四川大學

《计算机网络》实验报告（8）

Wireshark Lab: TCP v7.0

专　　业　软件工程

姓　　名　郭　政

学　　号　2023141461076

指导老师　程艳红

成绩分数

二零二五 年 六 月 二 日

Wireshark Lab: TCP v7.0

1. 将文件传输到 gaia.cs.umass.edu 的客户端计算机的 IP 地址和 TCP 端口号是多少？

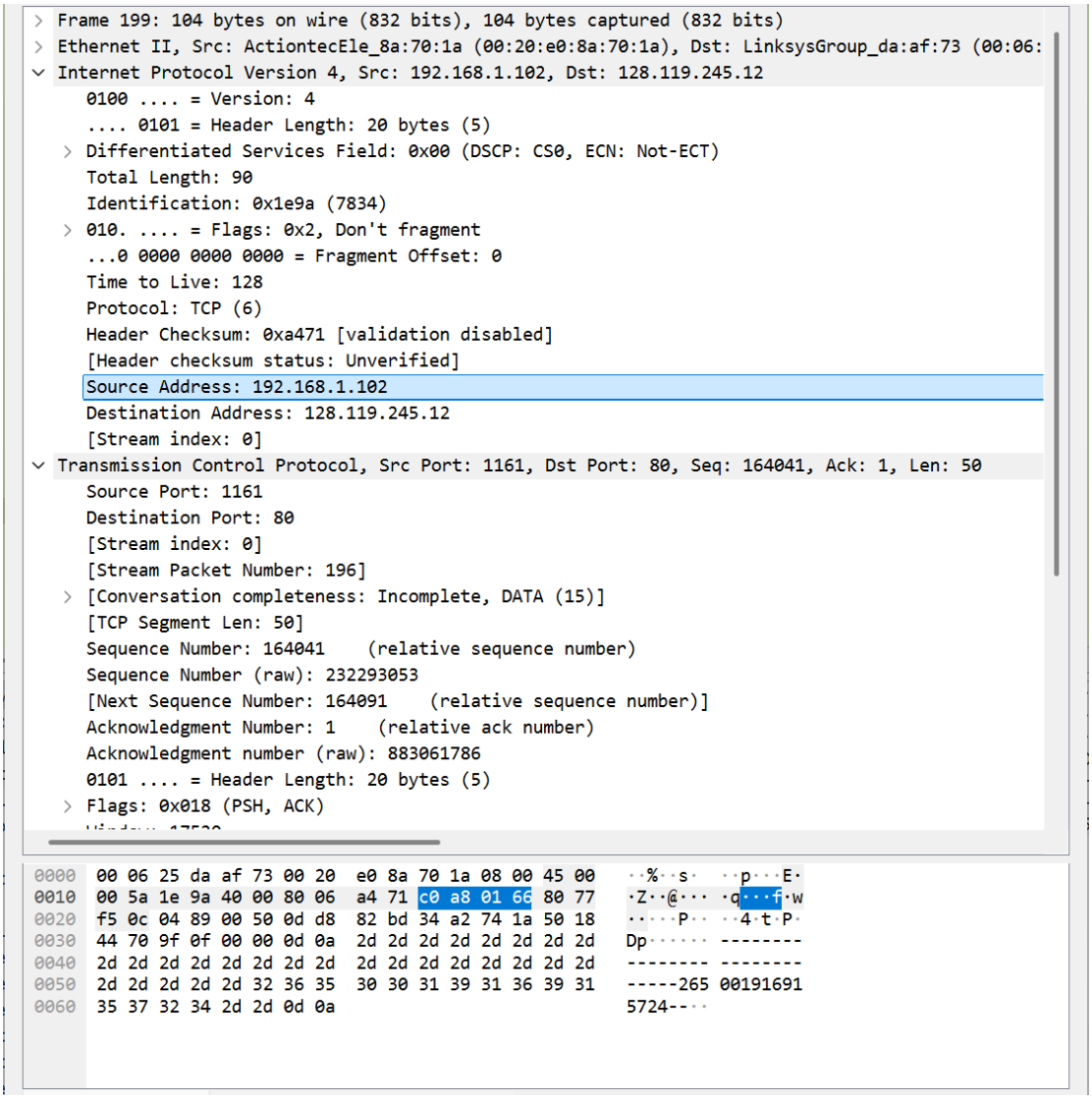
客户端 IP：192.168.1.102

端口号：1161

