

1. What is the IP address and TCP port number used by the client computer (source) that is transferring the file to gaia.cs.umass.edu? To answer this question, it's probably easiest to select an HTTP message and explore the details of the TCP packet used to carry this HTTP message, using the "details of the selected packet header window" (refer to Figure 2 in the "Getting Started with Wireshark" Lab if you're uncertain about the Wireshark windows ?

Answer: IP Address: 192.168.1.102 – TCP Port: 1161

2. What is the IP address of gaia.cs.umass.edu? On what port number is it sending and receiving TCP segments for this connection?

Answer: IP Address: 128.119.245.12 – TCP Port: 80

3. What is the IP address and TCP port number used by your client computer (source) to transfer the file to gaia.cs.umass.edu?

Answer: From my trace

The IP Address: 192.168.1.102

The TCP Port: 50623

```
> Internet Protocol Version 4, Src: 128.119.245.12, Dst: 10.128.183.215
▼ Transmission Control Protocol, Src Port: 80, Dst Port: 50623, Seq: 0, Ack: 1, Len: 0
    Source Port: 80
    Destination Port: 50623
    [Stream index: 4]
    > [Conversation completeness: Incomplete (14)]
```

4. What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and gaia.cs.umass.edu? What is it in the segment that identifies the segment as a SYN segment?

Answer: The sequence number of TCP SYN segment is 0 (seq = 0).

No.	Time	Source	Destination	Protocol	Length	Info
203	5.461175	128.119.245.12	192.168.1.102	TCP	784	80 → 1161 [PSH, ACK] Seq=1 Ack=164091 Win=62780 Len=730
206	5.651141	192.168.1.102	128.119.245.12	TCP	54	1161 → 80 [ACK] Seq=164091 Ack=731 Win=16790 Len=0
213	7.595557	192.168.1.102	199.2.53.206	TCP	62	1162 → 631 [SYN] Seq=0 Win=16384 Len=0 MSS=1460 SACK_PERM


```
> Frame 213: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on 0
> Ethernet II, Src: ActiontecEle_8a:70:1a (00:20:e0:8a:70:1a), Dst: LinksysGroup_da:af:73 (00:06:25:da:af:73)
> Internet Protocol Version 4, Src: 192.168.1.102, Dst: 199.2.53.206
▼ Transmission Control Protocol, Src Port: 1162, Dst Port: 631, Seq: 0, Len: 0
    Source Port: 1162
    Destination Port: 631
    [Stream index: 1]
    > [Conversation completeness: Incomplete, SYN_SENT (1)]
    [TCP Segment Len: 0]
    Sequence Number: 0 (relative sequence number)
    Sequence Number (raw): 234062521
    [Next Sequence Number: 1 (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    0111 ... = Header Length: 28 bytes (7)
    > Flags: 0x002 (SYN)
    Window: 16384
```

```
0000 00 06 25 da af
0010 00 30 1e 9e 40
0020 35 ce 04 8a 02
0030 40 00 ec 92 00
```

We can identify this segment as a SYN segment because the flag header has SYN hexadecimal number (0x002) this flag set to 1.

```

Flags: 0x002 (SYN)
000. .... = Reserved: Not set
...0 .... = Accurate ECN: Not set
.... 0... = Congestion Window Reduced: Not set
.... .0.. = ECN-Echo: Not set
.... ..0. = Urgent: Not set
.... ...0 = Acknowledgment: Not set
.... .... 0... = Push: Not set
.... ..... 0.. = Reset: Not set
> .... .... ..1. = Syn: Set

```

- What is the sequence number of the SYN ACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN? What is the value of the Acknowledgement field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? What is it in the segment that identifies the segment as a SYNACK segment?

Answer: The sequence number of the SYN ACK segment is 0 (seq = 0)

The value of the Acknowledgement field is 1 (in raw is 3852961089).

```

Acknowledgment Number: 1 (relative ack number)
Acknowledgment number (raw): 232129013

```

The Acknowledgement is determined by adding 1 to initial seq number of SYN segment from client computer ($0 + 1 = 0$).

```

Flags: 0x012 (SYN, ACK)
000. .... = Reserved: Not set
...0 .... = Accurate ECN: Not set
.... 0... = Congestion Window Reduced: Not set
.... .0.. = ECN-Echo: Not set
.... ..0. = Urgent: Not set
.... ...1 = Acknowledgment: Set
.... .... 0... = Push: Not set
.... ..... 0.. = Reset: Not set
> .... .... ..1. = Syn: Set
.... .... ...0 = Fin: Not set
[TCP Flags: .....A..S.]

