实验三、传输层 TCP 协议分析实验报告

组号：_____

姓名：_____ 学号：_____ 班级：_____

姓名：_____ 学号：_____ 班级：_____

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。

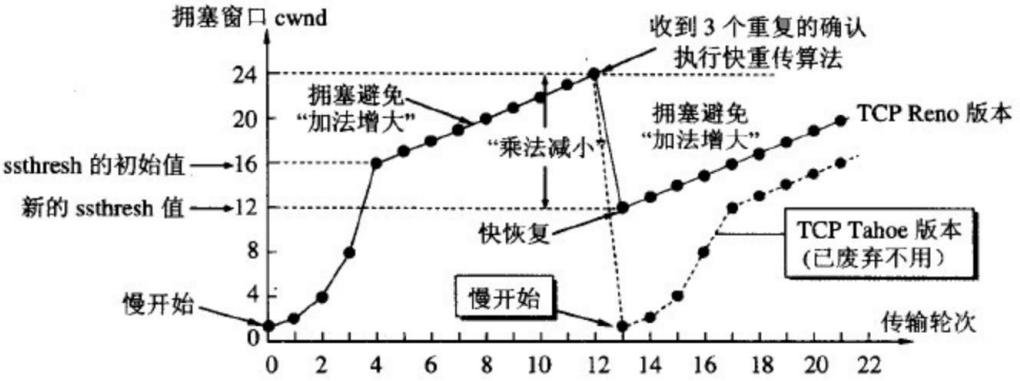


图 4-0 典型的 TCP 拥塞控制过程图例

三、 实验环境与分组

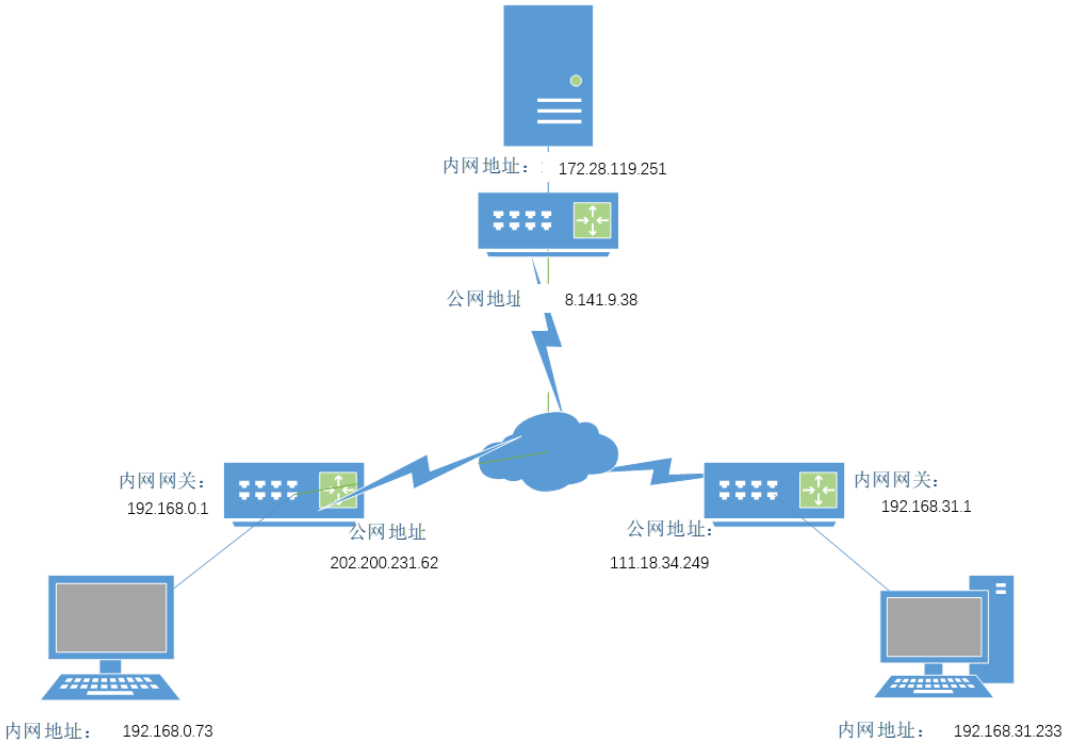
1）云服务器一台，启动 Apache2 服务（或其他服务器程序）。

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。

四、 实验组网

下图是本实验的组网图。



五、 实验过程及结果分析

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）。

在云服务器上安装 Apache2，有的云服务器软件包管理器 apt 需要更新，可以使用下面的命令。

```
sudo apt update
sudo apt upgrade
```

安装 Apache2 可以使用下面的命令。

```
sudo apt-get install apache2

sudo ufw allow 'Apache Full'"
```

Apache2 安装后可能无法通过 web 访问云服务器，这是因为其默认安全组将拒绝访问。可以将安全组的入口全部开启。
访问服务器（8.141.9.38），可以发现 Apache2 已经成功部署，见下图。

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）。

在服务器上可以使用 tcpdump 命令抓包，见下。

```
tcpdump -n tcp -w server.pcap
```

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。

从 PC1 上传文件至服务器，再从服务器上下载文件。具体地，即将文件上传至 /var/www/html/my 中（my 目录自行创建）。上传可以使用 scp 命令,见下。可以发现文件上传成功且可以被用户下载。

```
scp /local root@server_ip:/target_location
```

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。

在截获的报文中可以发现有的报文 Options 项被填充，有的没有。这些选项都是为了增强 TCP 协议的性能、可靠性和灵活性而设计的。MSS 用于告知最大报文段长度，窗口扩大因子用于扩大窗口以提高网络吞吐量，SACK 则用于提高 TCP 丢包后的重传效率，而时间戳选项则用于实现更精确的拥塞控制和 RTT 测量。以图 5.4.1 为例说明。