2. gaia.cs.umass.edu 的 IP 地址是多少？它在这个连接中使用哪个端口号发送和接收 TCP 段？

gaia.cs.umass.edu 的 IP 是 128.119.245.12

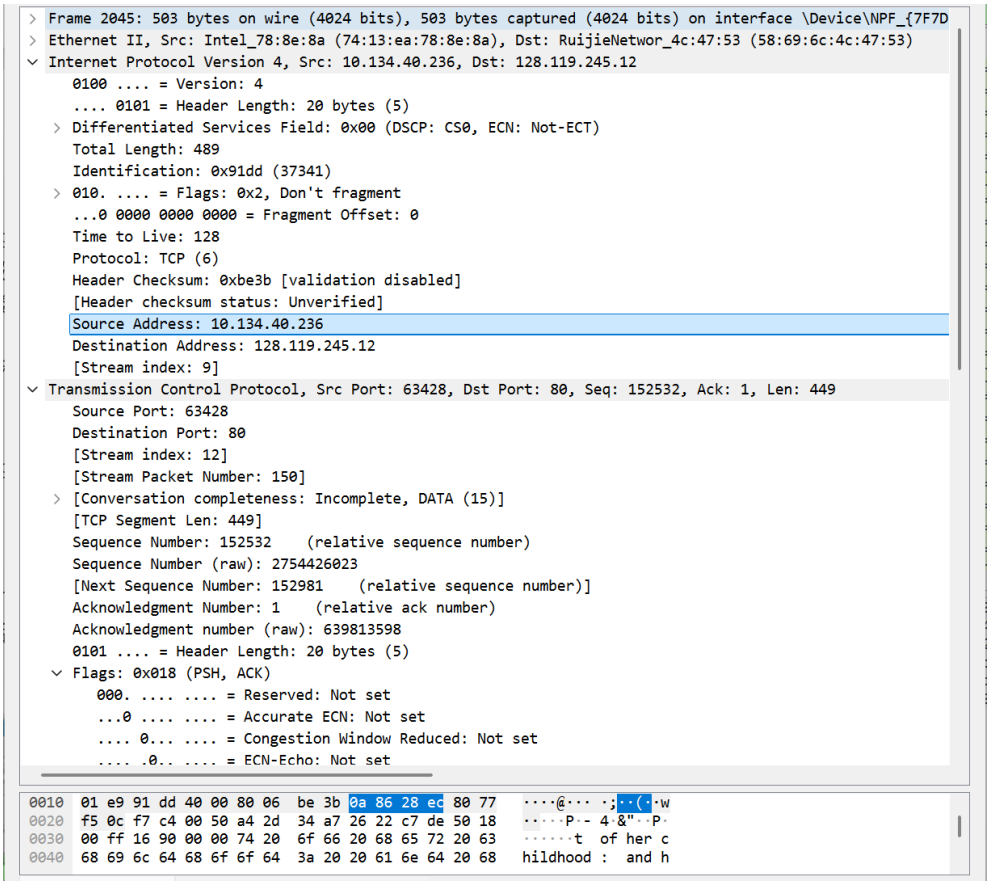
端口号：80



3. 你的客户端计算机在将文件传输到 gaia.cs.umass.edu 时使用的 IP 地址和 TCP 端口号是多少？

我的 IP：10.14.40.236

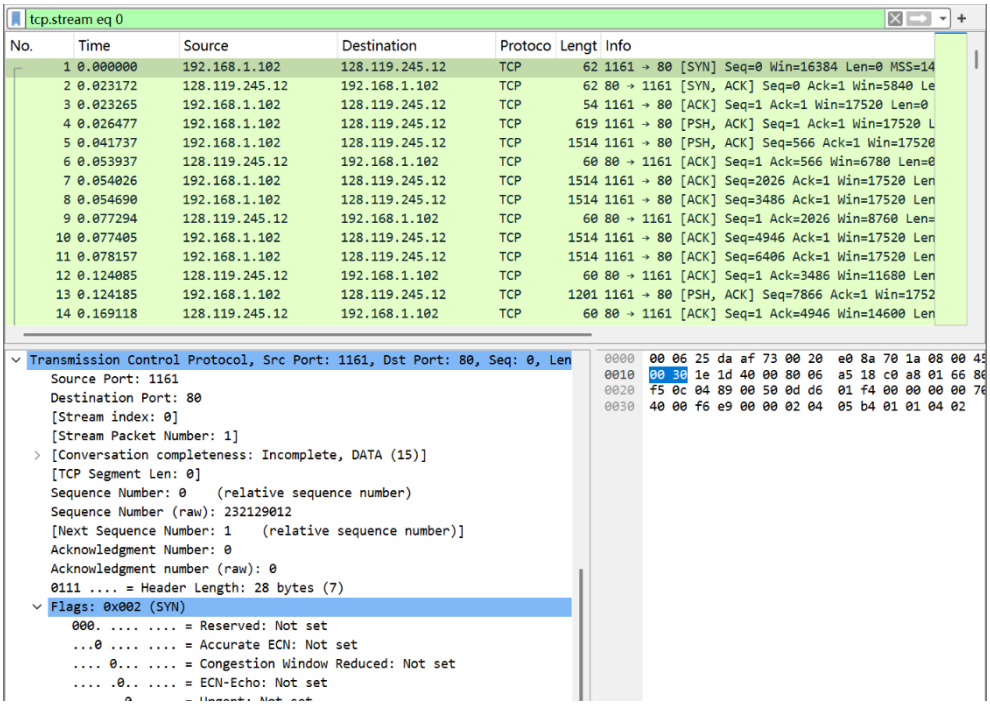
端口号：63428

