**实验三、传输层TCP协议分析实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 组号： |  |  |  |  |  |
| 姓名： | 向胤兴 | 学号： | 2215012469 | 班级： | 计算机2102 |
| 姓名： |  | 学号： |  | 班级： |  |
|  |  |  |  |  |  |

1. 实验目的
2. 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。
3. 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。
4. 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。
5. 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。
6. 实验内容
7. 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**。
8. 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。
9. 观察和估计客户机到服务器的RTT，双方各自的MSS。
10. 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】。

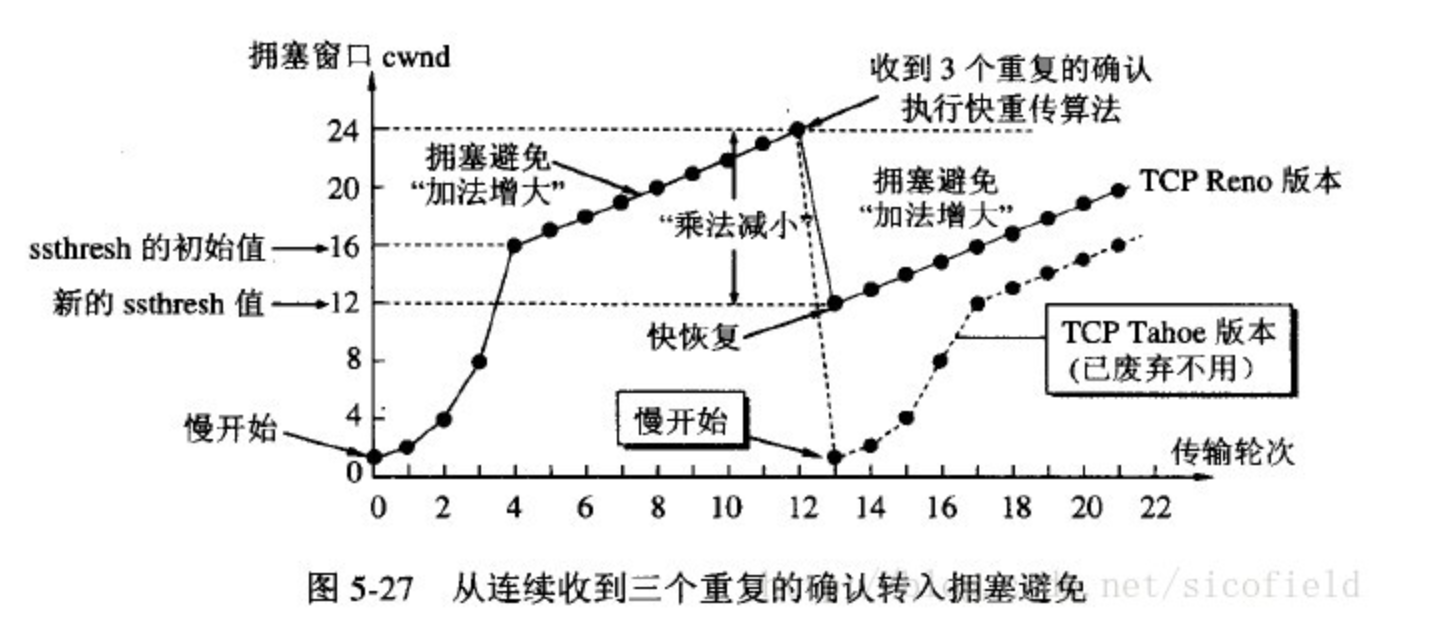


图4-0 典型的TCP 拥塞控制过程图例

1. \*（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】

1. 实验环境与分组

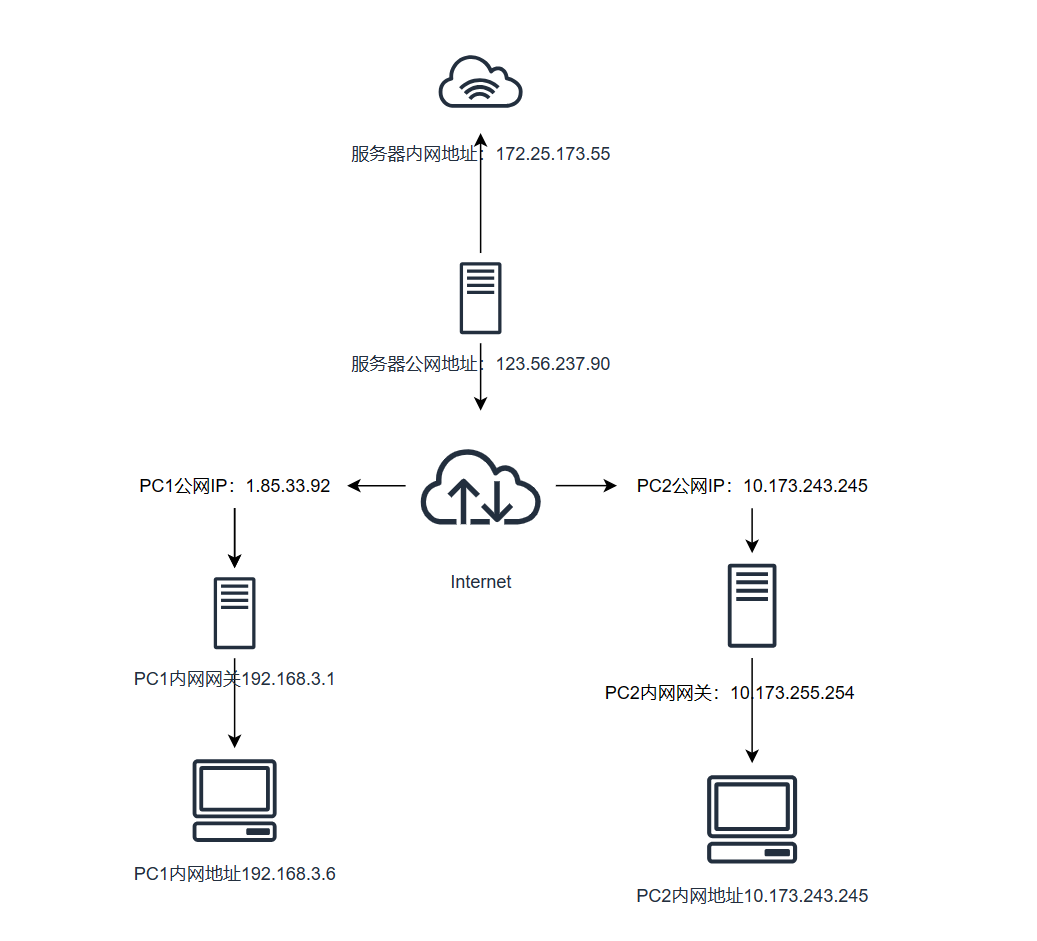
1）云服务器一台，启动Apache2服务（或其他服务器程序）。

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。】

1. 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。

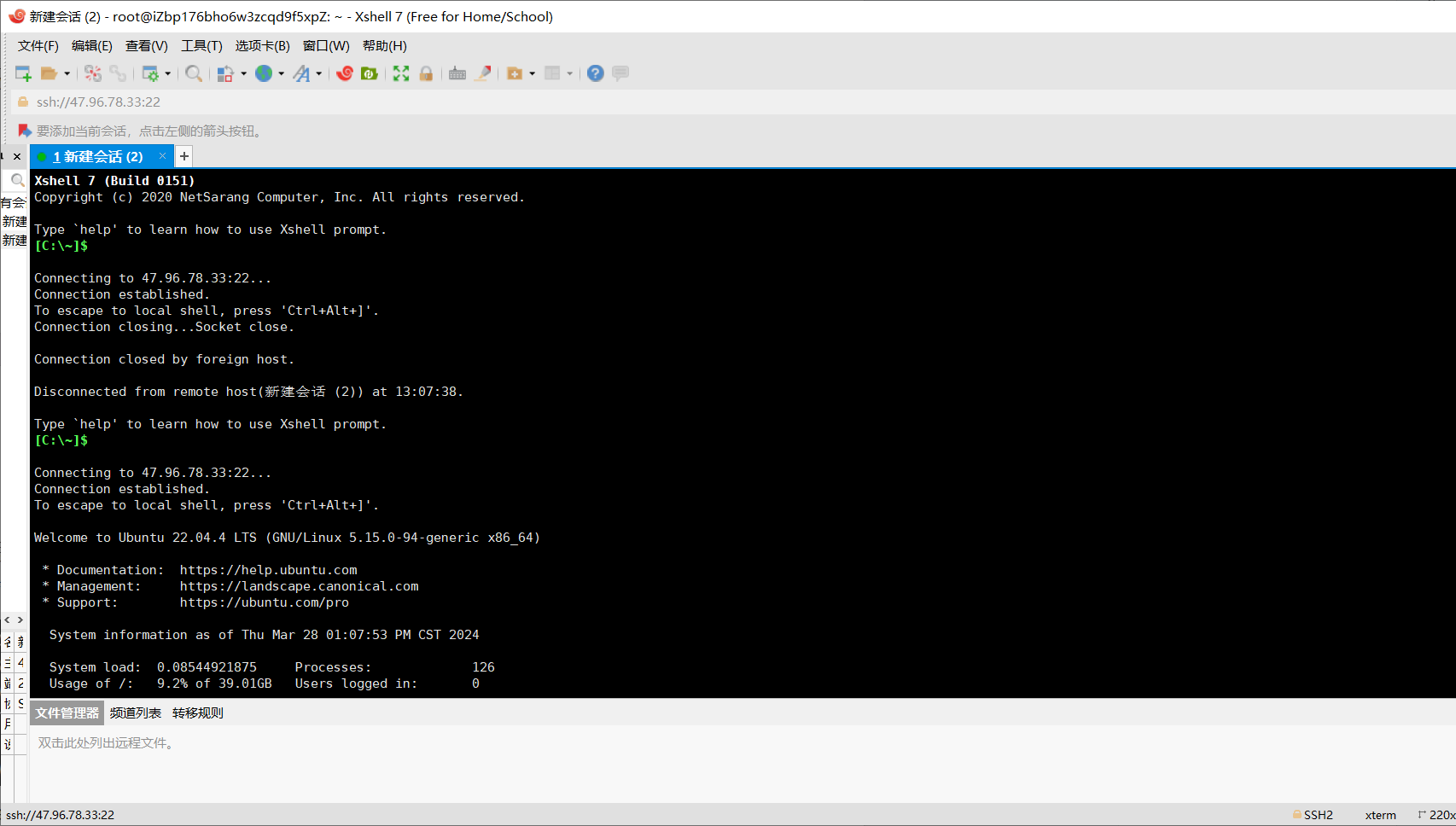


1. 实验过程及结果分析