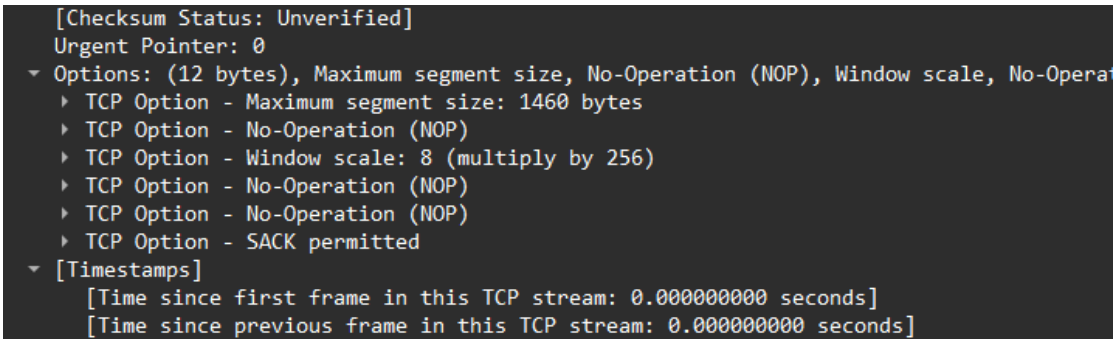


图 5.4.1

图 5.4.2

Maximum Segment Size (MSS)：最大报文段长度，指示了发送方所能接收的最大 TCP 报文段的大小。在这个报文中，MSS 被设置为 1460 字节。
Window Scale：窗口扩大因子，用于在 TCP 通信中对窗口进行扩大，以支持更大的窗口大小。在这个报文中，窗口扩大因子被设置为 8，表示接收窗口大小需要乘以 256（2<<8） 才能得到真实的大小。这在后面的报文中可以体现，即 $Calculated\ window\ size = Widow(size) * (2 << Window\ scale)$，可见图 5.4.2。
Selective Acknowledgment (SACK)：选择确认，用于在 TCP 通信中允许接收方指定已经成功接收的 TCP 报文段范围，以实现重传。在这个报文中，SACK 被设置为允许。
Timestamps：时间戳选项，用于测量往返时间（RTT）和实现更精确的拥塞控制。TCP Timestamps Option 由四部分构成：类别（kind）、长度（Length）、发送方时间戳（TS value）、回显时间戳（TS Echo Reply）。在这个报文中，时间戳选项被用于记录从 TCP 会话开始到当前的时间。在图 5.4.1 中，Time since first frame in this TCP stream 和 Time since previous frame in this TCP stream 都为 0 秒，这是因为在 TCP 连接的初始阶段，时间戳还没有开始计时，且没有前一个帧来计算时间间隔。

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。】

在实验中我们遇到的问题主要是在传输文件较大时出现的集中重传和窗口到达上限。