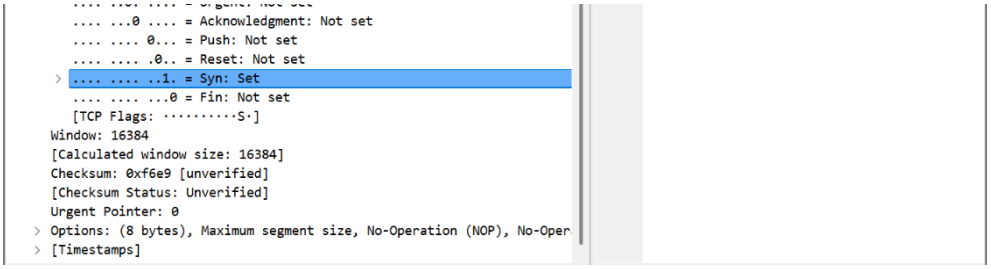


4. 客户端计算机和 gaia.cs.umass.edu 建立 TCP 连接时所使用的 TCP SYN 段的序列号是多少？段中的哪一项标识这是一个 SYN 段？

序列号是 0

SYN 标志位是 1



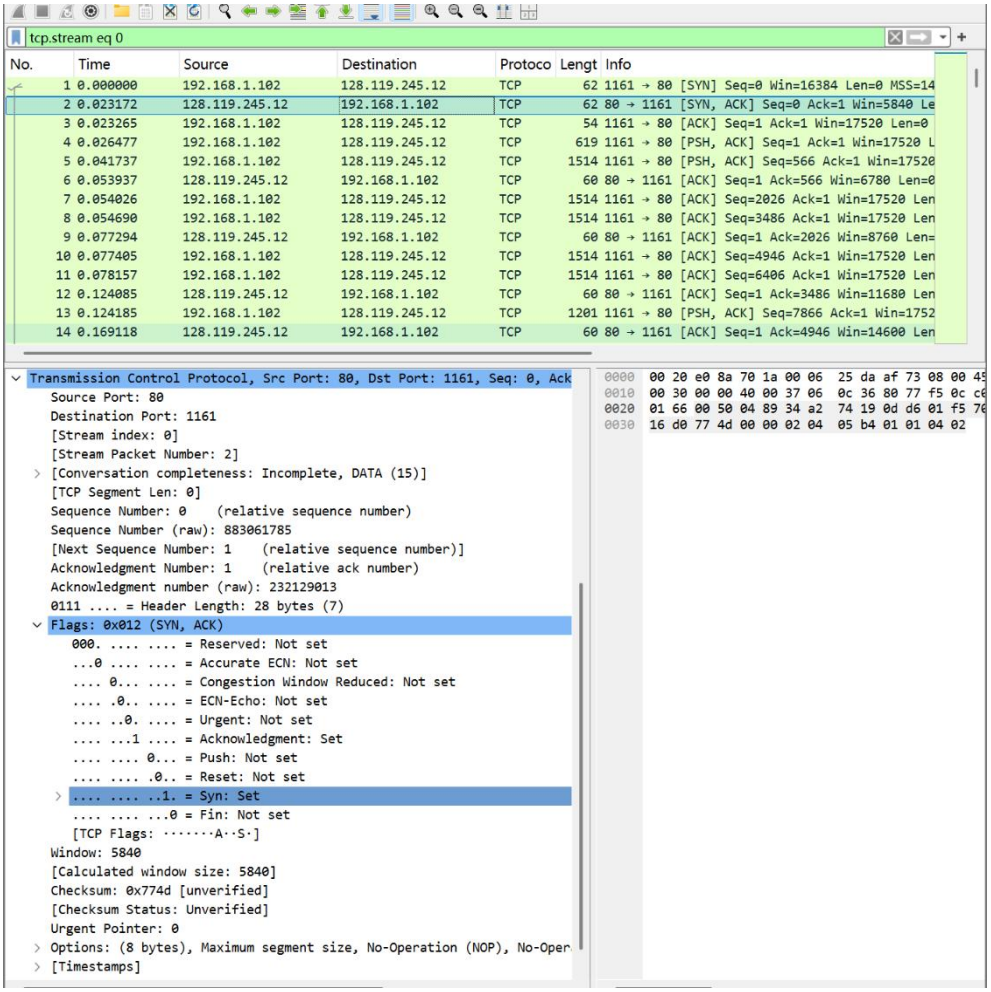


5. gaia.cs.umass.edu 回复 SYN 时发送的 SYNACK 段的序列号是多少？该段的确认号是多少？gaia.cs.umass.edu 是如何确定这个确认号的？段中的哪一项标识这是一个 SYNACK 段？