```

- What is the sequence number of the TCP segment containing the HTTP POST command? Note that in order to find the POST command, you'll need to dig into the packet content field at the bottom of the Wireshark window, looking for a segment with a "POST" within its DATA field.

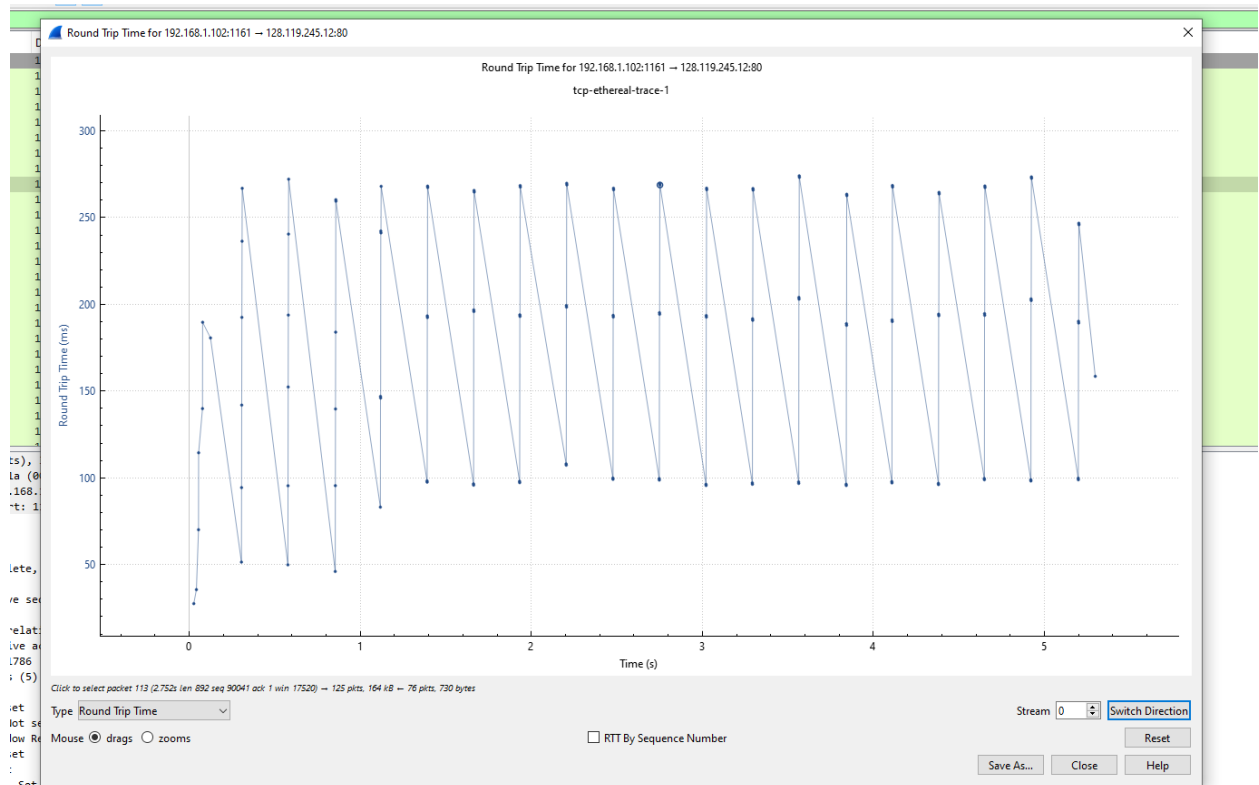
```
> 199 5.297341      192.168.1.102      128.119.245.12      HTTP      104 [POST /ethereal-labs/lab3-1-reply.htm HTTP/1.1 (text/plain)]

> Frame 199: 104 bytes on wire (832 bits), 104 bytes captured (832 bits) on interface 0
> Ethernet II, Src: ActiontecEle_8a:70:1a (00:20:e0:8a:70:1a), Dst: LinksysGroup_da:af:73 (00:06:25:da:af:73)
> Internet Protocol Version 4, Src: 192.168.1.102, Dst: 128.119.245.12
> Transmission Control Protocol, Src Port: 1161, Dst Port: 80, Seq: 164041, Ack: 1, Len: 50
  0000  00 06 25 da af 73 00 20  e0 8a 70 1a 08 00 45 00
  0010  00 5a 1e 9a 40 00 00 06  47 71 c0 a8 01 66 80 77
  0020  f5 8c 04 89 00 50 bd d8  82 bd 34 a2 74 1a 50 15
  0030  44 70 9f 0f 00 00 0d 0a  2d 2d 2d 2d 2d 2d 2d 2d
  0040  2d 2d 2d 2d 2d 2d 2d 2d  2d 2d 2d 2d 2d 2d 2d 2d
```