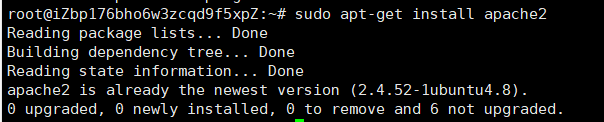
【过程记录应当详尽，截图并加以说明。以下过程和表格仅供参考。】

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）。

首先通过Xshell连接到云服务器：



本实验使用aliyun配置ubuntu系统，使用Apache2应用程序。

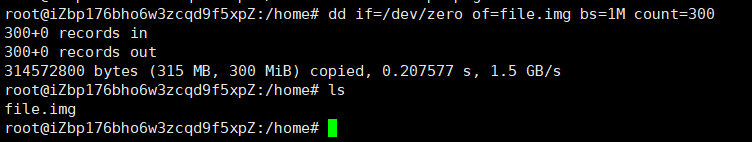


使用sudo apt-get install apache2在云服务器A上安装apache2。

可以看到安装成功（第一次安装忘记截屏了）。

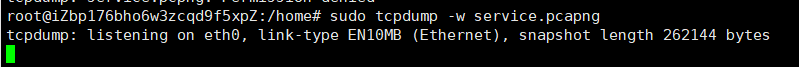
步骤2：在Z上首先创建一个文件，然后在PC1和Z上启动报文捕获软件，开始截获报文（注意抓包一定要包括建立连接的报文）。

首先需要创建一个文件，在shell中执行/home# dd if=/dev/zero of=file.img bs=1M count=500命令，创建一个大小为500M的，格式为.img的文件。创建完成后使用ls命令可以看到当前目录下已经存在file.img项。