序列号为 0

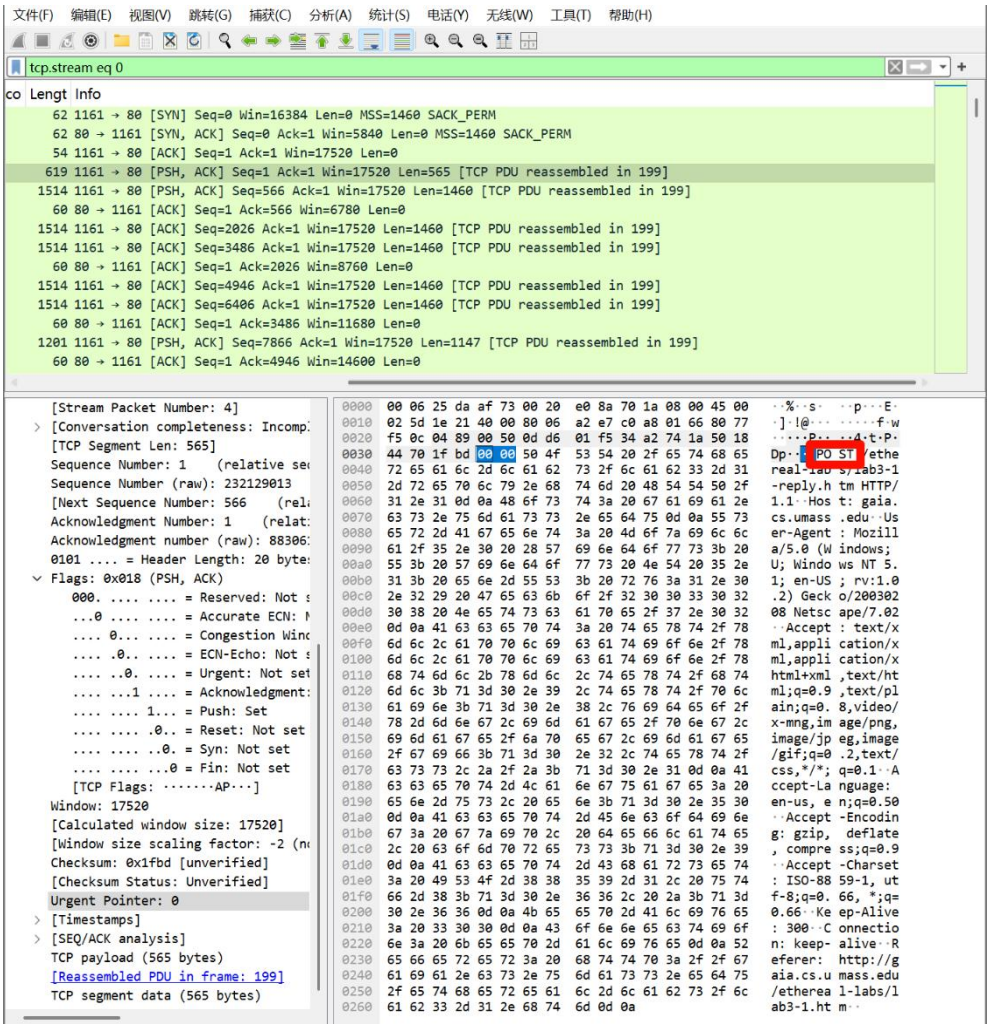
确认号为 1

gaia.cs.umass.edu 通过对收到的序列号为 0 的 SYN 请求的 SEQ 加 1 确定 ACK 的值，SYN、ACK 标志位都为 1



6. 包含 HTTP POST 命令的 TCP 段的序列号是多少？请注意，为了找到 POST 命令，你需要深入查看 Wireshark 窗口底部的数据包内容字段，查找其 DATA 字段中包含 “POST” 的段。

序列号为 1（标注如图中红框所示）



7. 将包含 HTTP POST 命令的 TCP 段视为该 TCP 连接中的第一个段，前六个 TCP 段（包括包含 HTTP POST 的那个）的序列号分别是多少？每个段是在什么时候发送的？每个段的 ACK 是在什么时候收到的？根据每个 TCP 段发送时间与其确认 ACK 的接收时间之间的差值，计算每个段的往返时间（RTT）。然后，根据课本第 3.5.3 节第 242 页中的 EstimatedRTT 公式，计算在收到每个 ACK 后的 EstimatedRTT 值。假设第一个段的 EstimatedRTT 值等于测得的 RTT 值，之后使用公式计算。