- Given the difference between when each TCP segment was sent, and when its acknowledgement was received, what is the RTT value for each of the six segments?

Assume that the value of the EstimatedRTT is equal to the measured RTT for the first segment, and then is computed using the EstimatedRTT equation on page 242 for all subsequent segments.

1	0.000000	192.168.1.102	128.119.245.12	TCP	62	1161 + 80	[SYN]	Seq=0	Win=16384	Len=0	MSS=1460	SACK_PERM	
2	0.023172	128.119.245.12	192.168.1.102	TCP	62	80 + 1161	[SYN, ACK]	Seq=0	Ack=1	Win=5840	Len=0	MSS=1460	SACK_PERM
3	0.023265	192.168.1.102	128.119.245.12	TCP	54	1161 + 80	[ACK]	Seq=1	Ack=1	Win=17520	Len=0		
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 segment of a reassembled PDU]	
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 segment of a reassembled PDU]	
6	0.053937	128.119.245.12	192.168.1.102	TCP	60	80 + 1161	[ACK]	Seq=1	Ack=566	Win=6780	Len=0		



8. What is the length of each of the first six TCP segments ?

Answer: The length is shown below

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.102	128.119.245.12	TCP	62	1161 → 80 [SYN] Seq=0 Win=16384 Len=0 MSS=1460 SACK_PERM
2	0.023172	128.119.245.12	192.168.1.102	TCP	62	80 → 1161 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM
3	0.023265	192.168.1.102	128.119.245.12	TCP	54	1161 → 80 [ACK] Seq=1 Ack=1 Win=17520 Len=0
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 segment of a reassembled PDU]
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 segment of a reassembled PDU]
6	0.053937	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=566 Win=6780 Len=0

9. What is the minimum amount of available buffer space advertised at the receiver for the entire trace? Does the lack of receiver buffer space ever throttle the sender?

Answer:

The minimum amount of available buffer space (receiver window) is 5840 bytes.

The sender is never throttled because we never reach full capacity of the window (62780 bytes).

10. Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?

Answer: There are retransmitted segments in trace file. Because the sequence number do not increases or decrease by time no duration has increased sequence number.

11. How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment (see Table 3.2 on page 250 in the text).

Answer: Each ACK has different lengths of data. But you can see in my case the length of 0 and 1460 usually appear in ACKs. There are cases where the receivers is ACKing every other segment. According to table 3.2 segment 80 with 2920 bytes = 1460 * 2.

