实验三、传输层 TCP 协议分析实验报告

组号：[redacted]

姓名：[redacted] 学号：[redacted] 班级：计算机[redacted]

姓名：[redacted] 学号：[redacted] 班级：计算机[redacted]

一、 实验目的

1. 理解 TCP 报文首部格式和字段的作用，TCP 连接的建立和释放过程，TCP 数据传输中的编号与确认的过程。

2. 理解 TCP 的错误恢复的工作原理和字节流的传输模式，分析错误恢复机制中 TCP 双方的交互情况。

3. 理解 TCP 的流量控制的工作原理，分析流量控制中 TCP 双方的交互情况。

4. 理解 TCP 的拥塞控制的工作原理，分析拥塞控制中 TCP 双方的交互情况。

二、 实验内容

1. 使用基于 TCP 的应用程序（如浏览器下载文件）传输文件，在**客户端和服务器均要捕获 TCP 报文**。

2. 分析 TCP 报文首部信息、TCP 连接的建立和释放过程、TCP 数据的编号与确认机制。观察几个典型的 TCP 选项：MSS、SACK、Window Scale、Timestamp 等，查资料说明其用途。

3. 观察和估计客户机到服务器的 RTT，双方各自的 MSS。

4. 观察 TCP 的流量控制过程，和拥塞控制中的慢启动、快速重传、拥塞避免，快速恢复等过程【观察拥塞控制的难度较大，观察到前两个过程即可】。

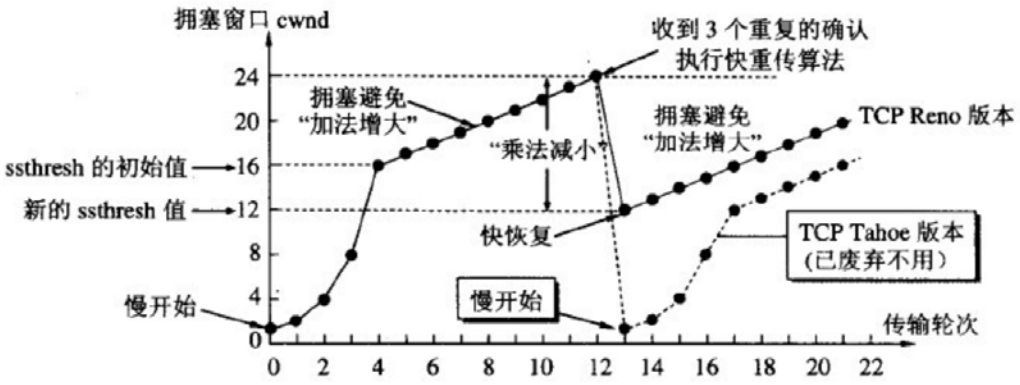


图 4-0 典型的 TCP 拥塞控制过程图例

5. *（可选）注意观察初始的 cwnd 是多少，看看不同的操作系统初始 cwnd 的差别。

【可以增加题目规定以外的分析】

三、 实验环境与分组

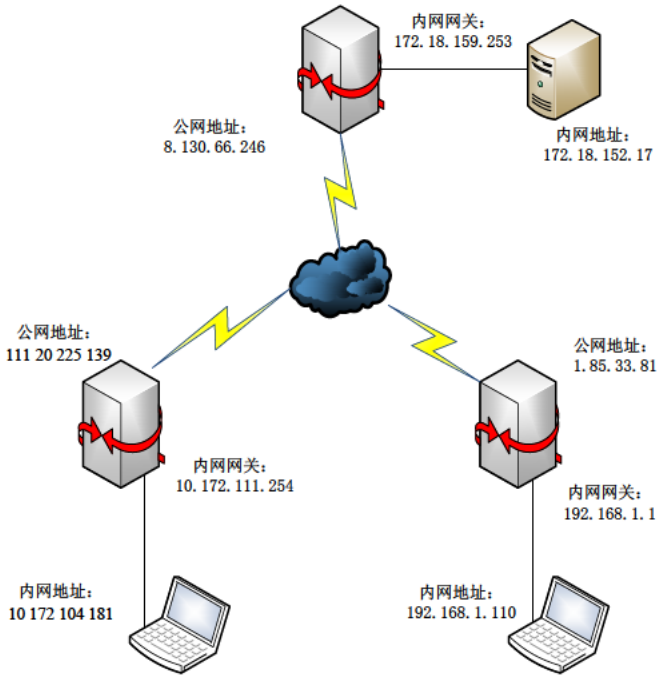
1）云服务器一台，启动 Apache2 服务（或其他服务器程序）。

2）每 2 名同学一组，各自在电脑上运行客户端程序（浏览器或其他客户端程序）。

3）使用客户端程序下载数据，运行 Wireshark 软件捕获报文。【注意：可以关闭 Wireshark 的 HTTP 协议分析，专注在 TCP 协议上，关闭方法是：菜单‘分析’—>‘启用的协议’中，取消‘HTTP’的选择。】