服务器端开始抓包：

使用sudo tcpdump -w service.pcapng命令开始抓包并保存为service.pcapng



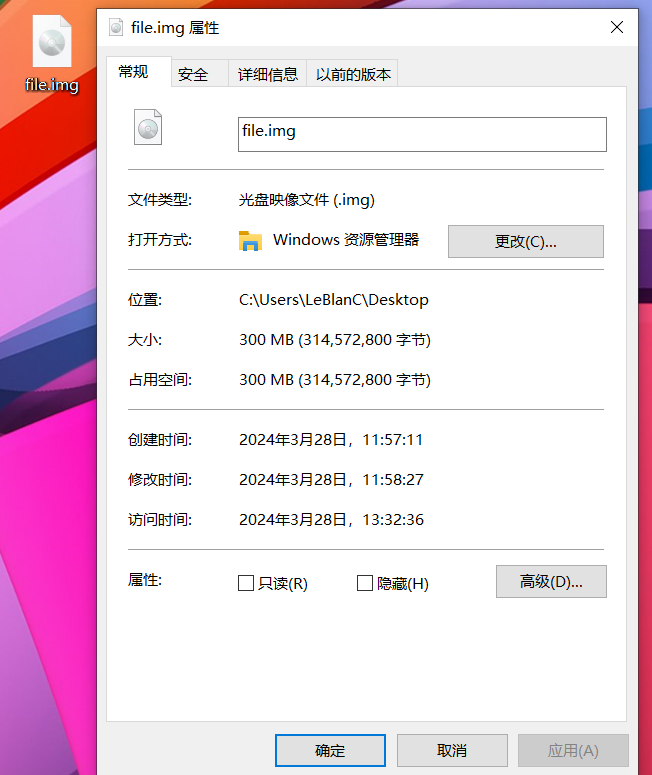
PC1端打开wireshark抓包即可。

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

在PC1使用Powershell下载服务器文件：



文件下载完成



下载完成后云服务器和PC1关闭抓包，在shell中执行ls查看目录：

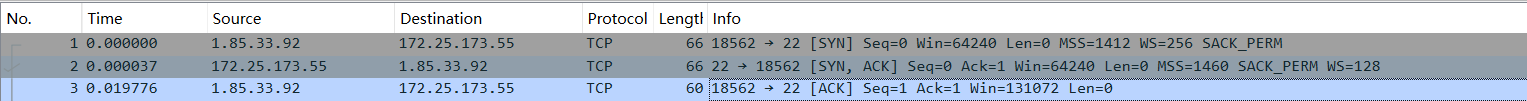


可以看到服务器端的抓包成功保存，同样从服务器端下载service.pcapng到PC1进行分析。

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。

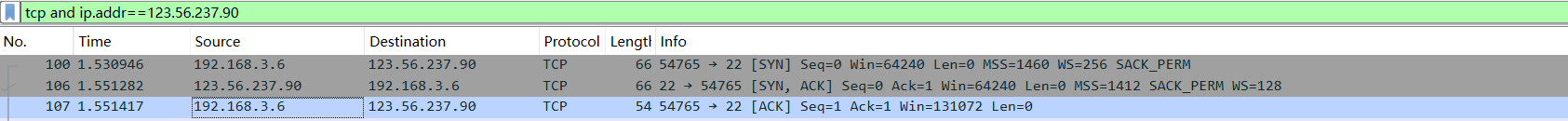
**三次握手建立TCP连接：**

服务器截获的TCP建立连接的报文：



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 报文序号 | 序号字段Seq Number | ACK flag | 确认号Ack Number | SYN flag | 源IP | 目的IP |
| 1 | 0 | 0 | 0 | 1 | 1.85.33.92 | 172.25.173.55 |
| 2 | 0 | 1 | 1 | 1 | 172.25.173.55 | 1.85.33.92 |
| 3 | 1 | 1 | 1 | 0 | 1.85.33.92 | 172.25.173.55 |

客户端截获的TCP建立连接的报文：



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 报文序号 | 序号字段Seq Number | ACK flag | 确认号Ack Number | SYN flag | 源IP | 目的IP |
| 100 | 0 | 0 | 0 | 1 | 192.168.3.6 | 123.56.237.90 |
| 106 | 0 | 1 | 1 | 1 | 123.56.237.90 | 192.168.3.6 |
| 107 | 1 | 1 | 1 | 0 | 192.168.3.6 | 123.56.237.90 |

可以看到不管是客户端还是服务器端截获的TCP建立连接报文都是经由三次握手：

第一次握手：由客户端发起第一次请求，SYN标志为1，序列号为0；

第二次握手：有服务器向客户端应答，SYN、ACK标志为1，序列号为0，ACK标志置1；

第三次握手：客户端应答服务器，ACK置为1，序列号为1，应答第二次握手中的ACK=1。

可以看到服务器端和客户端截获到的报文基本相同，区别只在于源IP地址和目的IP地址的区别。IP地址区别的原因是一台设备在访问另一网段的设备时，目标地址只能是目标设备的公网地址，但源地址是自身的私有地址。

**TCP连接的释放：**

本次抓包未能成功抓到四次握手FIN标志，最终是通过RST来强制关闭连接：

RST包是一个特殊的TCP包，用于异常关闭一个现有的连接。当一个TCP端点（客户端或服务器）检测到某些严重的问题，无法或不愿继续与另一个端点进行通信时，它可能会发送一个RST包来强制关闭连接。

服务器端：



客户端：



工作过程：  
 客户端向服务器端发送一个RST置位为1的包，来断开TCP连接。服务器接收到此包后，不再回应，断开连接。

RST包与正常的TCP连接关闭过程（即使用FIN和ACK包）不同。正常的关闭过程是一个有序的四次握手过程，而RST包则是一个突发的、强制性的关闭方法。

这些问题可能包括：

无效的连接：例如，当一个数据包到达一个未被监听的端口时。

连接超时：如果连接在一段时间内没有任何活动，某些系统或应用可能会决定关闭它。

应用层错误：某些应用层协议可能会在检测到错误时发送RST包。

系统资源不足：当系统资源（如内存）不足时，可能会发送RST包来关闭一些连接。

**关于报文中MSS、SACK、Window Scale、Timestamp字段的分析：**

以服务器端第一个报文为例：



MSS (Maximum Segment Size) = 1412：

MSS表示TCP连接中每个报文段所能承载的最大数据长度（不包括TCP头部和IP头部）。这里，MSS的值为1412字节。这意味着在TCP连接建立时，收发双方协商确定，每个报文段的数据部分最大不能超过1412字节。这有助于避免IP层分片，从而提高网络性能。

SACK\_PERM：

SACK\_PERM标志表示TCP连接支持Selective Acknowledgment（选择性确认）功能。这意味着接收方能够选择性地确认接收到的数据段，而不是仅仅确认最后一个连续收到的数据段。这种机制有助于发送方更精确地确定哪些数据段丢失了，并只重传那些丢失的数据段，从而提高数据传输的效率。

