# TCP Basics :

1. The first segment is the number 4, and the last is 199 (the one seen as POST in Wireshark).
2. The IP address of the client is 192.168.1.102 and it uses the source port 1161.
3. The IP address of gaia is 128.119.245.12, and receiving on port 80 (since it’s an HTTP request) and sending on 1161.
4. The (absolute) sequence number of the TCP SYN segment is 232129012. The Flag 0x002 identifies the segment as SYN segment.
5. The (absolute) sequence number of the SYN ACK segment is 883061785. The value of the ACKnowledge field is 232129013 (which is the sequence number of the TCP SYN segment+1).

Gaia determines this value by using the sequence number of the client reaching it. In the segment, the flag 0x012 identifies it as the SYN ACK segment.

1. The sequence number of the segment containing the POST is: 232293053.
2. The sequence numbers of the first 6 segments are respectively:

(From frame 4, since when we have a look at the raw data, we see it is the first TCP segment to contain POST)

Delta1 (Sample RTT1): 0,02746

Delta2 (Sample RTT2): 0,035557

Delta3 (Sample RTT3): 0,070059

Delta4 (Sample RTT4): 0,114428

Delta5 (Sample RTT5): 0,139894

Detla6 (Sample RTT6): 0,189645

EstimatedRTT = (1-0.125) \* PreviousEstimatedRTT + 0.125 \* Sample RTT

EstimatedRTT1 = 0,02746

EstimatedRTT2 = 0.875\*0,02746 + 0.125\*0,035557 = 0.028472125

EstimatedRTT3 = 0.875\*0.028472125 + 0.125\*0,070059 = 0,033670484375

EstimatedRTT4 = 0.875\*0,033670484375 + 0.125\*0,114428 = 0,043765173828125

EstimatedRTT5 = 0.875\*0,043765173828125 + 0.125\*0,139894 = 0,055781277099609375

EstimatedRTT6 = 0.875\*0,055781277099609375 + 0.125\*0,189645 = 0,07251424246215820

|  |  |  |
| --- | --- | --- |
| Sequence number | Sent time | Ack received time |
| 232129013 | Aug 21, 2004 15:44:20,596858000 | Aug 21, 2004 15:44:20,624318000 |
| 232129578 | Aug 21, 2004 15:44:20,612118000 | Aug 21, 2004 15:44:20,647675000 |
| 232131038 | Aug 21, 2004 15:44:20,624407000 | Aug 21, 2004 15:44:20,694466000 |
| 232138498 | Aug 21, 2004 15:44:20,625071000 | Aug 21, 2004 15:44:20,739499000 |
| 232133958 | Aug 21, 2004 15:44:20,647786000 | Aug 21, 2004 15:44:20,787680000 |
| 232135418 | Aug 21, 2004 15:44:20,648538000 | Aug 21, 2004 15:44:20,838183000 |

1. Segment1: TCP Length = 565

Segment2: TCP Length = 1460

Segment3: TCP Length = 1460

Segment4: TCP Length = 1460

Segment5: TCP Length = 1460

Segment6: TCP Length = 1460

1. The minimum amount is 5840 bytes, and it's advertised by the receiver during the handshake (it's in the Window Size field). Since it's always > 0, it means there was no time t where the buffer was full.
2. There are no retransmissions. First, I checked the window size, and since it's always > 0, there is no packet drop. Then I checked for DUP ACKs, and in this trace we don't have them. Finally, I checked if it has a duplicate sequence number at sender side, trying to find retransmission a because of timeout.
3. To do this, we should do an average on the difference between each pair of consecutive ACK number sent by the receiver, however it looks like it acknowledges about 2000 bytes at a time.
4. During the TCP connection the throughput is roughly 23KB/s. We calculated it by taking the total number of bytes sent, dividing it with the whole TCP connection time. (the precise calculation is: 177851/7.6 = 23KB/s

Task B:

1. First, the two graphs are very different.

The second one for example seems to apply the theoretical slow start mechanism. It starts by sending a lot of packets at a very fast speed, and then (probably after some treshold) sends them linearly with the time. The window size at receiver size never decreases, so it's probable that the congestion phase never takes over. Moreover, besised just one DUPACK (in figure 2b), there are no packet retransmission (since the number of the sequence number is always strictly monotonically increasing). However, the two implementations work differently since the first doesn't seem to use the AIMD phase.

1. The RWND is the number of bytes the receiver buffer can handle. This way, enables the congestion window (CWND) to determine the amount of data it can reliably transmit without an ACK. A sender can send unacknowledged data as long as it doesn't overflow the RWND. In that case, it must wait for an ACK before sending other data, so that it will be sure not to fall into congestion avoidance mechanisms. The sender will adjust its "effective window" accordingly to the minumun value between (RWND and CWND).
2. From the traces is impossible to spot the CWND size. We can only assume that since in both traces, the sender never reduces its effective window (no CA mechanisms activated), then the CWND will never decrease. (MA PERCHE AUMENTA SEMPRE IL NUMERO DI BYTE MANDATI IN MANIERA LINEARE ?? SIGNIFICA CHE LA RWND STA AUMENTANDO ?? IO NON LO VEDO DAL COSO, ANZI RIMANE LA STESSA…SIGNIFICA CHE ALLORA LA MIA EFFECTIVE WINDOW ( O CWND perche rwnd sta sempre al top IN QUESTO CASO ) AUMENTA.

Task C :

1. The TCP is fair, in the sense that for each connection it will use pretty much the same amount of bandwidth. Supposing no other connection are going on the client, since the bandwidth in this case is equally shared, it will be 4 \* throughput.

Throughput is approximately: 317KB/s for all connections.

|  |  |
| --- | --- |
| **Connection** | **Throughput** |
| 1 | 2903,545 KB/s |
| 2 | 1955,509 KB/s |
| 3 | 1687,717 KB/s |
| 4 | 1559,872 KB/s |
| 5 | 1206,785 KB/s |
| 6 | 784,941 KB/s |
| 7 | 730,499 KB/s |
| 8 | 480,143 KB/s |
| 9 | 435,805 KB/s |

Supposing that the TCP is fair, it will give each connection the same amount of bandwidth. The reason of the low throughput is because of the Round-Trip Time.