四、 实验组网

下图是本实验的组网图，图中参数请根据实际情况标注。



五、 实验过程及结果分析

【过程记录应当详尽，截图并加以说明。以下过程和表格仅供参考。】

步骤 1：PC2 登录到服务器 Z 上，在云服务器 Z 上启动 web 服务器（Apache、IIS 等）。

服务器系统为 Windows，开启的服务器程序为 Apache（2.4.55 版本）

```
administrator@IZ0D7ZIQWIN3WSZ C:\Apache\Apache24\bin>httpd -v
Server version: Apache/2.4.55 (Win64)
Server built:   Jan 15 2023 10:24:46
```

```
administrator@IZ0D7ZIQWIN3WSZ C:\Apache\Apache24\bin>httpd -k start
```

服务器上搭建的网页如下图所示：

步骤 2：在 PC1 和 Z 上启动报文捕获软件，开始截获报文（注意抓包一定要包括建立连接的报文）。

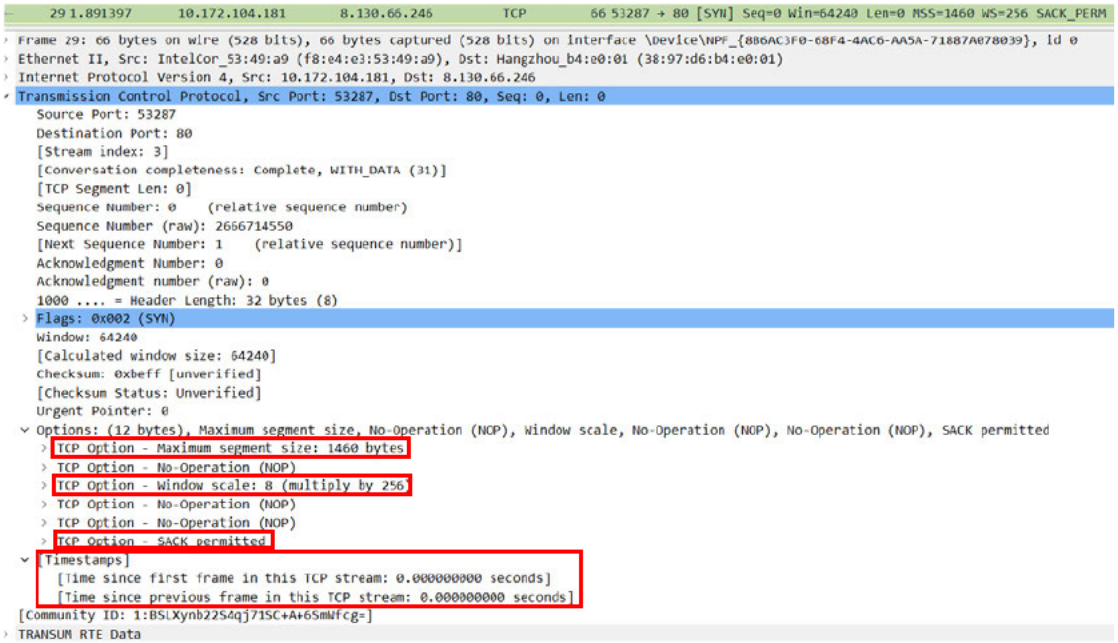
在 PC1 上打开 Wireshark 进行抓包；在 PC2 上通过 ECS 远程连接到服务器，同样在服务器上也打开 Wireshark 进行抓包。

步骤 3：在 PC1 上运行客户端软件，发送和接收一个约 500KB 的文件。文件传输完成后，停止报文截获。

在 PC1 上打开浏览器，访问服务器公网 IP 地址，然后开始下载文件。

步骤 4：对比观察客户端和服务器截获的报文，分析 TCP 协议的选项：MSS、SACK、Window Scale、Timestamp 等，查资料说明其用途。

①TCP 报文选项字段分析：



MSS：最大报文长度值，用于告诉对方期望收到的 TCP 报文数据部分的最大长度值。

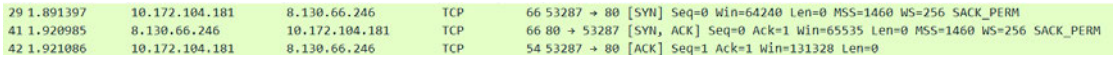
SACK：选择性确认技术，使 TCP 只重新发送通信过程中丢失的包，不用发送后续的所有包。

Window Scale：窗口缩放因子，用于计算实际窗口的大小（窗口值乘以缩放因子得到）。

Timestamp：时间戳，记录了从该 TCP 流的第一帧发送及上一帧发送起到当前帧发送经历的时间，可用于计算 RTT。

②TCP 建立连接：

客户端截获的三次握手报文



服务器截获的三次握手报文

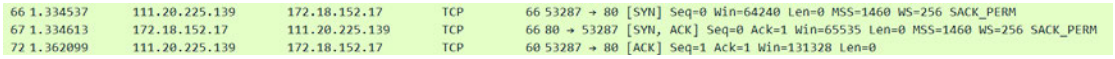


表 3-1 TCP 连接建立过程的三个报文信息

字段名称	第 1 条报文	第 2 条报文	第 3 条报文
报文序号 NO.	29	41	42
Seq #	0	0	1
Ack #	无	1	1
ACK Flag	0	1	1
SYN Flag	1	1	0