Window Scale = 256：

Window Scale用于放大TCP窗口大小。原始的TCP窗口大小是一个16位的值，最大为65535字节。但在高速网络中，这个大小可能不足以充分利用网络带宽。Window Scale通过一个缩放因子来扩展窗口大小，这里缩放因子为256。这意味着实际的窗口大小是原始窗口大小乘以256。这有助于在高带宽网络中提高TCP的吞吐量。

报文中没有Timestamp，Timestamp（时间戳）在TCP协议中是一个重要的组成部分，用于记录发送和接收数据的时间信息。具体来说，Timestamp的内容包括发送方的时间戳（TSval）和接收方的时间戳回显（TSecr）。

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。】

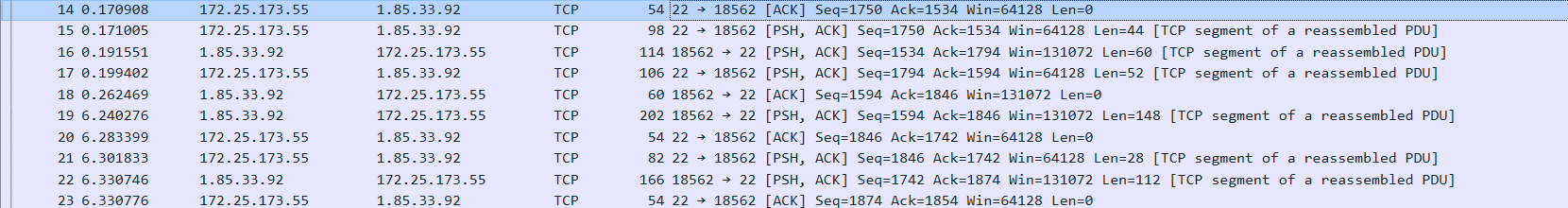


表3-1 正常通信记录TCP数据传送阶段的报文

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |
| 14 | 确认 | 1750 | 1534 | 0 | 10 | 64128 |
| 15 | 数据 | 1750 | 1534 | 44 | 10 | 64128 |
| 16 | 数据 | 1534 | 1794 | 60 | 15 | 131072 |
| 17 | 数据 | 1794 | 1594 | 52 | 16 | 64128 |
| 18 | 确认 | 1594 | 1846 | 0 | 17 | 131072 |
| 19 | 数据 | 1594 | 1846 | 148 | 17 | 131072 |
| 20 | 确认 | 1846 | 1742 | 0 | 19 | 64128 |
| 21 | 数据 | 1846 | 1742 | 28 | 19 | 64128 |
| 22 | 数据 | 1742 | 1874 | 112 | 21 | 131072 |
| 23 | 数据 | 1874 | 1854 | 0 | 22 | 64128 |

