实验三、传输层 TCP 协议分析实验报告

组号：	7-1				
姓名：	白佳兴	学号：	2204311549	班级：	计算机 2105
姓名：	廖立彬	学号：	2213611635	班级：	计算机 2105

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】。

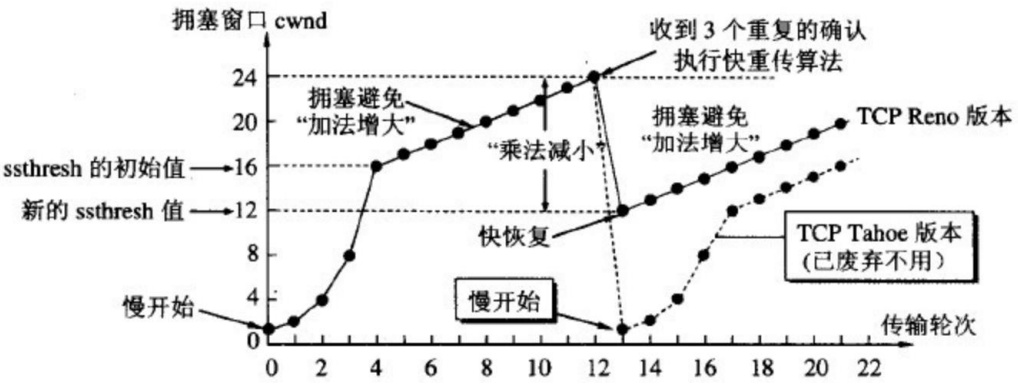


图 4-0 典型的 TCP 拥塞控制过程图例

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

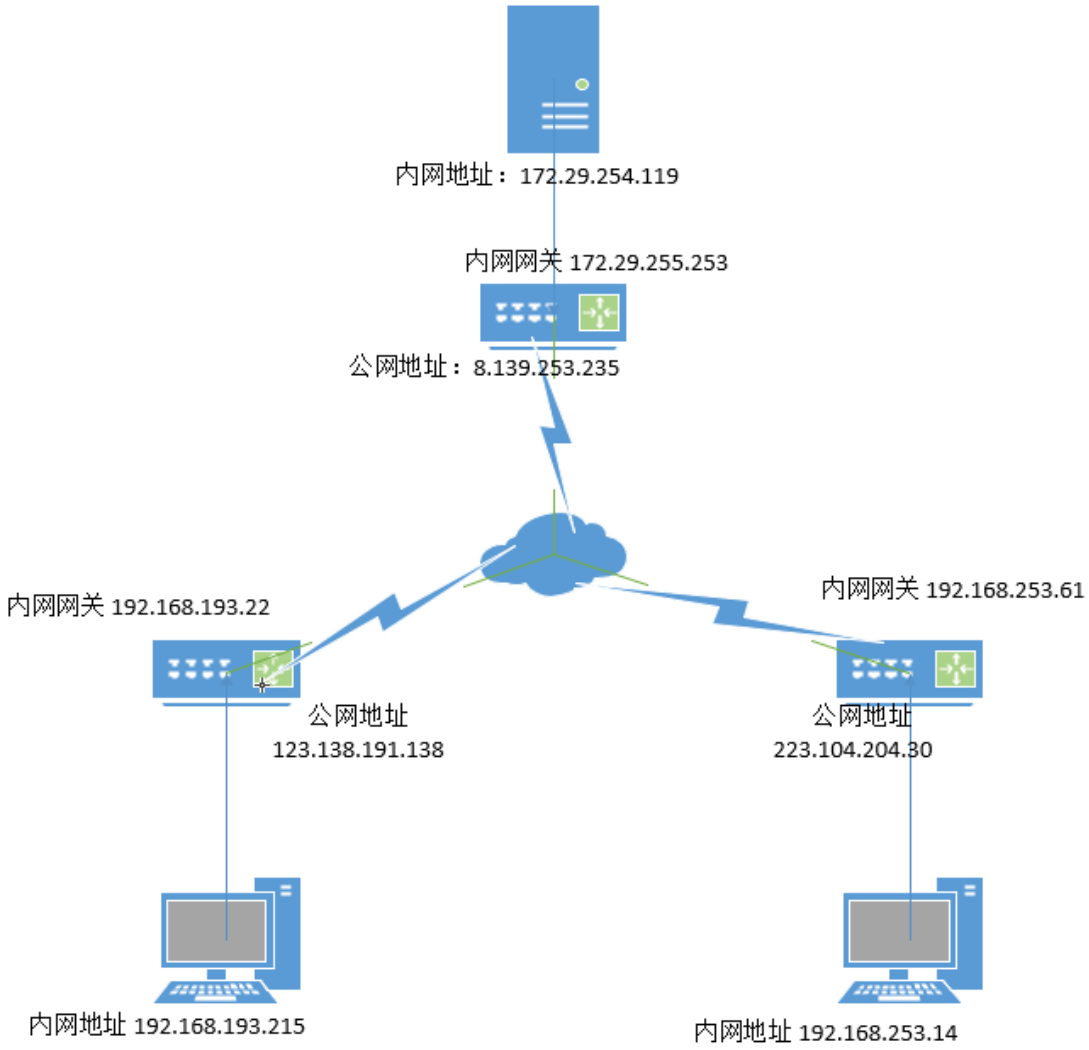
1）云服务器一台，启动 Apache2 服务（或其他服务器程序）。

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）。

在服务器上输入 `python3 -m http.server 80` 命令启动 web 服务器，在 PC1 上输入服务器公网地址 `8.139.253.235`，进入指定的服务器目录

Directory listing for /

- .bash_history
- .bashrc
- .cache/
- .config/
- .pip/
- .profile
- .pydistutils.cfg
- .python_history
- .rpmdb/
- .ssh/
- .vscode-server/
- .wget-hsts
- .Xauthority
- snap/
- test3.pcap

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）。

```
root@iZbp15o1gwawi1azkj5a6lZ:~#  tcpdump -n -s 500 tcp and host 8.139.253.235 and port 80 -w server.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 500 bytes
```

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。

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。

本机：

```
1450 5.916640  192.168.253.14  8.139.253.235   TCP    66 5254 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
1642 6.077349  8.139.253.235   192.168.253.14  TCP    66 80 → 5254 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1370 SACK_PERM WS=128
1643 6.077466  192.168.253.14  8.139.253.235   TCP    54 5254 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
```

服务器：

```
