# 实验三、传输层 TCP 协议分析实验报告


组号： 5-6

姓名： 施炎江 学号： 2186113847 班级： 计算机 82

姓名： 高浩翔 学号： 2181411962 班级： 计算机 82


## 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，计算丢包率及重传的流量。

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。

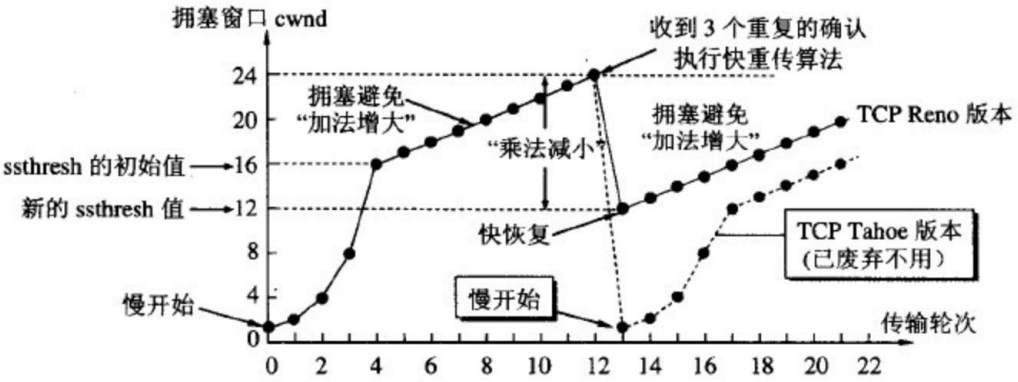



图 4-0 典型的 TCP 拥塞控制过程图例

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。观察有没有 Delay ACK 的应答模式，注意不同操作系统的差异。

6.*（可选）如果服务器为可控的 Linux，并且支持拥塞控制机制的更换，则测试两个不同拥塞控制机制下的传输，对比两种机制的特点。

## 三、 实验环境与分组

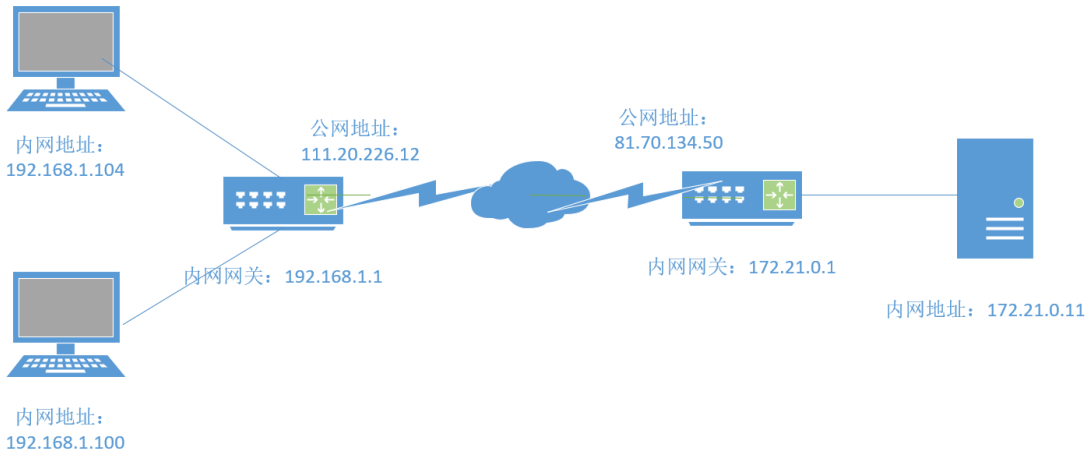
1）云服务器一台，启动 Apache2 服务（或其他服务器程序）。

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

步骤 1：PC2 通过 ssh 登录到服务器 Z 上，在云服务器 Z 上启动合适的服务器程序。

服务器系统为 CentOS，开启的服务器程序为 Apache（2.4.6 版本）。

```
[root@VM-0-11-centos html]# httpd -v
Server version: Apache/2.4.6 (CentOS)
Server built:   Nov 16 2020 16:18:20
```

下图是服务器网页的结构，其中 download 中有用来下载的数据，index.html 是访问服务器时的默认网页。

```
[root@VM-0-11-centos html]# ls
download  index.html
```

Html 文件源码很简单，就是提供了两个不同大小文件的下载地址。

```
<a href="./download/Zelda.rar">DOWNLOAD(3.2MB)</a>
<br>
<a href="./download/FF7.rar">DOWNLOAD(500KB)</a>
```

下面是网页展示，点击“DOWNLOAD”即可下载文件。


步骤 2：在 PC1 和 Z 上启动报文捕获软件，开始截获报文【注意加过滤器，比如 host w.x.y.z；不熟悉 tcpdump 的可以用 tcpdump -n -s 500 tcp and host A.B.C.D and port P -w server.pcap 选项，把报文记录到文件中，传输到客户端用 Wireshark 分析。其中 A.B.C.D 是客户端的**公网地址，P 是服务端口，如 80**】。