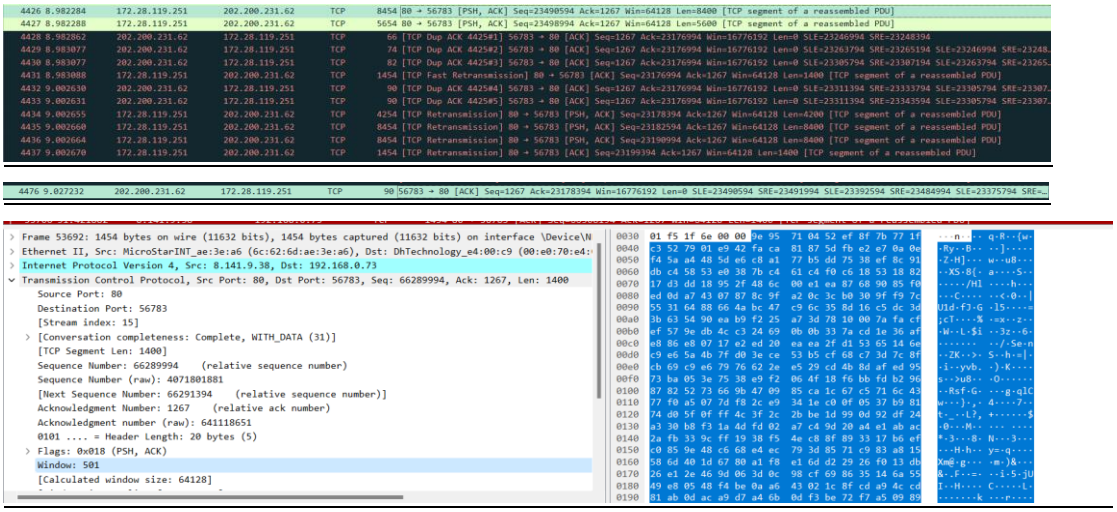


表 3-3（1） 记录 TCP 数据传送阶段的报文——TCP Retransmission

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
4426	数据	23490594	1267	8400	23176994	64128
4427	数据	23498994	1267	5600	23176994	64128
4428	确认	1267	23176994	0	23176994	16776192
4429	确认	1267	23176994	0	23176994	16776192
4430	确认	1267	23176994	0	23176994	16776192
4431	数据	23176994	1267	1400	23176994	64128
4432	确认	1267	23176994	0	23176994	16776192
4433	数据	1267	23176994	0	23176994	16776192
4434	数据	23178394	1267	4200	23176994	64128
4435	数据	23182594	1267	8400	23176994	64128
4436	数据	23190994	1267	8400	23176994	64128
4476	确认	1267	23178394	0	23178394	16776192

由于重传跨度极大，到发送接收端同步需要上百个报文，因此仅选取具有代表性的几个，如表中 4426-4427 为发送端在得知需要重传前发送 SEQ 号靠后（已经发送到 SEQ 23498994）的内容，在 4428-4430 及 4432-4433 的重复 ACK 后得知需要重传，则回到断点 23176994 开始重传，直到 4476 时接收端回复已收到重传