15 0.597891  223.104.204.30  172.29.254.119  TCP     66 22664 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1370 WS=256 SACK_PERM
16 0.597924  172.29.254.119  223.104.204.30  TCP     66 80 → 22664 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
17 0.600017  223.104.204.30  172.29.254.119  TCP     66 36550 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1370 WS=256 SACK_PERM
18 0.600035  172.29.254.119  223.104.204.30  TCP     66 80 → 36550 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
19 0.749295  223.104.204.30  172.29.254.119  TCP     60 22664 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
```

MSS: Maximum Segment Size 即最大报文段长度，表示 TCP 传往另一端的一个报文段的最大长度。当一个连接建立时，连接的双方都要通告各自的 MSS，通过取最小值来确定双方通信时的 MSS。当通信双方结果路由器中转数据包的时候，MSS 的值还可能收到路由器的影响。

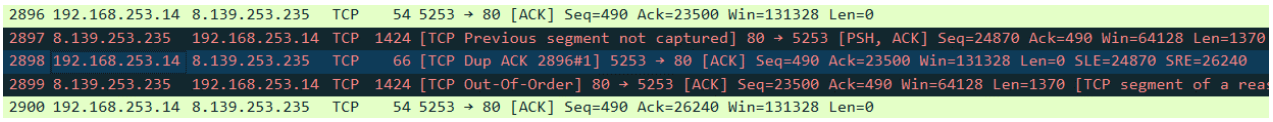
拿本次通信举例，在 1450 号报文中显示本机向服务器传送的报文中说 MSS 的大小为 1460 字节，但是服务器收到的 15 号报文显示，本机声称的 MSS 为 1370 字节。再看服务器向本机发送的 16 号报文中说自己的 MSS 大小为 1460 字节，但是在本机收到的 1642 号报

文中显示，服务器声称的 MSS 大小为 1370 字节，合理推测这是因为在通信路径上的某一个路由器的 MSS 大小为 1370 字节，在中转信息的时候发现报文中的 MSS 大于 1370 字节，于是将报文中的 MSS 更改为 1370 字节。

SACK：Selective Acknowledgment，是一个 TCP 的选项，来允许 TCP 单独确认非连续的片段，用于告知真正丢失的包，只重传丢失的片段。要使用 SACK，2 个设备必须同时支持 SACK 才可以，建立连接的时候需要使用 SACK Permitted 的 option，如果允许，后续的传输过程中 TCP segment 中的可以携带 SACK option，这个 option 内容包含一系列的非连续的没有确认的数据的 seq range。

```
▾ TCP Option - SACK permitted
    Kind: SACK Permitted (4)
    Length: 2
```

可以看到本机和服务器建立连接的时候用到了 SACK Permitted 的 option，通信双方同时支持 SACK，后续的传输过程中 TCP segment 中的可以携带 SACK option，这个 option 内容包含一系列的非连续的没有确认的数据的 seq range。