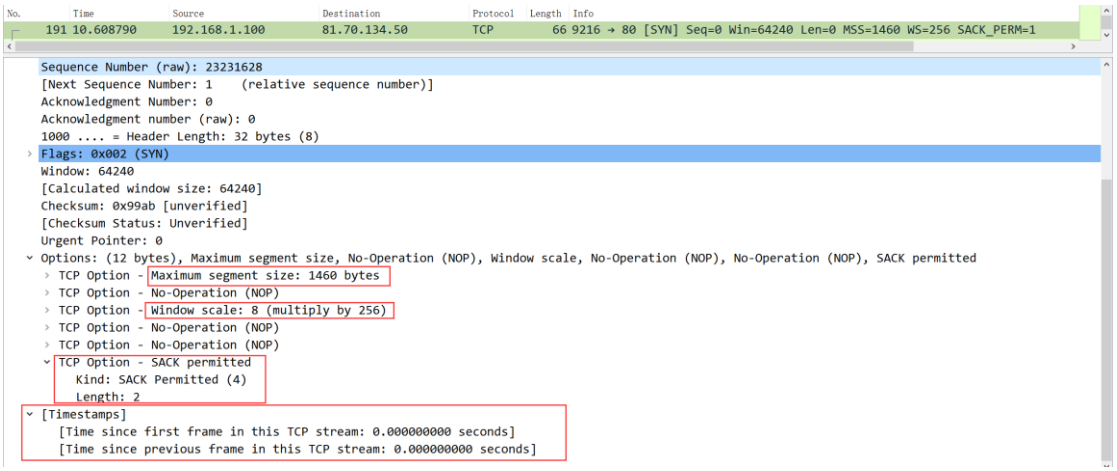
```
[root@VM-0-11-centos ~]# tcpdump -n -s 500 tcp and host 111.20.226.12 and port 80 -w server.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 500 bytes
```

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。

这一步在浏览器客户端点击下载文件即可。

步骤 4：对比观察客户端和服务器截获的报文，分析 TCP 协议的建立过程的三个报文并填写表 3-1。分析 TCP 连接的释放过程，选择 TCP 连接撤销的四个报文并填写表 3-2。

① TCP 报文典型选项分析：




MSS：最大段长度，用于告诉对方期望接收到的 TCP 报文段数据部分最大长度。

Window scale：用于计算接收窗口的大小。将接收窗口乘以 256（即 2 的 8 次方）后才是真正的接收窗口的大小。

```
TCP Option - Window scale: 8 (multiply by 256)
   Kind: Window Scale (3)
   Length: 3
   Shift count: 8
   [Multiplier: 256]
```

SACK：选择性确认技术，使 TCP 只重新发送交互过程中丢失的包，不用发送后续所有的包。

Timestamps：记录了这一 TCP 流中第一帧和上一帧发送时刻，用于计算 RTT。

② TCP 三次握手建立连接截图：

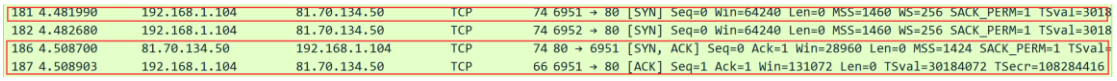



表 3-1 TCP 连接建立过程的三个报文信息【如果有多条，全部列出】

| 字段名称 | 第 1 条报文 | 第 2 条报文 | 第 3 条报文 |
|---|---|---|---|
| 报文序号 NO. | 181 | 186 | 187 |
| Seq # | 0 | 0 | 1 |

| | | | |
|---|---|---|---|
| Ack # | \ | 1 | 1 |
| ACK Flag | 0 | 1 | 1 |
| SYN Flag | 1 | 1 | 0 |

③ TCP 四次挥手断开连接截图：


表 3-2 TCP 连接撤销的四个报文信息

| 字段名称 | 首条报文 | 二条报文 | 三条报文 | 四条报文 |
|---|---|---|---|---|
| 报文捕获序号 NO. | 247 | 248 | 257 | 266 |
| Seq # | 426 | 382 | 382 | 427 |
| Ack # | 382 | 427 | 427 | 383 |
| ACK | 1 | 1 | 1 | 1 |
| FIN | 1 | 0 | 1 | 0 |

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[TCP Window Full]标记。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：