服务器端在向客户端发送数据时，会提前向客户端发送一个数据长度为0的确认报文，确认可以传下一个报文。

客户端在应答客户端时，原本不应该有数据报文。但由于远程服务器由ssh连接，客户端向服务器端在应答报文中捎带了数据，在计算机网络中把这一现象称为捎带应答。

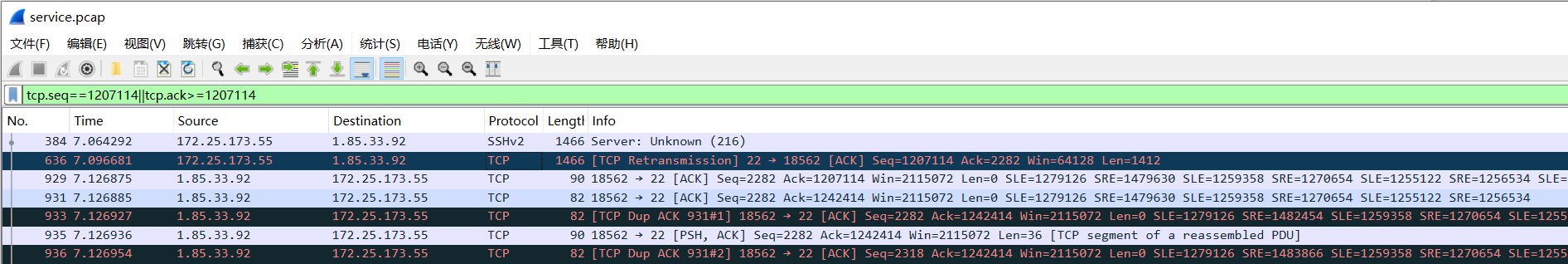
可以看到序号字段Seq与确认号字段Ack之间的对应关系。

错误恢复机制：

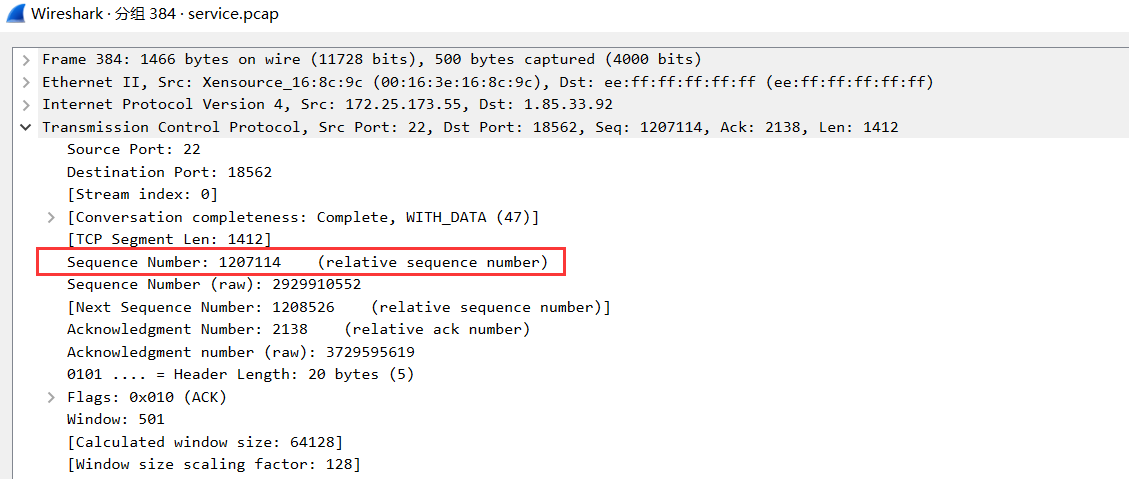
超时重传机制：

当TCP发送端发送一个数据报文段后，它会启动一个定时器，这个定时器被设置为一个初始的超时时间值。如果在定时器超时之前，发送端收到了接收端发送的对于该数据报文段的确认应答（ACK），继续正常工作；如果在定时器超时的时候，发送端仍然没有收到对应的确认应答，那么发送端就会认为该数据报文段在网络中丢失或被丢弃了。此时，发送端会重新发送该数据报文段，并重新启动定时器进行等待。这个过程可能会重复多次，直到收到确认应答或达到最大重传次数为止。

从服务器端截获的针对seq为1207114的报文：



第一套报文因为sshv2协议加密无法直接查看，打开查看详细内容：



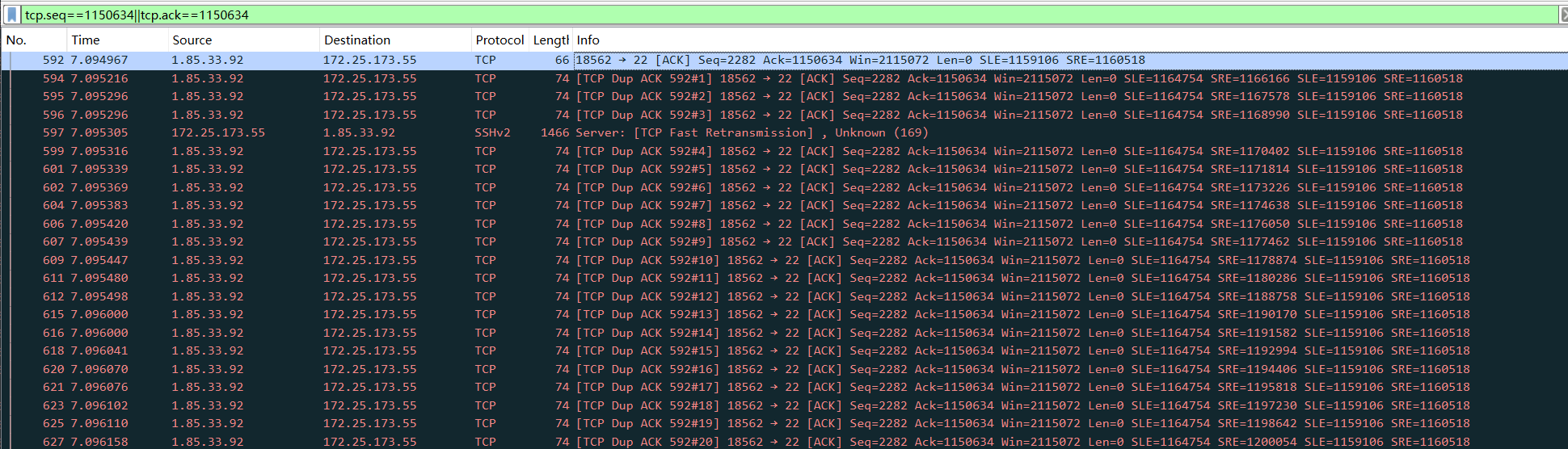
可以看到sequence号为1207114。

在服务器第一次发送完Sequence号为1207114的包后，等待一段时间后还没有接收到没有收到过ACK号大于1207114的包，于是启动超时重发TCP Retransiation机制重新发送。后续收到ACK号大于1207114的包后，说明sequence为1207114的包被正常接收。