Top 640 packets == 192.168.1.7 >> 128.119.245.12

No.	Time	Source	Destination	Protocol	Length	Info
368	7.693	192.168.1.7	128.119.245.12	TCP	1506	55616 → 80 [ACK] Seq=89288 Ack=1 Min=516 Len=1452 [TCP segment of a reassembled PDU]
369	7.693	192.168.1.7	128.119.245.12	TCP	1506	[TCP Out-Of-Order] 55616 → 80 [ACK] Seq=87836 Ack=1 Min=516 Len=1452 [TCP segment of a reassembled PDU]
370	7.693	192.168.1.7	128.119.245.12	TCP	1506	[TCP Out-Of-Order] 55616 → 80 [ACK] Seq=87836 Ack=1 Min=516 Len=1452 [TCP segment of a reassembled PDU]
371	7.693	192.168.1.7	128.119.245.12	TCP	1506	55616 → 80 [ACK] Seq=90740 Ack=1 Min=516 Len=1452 [TCP segment of a reassembled PDU]
372	7.961	128.119.245.12	192.168.1.7	TCP	60	80 → 55616 [ACK] Seq=1 Ack=89288 Min=1432 Len=0
373	7.961	128.119.245.12	192.168.1.7	TCP	1506	55616 → 80 [ACK] Seq=92192 Ack=1 Min=516 Len=1452 [TCP segment of a reassembled PDU]
374	7.962	128.119.245.12	192.168.1.7	TCP	60	80 → 55616 [ACK] Seq=1 Ack=90740 Min=1432 Len=0
375	7.962	128.119.245.12	192.168.1.7	TCP	1506	55616 → 80 [ACK] Seq=93644 Ack=1 Min=516 Len=1452 [TCP segment of a reassembled PDU]
376	7.962	128.119.245.12	192.168.1.7	TCP	60	[TCP Dup ACK 374#1] 80 → 55616 [ACK] Seq=1 Ack=90740 Min=1432 Len=0
377	7.962	128.119.245.12	192.168.1.7	TCP	60	80 → 55616 [ACK] Seq=1 Ack=92192 Min=1432 Len=0
378	7.962	128.119.245.12	192.168.1.7	TCP	1506	55616 → 80 [ACK] Seq=96096 Ack=1 Min=516 Len=1452 [TCP segment of a reassembled PDU]
383	8.229	128.119.245.12	192.168.1.7	TCP	60	80 → 55616 [ACK] Seq=1 Ack=93644 Min=1432 Len=0
384	8.229	128.119.245.12	192.168.1.7	TCP	1506	55616 → 80 [ACK] Seq=96548 Ack=1 Min=516 Len=1452 [TCP segment of a reassembled PDU]
385	8.230	128.119.245.12	192.168.1.7	TCP	60	80 → 55616 [ACK] Seq=1 Ack=96096 Min=1432 Len=0
386	8.230	128.119.245.12	192.168.1.7	TCP	1506	55616 → 80 [PSH, ACK] Seq=98000 Ack=1 Min=516 Len=1452 [TCP segment of a reassembled PDU]
387	8.231	128.119.245.12	192.168.1.7	TCP	60	80 → 55616 [ACK] Seq=1 Ack=96548 Min=1432 Len=0
388	8.231	128.119.245.12	192.168.1.7	TCP	1506	55616 → 80 [ACK] Seq=98452 Ack=1 Min=516 Len=1452 [TCP segment of a reassembled PDU]
390	8.497	128.119.245.12	192.168.1.7	TCP	60	80 → 55616 [ACK] Seq=1 Ack=98000 Min=1432 Len=0
391	8.497	128.119.245.12	192.168.1.7	TCP	1506	55616 → 80 [ACK] Seq=100000 Ack=1 Min=516 Len=1452 [TCP segment of a reassembled PDU]
392	8.499	128.119.245.12	192.168.1.7	TCP	60	80 → 55616 [ACK] Seq=1 Ack=98452 Min=1432 Len=0
393	8.499	128.119.245.12	192.168.1.7	TCP	60	80 → 55616 [ACK] Seq=1 Ack=100000 Min=1426 Len=0
394	8.499	128.119.245.12	192.168.1.7	TCP	1506	55616 → 80 [ACK] Seq=102336 Ack=1 Min=516 Len=1452 [TCP segment of a reassembled PDU]
395	8.499	128.119.245.12	192.168.1.7	TCP	1506	55616 → 80 [ACK] Seq=103808 Ack=1 Min=516 Len=1452 [TCP segment of a reassembled PDU]
396	8.765	128.119.245.12	192.168.1.7	TCP	60	80 → 55616 [ACK] Seq=1 Ack=102336 Min=1432 Len=0
397	8.765	128.119.245.12	192.168.1.7	TCP	1506	55616 → 80 [ACK] Seq=105200 Ack=1 Min=516 Len=1452 [TCP segment of a reassembled PDU]
398	8.767	128.119.245.12	192.168.1.7	TCP	60	80 → 55616 [ACK] Seq=1 Ack=105200 Min=1426 Len=0
399	8.767	128.119.245.12	192.168.1.7	TCP	1506	55616 → 80 [ACK] Seq=108712 Ack=1 Min=516 Len=1452 [TCP segment of a reassembled PDU]
400	8.767	128.119.245.12	192.168.1.7	TCP	1506	55616 → 80 [ACK] Seq=108164 Ack=1 Min=516 Len=1452 [TCP segment of a reassembled PDU]
401	9.033	128.119.245.12	192.168.1.7	TCP	60	80 → 55616 [ACK] Seq=1 Ack=108712 Min=1432 Len=0
402	9.033	128.119.245.12	192.168.1.7	TCP	1506	55616 → 80 [ACK] Seq=109616 Ack=1 Min=516 Len=1452 [TCP segment of a reassembled PDU]
403	9.033	128.119.245.12	192.168.1.7	TCP	1506	55616 → 80 [ACK] Seq=111068 Ack=1 Min=516 Len=1452 [TCP segment of a reassembled PDU]
404	9.033	128.119.245.12	192.168.1.7	TCP	60	80 → 55616 [ACK] Seq=1 Ack=108164 Min=1432 Len=0