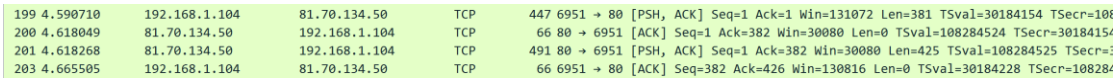
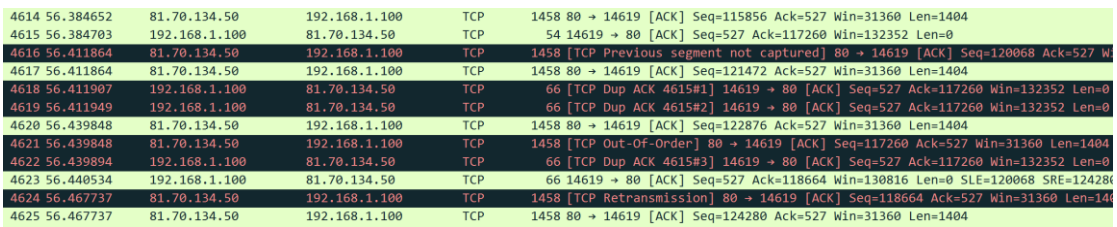



表 3-3 记录 TCP 数据传送阶段的报文

| 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 |
|---|---|---|---|---|---|---|
| 199 | 数据 | 1 | 1 | 381 | 187 | 131072 |
| 200 | 确认 | 1 | 382 | 0 | 199 | 30080 |
| 201 | 数据 | 1 | 382 | 425 | 199 | 30080 |
| 203 | 确认 | 382 | 426 | 0 | 201 | 130816 |

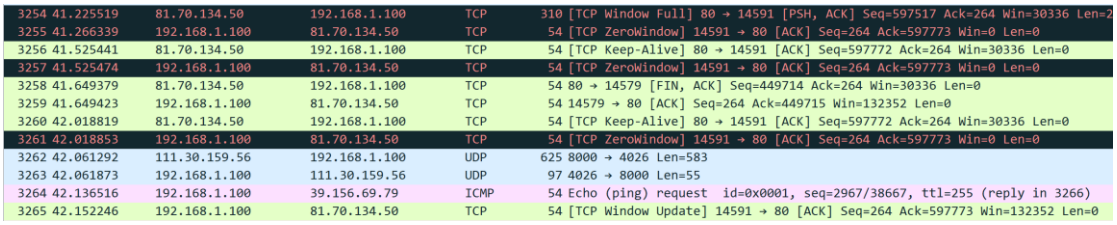
② 错误恢复时抓到的数据包：



4615 数据包为客户机的确认报文，确认序号为 117260，表示客户端接下来希望收到第 117260 字节起始的数据，在 4616 与 4617 数据包中，服务器向客户端发送的数据包起始序号为 120068 与 121472，这并不是按序到达的数据包，因此，客户端对 117260 字节之前数据进行了三次重复确认（4618、4619、4622），之后服务器对 117260 起始的数据进行了重传（4621、4624）。

