接报文，向客户端回送这个报文的确认报文（ACK 字段为 1），告诉服务器已经接收到 FIN 报文，并允许断开连接；
3. 服务器发送完确认报文后，服务器的 TCP 程序创建一条自己的断开连接报文，此报文的 FIN 字段被置为 1，然后发往客户端；
4. 客户端接收到服务器发来的 FIN 报文段，则产生一条确认报文（ACK 为 1），发送给服务器，告知服务器已经接收到了它的断开报文。服务器接收到这条 ACK 报文段后，释放 TCP 连接相关的资源（缓存和变量），而客户端等待一段时间后（半分钟、一分钟或两分钟），也释放处于客户端的缓存和变量。

在 Wireshark 中，我们可以看到四次挥手的过程如下：

图 8: 四次挥手

画出四次挥手协议的步骤图如下：

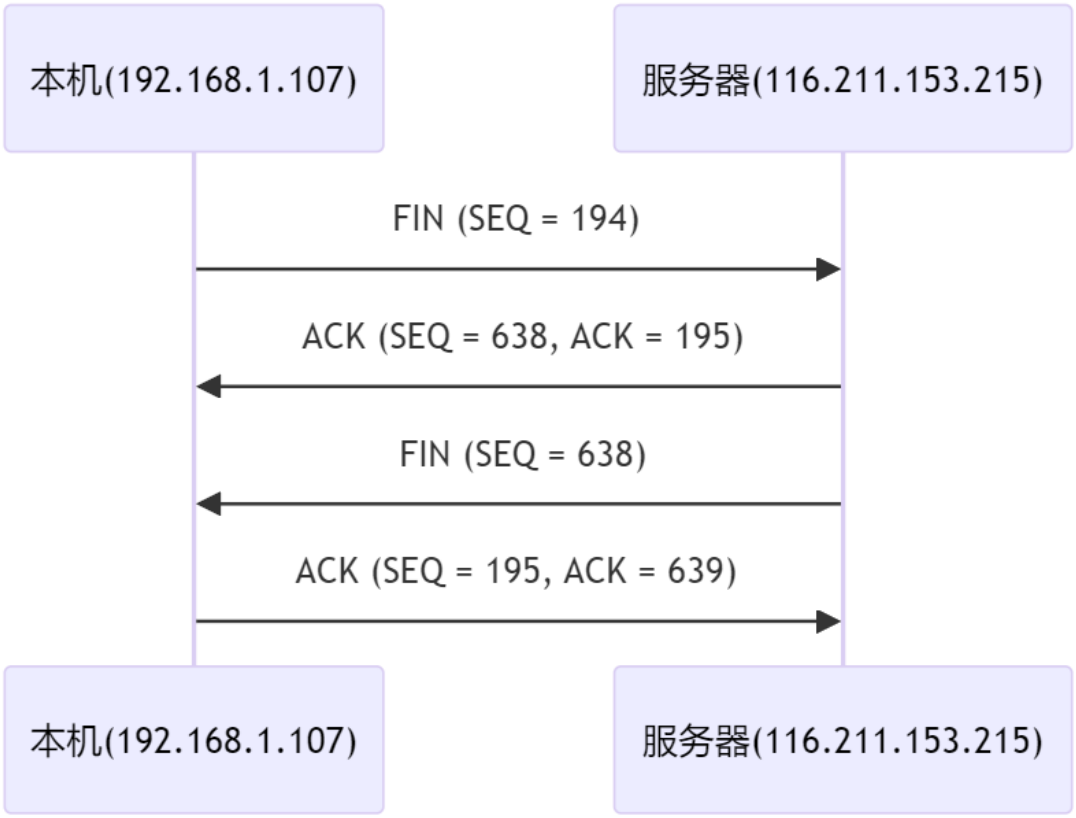

图 9: 四次挥手协议的步骤图

回答问题：

- 思考为什么连接的时候是三次握手，关闭的时候却是四次挥手？
  因为一方在收到对方发来的 `FIN` 包时，有可能还有未发送完的数据包，所以这时它会先给对方发送一个 `ACK` 包，等它将数据包都处理完毕并发送后，再向对方发送 `FIN ACK` 报文来同意关闭连接，最后对方收到 `FIN ACK` 包，向它回复 `ACK` 包。而建立连接时，不会出现这个问题，所以只需要三次握手即可建立连接。