的 SEQ 为 23176994 的长度为 1400 的报文，因此回复 SEQ 号 23178394，两端开始同步。

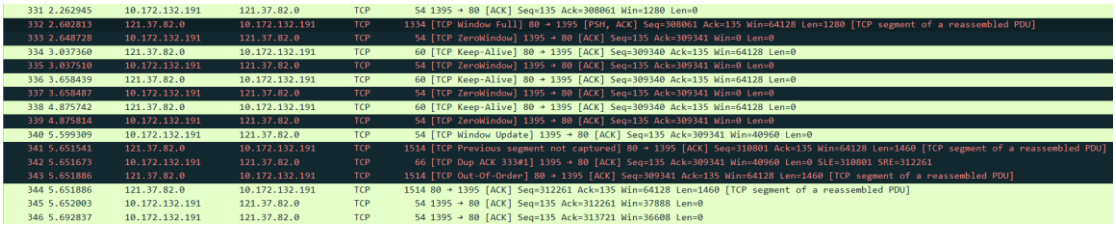
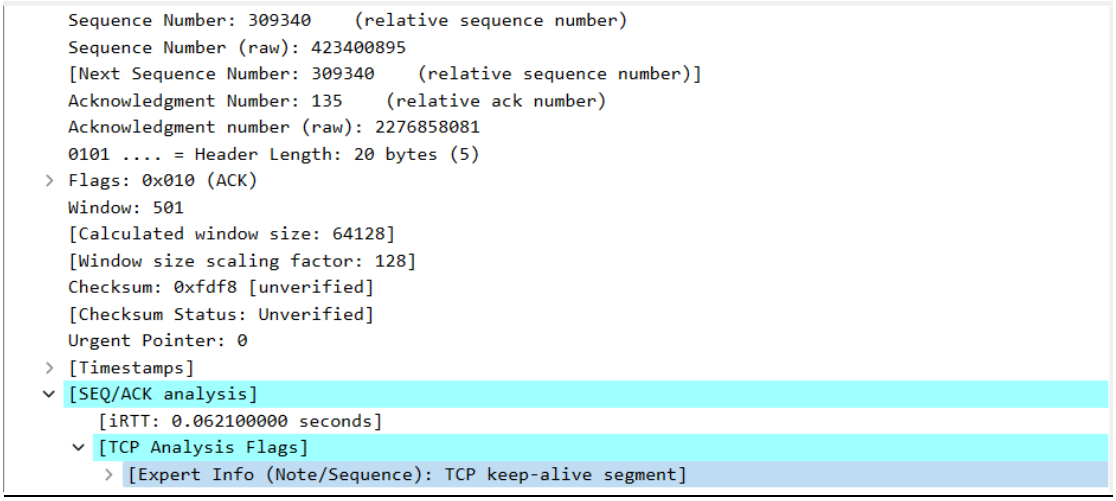


表 3-3（2） 记录 TCP 数据传送阶段的报文——Window Full

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）
332	数据	308061	135	1334	308061
333	确认	135	309341	54	309341
334	数据（0）	309340	135	60	309340
335	确认	135	139341	54	309341
336	数据（0）	309340	135	60	309340
337	确认	135	309341	54	309341
338	数据（0）	309340	135	60	309340
339	确认	135	309341	54	309341
340	确认（update）	135	309341	54	309341
341	数据	310801	135	1514	310801
342	确认	135	309341	66	309341
343	数据	309341	135	1514	309341