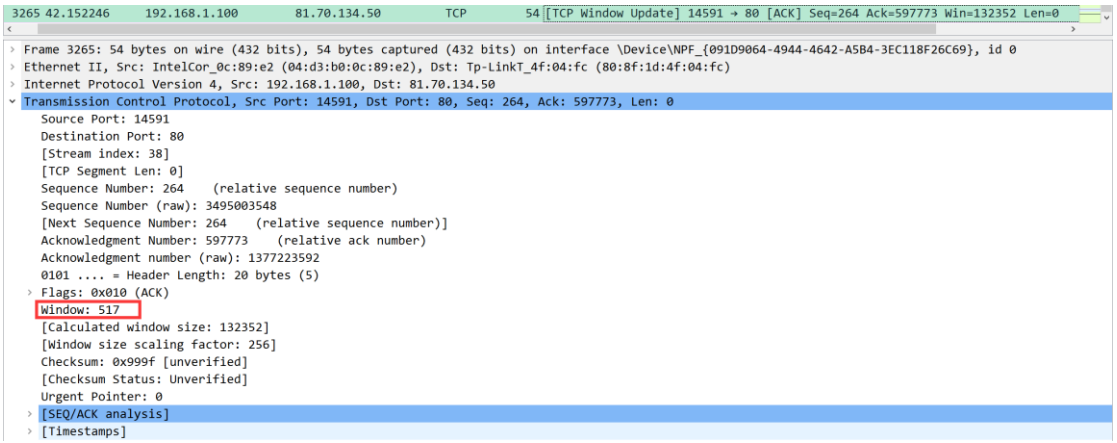
③ 流量控制时抓到的数据包：

由于正常下载中未能观察到明显的流量控制，因此采用迅雷限速下载的方式重新对数据包进行抓取，观察到了明显的流量控制现象如下图所示：



| | | | | | |
|---|---|---|---|---|---|
| 3254 | 41.225519 | 81.70.134.50 | 192.168.1.100 | TCP | 310 [TCP Window Full] 80 → 14591 [PSH, ACK] Seq=597517 Ack=264 Win=30336 Len=2 |
| 3255 | 41.266339 | 192.168.1.100 | 81.70.134.50 | TCP | 54 [TCP ZeroWindow] 14591 → 80 [ACK] Seq=264 Ack=597773 Win=0 Len=0 |
| 3256 | 41.525441 | 81.70.134.50 | 192.168.1.100 | TCP | 54 [TCP Keep-Alive] 80 → 14591 [ACK] Seq=597772 Ack=264 Win=30336 Len=0 |
| 3257 | 41.525474 | 192.168.1.100 | 81.70.134.50 | TCP | 54 [TCP ZeroWindow] 14591 → 80 [ACK] Seq=264 Ack=597773 Win=0 Len=0 |
| 3258 | 41.649379 | 81.70.134.50 | 192.168.1.100 | TCP | 54 80 → 14579 [FIN, ACK] Seq=449714 Ack=264 Win=30336 Len=0 |
| 3259 | 41.649423 | 192.168.1.100 | 81.70.134.50 | TCP | 54 14579 → 80 [ACK] Seq=264 Ack=449715 Win=132352 Len=0 |
| 3260 | 42.018819 | 81.70.134.50 | 192.168.1.100 | TCP | 54 [TCP Keep-Alive] 80 → 14591 [ACK] Seq=597772 Ack=264 Win=30336 Len=0 |
| 3261 | 42.018853 | 192.168.1.100 | 81.70.134.50 | TCP | 54 [TCP ZeroWindow] 14591 → 80 [ACK] Seq=264 Ack=597773 Win=0 Len=0 |
| 3262 | 42.061292 | 111.30.159.56 | 192.168.1.100 | UDP | 625 8000 → 4026 Len=583 |
| 3263 | 42.061873 | 192.168.1.100 | 111.30.159.56 | UDP | 97 4026 → 8000 Len=55 |
| 3264 | 42.136516 | 192.168.1.100 | 39.156.69.79 | ICMP | 54 Echo (ping) request id=0x0001, seq=2967/38667, ttl=255 (reply in 3266) |
| 3265 | 42.152246 | 192.168.1.100 | 81.70.134.50 | TCP | 54 [TCP Window Update] 14591 → 80 [ACK] Seq=264 Ack=597773 Win=132352 Len=0 |

通过数据包 3254 可以看出，本地主机缓存窗口已满，因此，本地主机向服务器发送数据包 3255 进行流量控制，将接受窗口调整为 0。此时服务器设定计时器，在计时器归零后发送一个零窗口探测数据包 3256，本地主机对此进行回复，表示接收窗口大小仍为 0（数据包 3257），服务器继续等待。当服务器在计时器到时之后又发送一个窗口探测数据包 3260，本地主机再次对此进行回复，声明接收窗口仍为零窗口（数据包 3261）。当本地缓存有空余之后，本地主机向服务器发送一个窗口更新报文，调整接受窗口大小（数据包 3265）。通过下图可看出，窗口大小重新调整为 132352（517×256）。



```
3265 42.152246    192.168.1.100    81.70.134.50    TCP    54 [TCP Window Update] 14591 → 80 [ACK] Seq=264 Ack=597773 Win=132352 Len=0
Frame 3265: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_{091D9064-4944-4642-A5B4-3EC118F26C69}, id 0
Ethernet II, Src: IntelCor_0c:89:e2 (04:d3:b0:0c:89:e2), Dst: Tp-LinkT_4f:04:fc (80:8f:1d:4f:04:fc)
Internet Protocol Version 4, Src: 192.168.1.100, Dst: 81.70.134.50
Transmission Control Protocol, Src Port: 14591, Dst Port: 80, Seq: 264, Ack: 597773, Len: 0
    Source Port: 14591
    Destination Port: 80
    [Stream index: 38]
    [TCP Segment Len: 0]
    Sequence Number: 264    (relative sequence number)
    Sequence Number (raw): 3495003548
    [Next Sequence Number: 264    (relative sequence number)]
    Acknowledgment Number: 597773    (relative ack number)
    Acknowledgment number (raw): 1377223592
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window: 517
    [Calculated window size: 132352]
    [Window size scaling factor: 256]
    Checksum: 0x999f [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [SEQ/ACK analysis]
  > [Timestamps]
```

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，丢包率和重传流量，平均传输速度等参数。

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→主机方向的 RTT：

```
191 10.608790    192.168.1.100    81.70.134.50     TCP    66 9216 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
194 10.635209    81.70.134.50     192.168.1.100    TCP    66 80 → 9216 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1424 SACK_PERM=1 W
```

RTT=10.635209-10.608790=0.026419s.