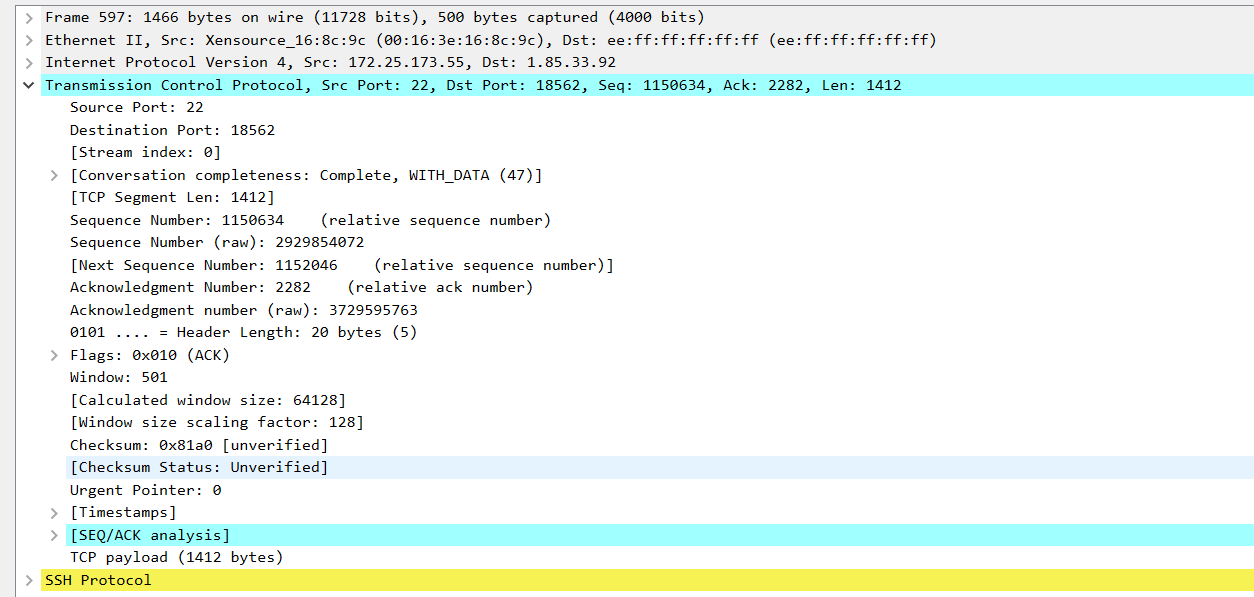
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 报文序号 | 报文种类 (数据/确认) | 序号字段Seq Number | 确认号Ack Number | 数据长度 | 确认到哪条报文（填序号） | 窗口大小 |
| 384 | 数据 | 1207114 | 2138 | 1412 | 172 | 64128 |
| 616 | 数据 | 1207114 | 2138 | 1412 | 172 | 64128 |
| 929 | 确认 | 2282 | 1207114 | 0 | 616 | 2115072 |

快速重传机制：快速重传的触发条件主要是基于接收端发送的重复确认应答（Duplicate Acknowledgment，简称Dup ACK）。当发送端连续收到三个或更多个重复的ACK时，它会预测某个数据包可能已丢失，并触发快速重传机制。这种情况下，发送端不需要等待超时定时器到期，而是立即重传丢失的数据包。

从客户端截获的针对seq为1150634的报文：



597号包的详细信息：



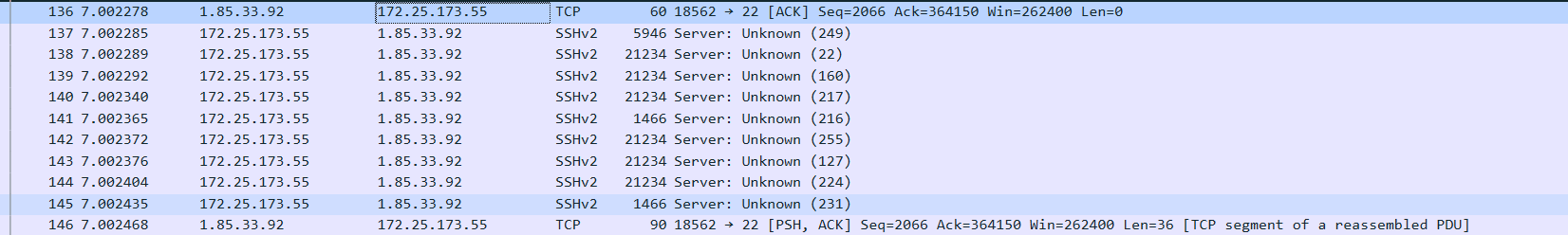
从服务器视角出发，在连续收到3个序列号为1150634的重复请求后，不等超时计时器到达触发值就开启超时重传。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 报文序号 | 报文种类 | 序号字段Seq Number | 确认号Ack Number | 数据长度 | 窗口大小 |
| 592 | 确认 | 2282 | 1150634 | 0 | 2115072 |
| 594 | 确认 | 2282 | 1150634 | 0 | 2115072 |
| 595 | 确认 | 2282 | 1150634 | 0 | 2115072 |
| 596 | 确认 | 2282 | 1150634 | 0 | 2115072 |
| 597 | 数据 | 1150634 | 2282 | 1412 | 64128 |