其中，数据（0）是指为保持 TCP 连接的空数据包，确认（update）更新了零窗口状态，如图所示。



```
Acknowledgment Number: 309341    (relative ack number)
Acknowledgment number (raw): 423400896
0101 .... = Header Length: 20 bytes (5)
> Flags: 0x010 (ACK)
Window: 160
[Calculated window size: 40960]
[Window size scaling factor: 256]
Checksum: 0x5aab [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
> [Timestamps]
v [SEQ/ACK analysis]
    [iRTT: 0.062100000 seconds]
  v [TCP Analysis Flags]
    v [Expert Info (Chat/Sequence): TCP window update]
        [TCP window update]
        [Severity level: Chat]
        [Group: Sequence]
```

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。

RTT(Round-Trip Time)是一个重要的性能指标，表示从发送端发送数据开始，到发送端收到来自接收端的确认（接收端收到数据后便立即发送确认），总共经历的时延，见图 5.6.1。图 5.6.2 是在服务器报文中截取的，图 5.6.3 是在客户机报中文截取的。

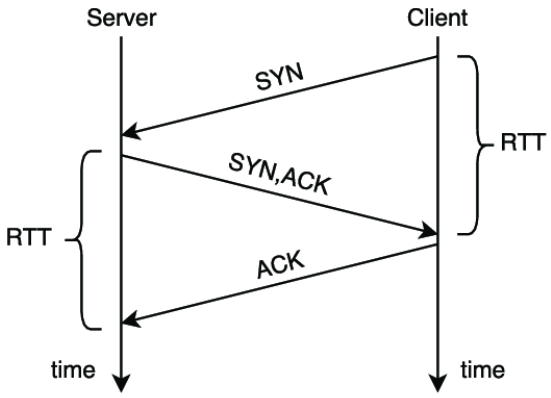


图 5.6.1

```
[SEQ/ACK analysis]
  [iRTT: 0.044097000 seconds]
  [Bytes in flight: 2619400]
  [Bytes sent since last PSH flag: 5600]
```

图 5.6.2

```
[SEQ/ACK analysis]
  [iRTT: 0.044255000 seconds]
  [Bytes in flight: 4200]
  [Bytes sent since last PSH flag: 1400]
```