序列号：1、556、2026、3486、4946、6406

发送时间：0.026477,0.041737,0.054026,0.054690,0.077405,0.078157

ACK 时间：0.053937,0.077294,0.124085,0.169118,0.217299,0.267802

RTT：0.02746,0.035557,0.070059,0.11443,0.13989,0.18964

根据公式：EstimatedRTT = (1 - a) × EstimatedRTT + a × SampleRTT, a = 0.125

$EstimatedRTT_1 = SampleRTT_1 = 0.02746$

$EstimatedRTT_2 = (1 - 0.125) \times 0.02746 + 0.125 \times 0.035557$

$= 0.875 \times 0.02746 + 0.125 \times 0.035557$

$= 0.0240275 + 0.004444625$

$= 0.028472125$

$EstimatedRTT_3 = 0.875 \times 0.028472125 + 0.125 \times 0.070059$

= 0.024411109375 + 0.008757375

= 0.033168484375

$EstimatedRTT_4 = 0.875 \times 0.033168484375 + 0.125 \times 0.11443$

= 0.029521424 + 0.01430375

= 0.043825174375

$EstimatedRTT_5 = 0.875 \times 0.043825174375 + 0.125 \times 0.13989$

= 0.0383445276 + 0.01748625

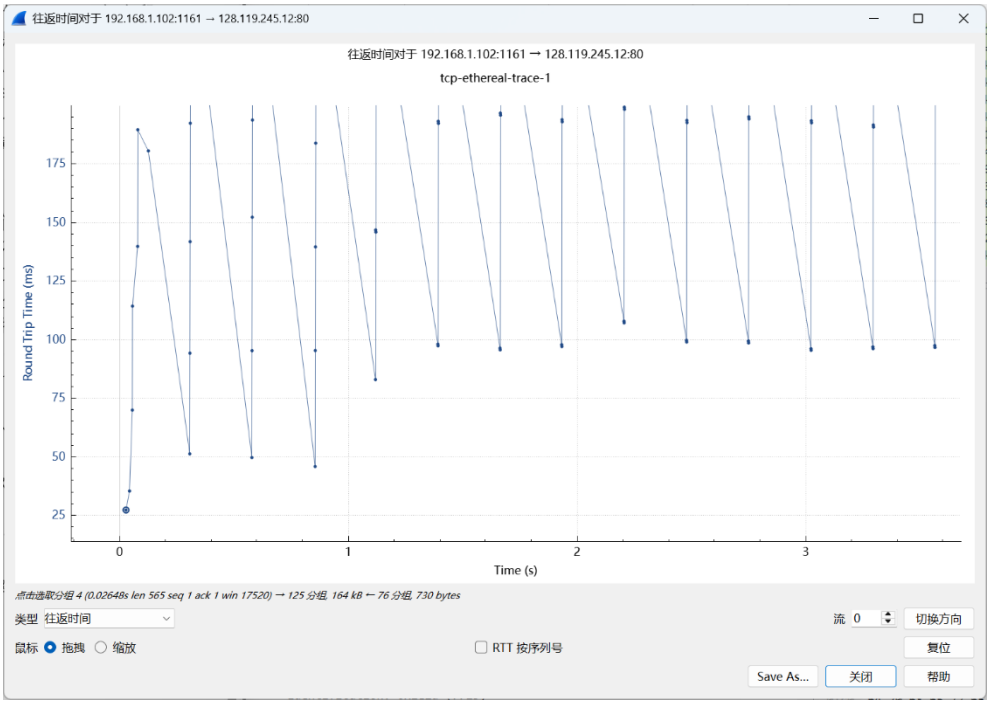
= 0.0558307776

$EstimatedRTT_6 = 0.875 \times 0.0558307776 + 0.125 \times 0.18964$

= 0.0488519309 + 0.023705

= 0.0725569309

EstimatedRTT：0.02746，0.02847，0.03317，0.04383，0.05583，0.07256



8. 前六个 TCP 段的长度分别是多少？

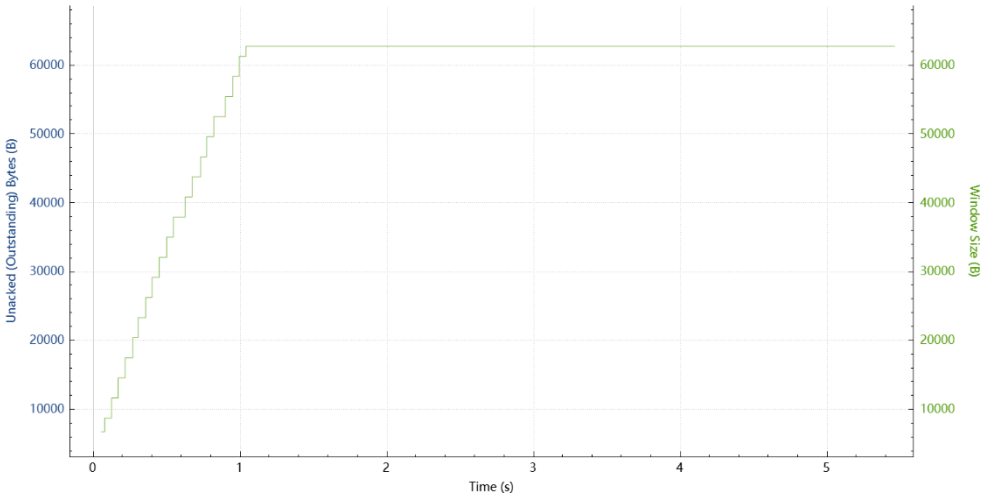
4	0.026477	192.168.1.102	128.119.245.12	TCP	619 1161 → 80 [PSH, ACK] Seq=1 Ack=1 Win=17520 Len=565 [TCP PDU reassembled in 199]
5	0.041737	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [PSH, ACK] Seq=566 Ack=1 Win=17520 Len=1460 [TCP PDU reassembled in 199]
6	0.053937	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=566 Win=6780 Len=0
7	0.054026	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=2026 Ack=1 Win=17520 Len=1460 [TCP PDU reassembled in 199]
8	0.054690	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=3486 Ack=1 Win=17520 Len=1460 [TCP PDU reassembled in 199]
9	0.077294	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=2026 Win=8760 Len=0
10	0.077405	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=4946 Ack=1 Win=17520 Len=1460 [TCP PDU reassembled in 199]
11	0.078157	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=6406 Ack=1 Win=17520 Len=1460 [TCP PDU reassembled in 199]
12	0.124085	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=3486 Win=11680 Len=0
13	0.124185	192.168.1.102	128.119.245.12	TCP	1201 1161 → 80 [PSH, ACK] Seq=7866 Ack=1 Win=17520 Len=1147 [TCP PDU reassembled in 199]
14	0.169118	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=4946 Win=14600 Len=0
15	0.217299	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=6406 Win=17520 Len=0
16	0.267802	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=7866 Win=20440 Len=0