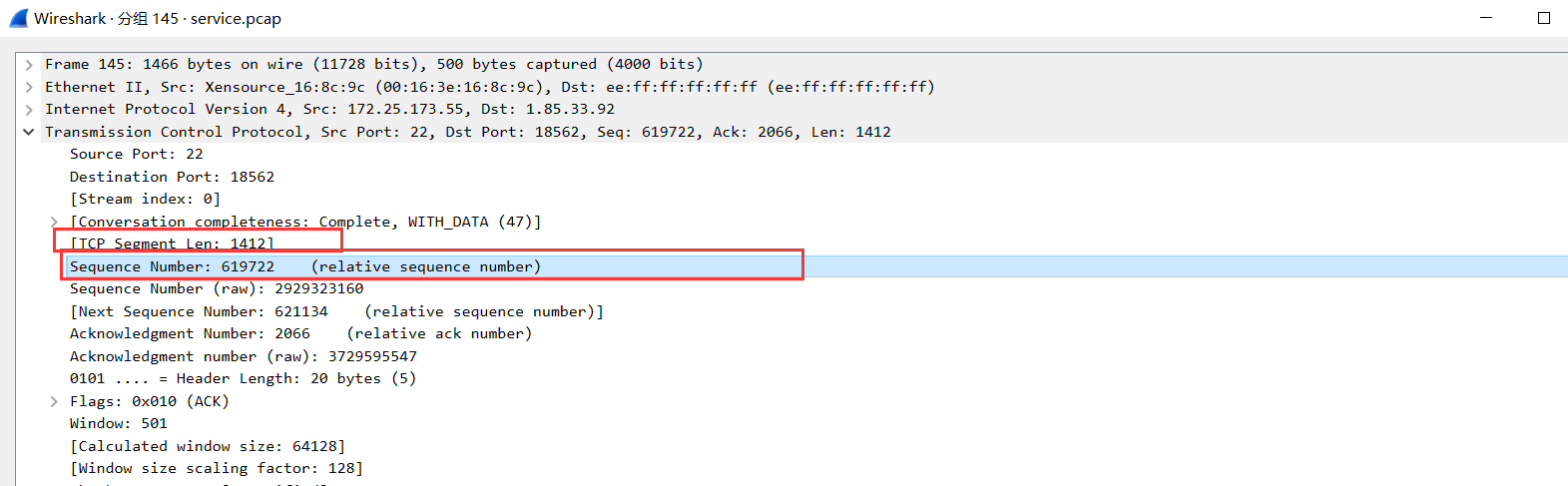
流量控制：

TCP流量控制的主要目的是根据接收方的实际接收能力，来控制发送方的数据发送速率。换句话说，流量控制旨在确保接收方能够按照其处理能力来处理数据，避免数据发送过快而导致接收方缓冲区溢出或数据丢失。

从服务器抓包：



第145号报文的具体内容如下：



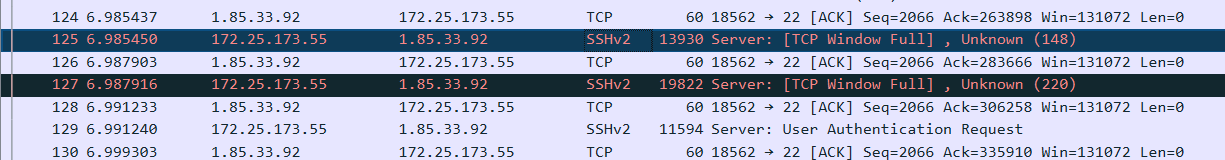
客户端最后一次向服务器发送应答的ACK为364150，窗口大小为262400，服务器发送的最后一个报文Sequence值为619722，长度为1412，

已发送未应答的长度：619722+1412-364150=256984

可能剩余的最小剩余缓冲区大小：262400-256984=5416

剩余窗口大小只剩5416，为了保证数据不溢出，服务器直到收到客户端的应答报文后才继续发送。

在流量控制失效或因其他原因没有正常工作时，还可以通过在窗口满时发送TCP Window Full的报文，以防止在接收方缓冲区满的情况下发送方仍一直发送数据，造成网络资源的浪费：



如图服务器发送125，127两个TCP Window Full的报文后，客户端不会发送数据，只会发送请求询问，直到服务器发送恢复正常工作的包后，客户端才可以向服务器发送数据。

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。

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）。

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。

1. 互动讨论主题

1）TCP的流量控制和拥塞控制有什么不同？

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？

3）讨论传输层与其上下相邻层的关系；

4）讨论TCP协议在传输实时语音流方面的优缺点。