Frame 385: 1506 bytes on wire (12048 bits), 1506 bytes captured (12048 bits) on interface vpp50780261-BB-000000
 Ethernet II, Src: Hecaton_A4:26:e2 (c0:9f:04:a4:26:e2), Dst: zte_c3:ff:74 (c0:9f:04:c3:ff:74)
 Internet Protocol Version 4, Src: 192.168.1.7, Dst: 128.119.245.12
 Transmission Control Protocol, Src Port: 55616, Dst Port: 80, Seq: 61780, Ack: 1, Len: 1452
 Source Port: 55616
 Destination Port: 80
 [Stream index: 7]
 [Conversation completeness: Incomplete (20)]
 [TCP Segment Len: 1452]
 Sequence Number: 61780 (relative sequence number)
 Sequence Number (raw): 265807621
 [Next Sequence Number: 63152 (relative sequence number)]
 Acknowledgment Number: 1 (relative ack number)

Reset (tcp.flags.reset), 1 byte

Packets: 575 - Displayed: 325 (56.5%) - Dropped: 0 (0.0%) Profile: Default

12. What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

Answer: The throughput can be calculated by using seq of last ack – the first seq number and divide by time since first frame : $(153037 - 1) / 11.829099 = 12937.07$ (byte per second)

545	17.829036	128.119.245.12	192.168.1.7	TCP	60	80 → 55616 [FIN, ACK] Seq=778 Ack=153037 Win=1432 Len=0
546	17.829099	192.168.1.7	128.119.245.12	TCP	54	55616 → 80 [ACK] Seq=153037 Ack=779 Win=513 Len=0

Acknowledgment Number: 779 (relative ack number)	0000 c0 9f e1 c3 ff 74 c0 ff d4 a4 26 e2 08 00 45 00
Acknowledgment number (raw): 3016442908	0010 00 28 33 4e 40 00 00 06 90 4e c0 a8 01 07 80 77

13. Use the Time-Sequence-Graph(Stevens) plotting tool to view the sequence number versus time plot of segments being sent from the client to the gaia.cs.umass.edu server. Can you identify where TCP's slowstart phase begins and ends, and where congestion avoidance takes over? Comment on ways in which the measured data differs from the idealized behavior of TCP that we've studied in the text.

Answer: The TCP slow start begins at the start of the connection (when HTTP POST segment was sent out). TCP slow start phase and congestion avoidance phase can be identified by the value of window size of TCP sender. However, the value of the congestion window size cannot be obtained directly from the Time-Sequence-Graph (Stevens) graph. Nevertheless, we can estimate the lower bound of the TCP window size by the amount of outstanding data because the outstanding data is the amount of

data without acknowledgement. We also know that TCP window is constrained by the receiver window size and the receiver buffer can act as the upper bound of the TCP window size. In this trace, the receiver buffer is not the bottleneck; therefore, this upper bound is not quite useful to infer the TCP window size. Hence, we focus on the lower bound of the TCP window size.