4.4 TCP 数据传输

观察 `Wireshark` 生成的 `IO` 图表, 如下所示：

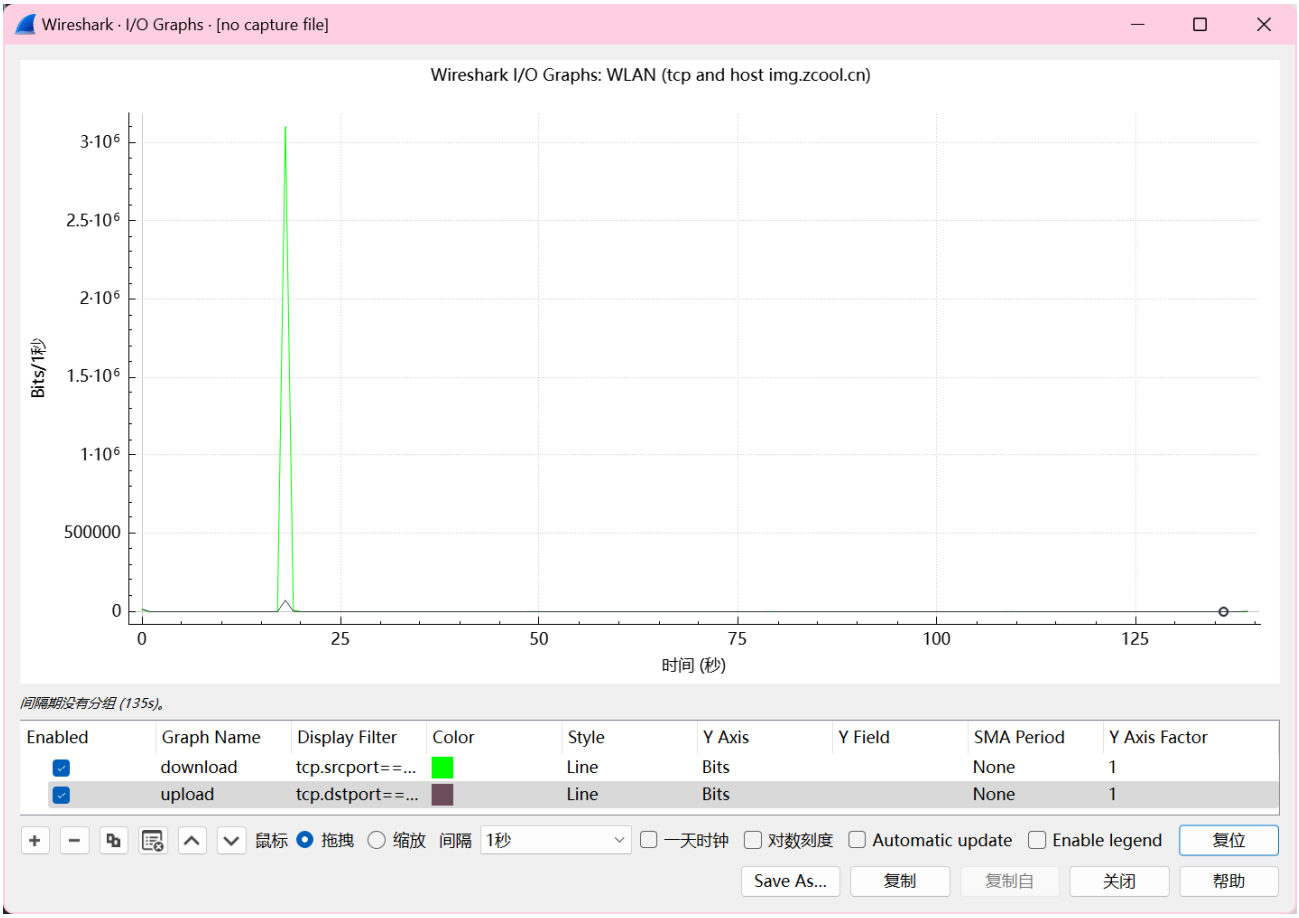

图 10: IO 图表

下载速度为 3107000 Bits/s，上传速度为 732900 Bits/s

观察数据包传输过程，回答问题：

- 观察数据传输过程中 Acknowledgment number，sequence number 和 Segment Len 之间的变化
- 如果最近从服务器收到的 TCP 数据段的序列号是 X，那么下一个发送的 ACK 是多少？
  下一个发送的 ACK 是 X + Segment Len