拿本机抓到的 2896-2900 号报文举例,在 2896 号报文中本机向服务器发消息，说下一条服务器发过来的消息从 23500 号开始，然后在 2897 号报文中本机接收到服务器的数据包序号是从 24870 开始的，这意味着本机在 24870 号之前的 23500 号报文丢失了，然后在 2898 号报文中本机向服务器发消息，说本机需要的下一条报文开始序号为 23500（也就是 2896 号报文中 ACK 的值）另外 SLE 为 24870，SRE 为 26420，这表示从 24870 号到 26420 号的长度为 1370 字节的数据包本机已经接收到了，这一段数据不需要重复发送。在 2899 号报文中本机接收到了开始序号为 23500 长度为 1370 字节的数据包，补全了缺失的数据，所以接收到 2899 号报文的时候，本机立即给服务器发送了一条确认信息，其中 ACK 为 26420，表示 26420 号之前的数据包全部收到。

```
▾ TCP Option - SACK 24870-26240
    Kind: SACK (5)
    Length: 10
    left edge = 24870 (relative)
    right edge = 26240 (relative)
    [TCP SACK Count: 1]
```

Window scale: 在 TCP 的报头中，有一个 16bit 的字段是表示 window 的大小的，而 2^16 = 65536，即由于字段的长度，窗口最大为 64KB，显然已经成为性能瓶颈。如果要修改该字段的长度，则对 TCP 的兼容性会造成比较大的影响，因此采用了 Window scale 的方法。即在 TCP 的报头的 option 中增加一个 Window scale，计算方式如下:

实际的窗口大小 = （window size value） ✖（2^(Window scale)）

```
1450 5.916640  192.168.253.14  8.139.253.235   TCP   66 5254 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
1642 6.077349  8.139.253.235   192.168.253.14  TCP   66 80 → 5254 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1370 SACK_PERM WS=128
1643 6.077466  192.168.253.14  8.139.253.235   TCP   54 5254 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
```

可以看到，在通信双方建立 TCP 连接的时候双方就交换了各自的窗口大小，1450 号报文表示本机的窗口大小为 64240✖256=16445440 字节，1642 号报文表示服务器的窗口大小为 64240✖128=8222720 字节。最终在 1643 号报文中确认了双方通信的窗口大小为 513✖256=131328 字节。

```
Window: 513
[Calculated window size: 131328]
[Window size scaling factor: 256]
```

Timestamp，即时间戳，用来表示发送数据包时的时刻，用于计算 RTT 比如现在 a 向 b 发送一个报文 s1，b 向 a 回复 ACK 报文 s2，那么：a 向 b 发送报文时，timestamp 中存放的时间戳就是 a 主机此时的内核时间 ta1b 向 a 回复报文时，timestamp 中存放的就是 b 主机此时的时间 tb1，timestamp echo 的值为从 s1 报文解析出来的时间 ta1，a 收到 b 回复的 s2 报文之后，此时 a 主机的内核时间为 ta2，而在 s2 报文的 timestamp echo 存放的是 ta1 最终：RTT = ta2 - ta1。时间戳还能用于防止序列号回绕

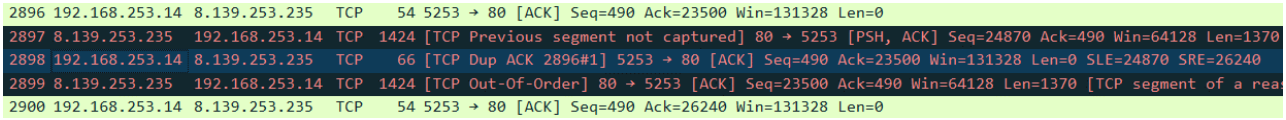
步骤 5：分析 TCP 数据传送阶段的报文，分析其错误恢复和流量控制机制，并填表。【注：出现明显的流量控制的地方，Wireshark 会有应答窗口的变化，乃至[TCP Window Full]或[TCP Zero Window]标记的报文出现。如果没有观察到明显的流量控制过程，可以再单独设计实验测试。比如编程设计接收端缓慢接收数据。】

① 数据传送阶段，正常下载服务器文件时抓到的数据包：

```
2889 8.139.253.235   192.168.253.14  TCP  1424 80 → 5253 [PSH, ACK] Seq=16650 Ack=490 Win=64128 Len=1370
2890 192.168.253.14  8.139.253.235   TCP    54 5253 → 80 [ACK] Seq=490 Ack=18020 Win=131328 Len=0
2891 8.139.253.235   192.168.253.14  TCP  1424 80 → 5253 [ACK] Seq=18020 Ack=490 Win=64128 Len=1370 [TCP
2892 8.139.253.235   192.168.253.14  TCP  1424 80 → 5253 [PSH, ACK] Seq=19390 Ack=490 Win=64128 Len=1370
2893 192.168.253.14  8.139.253.235   TCP    54 5253 → 80 [ACK] Seq=490 Ack=20760 Win=131328 Len=0
```