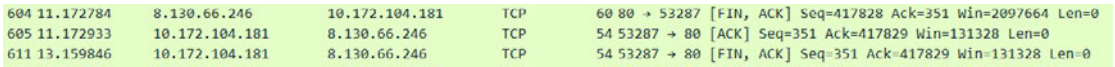
首先，客户端向服务器发送一个 SYN 位置 1 的连接请求报文，当服务器收到该报文后，发送 SYN 和 ACK 位均置 1 的报文来对第一个 SYN 报文段进行确认。最后，客户端再发送一个 ACK 位置 1 的确认报文，来通知服务器双方已完成连接的建立。

三次握手除了完成可靠连接的建立外，还使双方确认了各自的初始序号。从上图中可以看出，客户端在发送连接建立请求报文时，同时携带了序号 0（Seq=0）。在服务器对连接请求进行响应时，一方面对客户端的起始序号进行了确认（Ack=1），另一方面也发送了自己的起始序号 0（Seq=0）。最后，客户端在确认中携带了对服务器的起始序号 0 的确认（Ack=1）。

此外，为了达到最佳传输性能，TCP 在建立连接时还需要协商双方可接受的最大报文长度（MSS）及窗口缩放因子（WS）等参数。

③TCP 释放连接：

客户端截获的四次挥手报文



服务器截获的四次挥手报文

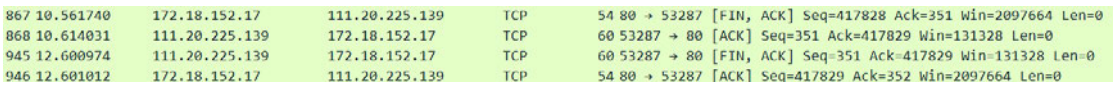


表 3-2 TCP 连接撤销过程的四个报文信息

字段名称	首条报文	二条报文	三条报文	四条报文

报文序号 NO.	604	605	611	622
Seq #	417828	351	351	417829
Ack #	351	417829	417829	352
ACK Flag	1	1	1	1
SYN Flag	1	0	1	0

TCP 采用对称的连接释放方式，在关闭一个方向的连接时，连接释放的发起方发送一个 FIN 位置 1 的报文，响应方的 TCP 进程收到报文后，对 FIN 报文段进行确认，并通知应用程序，整个通信会话已结束，此后，在该方向上无法继续传送数据。当连接的两个方向都已关闭时，该连接的两个端点的 TCP 进程将删除这个连接记录。

步骤 5：分析 TCP 数据传送阶段的报文，分析其错误恢复和流量控制机制，并填表。【注：出现明显的流量控制的地方，Wireshark 会有应答窗口的变化，乃至[TCP Window Full]或[TCP Zero Window]标记的报文出现。如果没有观察到明显的流量控制过程，可以再单独设计实验测试。比如编程设计接收端缓慢接收数据。】

①正常情况下：

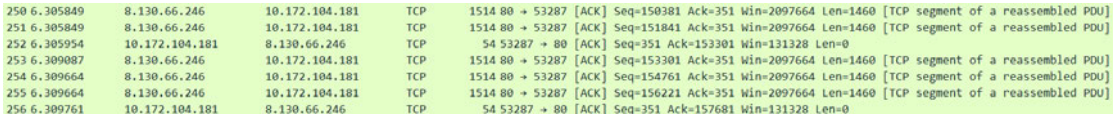


表 3-3 记录 TCP 数据传送阶段的报文

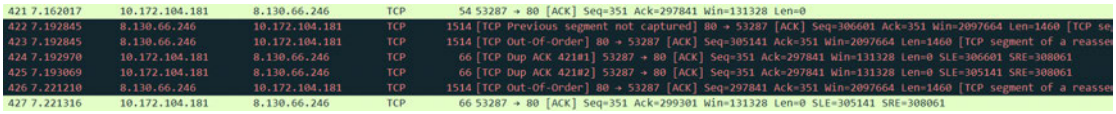
报文序号	报文种类(数据/确认)	序号字段 Seq Number	确认号 Ack Number	数据长度	确认到哪条报文（填序号）	窗口大小
250	数据	150381	351	1460	249	2097664
251	数据	151841	351	1460	249	2097664
252	确认	351	153301	0	251	131328
253	数据	153301	351	1460	252	2097664
254	数据	154761	351	1460	252	2097664
255	数据	156221	351	1460	252	2094664
256	确认	351	157681	0	255	131328

TCP 报文中的序号字段是指数据部分起始字节的序号，当接收端收到报文后，使用确认字段指出下一个期望接收的字节序号，也就是告诉对方，这个序号之前的字节都已经正确收到。