4.5 问题讨论

下载一个大文件，观察 TCP 流图表，如下所示：

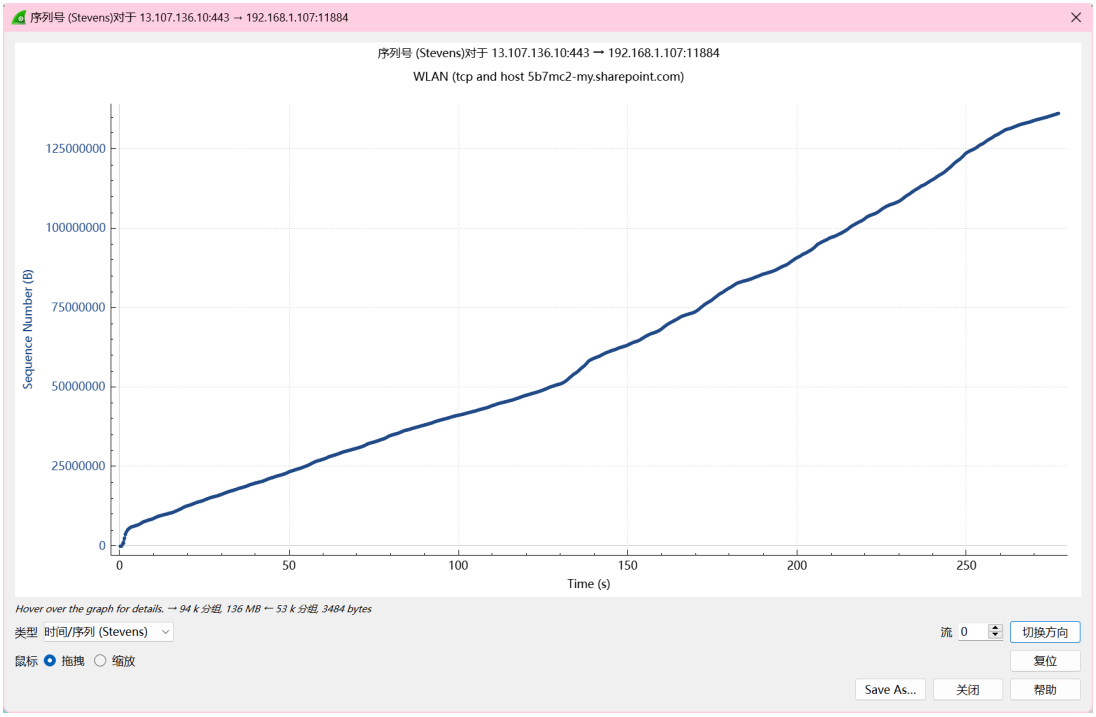

图 11: 序列号

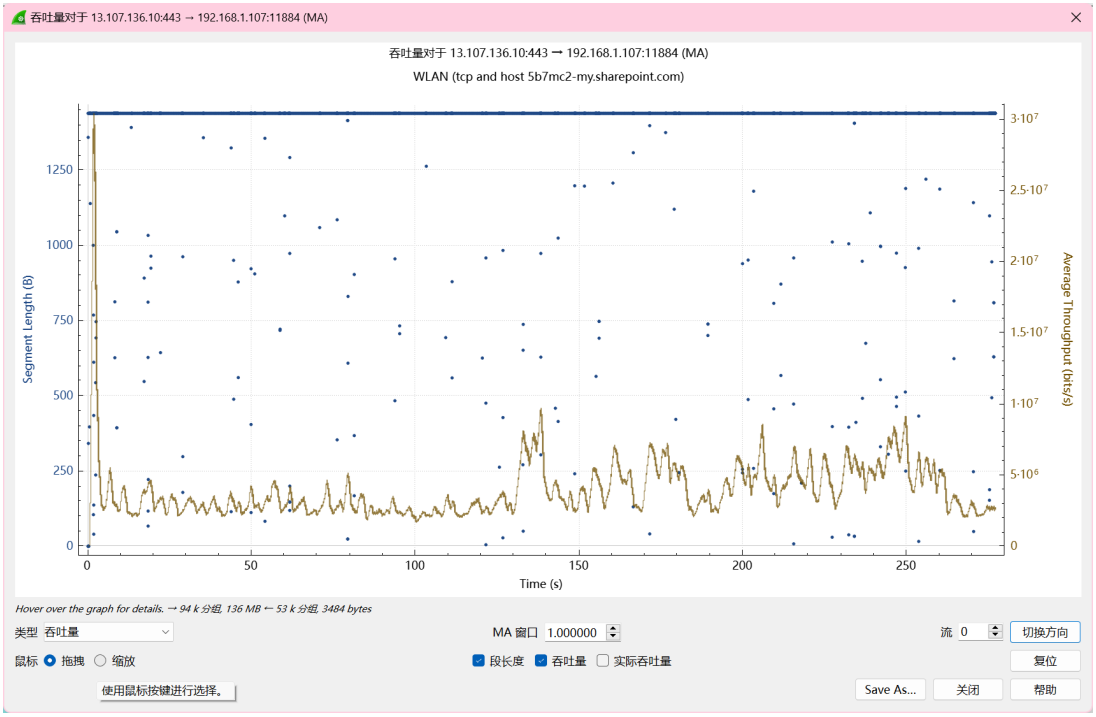

图 12: 吞吐量

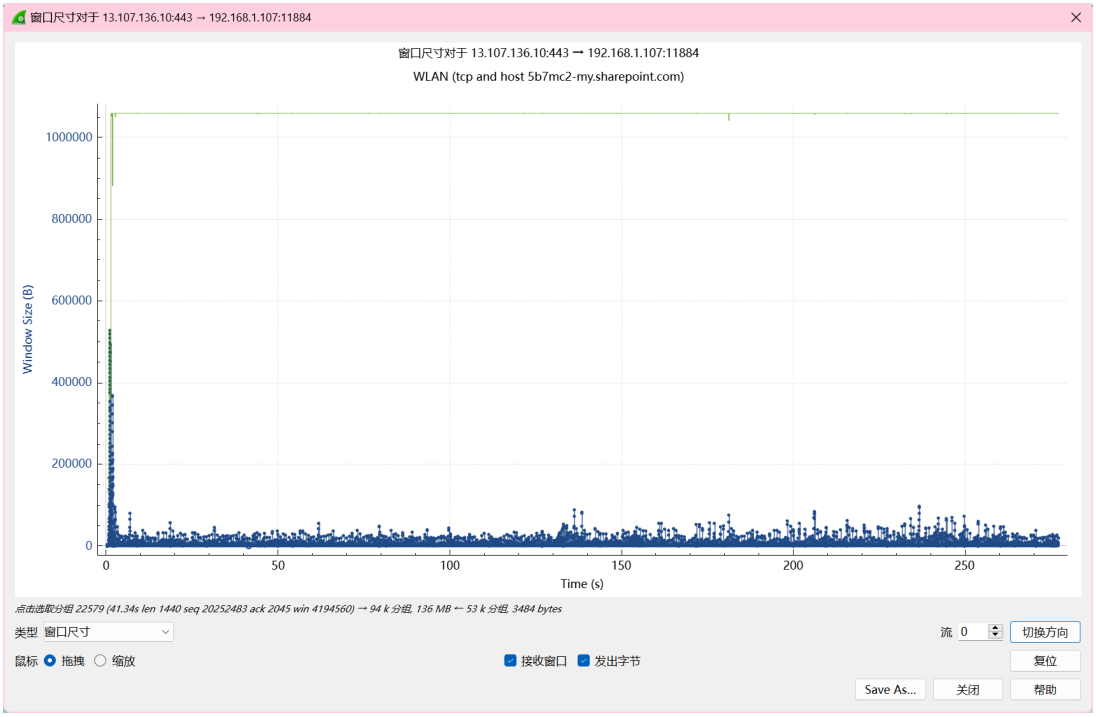

图 13: 窗口尺寸

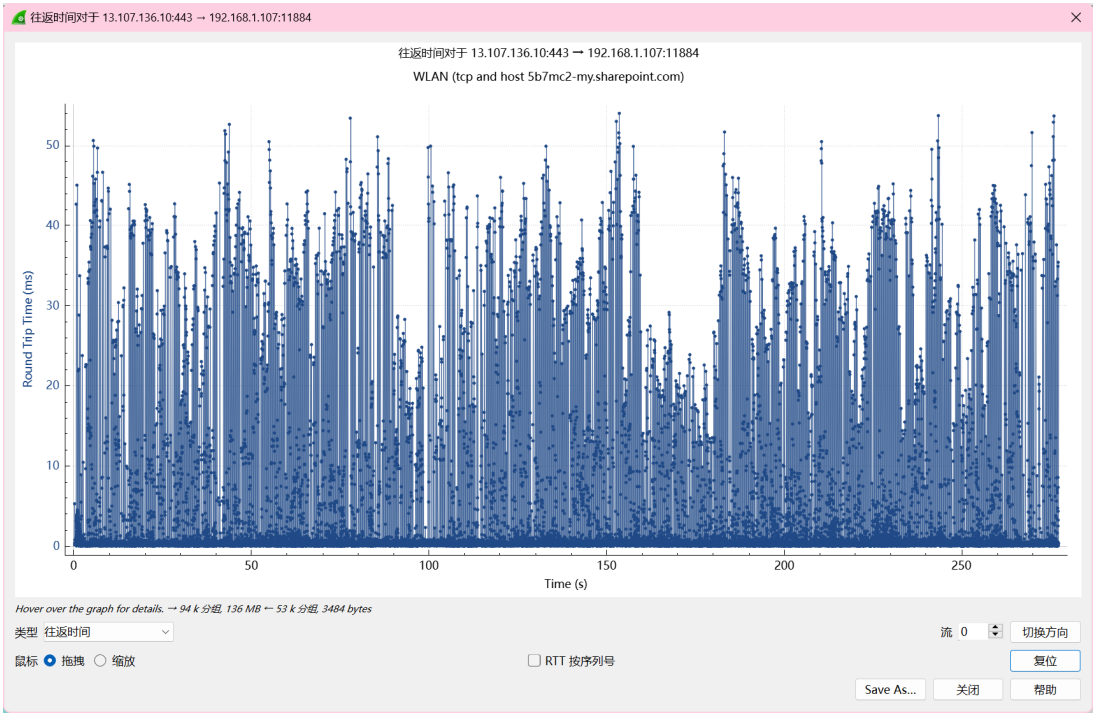

图 14: 往返时间

回答下列问题：

- Explore the congestion control and the classic AIMD behavior of TCP. To do this, you will likely want to capture a trace while you are