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
2889	数据	16650	490	1370	489	64128
2890	确认	490	18020	0	16830	131328
2891	数据	18020	490	1370	489	64128
2892	数据	19390	490	1370	489	64128
2893	确认	490	20760	0	19390	131328

② 错误恢复时抓到的数据包:

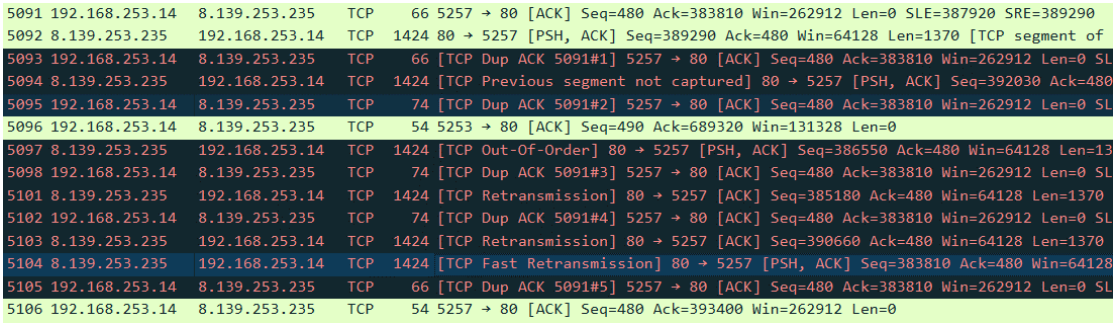


在 2896 号报文中本机告诉服务器 23500 号之前的报文已经收到，然后再 2897 号报文中服务器发送给本机的报文起始序号为 24870，本机没有收到 23500 到 24870 号之前的数据，所以在 2898 号报文中本机告诉服务器需要发送 23500 号报文，同时 24870 到 26240 号报文的数据已经收到，之后在 2899 号报文中服务器发送了 23500 号报文，通信回复正常，在 2900 号报文中本机发送的信息中 ACK 字段的值为 26240.

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
2896	TCP	490	23500	0	22130	131328
2897	TCP	24870	490	1370	489	64128
2898	TCP	490	23500	0	22130	131328
2899	TCP	23500	490	1370	489	64128
2900	TCP	490	26240	0	22130	64128

③快速重传



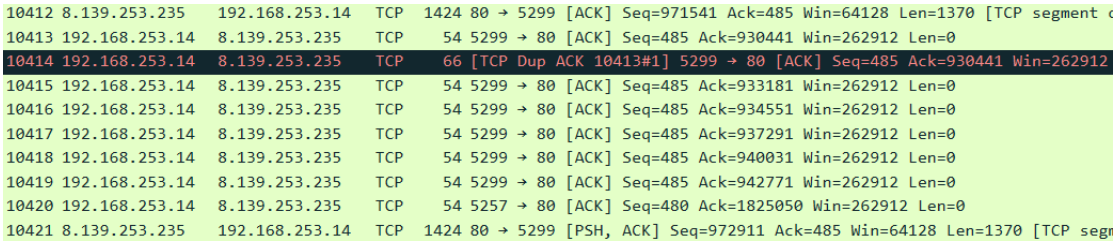
可以看到本机发送 5091 报文表示 383810 号报文之前的数据已经全部收到，下一次需要接收以 383810 号开头的报文，在 5092，5094 号报文中服务器都没有发送以 383810 号报文开头的报文，所以本机连续向服务器发送 5093，5095，5098，5102 号报文告诉服务器他需要发送以 383810 号开头的报文，随后服务器在接收到多个 ACK 为 383810 的确认报文后，向本机发送了以 383810 号开头的 5104 报文。实现了快速重传。

表 3-4 快速重传的报文

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
5091	确认	480	383810	0	382,440	262912
5092	数据	389290	480	1370	479	64128

5093	确认	480	383810	0	382440	262912
5094	数据	392030	480	1370	479	64128
5095	确认	480	383810	0	382440	262912
5097	数据	386550	480	1370	479	64128
5098	确认	480	383810	0	382440	262912
5101	数据	385180	480	1370	479	64128
5102	确认	480	383810	0	382440	262912
5103	数据	390660	480	1370	479	64128
5104	数据	383810	480	1370	479	64128