图 5.6.3

则

$$RTT1 = 44.1ms$$
$$RTT2 = 44.3ms$$

因此双方的 RTT 可以估计为：

$$RTT = \frac{RTT1 + RTT2}{2} = 44.2\,ms$$

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）。

整体的窗口情况如图图 5.7.1，慢启动和拥塞避免可见图 5.7.2。

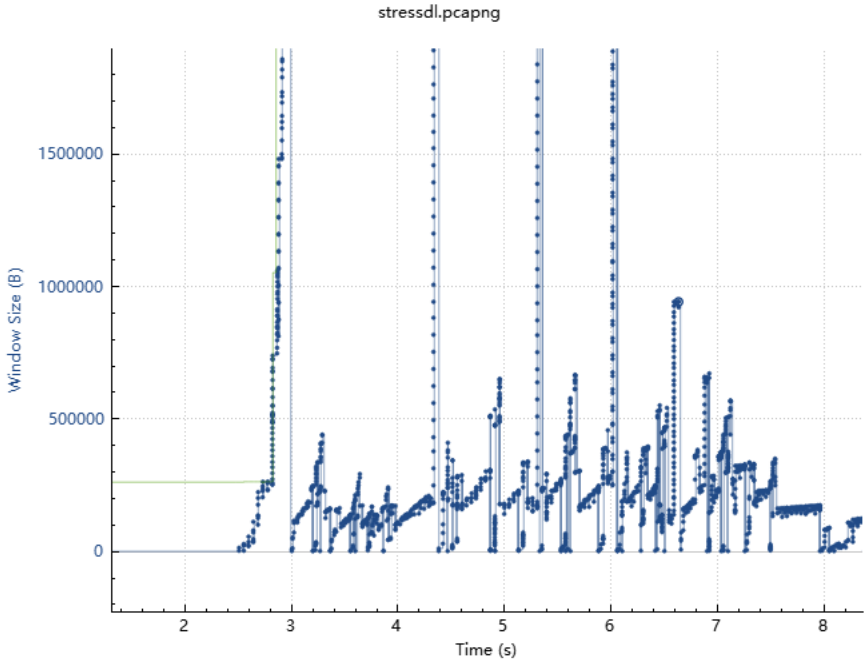


图 5. 7. 1

慢启动：算法通过不断增加发送窗口的大小来逐步增加发送数据的速率。初始阶段，发送窗口大小被设置为一个较小的值（通常为一个 MSS，即最大报文段长度），然后每当收到一个确认（ACK）时，发送窗口大小会翻倍。这样逐渐增大发送窗口大小，直到达到一个阈值（慢启动阈值）可以看出，窗口大小存在一个骤减，骤减后，窗口大小很小并在之后呈迅速增长。

拥塞避免：由于这是离散的数据，因此往往会呈阶梯形，但是仍然可以发现，在前一段时间内窗口大小迅速增加，之后以较慢的速度增加，这就是拥塞避免。

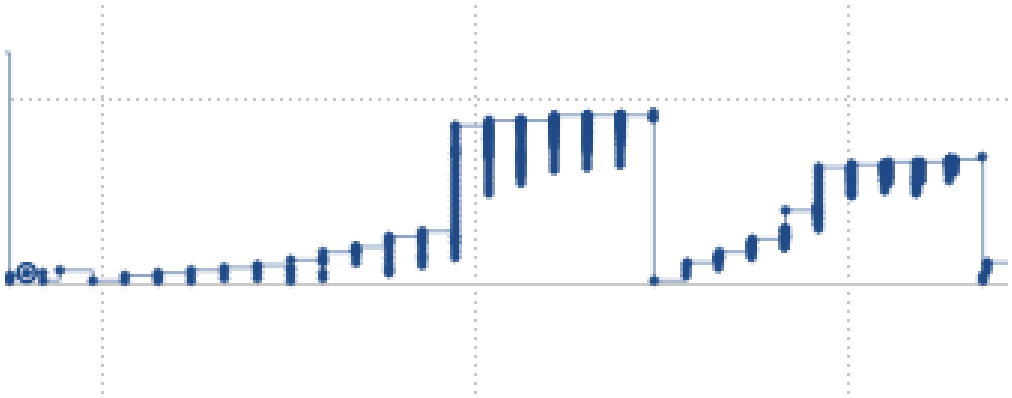


图 5.7.2

快速重传：当接收方收到乱序的数据包时，会立即发送一个冗余的确认。如果发送方连续收到三个相同的冗余确认（即确认了同一个数据包丢失），其将 cwnd 设置为当前的一半，跳过慢启动阶段，这样可以更快地恢复丢失的数据包，提高传输效率。图 5.7.3 和图 5.7.4 描述了这一情形。