主机→服务器方向的 RTT：

```
2 0.000047    172.21.0.11      111.20.226.12    TCP    66 80 → 9216 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 W
3 0.026320    111.20.226.12    172.21.0.11      TCP    54 9216 → 80 [ACK] Seq=1 Ack=1 Win=132352 Len=0
```

RTT=0.026320-0.000047=0.026273s.

② 丢包率：

下图可以看出，TCP 数据包共有 4642 个。

| 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 |
|---|---|---|---|---|---|---|---|---|
| Frame | 100.0 | 4642 | 100.0 | 4138539 | 399k | 0 | 0 | 0 |
| Ethernet | 100.0 | 4642 | 1.6 | 64988 | 6277 | 0 | 0 | 0 |
| Internet Protocol Version 4 | 100.0 | 4642 | 2.2 | 92840 | 8968 | 0 | 0 | 0 |
| Transmission Control Protocol | 100.0 | 4642 | 96.2 | 3980711 | 384k | 2378 | 791420 | 76k |
| Data | 48.8 | 2264 | 76.0 | 3144011 | 303k | 2264 | 3144011 | 303k |

丢失的包有 228 个。

| 严重 | 概要 | 组 | 协议 | 计数 |
|---|---|---|---|---|
| Warning | No response seen to ICMP request | Sequence | ICMP | 8 |
| Warning | Unsupported header type: Elasticsearch version ... | Undecoded | Elasticsearch | 5 |
| Warning | TCP window specified by the receiver is now com... | Sequence | TCP | 11 |
| Warning | TCP Zero Window segment | Sequence | TCP | 41 |
| Warning | This frame is a (suspected) out-of-order segment | Sequence | TCP | 135 |
| Warning | D-SACK Sequence | Sequence | TCP | 808 |
| Warning | Previous segment(s) not captured (common at ca... | Sequence | TCP | 228 |
| Warning | Connection reset (RST) | Sequence | TCP | 42 |
| Note | ACK to a TCP keep-alive segment | Sequence | TCP | 16 |
| Note | TCP keep-alive segment | Sequence | TCP | 45 |
| Note | This frame is a (suspected) retransmission | Sequence | TCP | 451 |
| Note | Duplicate ACK (#1) | Sequence | TCP | 443 |
| Chat | TCP window update | Sequence | TCP | 24 |
| Chat | NOTIFY * HTTP/1.1\r\n | Sequence | SSDP | 68 |
| Chat | Connection finish (FIN) | Sequence | TCP | 179 |
| Chat | Connection establish acknowledge (SYN+ACK): s... | Sequence | TCP | 88 |
| Chat | Connection establish request (SYN): server port 4... | Sequence | TCP | 101 |

故丢包率=228/4642*100%=4.912%。

③ 重传率：

重传的包共有 451 个。

| 严重 | 概要 | 组 | 协议 | 计数 |
|---|---|---|---|---|
| Warning | No response seen to ICMP request | Sequence | ICMP | 8 |
| Warning | Unsupported header type: Elasticsearch version ... | Undecoded | Elasticsearch | 5 |
| Warning | TCP window specified by the receiver is now com... | Sequence | TCP | 11 |
| Warning | TCP Zero Window segment | Sequence | TCP | 41 |
| Warning | This frame is a (suspected) out-of-order segment | Sequence | TCP | 135 |
| Warning | D-SACK Sequence | Sequence | TCP | 808 |
| Warning | Previous segment(s) not captured (common at ca... | Sequence | TCP | 228 |
| Warning | Connection reset (RST) | Sequence | TCP | 42 |
| Note | ACK to a TCP keep-alive segment | Sequence | TCP | 16 |
| Note | TCP keep-alive segment | Sequence | TCP | 45 |
| Note | This frame is a (suspected) retransmission | Sequence | TCP | 451 |
| Note | Duplicate ACK (#1) | Sequence | TCP | 443 |
| Chat | TCP window update | Sequence | TCP | 24 |
| Chat | NOTIFY * HTTP/1.1\r\n | Sequence | SSDP | 68 |
| Chat | Connection finish (FIN) | Sequence | TCP | 179 |
| Chat | Connection establish acknowledge (SYN+ACK): s... | Sequence | TCP | 88 |
| Chat | Connection establish request (SYN): server port 4... | Sequence | TCP | 101 |