④流量控制时抓到的数据包：



流量控制的过程之一如上图所示，当服务器发送完 971541 号报文（No. 10412）后，已发出且未确认的报文总长已达 (971541+1370-930441) BB = 42,470 BB，即将超过接收方接收窗口的大小。因此，服务器并未继续发送数据报文，而是等待新的确认报文（No. 10419）到来后再开始发送。

表 3-5 流量控制的报文

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
10412	数据	971541	485	1370	484	64128
10413	确认	485	930441	0	929,071	262912
10414	确认	485	930441	0	929,071	262912
10415	确认	485	933181	0	931,811	262912
10416	确认	485	934551	0	933,181	262912
10417	确认	485	937291	0	935,921	262912
10418	确认	485	940031	0	938,661	262912
10419	确认	485	942771	0	941,401	262912
10421	数据	972911	485	1370	484	64128

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 RTT:

根据 RTT 的定义，RTT（往返时延）等于发送方收到第一个 ACK 报文的时刻与发送方首个分组的最后 1 比特被传输的时间之差。

服务器→本机方向的 RTT:

```
15 0.597891  223.104.204.30  172.29.254.119  TCP   66 22664 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1370 WS=256 SACK_PERM
16 0.597924  172.29.254.119  223.104.204.30  TCP   66 80 → 22664 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
```

RTT=0.597924-0.597891=0.000033s

主机→服务器方向的 RTT:

```
6.077349  1642 8.139.253.235  192.168.253.14  TCP   66 80 → 5254 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1370 SACK_PERM WS=128
6.077466  1643 192.168.253.14 8.139.253.235   TCP   54 5254 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
```

RTT=6.077466-6.077349=0.000117s

② 丢包率：

Wireshark · 协议分级统计 · 计网2.pcapng

协议	按分组百分比	分组	按字节百分比	字节	比特/秒	结束 分组	结束 字节	结束 位/秒	PDUs
Frame	100.0	47527	100.0	41483037	8142 k	0	0	0	47527
Ethernet	100.0	47527	1.6	665378	130 k	0	0	0	47527
Internet Protocol Version 4	100.0	47527	2.3	950540	186 k	0	0	0	47527
Transmission Control Protocol	100.0	47527	96.1	39867119	7825 k	47505	39858266	7823 k	47527
Hypertext Transfer Protocol	0.0	22	93.3	38717044	7599 k	12	5771	1132	22
Media Type	0.0	9	93.3	38708548	7598 k	9	38708548	7598 k	9
Line-based text data	0.0	1	0.0	704	138	1	704	138	1

可以看到，一共有 47527 个 TCP 包

Wireshark · 专家信息 · 计网2.pcapng

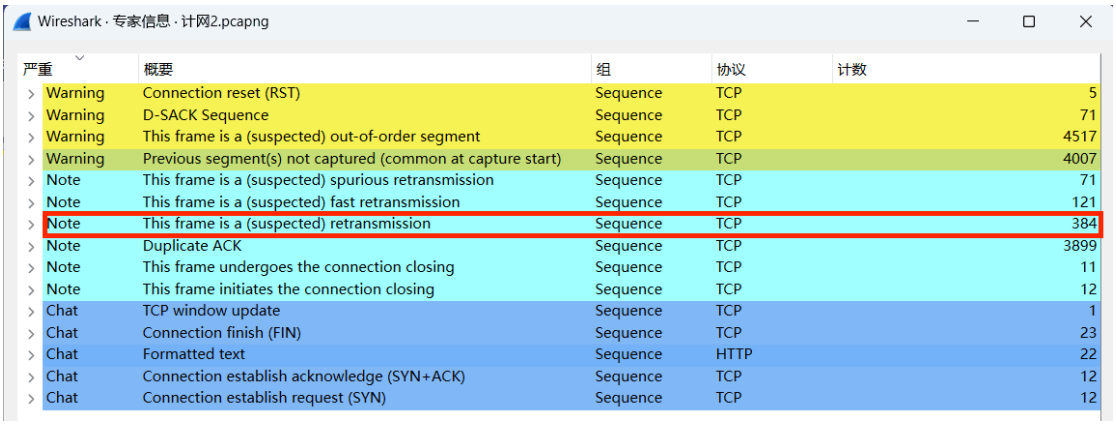
严重	概要	组	协议	计数
Warning	Connection reset (RST)	Sequence	TCP	5
Warning	D-SACK Sequence	Sequence	TCP	71
Warning	This frame is a (suspected) out-of-order segment	Sequence	TCP	4517
Warning	Previous segment(s) not captured (common at capture start)	Sequence	TCP	4007
Note	This frame is a (suspected) spurious retransmission	Sequence	TCP	71
Note	This frame is a (suspected) fast retransmission	Sequence	TCP	121
Note	This frame is a (suspected) retransmission	Sequence	TCP	384
Note	Duplicate ACK	Sequence	TCP	3899
Note	This frame undergoes the connection closing	Sequence	TCP	11
Note	This frame initiates the connection closing	Sequence	TCP	12
Chat	TCP window update	Sequence	TCP	1
Chat	Connection finish (FIN)	Sequence	TCP	23
Chat	Formatted text	Sequence	HTTP	22
Chat	Connection establish acknowledge (SYN+ACK)	Sequence	TCP	12
Chat	Connection establish request (SYN)	Sequence	TCP	12