（其中 No.4、5、7、8、10、11 为前 6 个报文段，No.6、9、12、14、15、16 为 ACK）

由上图 Len 可知除第一段长度为 556 字节，其他 5 段都是 1460 字节

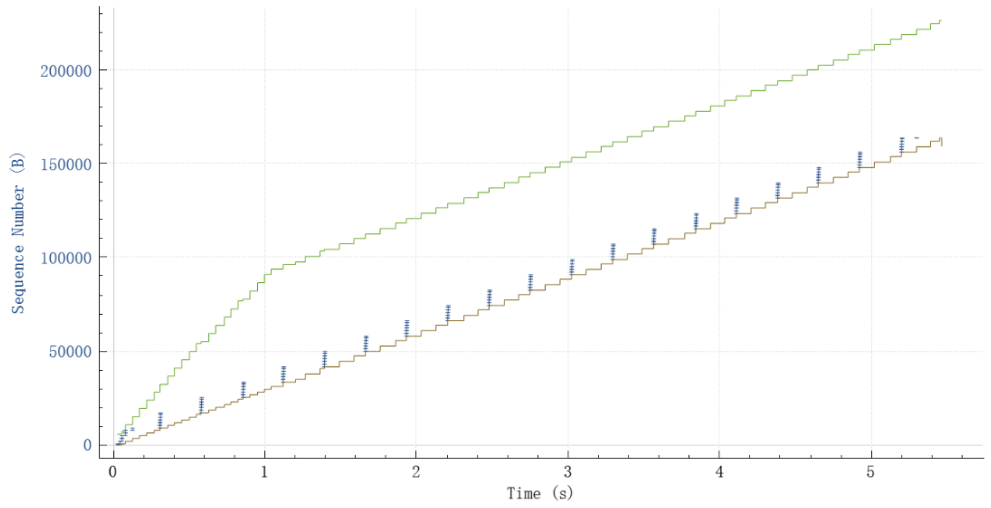
9. 整个抓包过程中，接收方通告的最小可用缓冲区空间是多少？接收方缓冲区空间不足是否曾经限制过发送方的发送速率？

最开始时接收端公布的可用缓冲空间最小，为 5840 字节，之后一直增大到 62780 字节，说明这里不会因为缺少接收端缓存而限制发送端



10. 在抓包文件中是否有重传的段？你是通过查看抓包中的哪些信息来回答这个问题的？

没有，通过图表发现序列号随时间一直增加，说明没有重传



11. 接收方通常在一个 ACK 中确认了多少数据？你能否找到接收方每接收两个段才发送一个 ACK 的情况？（参考课本第 250 页的表 3.2）

有时每收到一个数据段就 ACK 一次，第一次 ACK 确认 556 字节，大多数时候每次确认 1460 字节，有时是累计确认，比如 NO.88 确认了两段，确认的数据段长度为 2920 字节

4	0.026477	192.168.1.102	128.119.245.12	TCP	619 1161 → 80 [PSH, ACK] Seq=1 Ack=1 Wi
5	0.041737	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [PSH, ACK] Seq=566 Ack=1
6	0.053937	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=566 Win=6
7	0.054026	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=2026 Ack=1 Win=
8	0.054690	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=3486 Ack=1 Win=
9	0.077294	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=2026 Win=
10	0.077405	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=4946 Ack=1 Win=
11	0.078157	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=6406 Ack=1 Win=
12	0.124085	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=3486 Win=
13	0.124185	192.168.1.102	128.119.245.12	TCP	1201 1161 → 80 [PSH, ACK] Seq=7866 Ack=1
14	0.169118	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=4946 Win=
15	0.217299	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=6406 Win=
16	0.267802	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=7866 Win=
17	0.304807	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=9013 Win=