②错误恢复：

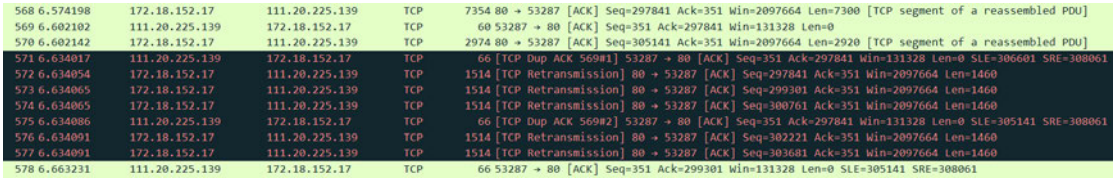
a）超时重传机制：

客户端截获的报文



从客户端角度来看，客户端首先发送了 Ack=297841 的确认报文，表示此序号前的报文均已收到，期望下次收到的是 Seq=297841 的报文。但是，紧接着客户端却收到了 Seq=306601 的报文，说明出现了丢包现象。之后，客户端又收到了 Seq=305141 的报文，说明出现了乱序现象。于是，客户端发送两个重复的 Ack=297841 的报文以提醒服务器发生了错误。最后，客户端收到了期望的 Seq=297841 的报文，并发送 Ack=299301 的报文进行确认。

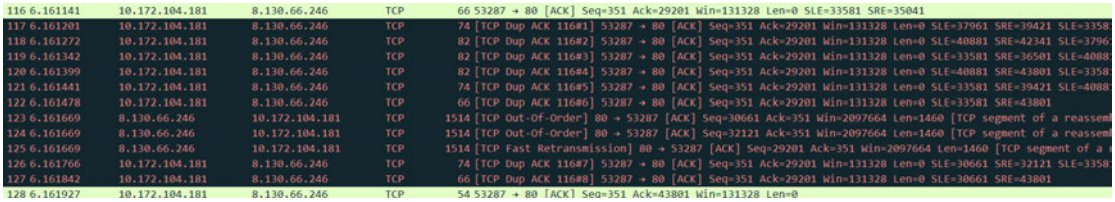
服务器截获的报文



从服务器角度来看，服务器首先发送了 Seq=297841 的报文，然后收到了客户端发来的 Ack=297841（表示客户端期望收到的下一个报文序号）的报文，接着服务器发送了 Seq=305141 的报文。但是，服务器却再次收到了客户端发来的 Ack=297841 的报文，且在一段时间内没有收到对先前发送报文的确认，这说明先前发送的数据丢失，于是服务器启动超时重传机制，重新发送之前的报文。最后，服务器收到了客户端发来的 Ack=299301 的报文，说明客户端收到了重传的报文。

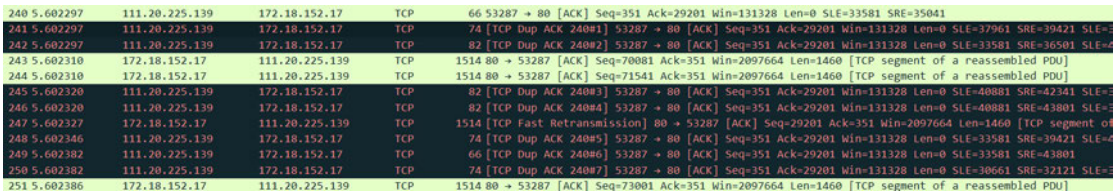
b）快速重传机制：

客户端截获的报文



从客户端角度来看，客户端在发送了 6 个重复的 Ack=29201 的报文后，收到了服务器快速重传的 Seq=29201 的报文。

服务器截获的报文



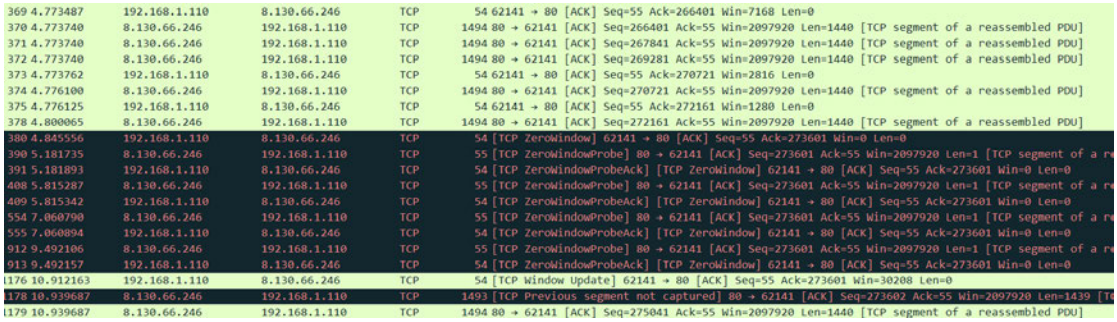
从服务器角度来看，服务器在收到客户端发来的 4 个重复的 Ack=29201 的报文后，启动快速重传机制，发送了 Seq=29201 的报文。紧接着，服务器又收到了 3 个重复的 Ack=29201 的报文，由于刚才已经重传过，故不再重传。