丢失了 4007 个数据包

所以丢包率=4007/47527*100%=8.43%

③ 重传率:

重传的包共有 384 个。

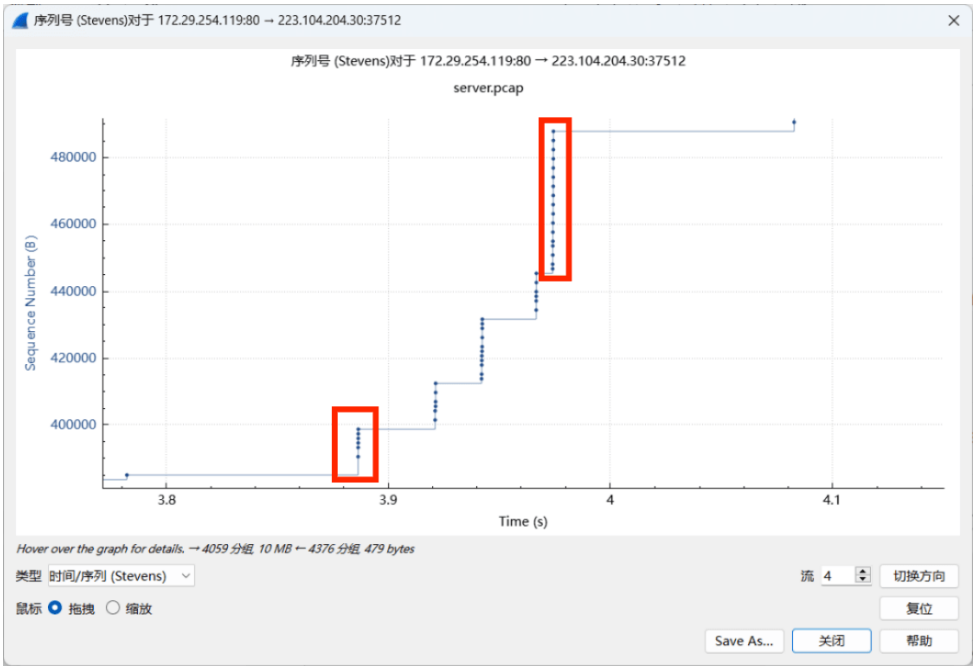


严重	概要	组	协议	计数
Warning	Connection reset (RST)	Sequence	TCP	5
Warning	D-SACK Sequence	Sequence	TCP	71
Warning	This frame is a (suspected) out-of-order segment	Sequence	TCP	4517
Warning	Previous segment(s) not captured (common at capture start)	Sequence	TCP	4007
Note	This frame is a (suspected) spurious retransmission	Sequence	TCP	71
Note	This frame is a (suspected) fast retransmission	Sequence	TCP	121
Note	This frame is a (suspected) retransmission	Sequence	TCP	384
Note	Duplicate ACK	Sequence	TCP	3899
Note	This frame undergoes the connection closing	Sequence	TCP	11
Note	This frame initiates the connection closing	Sequence	TCP	12
Chat	TCP window update	Sequence	TCP	1
Chat	Connection finish (FIN)	Sequence	TCP	23
Chat	Formatted text	Sequence	HTTP	22
Chat	Connection establish acknowledge (SYN+ACK)	Sequence	TCP	12
Chat	Connection establish request (SYN)	Sequence	TCP	12

