**report.md**

PART 1

# Q1

*What is the IP address of gaia.cs.umass.edu?* **128.119.245.12**

*On what port number is it sending and receiving TCP segments for this connection?* **80 (port reserved for HTTP)**

*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?* **ip address 192.168.1.102, port 1161**

# Q2

*What is the sequence number of the TCP segment containing the HTTP POST command?* **232129013**

# Q3

| pkt | seq# | length (bytes) | sent @ | ACK recv'd @ | RTT | EstRTT |
|---|---|---|---|---|---|---|
| 4 | 232129013 | 565 | 0.026477 | 0.053937 | 0.027460 | 0.027460 |
| 5 | 232129578 | 1460 | 0.041737 | 0.077294 | 0.035557 | 0.028472125 |
| 7 | 232131038 | 1460 | 0.054026 | 0.124085 | 0.070059 | 0.033670484 |
| 8 | 232132498 | 1460 | 0.054690 | 0.169118 | 0.114428 | 0.043765173 |
| 10 | 232133958 | 1460 | 0.077405 | 0.217299 | 0.139894 | 0.055781276 |
| 11 | 232135418 | 1460 | 0.078157 | 0.267802 | 0.189645 | 0.072514242 |

*note that the length is the length of the payload, the header is another 20 bytes*

EstRTT = (1-alpha) * EstRTT + (alpha * SampleRTT), where alpha = 0.125

EstRTT(1) = SampleRTT(1) = 0.027460

EstRTT(2) = (0.875 * 0.027460) + (0.125 * 0.035557) = 0.028472125

EstRTT(3) = (0.875 * 0.028472125) + (0.125 * 0.070059) = 0.033670484

EstRTT(4) = (0.875 * 0.033670484) + (0.125 * 0.114428) = 0.043765173

EstRTT(5) = (0.875 * 0.043765173) + (0.125 * 0.139894) = 0.055781276

EstRTT(6) = (0.875 * 0.055781276) + (0.125 * 0.189645) = 0.072514242

# Q4

see table in q3

# Q5

*What is the minimum amount of available buffer space advertised at the receiver for the entire trace?* **"Statistics -> TCP stream graphs -> window scaling" shows the recevier window size is at its minimum at the start. This appears to be 5840 bytes, indicated by packet 2**

*Does the lack of receiver buffer space ever throttle the sender?* **as the graph shows the windows size steadily increasing over duration, I do not beleive the window size is 0 at any stage, and hence there is always buffer space and the sender is not throttled (window size seems to always be greater than the segments)**

# Q6

*Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?* **No, tcp.analysis.retransmission returns no results. I did start manually by checking the sequence numbers but there are way too many so I used the previous method (and also looked at both of the "Statistics -> TCP stream graphs -> Time Sequence" graphs which seem to always increase over time)**

# Q7

*How much data does the receiver typically acknowledge in an ACK?* **typically 1460. I found this out by turning relative sequence numbers back on, and then starting from packet 9 added 1460 to the ACK number and this almost always derived the next ACK number (I knew to add 1460 because that's the average payload size and was always the result from doing packet10acknum - packet9acknum before turning relative numbers back on)**

*Can you identify cases where the receiver is ACKing every other received segment?* **the receiver seems to wait longer than usual between ACK's at packet 191 and 198, and this results in the ack at packet 198 ackowledging 2920 bytes (instead of the usual 1460, and it is no cooincidence that 2920 = 2 * 1460, i.e. it ACK'd every other packet in this occasion)**

# Q8

*What is the throughput (bytes transferred per unit time) for the TCP connection (explain how you calculated this value)? **the last ACK value is 164091 (relative), meaning 164000 bytes were transferred (minus 1 to account for the 1 ACK consumed in the 3 way handshake). This took 5.45 - 0.02 = 5.43 seconds i.e. time of last data packet minus time of first data packet not including handshake setup and disconnect packets. Hence the throughput was approximately 164000/5.43 = 30,202.57 bytes/sec or ~30kb/s**

PART 2

# Q1

*What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and server?* **2818463618**

# Q2

*What is the sequence number of the SYNACK segment sent by the server to the client computer in reply to the SYN?* **1247095790**

*What is the value of the Acknowledgement field in the SYNACK segment?* **2818463619**

*How did the server determine that value?* **2818463618+1 because it consumed the 1 byte that is used in the 3 way handshake. It is acknowledging that it got that byte and so now asking the next one(s)**

# Q3

*What is the sequence number of the ACK segment sent by the client computer in response to the SYNACK? **2818463619**

*What is the value of the Acknowledgment field in this ACK segment?* **1247095791**

*Does this segment contain any data?* **no, the next packet that is sent by the client has the same seq#, meaning no bytes were sent in this one**

# Q4

*Who has done the active close? client or the server? how you have determined this? **both the server and client initiate the close independantly, thus they are both doing active close. This is evident by the ACK number on the servers FIN (second FIN to occur) being the same as the seq# of the clients FIN (first fin) i.e. at the time of the server doing it's FIN it has not seen the client FIN, despite it having already occured**

*What type of closure has been performed? 3 Segment (FIN/FINACK/ACK), 4 Segment (FIN/ACK/FIN/ACK) or Simultaneous close?* **simultaneous, because they both active close, they each ACK the close of one another and it's done**

# Q5

*How many data bytes have been transferred from the client to the server and from the server to the client during the whole duration of the connection? What relationship does this have with the Initial Sequence Number and the final ACK received from the other side?* **As the sequence numbers are essentially a byte counter for "sent" (but starting at some random offset i.e. ISN) and ACK#'s are essentially byte counters for "delivered", the last ACK# - the first seq# gives you the total amount of bytes successfully sent over the connection. Hence the client to server bytes was 2818463653 - 2818463618 = 35 bytes and the server to client bytes was 1247095832 - 1247095790 = 42 bytes**