③流量控制：

为了能够观察到明显的流量控制过程，使用 python 的 socket 编程，限制接收方的数据处理速度，源代码如下：

```
import socket
import time

HOST = '8.130.66.246'
PORT = 80
FILE_PATH = '/download/R-C.jpg'
DOWNLOAD_FILE_NAME = 'test.jpg'

# 创建 socket 连接
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# 连接远程服务器
client_socket.connect((HOST, PORT))
# 修改接收缓冲区大小
client_socket.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, 256)

# 构建 HTTP 请求，请求文件
request = f'GET {FILE_PATH} HTTP/1.1\r\nHost:
{HOST}\r\n\r\n'.encode()
# 发送 HTTP 请求
client_socket.sendall(request)

# 接收服务器响应
response = b''
while True:
    data = client_socket.recv(256)
    if not data:
        break
    time.sleep(0.01)
    response += data

# 解析响应，获取文件内容
header, content = response.split(b'\r\n\r\n', 1)
# 获取 HTTP 状态码
status_code = int(header.split(b' ')[1])
if status_code == 200:
    # 将文件内容保存到当前目录下
    with open(DOWNLOAD_FILE_NAME, 'wb') as f:
        f.write(content)
    print('文件下载成功！')
else:
    print(f'文件下载失败，状态码：{status_code}')
```

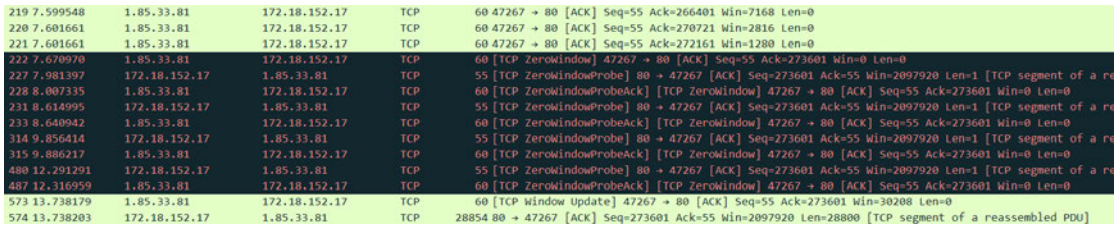
客户端截获的报文



369 4.773487	192.168.1.110	8.130.66.246	TCP	54 62141 → 80 [ACK] Seq=55 Ack=266401 Win=7168 Len=0
370 4.773740	8.130.66.246	192.168.1.110	TCP	1494 80 → 62141 [ACK] Seq=266401 Ack=55 Win=2097920 Len=1440 [TCP segment of a reassembled PDU]
371 4.773740	8.130.66.246	192.168.1.110	TCP	1494 80 → 62141 [ACK] Seq=267841 Ack=55 Win=2097920 Len=1440 [TCP segment of a reassembled PDU]
372 4.773740	8.130.66.246	192.168.1.110	TCP	1494 80 → 62141 [ACK] Seq=269281 Ack=55 Win=2097920 Len=1440 [TCP segment of a reassembled PDU]
373 4.773762	192.168.1.110	8.130.66.246	TCP	54 62141 → 80 [ACK] Seq=55 Ack=270721 Win=2816 Len=0
374 4.776100	8.130.66.246	192.168.1.110	TCP	1494 80 → 62141 [ACK] Seq=270721 Ack=55 Win=2097920 Len=1440 [TCP segment of a reassembled PDU]
375 4.776125	192.168.1.110	8.130.66.246	TCP	54 62141 → 80 [ACK] Seq=55 Ack=272161 Win=1280 Len=0
378 4.800065	8.130.66.246	192.168.1.110	TCP	1494 80 → 62141 [ACK] Seq=272161 Ack=55 Win=2097920 Len=1440 [TCP segment of a reassembled PDU]
380 4.845556	192.168.1.110	8.130.66.246	TCP	54 [TCP ZeroWindow] 62141 → 80 [ACK] Seq=55 Ack=273601 Win=0 Len=0
390 5.181735	8.130.66.246	192.168.1.110	TCP	55 [TCP ZeroWindowProbe] 80 → 62141 [ACK] Seq=273601 Ack=55 Win=2097920 Len=1 [TCP segment of a re
391 5.181893	192.168.1.110	8.130.66.246	TCP	54 [TCP ZeroWindowProbeAck] [TCP ZeroWindow] 62141 → 80 [ACK] Seq=55 Ack=273601 Win=0 Len=0
408 5.815287	8.130.66.246	192.168.1.110	TCP	55 [TCP ZeroWindowProbe] 80 → 62141 [ACK] Seq=273601 Ack=55 Win=2097920 Len=1 [TCP segment of a re
409 5.815342	192.168.1.110	8.130.66.246	TCP	54 [TCP ZeroWindowProbeAck] [TCP ZeroWindow] 62141 → 80 [ACK] Seq=55 Ack=273601 Win=0 Len=0
554 7.060790	8.130.66.246	192.168.1.110	TCP	55 [TCP ZeroWindowProbe] 80 → 62141 [ACK] Seq=273601 Ack=55 Win=2097920 Len=1 [TCP segment of a re
555 7.060894	192.168.1.110	8.130.66.246	TCP	54 [TCP ZeroWindowProbeAck] [TCP ZeroWindow] 62141 → 80 [ACK] Seq=55 Ack=273601 Win=0 Len=0
912 9.492106	8.130.66.246	192.168.1.110	TCP	55 [TCP ZeroWindowProbe] 80 → 62141 [ACK] Seq=273601 Ack=55 Win=2097920 Len=1 [TCP segment of a re
913 9.492157	192.168.1.110	8.130.66.246	TCP	54 [TCP ZeroWindowProbeAck] [TCP ZeroWindow] 62141 → 80 [ACK] Seq=55 Ack=273601 Win=0 Len=0
176 10.912163	192.168.1.110	8.130.66.246	TCP	54 [TCP Window Update] 62141 → 80 [ACK] Seq=55 Ack=273601 Win=30208 Len=0
178 10.939687	8.130.66.246	192.168.1.110	TCP	1493 [TCP Previous segment not captured] 80 → 62141 [ACK] Seq=273602 Ack=55 Win=2097920 Len=1439 [T
179 10.939687	8.130.66.246	192.168.1.110	TCP	1494 80 → 62141 [ACK] Seq=275041 Ack=55 Win=2097920 Len=1440 [TCP segment of a reassembled PDU]

