**Wireshark Lab 3: TCP**

**Group Details:**

**Shayshu NR – 1005035196**

**Johnathan Yan - 1004745476**

**Mark:**



|  |  |  |
| --- | --- | --- |
|  | **Question** | **Answer** |
| 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? | Client IP: 192.168.1.102  Client source port: 1161 |
| Annotated Screenshots  (if needed) |  | |
| 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? | Server IP: 128.119.245.12  Server destination port: 80 |
| Annotated Screenshots  (if needed) |  | |
| 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? | Client IP: 10.0.0.230  Client source port: 58489 |
| Annotated Screenshots  (if needed) |  | |
| 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? | The relative sequence number is 0 (raw 23219012). The SYN flag is set in this segment. |
| Annotated Screenshots  (if needed) |  | |
| 5 | What is the sequence number of the SYNACK 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? | The relative sequence number in the SYNACK is 0 (raw 883061785). The relative acknowledgement number is 1 (raw 23219013). This value is the sequence number that the client sent incremented by one. The SYN, and ACK flags are set thus denoting it as a SYNACK segment. |
| Annotated Screenshots  (if needed) |  | |
| 6 | What is the sequence number of the TCP segment containing the HTTP POST command? | The relative sequence number is 1 (raw 232129013) |
| Annotated Screenshots  (if needed) |  | |
| 7 | Consider the TCP segment containing the HTTP POST as the first segment in the TCP connection. What are the sequence numbers of the first six segments in the TCP connection (including the segment containing the HTTP POST)? At what time was each segment sent? When was the ACK for each segment received?  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? What is the EstimatedRTT value after the  receipt of each ACK? | Sequence numbers:   1. 232129013 2. 232129578 3. 232131038 4. 232132498 5. 232133958 6. 232135418   Time sent:   1. 0.026477s 2. 0.041737s 3. 0.054026s 4. 0.054690s 5. 0.077405s 6. 0.078157s   Time received:   1. 0.053937s 2. 0.077294s 3. 0.124085s 4. 0.169118s 5. 0.217299s 6. 0.267802s   RTT:   1. 0.027460s 2. 0.035557s 3. 0.070059s 4. 0.114428s 5. 0.139894s 6. 0.190397s   Les α = 0.125  Estimate RTT:   1. 0.027460s 2. 0.028472s 3. 0.033670s 4. 0.043765s 5. 0.055781s 6. 0.072608s |
| Annotated Screenshots  (if needed) |  | |
| 8 | What is the length of each of the first six TCP segments? | 1. 565  1. 1460 2. 1460 3. 1460 4. 1460 5. 1460 |
| Annotated Screenshots  (if needed) |  | |
| 9 | What is the minimum amount of available buffer space advertised at the received  for the entire trace? Does the lack of receiver buffer space ever throttle the  sender? | Window size increases over time:   1. 5840 2. 6780 3. 8760 4. 11680 5. 14600 6. 17520   No, the window size doesn’t throttle the sender. Minimum is 5840. |
| Annotated Screenshots  (if needed) |  | |
| 10 | Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question? | There aren’t any retransmitted files.  You simply look for packets with the same sequence numbers being sent at different times. Also, you can check the time vs sequence number graph, or the retransmission analysis in Wireshark. |
| Annotated Screenshots  (if needed) |  | |
| 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 | The receiver usually acknowledges about 1460 bytes per ACK segment. No in the given trace, there is an acknowledgement for every segment. Meaning send 3 segments get 3 ACKs. |
| Annotated Screenshots  (if needed) |  | |
| 12 | What is the throughput (bytes transferred per unit time) for the TCP connection?  Explain how you calculated this value. | Throughput is defined as the amount of data transmitted in each period of time. I chose to use a period of 5 seconds, since that is when most of the data was done being transferred. By using the sequence number and the scaling factor you can find how much data has been transmitted after 5 seconds.  Throughput = 151197 bytes / 5 seconds  Throughput = 30239.1 bytes / second  Throughput = 30.24 Kb/s |
| Annotated Screenshots  (if needed) |  | |
| 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. | The slow start period is from 0s to about 0.1s. The Congestion avoidance starts at 0.1s and continues till the end of the transfer (around 5s).  The graph is discretized and not smooth, this is because there is a wait time for the client to receive the acknowledgements from the server. Also, in this graph the client is implementing pipelining thus causing the data points to stack on top of each other. |
| Annotated Screenshots  (if needed) |  | |
| 14 | Answer each of two questions above for the trace that you have gathered when you transferred a file from your computer to gaia.cs.umass.edu | Throughput = 153078 bytes/ 2.12s  Throughput = 72.21 Kb/s  The slow start region looks to be from 0s to 0.2s, followed by congestion avoidance from the rest.  This graph shows how real pipelining works. You can clearly see the staggered sending of multiple one right after the other.  Furthermore, this graph illustrates how the sender can transmit data faster than the receiver can reply, as evident by the large plateaus between sending. |
| Annotated Screenshots  (if needed) |  | |