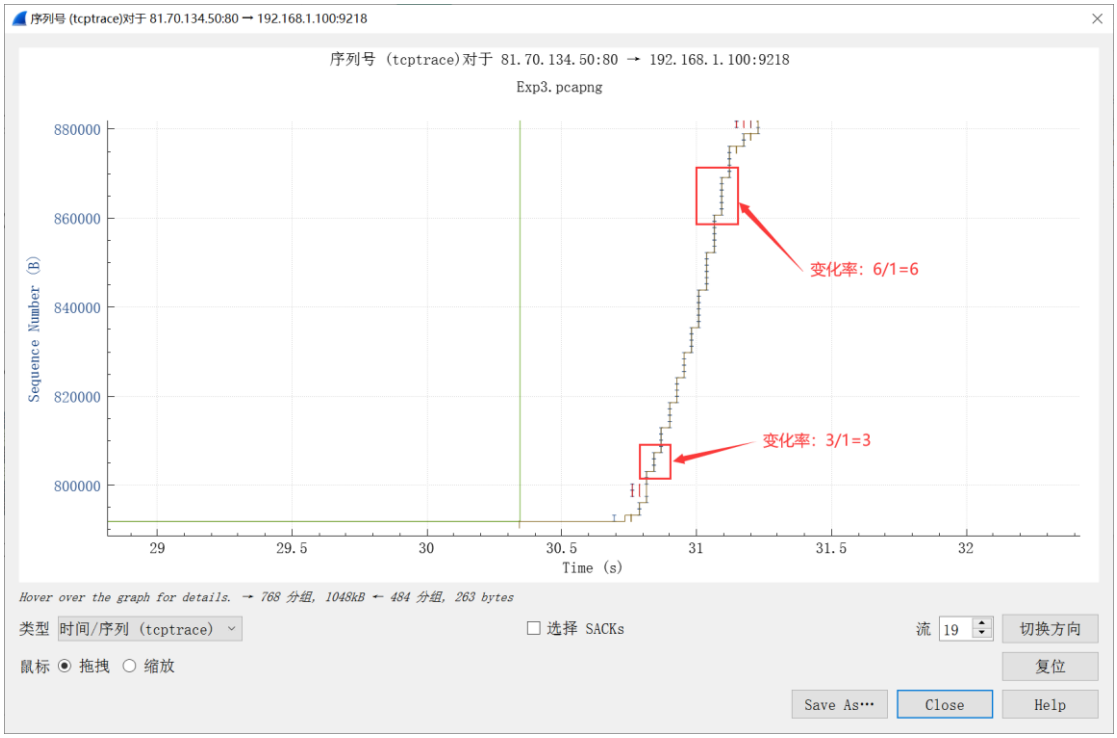
故重传率=451/4642*100%=9.7%。

④ 平均传速度：

报文的序列号代表着这是整个传输流过程中的字节序号，因此最后一个报文的序号就是整个过程中所传输的字节的数，用它除以整个传输时间就能计算出平均传输速度：1048902B/(53.38-10.74)s/1024=24.022KBps

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、拥塞避免，快速恢复），计算各个时期发送数据平均传输速度。