所以重传率为 384/47527*100%=0.8%

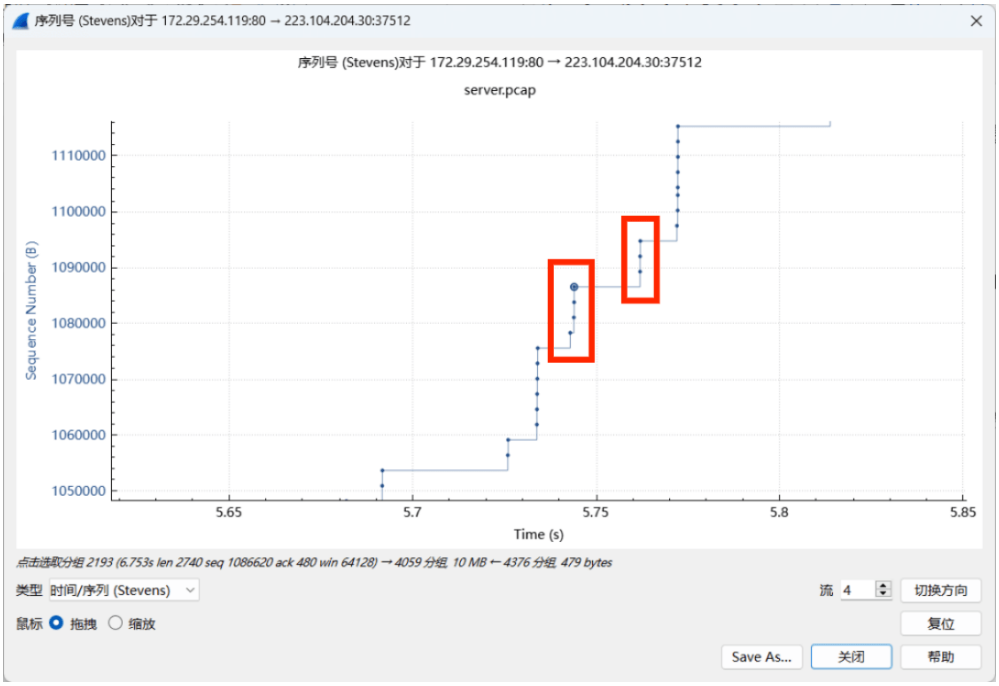
步骤 7、分析整个 TCP 流的拥塞控制，找到拥塞控制的几个典型过程（即慢启动、快速重传等）。

① 拥塞避免:



上图是报文序列号和时间的关系图，斜率即反映了报文的传输速率，而在正 常情况下，报文的传输速率是和拥塞窗口的大小成正比的，因此这条曲线的斜率 可以反映拥塞窗口的大小。 由上图可以看出，在 3.9s 左右时，曲线斜率大约是 （398880-385180）/（4.895-4.791）/1024=128.64KBps，随着时间增加，这条曲线的斜率也在线性增大，在 4s 左右达到(487930-445460)/(4.983-4.975)/1024=5.06MBps，反映了 TCP 报文拥塞控制中的拥塞避免阶段。

② 快恢复:



可以看出，快恢复前的斜率是(1086620-1075660)/(6.753-6.743)/1024=1070.31KBps，当开始丢包时，进入到快恢复，斜率变成(1094840-1086620)/(6.771-6.753)/1024=445.96KBps，基本上是之前的 1/2。之后同样进入拥塞避免阶段，斜率线性增大

③ 快重传:

```
5091 192.168.253.14   8.139.253.235    TCP    66 5257 → 80 [ACK] Seq=480 Ack=383810 Win=262912 Len=0 SLE=387920 SRE=389290
5092 8.139.253.235    192.168.253.14   TCP  1424 80 → 5257 [PSH, ACK] Seq=389290 Ack=480 Win=64128 Len=1370 [TCP segment of
5093 192.168.253.14   8.139.253.235    TCP    66 [TCP Dup ACK 5091#1] 5257 → 80 [ACK] Seq=480 Ack=383810 Win=262912 Len=0 SL
5094 8.139.253.235    192.168.253.14   TCP  1424 [TCP Previous segment not captured] 80 → 5257 [PSH, ACK] Seq=392030 Ack=480
5095 192.168.253.14   8.139.253.235    TCP    74 [TCP Dup ACK 5091#2] 5257 → 80 [ACK] Seq=480 Ack=383810 Win=262912 Len=0 SL
5096 192.168.253.14   8.139.253.235    TCP    54 5253 → 80 [ACK] Seq=490 Ack=689320 Win=131328 Len=0
5097 8.139.253.235    192.168.253.14   TCP  1424 [TCP Out-Of-Order] 80 → 5257 [PSH, ACK] Seq=386550 Ack=480 Win=64128 Len=13
5098 192.168.253.14   8.139.253.235    TCP    74 [TCP Dup ACK 5091#3] 5257 → 80 [ACK] Seq=480 Ack=383810 Win=262912 Len=0 SL
5101 8.139.253.235    192.168.253.14   TCP  1424 [TCP Retransmission] 80 → 5257 [ACK] Seq=385180 Ack=480 Win=64128 Len=1370
5102 192.168.253.14   8.139.253.235    TCP    74 [TCP Dup ACK 5091#4] 5257 → 80 [ACK] Seq=480 Ack=383810 Win=262912 Len=0 SL
5103 8.139.253.235    192.168.253.14   TCP  1424 [TCP Retransmission] 80 → 5257 [ACK] Seq=390660 Ack=480 Win=64128 Len=1370
5104 8.139.253.235    192.168.253.14   TCP  1424 [TCP Fast Retransmission] 80 → 5257 [PSH, ACK] Seq=383810 Ack=480 Win=64128
5105 192.168.253.14   8.139.253.235    TCP    66 [TCP Dup ACK 5091#5] 5257 → 80 [ACK] Seq=480 Ack=383810 Win=262912 Len=0 SL
5106 192.168.253.14   8.139.253.235    TCP    54 5257 → 80 [ACK] Seq=480 Ack=393400 Win=262912 Len=0
```