从客户端角度来看，刚开始客户端的接收窗口大小为 7168B，随着不断接收来自服务器的报文，窗口大小逐渐减小为 2816B、1280B。接着，当客户端再接收一个 1440B 的报文后，客户端接收窗口满，便向服务器发送一个标记为[TCP ZeroWindow]的报文（Win=0）。然后，客户端收到服务器发来的零窗口探测报文（标记为[TCP ZeroWindowProbe]），并向服务器发送一个标记为[TCP ZeroWindowProbeAck]的应答报文，用以告诉服务器当前的接收窗口大小。此过程重复多次，直至客户端处理完接收窗口内的数据，并向服务器发送一个标记为[TCP Window Update]的报文，用以通知服务器当前接收窗口大小的最新值。之后，客户端就又能收到服务器发来的数据了。

服务器截获的报文



219 7.599548	1.85.33.81	172.18.152.17	TCP	60 47267 → 80 [ACK] Seq=55 Ack=266401 Win=7168 Len=0
220 7.601661	1.85.33.81	172.18.152.17	TCP	60 47267 → 80 [ACK] Seq=55 Ack=270721 Win=2816 Len=0
221 7.601661	1.85.33.81	172.18.152.17	TCP	60 47267 → 80 [ACK] Seq=55 Ack=272161 Win=1280 Len=0
222 7.670970	1.85.33.81	172.18.152.17	TCP	60 [TCP ZeroWindow] 47267 → 80 [ACK] Seq=55 Ack=273601 Win=0 Len=0
227 7.981397	172.18.152.17	1.85.33.81	TCP	55 [TCP ZeroWindowProbe] 80 → 47267 [ACK] Seq=273601 Ack=55 Win=2097920 Len=1 [TCP segment of a re
228 8.007335	1.85.33.81	172.18.152.17	TCP	60 [TCP ZeroWindowProbeAck] [TCP ZeroWindow] 47267 → 80 [ACK] Seq=55 Ack=273601 Win=0 Len=0
231 8.614995	172.18.152.17	1.85.33.81	TCP	55 [TCP ZeroWindowProbe] 80 → 47267 [ACK] Seq=273601 Ack=55 Win=2097920 Len=1 [TCP segment of a re
233 8.640942	1.85.33.81	172.18.152.17	TCP	60 [TCP ZeroWindowProbeAck] [TCP ZeroWindow] 47267 → 80 [ACK] Seq=55 Ack=273601 Win=0 Len=0
314 9.856414	172.18.152.17	1.85.33.81	TCP	55 [TCP ZeroWindowProbe] 80 → 47267 [ACK] Seq=273601 Ack=55 Win=2097920 Len=1 [TCP segment of a re
315 9.886217	1.85.33.81	172.18.152.17	TCP	60 [TCP ZeroWindowProbeAck] [TCP ZeroWindow] 47267 → 80 [ACK] Seq=55 Ack=273601 Win=0 Len=0
480 12.291291	172.18.152.17	1.85.33.81	TCP	55 [TCP ZeroWindowProbe] 80 → 47267 [ACK] Seq=273601 Ack=55 Win=2097920 Len=1 [TCP segment of a re
487 12.316959	1.85.33.81	172.18.152.17	TCP	60 [TCP ZeroWindowProbeAck] [TCP ZeroWindow] 47267 → 80 [ACK] Seq=55 Ack=273601 Win=0 Len=0
573 13.738179	1.85.33.81	172.18.152.17	TCP	60 [TCP Window Update] 47267 → 80 [ACK] Seq=55 Ack=273601 Win=30208 Len=0
574 13.738203	172.18.152.17	1.85.33.81	TCP	28854 80 → 47267 [ACK] Seq=273601 Ack=55 Win=2097920 Len=28800 [TCP segment of a reassembled PDU]