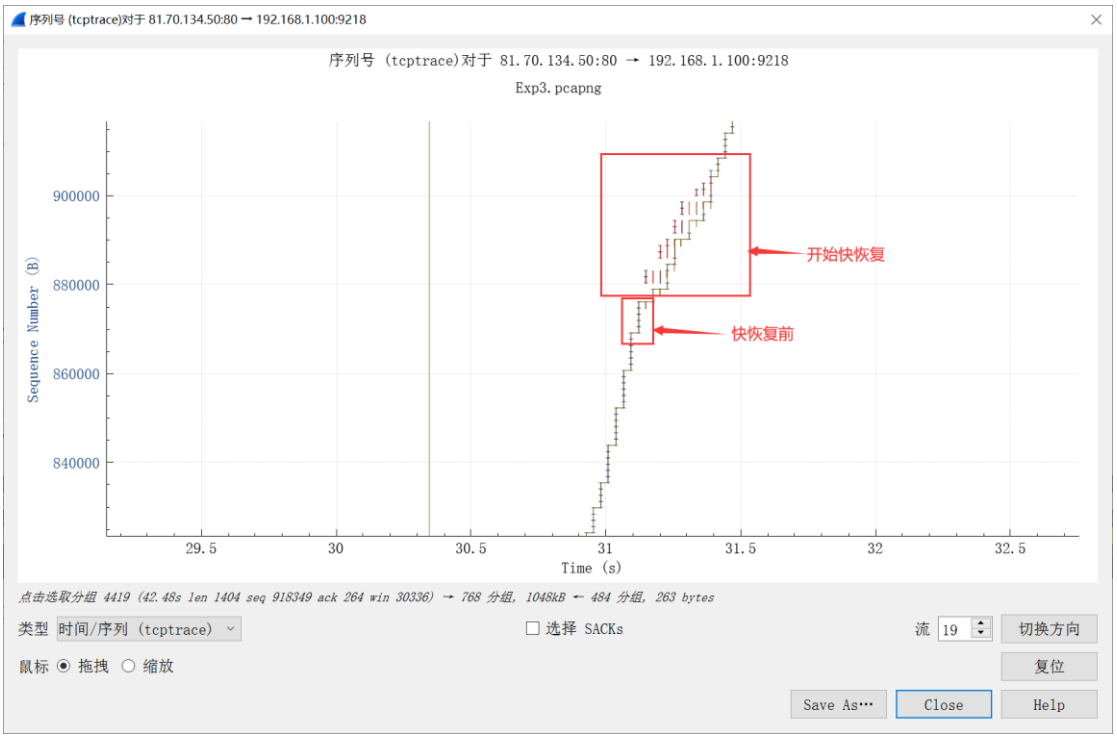
① 拥塞避免：




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。

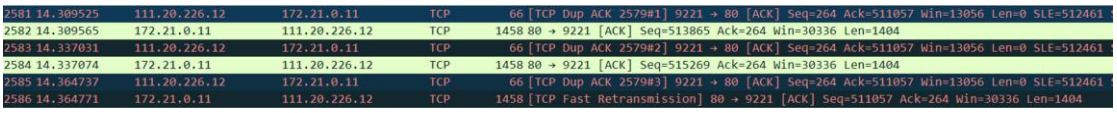
由上图可以看出，在 30.9s 左右时，曲线斜率大约是 3 左右，随着时间增加，这条曲线的斜率也在线性增大，在 31.1s 左右达到 6 左右，反映了 TCP 报文拥塞控制中的拥塞避免阶段。

② 快恢复




可以看出，快恢复前的斜率大约是 5/1=5，当开始丢包时，进入到快恢复，斜率变成大约 4/2=2 左右，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：

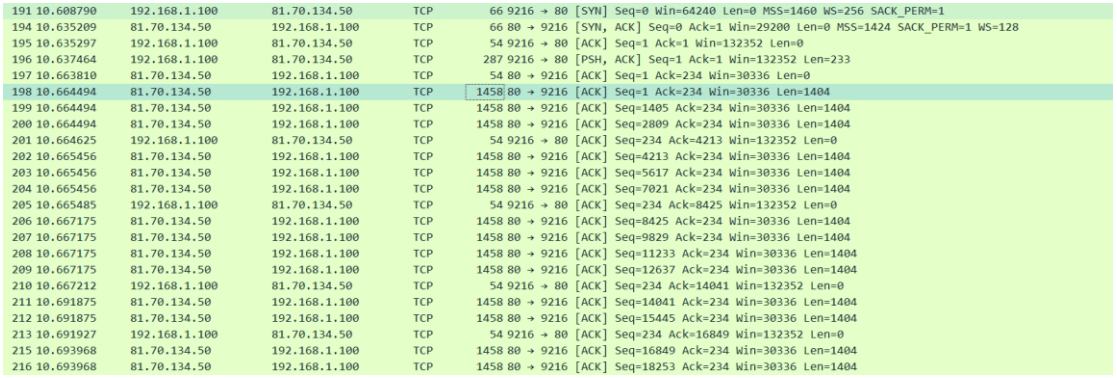



在 2581、2583、2585 三个数据包，主机向服务器发送三个对于 51107 之前数据的重复确认，服务器在收到这三次重复确认之后不等重传计时器到时就立即重传（2568）。

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。

（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。观察有没有 Delay ACK 的应答模式，注意不同操作系统的差异。