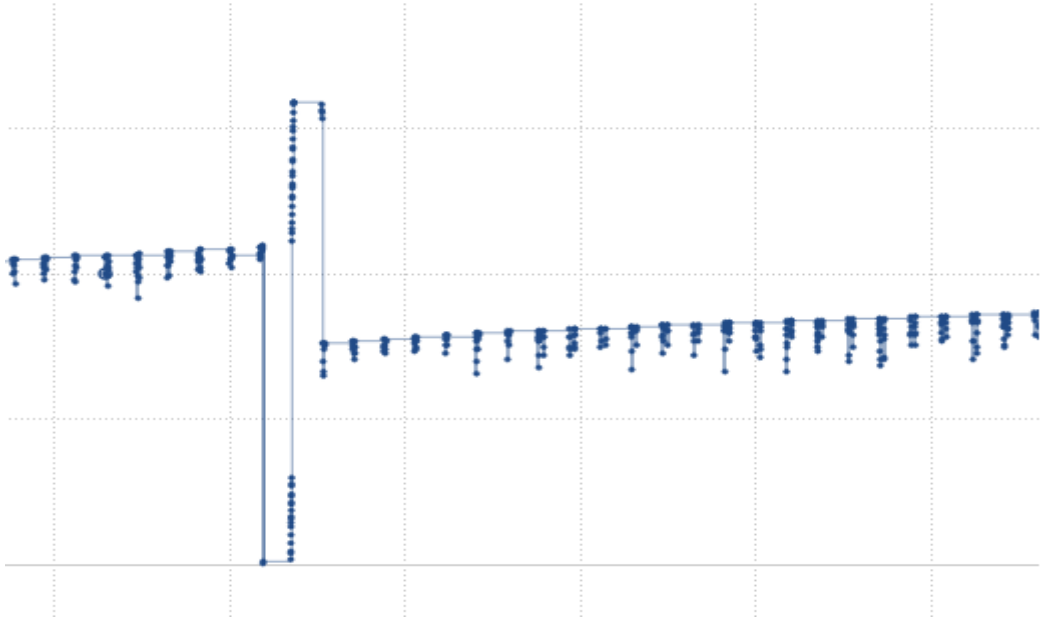


图 5.7.3

940	5.048737	172.28.119.251	202.200.231.62	TCP	5654 [TCP Retransmission] 80 → 56783 [PSH, ACK]
941	5.048745	172.28.119.251	202.200.231.62	TCP	9854 [TCP Retransmission] 80 → 56783 [PSH, ACK]
942	5.048756	172.28.119.251	202.200.231.62	TCP	7054 [TCP Retransmission] 80 → 56783 [PSH, ACK]
943	5.048775	202.200.231.62	172.28.119.251	TCP	90 [TCP Window Update] 56783 → 80 [ACK] Seq=1

图 5.7.4

六、 互动讨论主题

1）TCP 的流量控制和拥塞控制有什么不同？

（a） 流量控制是指在数据发送方和接收方之间进行控制，以确保发送方发送的数据不会超出接收方处理能力的能力范围。在 TCP 中，流量控制通过接收窗口来实现。接收方会在 TCP 报文中的窗口字段中通知发送方，告诉发送方它还能够接收多少数据。发送方根据接收方提供的窗口大小来调整发送数据的速率，以

确保不会导致接收方缓冲区溢出。
（b） 拥塞控制是指在整个网络中进行控制，防止过多的数据注入到网络中以避免网络拥塞。在 TCP 中，拥塞控制通过动态调整发送方的发送速率来实现，常用的方法有：（1）慢启动、拥塞避免（2）快速重传、快恢复。

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 的流量控制是由接收方来主导的。在 TCP 中，接收方通过发送窗口字段来告知发送方自己的可用缓冲区大小，即接收窗口大小。发送方根据接收窗口大小来控制自己的发送速率，以确保不会发送超过接收方处理能力的数据量。如果接收方的缓冲区已满，接收窗口为零，发送方将停止发送数据，直到接收方的缓冲区有空间可用。因此，流量控制主要发生在接收方的一侧，由接收方来主导。

3）讨论传输层与其上下相邻层的关系；
（a）传输层与应用层（上层）
传输层为应用层提供了端到端的数据传输服务，使得应用程序之间能够进行通信。应用层向传输层提供数据，并指定了所需的服务质量和传输参数。传输层将应用层数据进行分段（Segmentation）并添加相应的传输控制信息，以便在网络中正确传输。例如，TCP 协议在数据上添加序号、确认号、校验和等信息。
（b）传输层与网络层（下层）
传输层位于网络层之上，利用网络层提供的路由服务来实现端到端的数据传输。传输层通过网络层提供的 IP 地址来识别网络中的主机，并利用 IP 地址来确定数据报的传输路径。

4）讨论 TCP 协议在传输实时语音流方面的优缺点。
（a）优点：TCP 提供可靠的端到端数据传输机制。从而确保语音数据的稳定传输。
（b）缺点：TCP 协议在保证可靠性和流量控制的同时，引入了较大的延迟，影响语音通话的实时性和流畅性。