从服务器角度来看，服务器收到的来自客户端的确认报文中携带有客户端接收窗口大小的信息。当服务器收到客户端的[TCP ZeroWindow]报文后，便停止向客户端发送数据。但由于此时服务器仍有数据需要发送给客户端，于是服务器会向客户端发送零窗口探测报文（[TCP ZeroWindowProbe]），并收到客户端的应答报文（[TCP ZeroWindowProbeAck]），以便及时了解客户端的接收窗口大小值。此过程不断重复，直至服务器收到客户端发来的[TCP Window Update]报文，并从中获取了客户端接收窗口大小的最新消息。之后，服务器就能继续向客户端发送数据了。

步骤 6、分析客户机和服务器两边各自捕获到的分组，分析整个 TCP 流，估计双方的 RTT。

①客户端：

客户端于 1.891397s 发出第一次握手报文，于 1.920985s 收到第二次握手报文，故 RTT_1=1.920985s - 1.891397s =29.588ms

客户端于 13.159846s 发出第三次挥手报文，于 13.188754s 收到第四次挥手

报文，故 RTT_2=13.188754s -13.159846s =28.908ms

故客户端的 RTT=(RTT_1+ RTT_2) / 2=29.248ms

②服务器：

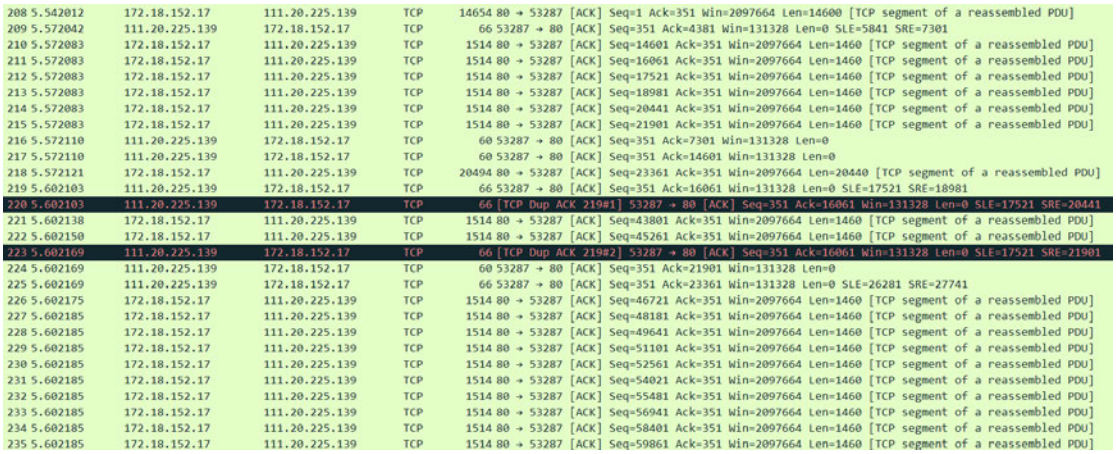
服务器于 1.334613s 发出第二次握手报文，于 1.362099s 收到第三次握手报文，故 RTT_1=1.362099s - 1.334613s =27.486ms

服务器于 10.561740s 发出第一次挥手报文，于 10.614031s 收到第二次挥手报文，故 RTT_2=10.614031s -10.561740s =52.291ms

故服务器的 RTT=(RTT_1+ RTT_2) / 2=39.889ms

步骤 7、分析整个 TCP 流的拥塞控制，找到拥塞控制的几个典型过程（即慢启动、快速重传等）。

①慢启动：

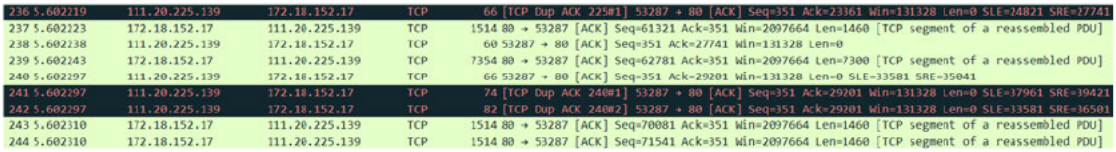


当客户端与服务器建立 TCP 连接后，客户端向服务器发送了一个 HTTP 请求，于是服务器便开始向客户端发送数据。刚开始，服务器发送了 10 个 MSS 报文段（初始拥塞窗口大小），在收到客户端发来的对 3 个 MSS 段的 ACK 报文后，服务器又继续向客户端发送 6 个 MSS 报文段。此后的过程简述如下：