| | | | | | | |
|---|---|---|---|---|---|---|
| 191 | 10.608790 | 192.168.1.100 | 81.70.134.50 | TCP | 66 | 9216 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1 |
| 194 | 10.635209 | 81.70.134.50 | 192.168.1.100 | TCP | 66 | 80 → 9216 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1424 SACK_PERM=1 WS=128 |
| 195 | 10.635297 | 192.168.1.100 | 81.70.134.50 | TCP | 54 | 9216 → 80 [ACK] Seq=1 Ack=1 Win=132352 Len=0 |
| 196 | 10.637464 | 192.168.1.100 | 81.70.134.50 | TCP | 287 | 9216 → 80 [PSH, ACK] Seq=1 Ack=1 Win=132352 Len=233 |
| 197 | 10.663810 | 81.70.134.50 | 192.168.1.100 | TCP | 54 | 80 → 9216 [ACK] Seq=1 Ack=234 Win=30336 Len=0 |
| 198 | 10.664494 | 81.70.134.50 | 192.168.1.100 | TCP | 1458 | 80 → 9216 [ACK] Seq=1 Ack=234 Win=30336 Len=1404 |
| 199 | 10.664494 | 81.70.134.50 | 192.168.1.100 | TCP | 1458 | 80 → 9216 [ACK] Seq=1405 Ack=234 Win=30336 Len=1404 |
| 200 | 10.664494 | 81.70.134.50 | 192.168.1.100 | TCP | 1458 | 80 → 9216 [ACK] Seq=2809 Ack=234 Win=30336 Len=1404 |
| 201 | 10.664625 | 192.168.1.100 | 81.70.134.50 | TCP | 54 | 9216 → 80 [ACK] Seq=234 Ack=4213 Win=132352 Len=0 |
| 202 | 10.665456 | 81.70.134.50 | 192.168.1.100 | TCP | 1458 | 80 → 9216 [ACK] Seq=4213 Ack=234 Win=30336 Len=1404 |
| 203 | 10.665456 | 81.70.134.50 | 192.168.1.100 | TCP | 1458 | 80 → 9216 [ACK] Seq=5617 Ack=234 Win=30336 Len=1404 |
| 204 | 10.665456 | 81.70.134.50 | 192.168.1.100 | TCP | 1458 | 80 → 9216 [ACK] Seq=7021 Ack=234 Win=30336 Len=1404 |
| 205 | 10.665485 | 192.168.1.100 | 81.70.134.50 | TCP | 54 | 9216 → 80 [ACK] Seq=234 Ack=8425 Win=132352 Len=0 |
| 206 | 10.667175 | 81.70.134.50 | 192.168.1.100 | TCP | 1458 | 80 → 9216 [ACK] Seq=8425 Ack=234 Win=30336 Len=1404 |
| 207 | 10.667175 | 81.70.134.50 | 192.168.1.100 | TCP | 1458 | 80 → 9216 [ACK] Seq=9829 Ack=234 Win=30336 Len=1404 |
| 208 | 10.667175 | 81.70.134.50 | 192.168.1.100 | TCP | 1458 | 80 → 9216 [ACK] Seq=11233 Ack=234 Win=30336 Len=1404 |
| 209 | 10.667175 | 81.70.134.50 | 192.168.1.100 | TCP | 1458 | 80 → 9216 [ACK] Seq=12637 Ack=234 Win=30336 Len=1404 |
| 210 | 10.667212 | 192.168.1.100 | 81.70.134.50 | TCP | 54 | 9216 → 80 [ACK] Seq=234 Ack=14041 Win=132352 Len=0 |
| 211 | 10.691875 | 81.70.134.50 | 192.168.1.100 | TCP | 1458 | 80 → 9216 [ACK] Seq=14041 Ack=234 Win=30336 Len=1404 |
| 212 | 10.691875 | 81.70.134.50 | 192.168.1.100 | TCP | 1458 | 80 → 9216 [ACK] Seq=15445 Ack=234 Win=30336 Len=1404 |
| 213 | 10.691927 | 192.168.1.100 | 81.70.134.50 | TCP | 54 | 9216 → 80 [ACK] Seq=234 Ack=16849 Win=132352 Len=0 |
| 215 | 10.693968 | 81.70.134.50 | 192.168.1.100 | TCP | 1458 | 80 → 9216 [ACK] Seq=16849 Ack=234 Win=30336 Len=1404 |
| 216 | 10.693968 | 81.70.134.50 | 192.168.1.100 | TCP | 1458 | 80 → 9216 [ACK] Seq=18253 Ack=234 Win=30336 Len=1404 |

如图所示察第一次握手可以大致得到 RTT 时间约为（10.635209-10.608790）=0.026419s，文件传输中第一次数据报接受的时间为 10.664494s，这个时间加上一个 RTT 为 10.690913s，在这一段时间内一共收到了 10 个数据包，因此，可知基于 linux 系统的服务器的 initcwnd=10。

（可选）如果服务器为可控的 Linux，并且支持拥塞控制机制的更换，则测试两个不同拥塞控制机制下的传输，对比两种机制的特点。

可用拥塞控制：

```
[root@VM-0-11-centos ~]# cat /proc/sys/net/ipv4/tcp_allowed_congestion_control
cubic reno
```

当前算法：

```
[root@VM-0-11-centos ~]# cat /proc/sys/net/ipv4/tcp_congestion_control
cubic
```

通过对比发现，当使用 Reno 时，每收到一个 ACK 报文，CWND 加一，等出现丢包之后发送者会将发送速率减半。Cubic 则是在 BIC 机制的基础上，让拥塞窗口在接近上一个 CWND 最大值时保持的时间更长一些，并且让 CWND 的增长和 RTT 的长短无关，让 CWND 增长的三次函数仅与时间相关，无论 RTT 大小，一定时间后 CWND 一定会增长到某个值，从而让网络更公平，使得 RTT 小的连接不能挤占 RTT 大的连接的资源。

## 六、 互动讨论主题

1）TCP 的流量控制和拥塞控制有什么不同？

TCP 流量控制为接收方主导，而拥塞控制为发送方主导。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；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。

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？

流量控制为接收方主导。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。这个时候接收方就会向发送方发送报文告知自己的接受窗口大小，使其不要发送过快。

3）讨论传输层与其上下相邻层的关系

运输层为其上层网络层的应用进程提供了直接的通信服务，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运输层实体之间有一条端到端的逻辑通信信道，因此运输层协议又称为端到端协议。

而运输层的下层网络层为运输层提供服务，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。

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。

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。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。