12. 该 TCP 连接的吞吐量是多少（单位时间内传输的字节数）？请解释你是如何计算这个值的。

(1604091-1)/(5.455830-0.026477)=30.222KB/s

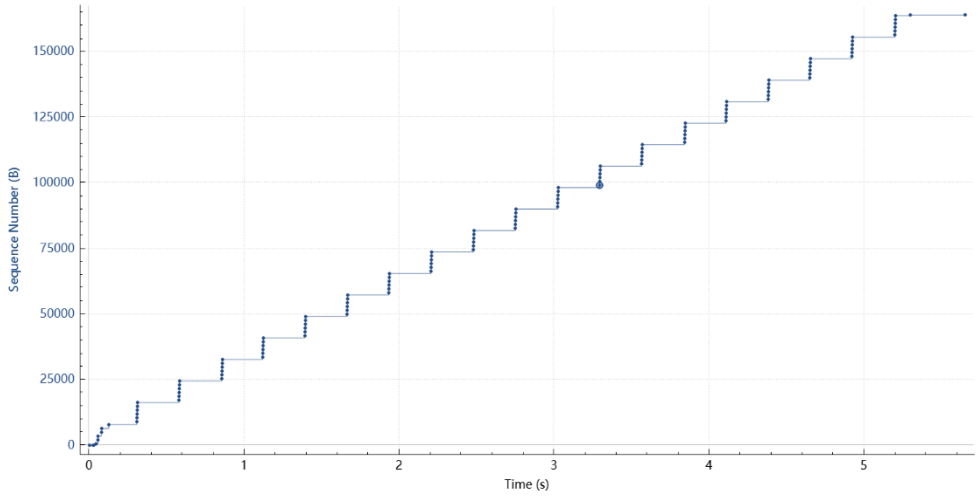
以握手后第一个 TCP 报文段为开始，到最后一个 ACK 结束，SEQ 差值即为数据总量，再除以它们的时间差值为吞吐量

201 5.447887	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=164041 Win=62780 Len=0
202 5.455830	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=164091 Win=62780 Len=0
203 5.461175	128.119.245.12	192.168.1.102	HTTP	784 HTTP/1.1 200 OK (text/html)

13. 使用 Time-Sequence-Graph (Stevens) 绘图工具查看从客户端发送到 gaia.cs.umass.edu 服务器的段的“序列号与时间”图。

你能否识别出 TCP 的慢启动（slow start）阶段是从哪里开始、到哪里结束？拥塞避免（congestion avoidance）阶段从哪里开始？

请对比测得的数据和我们在课本中学习的 TCP 理想行为，指出有哪些不同之处，并进行评论。



慢启动从发出了 HTTP POST 报文段后开始，但从图中并不能看出慢启动什么时候结束，拥塞避免是什么时候开始的

TCP 采用慢启动的目的是进行拥塞控制，但是在实际的网络通信中，对于一些数据量较小的小文件，在网络畅通的情况下发送非常快，甚至可能在慢启动结束之前就已经发送完毕。而这种情况下，采用慢启动方式反而来制约了文件的快速发送，从而影响通信的效率。

14. 对你在将文件从你的计算机传输到 gaia.cs.umass.edu 时收集到的抓包数据，回答以上两个问题

（1）TCP 的慢启动阶段从连接建立后第一个 HTTP POST 段发送时开始。虽然我们无法直接从时间-序列图中获取发送方的拥塞窗口大小，但可以通过未确认数据来估计拥塞窗口的下界。由于接收方缓冲区在本次实验中并未成为瓶颈，因此我们可以忽略其对窗口上限的影响，专注于通过未确认数据量近似反映拥塞窗口的增长。

（2）理论上，TCP 理想状态下的发送方应始终保持尽可能活跃，以充分利用可用带宽，同时通过 AIMD 算法控制拥塞风险。然而实际中 TCP 的表现受应用层行为强烈影响。本次实验中，尽管网络未出现显著丢包或延迟，TCP 连接在慢启动阶段结束之前就完成了数据传输。这是因为 Web 应用中传输的对象体积非常小，发送方没有更多数据可以传输，导致连接在慢启动过程中即被终止。这种现象表明：对于短小的 Web 对象，TCP 慢启动机制可能

引发不必要的启动延迟，降低了小对象传输的效率。这也是为什么现代 Web 应用倾向于使用持久连接或优化 TCP 初始窗口大小的原因。