服务器	报文（单位 MSS）	客户端
	7 ACK	←
→	14 Data	
	1 ACK	←
→	2 Data	
	5 ACK	←
→	10 Data	

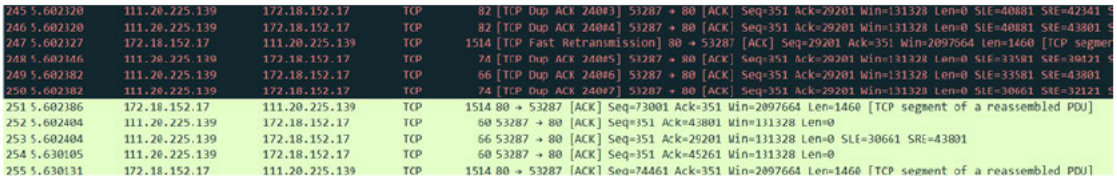
在这个过程中，服务器每收到对一个 MSS 数据段的 ACK 确认报文，拥塞窗口就增加一个 MSS 数据段。因此，此时拥塞控制处于慢启动阶段。

②拥塞避免：



在这个过程中，发送方的拥塞窗口增长与 ACK 报文的数量无关，而是以一个往返时间（RTT）为单位线性增长，故拥塞控制处于拥塞避免阶段。

③快速重传：



在这个过程中，服务器收到 4 个重复的 ACK 报文后，没有等待重传定时器超时就重传看来已经丢失的数据段。此时，拥塞控制处于快速重传阶段

步骤 8、如果拥塞控制的相关过程不明显，请设计合适的方法再次测试。

步骤 9、完成其他可选的实验步骤。

观察初始的 cwnd 是多少，看看不同的操作系统初始 cwnd 的差别。

①Windows：

服务器使用的操作系统为 Windows，在步骤 7 的拥塞控制慢启动阶段中，通过分析发现服务器最开始传输了 10 个 MSS 数据段给客户端，之后等待收到客户端的 ACK 报文后才继续发送数据，因此可推测其初始 cwnd 为 10 个 MSS。在命令行下查看其 TCP 参数为：

```
administrator@IZ0D7ZIQWIN3WSZ C:\Users\Administrator>netsh int tcp show supplemental

TCP 全局默认模板为 automatic


自动模式基于 TCP 连接参数在 Internet 和 Datacenter 模板
之间选取。

请使用 "netsh int tcp show supplementalports" 和
"netsh int tcp show supplementalsubnets" 命令查看活动的筛选器。
```

```
TCP 全局默认模板为 internet

TCP 补充参数
-------------------------------------------
最小 RTO (毫秒)                  : 300
初始拥塞窗口(MSS)      : 10
拥塞控制提供程序          : cubic
启用拥塞窗口重新启动     : disabled
延迟 ACK 超时(毫秒)           : 40
延迟 ACK 频率                : 2
启用 RACK                        : enabled
启用尾部丢失探测              : enabled

请使用 "netsh int tcp show supplementalports" 和
"netsh int tcp show supplementalsubnets" 命令查看活动的筛选器。
```

因此，Windows 操作系统的初始 cwnd 为 10 个 MSS。

②Linux：

在 Linux 下通过 ss -it 命令可查看 TCP 连接的相关参数，同样可以发现 Linux 的初始 cwnd 为 10 个 MSS。

六、 互动讨论主题

1）TCP 的流量控制和拥塞控制有什么不同?

流量控制由接收方主导，主要解决发送方与接收方速率不匹配的问题，通过滑动窗口机制控制发送方发送数据的速率，以便接收方来得及接收和处理数据;

拥塞控制由发送方主导，其目的是保证网络负荷不会超过通信子网的承载能力，通过慢启动、拥塞避免、快速重传和快速恢复算法来动态调整拥塞窗口。

2）TCP 的流量控制是哪一方（接收、发送）来主导的？什么情况下会发生流量控制?

TCP 流量控制由接收方主导，接收方通过确认报文通知发送方接收窗口的大小，以便发送方据此决定是否继续发送数据。

当发送方已经发出但尚未收到确认的报文长度总和即将超过接收窗口大小时，发送方便会暂缓发包，等待收到对一部分报文的确认后再继续发送，此时便发生了流量控制。

3）讨论传输层与其上下相邻层的关系;

传输层为上层应用程序提供了可靠的数据传输服务，为应用层提供了简单的接口，使得应用程序可以轻松地发送和接收数据;

传输层使用下层网络层提供的服务将数据传送给目的主机，传输层将应用层的数据分割成较小的数据包以提高网络效率，然后再将这些数据包递交给网络层进行转发。

4）讨论 TCP 协议在传输实时语音流方面的优缺点。

优点：TCP 协议具有高可靠性，可以保证传输数据的完整性;

缺点：TCP 协议连接建立过程较慢，且重传机制会导致语音流出现卡顿，从而使得传输的实时性较差。