sending (not receiving) a moderate amount of data on a TCP connection. You can then use the “TCP Stream Graph” tools as well as other analysis to observe how the congestion window changes over time. 从吞吐量图表中可以看出，吞吐量随着时间的增加而增加，但是在某些时刻会突然下降，不断波动，这是因为在这些时刻发生了拥塞，TCP 会减小拥塞窗口，然后再慢慢增加，这就是拥塞控制的过程。
- Explore the reliability mechanisms of TCP more deeply. Capture a trace of a connection that includes segment loss. See what triggers the retransmissions and when. Also look at the round-trip time estimator. 可以看到，当接受到的数据包顺序错误时（可能是之前的数据包丢失了），会触发重传机制，重传丢失的数据包。

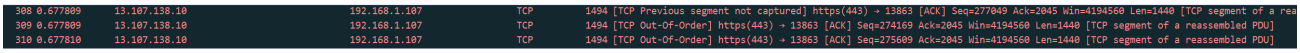

图 15: 错误重传

往返时间图表已在上方列出。

- Look at the use of options including SACK to work through the details. You should see information about ranges of received bytes during times of segment loss.
  观察一个 Options 中包含了 SACK 的数据包，如下图所示：

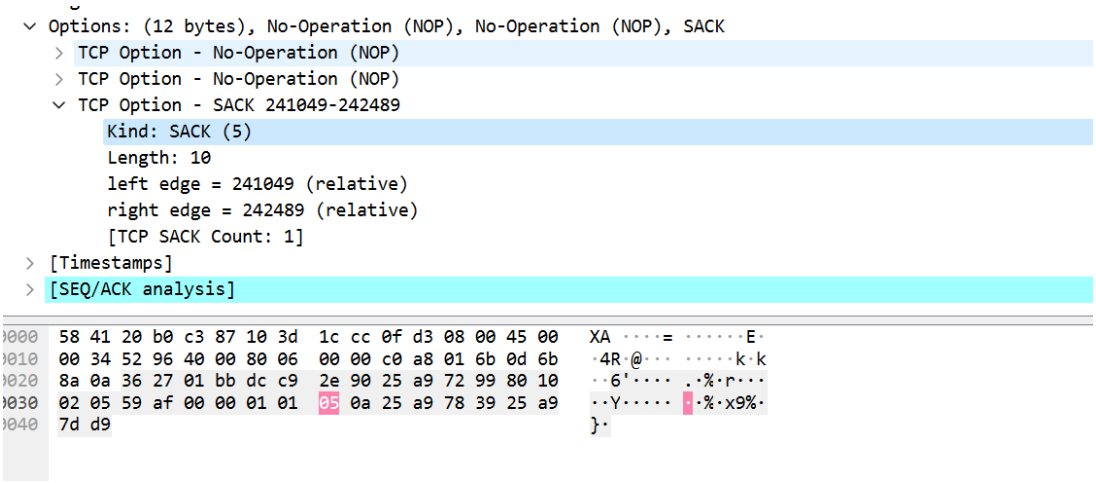

图 16: SACK

可以看到，SACK 中包含了丢失的数据包的序列号范围。

- TCP is the transport layer underlying the web. You can see how your browser makes use of TCP by setting up concurrent connections.
  浏览器处理用户请求时，会建立多个 TCP 连接，并发地处理用户请求。

5 实验结果总结

在本次实验中，我学习了 TCP 的数据包结构，以及 TCP 连接建立和释放的过程，还学习了 TCP 数据传输的过程。

同时，我还了解到了 TCP 的拥塞控制机制，以及 TCP 的可靠性机制。

6 附录

无