可以看到本机发送 5091 报文表示 383810 号报文之前的数据已经全部收到，下一次需要接收以 383810 号开头的报文，在 5092，5094 号报文中服务器都没有发送以 383810 号报文开头的报文，所以本机连续向服务器发送 5093，5095，5098，5102 号报文告诉服务器他需要发送以 383810 号开头的报文，随后服务器在接收到多个 ACK 为 383810 的确认报文后，向本机发送了以 383810 号开头的 5104 报文。实现了快速重传。

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。

六、 互动讨论主题

1)TCP 的流量控制和拥塞控制有什么不同?

流量控制是指发送方根据接收方的处理能力来控制数据的发送速率,以避免数据包的丢失或过载,以接收方为主导。拥塞控制是指在网络中防止过多数据注入到网络中，以避免网络拥塞和性能下降，以发送方为主导。

在 TCP 中，流量控制通过滑动窗口来实现。接收方会通知发送方当前它能够接收的数据量，发送方会根据这个信息来控制发送速率，确保不会超出接收方的处理能力。拥塞控制通过拥塞窗口来实现。发送方会根据网络的拥塞情况来调整发送速率,如果检测到网络拥塞,则会减小发送速率以减轻网络负担。

流量控制的目的是使接收方能够来得及接受发送方发送的数据,主要是为了解决接收方处理能力不足而设计的，它主要涉及到发送方和接收方之间的通信；而拥塞控制是通过调整发送窗口来有效避免以及处理网络拥塞的情况，它涉及到所有网络节点之间的通信，包括发送方、接收方以及网络中的路由器和交换机等设备。

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。接收方通过使用滑动窗口来告知发送方自己当前的处理能力，即它可以接收的数据量。发送方根据接收方提供的窗口大小来控制发送的数据量，以确保不会超过接收方的处理能力，从而避免数据的丢失或过载，如果发送方发送数据太快，接受方来不及接受出现接收缓存不够用的情况，就会发生流量控制。因此，TCP 的流量控制是由接收方来控制发送方的数据发送速率，以适应接收方的处理能力。

3）讨论传输层与其上下相邻层的关系

传输层为其上层网络层的应用进程提供了一种直接且可靠的通信服务,并且向高层用户屏蔽了下面网络核心的细节,它使得应用进程好像是在两个传输层实体之间有一条端到端的逻辑通信信道,因此传输层协议又称为端到端协议。应用层可以选择使用不同的传输层协议,根据自身的需求来决定是否需要可靠传输、连接建立等特性。

而传输层的下层网络层为传输层提供服务，网络层解决了庞大的互联网之间的节点如何互相识别、传输的问题，实现了主机到主机之间通信的重要部分。传输层利用网络层提供

的服务来实现数据的传输。网络层负责将数据包从源主机发送到目标主机，传输层则利用这些数据包来建立通信连接和传输数据。

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与可控制性。TCP 协议具有流量控制机制，可以根据接收方的处理能力来调整数据发送的速率，以避免数据的积压和丢失，从而保证语音流的稳定传输。TCP 协议保证了数据包的顺序传输，这对于实时语音流来说非常重要。语音数据需要按照正确的顺序传输到接收端，以保证语音的连贯性和正确性。

缺点：TCP 协议链接建立的过程较慢，并且 TCP 的重传机制会大大影响 实时语音的质量。TCP 协议对网络抖动的处理不够灵活。在遇到网络抖动时，TCP 会采取重新发送数据的策略，这会增加通信的延迟并影响语音通话的质量。当实时语音由于种种问题丢失了十几分之一秒的数据，TCP 协议会一直等待这段数据重传之后再交付上层应用，这就会造成实时语音的卡顿，实际上丢失的这一段数据对用户整体交流影响并不很大，用户更关心的是实时性。